End-to-End Error Correction and Online Diagnosis for On-Chip Networks

Saeed Shamshiri, Amirali Ghofrani, and Kwang-Ting Cheng
Department of Electrical and Computer Engineering
University of California, Santa Barbara, CA 93106
{saeedshamshiri, ghofrani, timcheng}@ece.ucsb.edu

Abstract

We propose a comprehensive solution for end-to-end (e2e) error correction and online defect diagnosis for on-chip networks. For e2e error correction, we propose an interleaved error-locality-aware code that efficiently corrects both random and burst errors. We demonstrate that for 64-bit wide network links, interleaving four of the proposed code, 2G4L(26,16), each of which supports 16-bit data, can correct as many as two random errors or 16 adjacent errors. Validated by the synthesis results, with the same code rate and cost as the traditional BCH code, the proposed 2G4L code provides higher reliability against burst errors.

In order to maintain the error correction capability of the Error Correcting Code (ECC) for transient and intermittent errors, we further propose an e2e data gathering and online diagnosis approach that locates the defective wires and replaces them with the spare wires embedded in the network. Our analytical and experimental studies show that under heavy noise, high escape rate, uncertainty about routing, and many other harmful effects, the diagnostic data collected by the proposed approach are accurate enough for the purpose of passive diagnosis.

1. Introduction

Multi-core processors, acclaimed as one of the top technologies of the decade [1], are quickly becoming dominant architectures for the next-generation high-performance processors [2-4]. Today's multi-core processors range from a few cores per chip, such as the 8-core IBM Cell processor [5], to tens and hundreds of cores, such as Intel's 80-core Terascale [6] and NVIDIA's 128-core Quadro [7]. To scale with the ever-increasing number of on-chip cores, network-on-chip (NoC) has become the most prominent architectural choice for the communication infrastructure of the multi-core SoCs [8-9].

Aggressive technology scaling has magnified the reliability challenges as it increases the number of permanent and transient faults due to the accelerated aging effects such as electro-migration, increased device variations, and significant noise margin reduction [10]. The growing concern about the reliability of NoCs has motivated many researchers to investigate innovative solutions to tolerating permanent or transient faults [11-13]. However, a comprehensive cost-efficient approach to address all reliability concerns has yet to be developed [14].

In an on-chip network, roughly 80% of the communication faults are transient [9]. Different fault tolerance approaches such as Forward Error Control (FEC), Automatic Repeat Query (ARQ), and multi-path routing have been used and compared in literature for reliable on-chip transmission [15-17]. These approaches tolerate transient faults, but they become ineffective in the presence of permanent faults. Permanent faults on wires occur both during manufacturing and in the field, causing yield degradation and service costs respectively. The overall system cost can be reduced by adding some spare wires per each link of the network to replace the defective wires [15,18]. Nevertheless, an in-field diagnosis mechanism is required to locate the defective wire and initiates the wire replacement.

The reliability requirement can be addressed in the data link layer, in a switch-to-switch (s2s) basis, or in the transport layer, in an end-to-end (e2e) fashion [11]. For example, automatic repeat query, and multi-path routing work on the e2e level, while forward error control methods, such as Error Correcting Codes (ECC), can be implemented in the s2s or e2e levels [15-17]. In the s2s approach, the encoders (decoders) of the ECC should be implemented in the output (input) ports of the routers of the NoC, while in the e2e scheme, the encoders and decoders of the ECC are implemented in the network interfaces of the cores. In the mesh architecture, the total number of input-output ports is about four times the number of cores; therefore, the hardware overhead of s2s ECC is about four times the overhead of e2e ECC.

In this paper, first, we propose a new error correcting code to provide e2e fault-tolerance in NoC communications. Second, we propose an on-line mechanism to collect information about the observed error patterns and use it for diagnosis of the defective wires. Once a defective wire is identified, it can be replaced by an embedded spare wire.

During the past several decades, the concept of error correcting codes has been used in variety of applications from satellite communications, magnetic storage device, on-chip memories, to on-chip communications. Hamming codes were proposed in 1960 for single error correction and since then, due to their simplicity, they have been one of the most commonly used codes. For multiple random error corrections (i.e. when errors are uncorrelated),

Double-Error-Correcting (DEC) BCH (Bose-Chaudhuri-Hocquenghem) [19-20], and Triple-Error-Correcting (TEC) Golay code [21] have been used for on-chip memories. For burst error corrections (i.e. where errors are spatially or temporally correlated), Abramson's [22], Reiger's [23], and Gilbert codes [24] were proposed for telephone lines and magnetic tapes. It has been shown that Reiger's [23], Reed-Solomon [25], Gilbert codes [24], and similar burst codes such as product codes [26] and array codes [27-28], provide some reliability against random errors and burst errors [29]. However, their random and burst error correction capability are not independent, and hence, they can't be efficiently correct a given number of random and burst errors [30]. In [30], for a given set of requirements such as the size of the codeword and the number of random and burst error corrections, a Boolean satisfiability approach has been taken to find an efficient error correcting code for on-chip memories. The proposed code, named *error-locality-aware* code provides immunity against errors which are more likely to happen. For example, 2G4L(26,16) is an error-locality-aware code designed for 16-bit memory words to correct two global (i.e. random) or four local errors (i.e. a burst error of size four). With the same overhead as the Double Error Correcting (DEC) BCH(26,16) [20], 2G4L(26,16) provides extra reliability against burst errors. A well-known alternative to provide burst error correction is code interleaving which has been widely used for on-chip memories to immune them against radiation-induced single-event multiple upsets [31-33].

In this paper, we propose to use four interleaved error-locality-aware 2G4L(26,16) codes for an exemplary 64-bit wide links of NoC. This fault-tolerance scheme offers the capability of two random bit error correction plus upto 16 bit burst error correction. Moreover, in order to maintain the error correction capability of the proposed ECC for transient and intermittent errors, we propose an e2e data collection and online diagnosis approach that locates the defective wires and triggers the replacement of them with the spare wires. Our analytical and experimental studies show that under heavy noise, a high escape rate, uncertainty about routing, and several other harmful effects, the diagnostic data collected by the proposed approach are sufficiently accurate to support passive diagnosis (i.e. online diagnosis without interrupting or interfering with the normal-mode operations). An online diagnosis approach has been proposed in [34] to diagnose the faulty switches and deflect the corresponding packet routings accordingly. In this paper, we address the diagnosis problem of faulty wires based on e2e information which is a new approach and should lead to a more cost-effective solution.

In Section 2, we give an overview for the types of errors occurred in an NoC. Then, we briefly summarize the ways of addressing reliability in Section 3. In Section 4, we propose a new code to be used for e2e error correction. In Section 5, we provide an analytical and experimental study to show the feasibility of online diagnosis based on passive data gathering of e2e error patterns. This follows by a conclusion in Section 6.

2. Sources of Errors

Communication errors in on-chip networks are due to the several causes:

1. *Crosstalk*: Crosstalk is the most common type of error for on-chip communications. In the worst case, crosstalk can change a bit pattern like 10101010 to 11111111 (or 00000000). So, the maximum number of errors would be *n/2* where *n* is the bandwidth of the link. In practice, the number of crosstalk errors is likely to be far less than this worst-case. The most common types of crosstalk errors are multiple-aggressor single-victim, and single-aggressor single-victim. Both types manifest as random errors rather than burst. Crosstalk can also manifest as a burst error (multiple/single aggressor, multiple victims), but it is not very likely to happen.

2. *Radiation-induced soft errors*: Radiation-induced soft errors can disturb several adjacent wires at the same time. This type of error has a strong locality in its appearance and can be modeled as a burst error [30].

3. *Voltage-induced delay errors*: In low-power designs, lowering the operational voltage of the links of an on-chip network delays the signal propagation through the wire and can manifest as random or burst errors on the communication channel.

4. *Hard errors*: hard errors are permanent errors which last forever. Some of the hard errors are manufacturing defects (e.g. bridging between adjacent wires) that can be detected, and possibly repaired by replacement with spare wires, duriung manufacturing testing. Another type of hard errors occurs after shipment due to infant mortality, aging, and stress in the field. The burn-in process detects most of the infant-mortality defects, but still there could be some escapes. The wear-out defects (a.k.a. aging) behave as random or burst errors.

A comprehensive reliability approach should address all of the above error types.

3. Reliability Approaches

In an on-chip network, reliability of communications can be provided at different layers (e.g. data link or transport layer). Moreover, the employed approach can be in the form of error correction, or detection-and-recovery.

3.1. Correction vs. Detection-&-Recovery

Errors can be detected and corrected using Forward Error Correction (FEC) approaches such as Hamming codes. On the other hand, we can use a simpler and cheaper approach such as even parity just for error detection, and in case of observing error, recovery can be done by requesting the source to resend the data.

In this work, we want to differentiate defects from soft errors in order to replace defective wires with spare wires. Therefore, error detection and recovery approaches do not suffice. In addition to error correction, we need to have a mechanism to differentiate between hard and soft errors, and further locate the faulty wires in case of hard errors,

3.2. Switch-to-Switch vs. End-to-End

Switch-to-switch (s2s): Reliable communication can be supported in the data link layer by adding redundancy to the s2s flits. For example, to implement a Single Error Correction (SEC) Hamming (71,64) on an NoC with 64-bit wide links, seven parity wires need to be added to every link of the network (resulting in a total of 71 wires per link). Moreover, each output port of each router needs a Hamming encoder and each input port of each router needs a Hamming decoder. An s2s ECC avoids error accumulation along the route, but it incurs significant overhead.

End-to-end (e2e): Reliable communications can be supported in the transport layer in packet level. This way, the redundant parity bits are part of the packet and they don't impose hardware overhead. The e2e ECC incurs lower overhead because it only requires each network interface, instead of each port, to have an encoder and a decoder.

In this paper we focus on e2e reliability. We propose a solution for e2e error correction, as well as for diagnostic data collection and analysis without relying on s2s ECCs.

3.3. Size of the Codeword

In addition to the network layer in which we implement the ECC, we also need to decide the size of the block on which the ECC is applied. For example, for s2s reliability of a 64-bit link, we can either apply Hamming (71,64), or break the 64 bits into two halves and apply two SEC Hamming (38,32). Using smaller blocks makes the encoder and decoder simpler and faster, but increases the number of parity bits and thus reduces the code rate (i.e. the ratio of the data bits to the encoded bits).

To evaluate the effect of the block size on area, power consumption, and latency of the encoder and decoder, we implemented SEC Hamming codes of different sizes and synthesized them using Synopsys Design Compiler with a 180nm standard cell library.Table I shows the results of the SEC Hamming (n,k) for k=4 to 256 for which k bits of data are encoded into n bits of codeword with a code rate of k/n.

Figure 1 shows the total area, power consumption, and latency of Hamming encoders and decoders of different sizes normalized to those of the Hamming (7,4). As this figure illustrates, the trend of power consumption increase is more significant than the increases of area and latency. With 64x growth in data width from 4-bit to 256-bit, the power consumption grows 140x, while the area and latency grow 75x and 12x respectively. Partitioning the data into smaller blocks is clearly a better strategy for lowering power consumption.

Table I: Area, power, and latency of the encoders and decoders of Hamming (n,k) for k=4 to 256.

	Area (um^2)		Dynamic Power (mW)		Latency (ns)	
H(n,k)	Encoder	Decoder	Encoder	Decoder	Encoder	Decoder
H(7,4)	280	864	0.128	0.470	0.31	0.67
H(12,8)	672	1784	0.362	1.113	0.45	1.04
H(21,16)	1512	3676	1.070	2.798	0.65	1.58
H(38,32)	3360	6940	3.029	5.717	0.99	2.02
H(71,64)	7112	13644	7.693	11.591	1.57	2.90
H(136,128)	14896	27124	17.025	23.492	1.89	4.76
H(265,256)	31048	54676	36.770	47.027	3.08	8.57

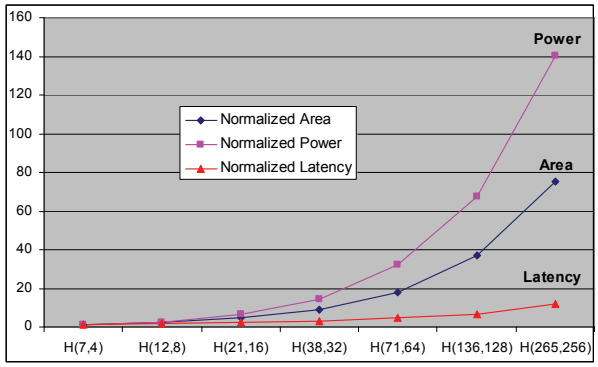

Figure 1. Area, power, and latency of Hamming (n,k) for k=4 to 256, normalized to those of Hamming (7,4).

For our exemplary 64-bit link, Figure 2 shows the effect of partitioning on area, power, latency, and the code rate of the SEC coding. As illustrated in this figure, while the total area of eight Hamming (12,8) is almost the same as that of Hamming (71,64), its total power consumption and latency is much lower. The major drawback of partitioning is that the code rate also decreases, meaning that more parity bits are needed; eight Hamming (12,8) use a total of 32 parities, while Hamming (71,64) has just seven parities. In s2s ECCs, we need extra wires for parity bits which consume power in data communication. Therefore, there is a tradeoff between the power consumption of the encoder/decoder and that of the link. Increasing the number of partitions reduces the former while increases the later.

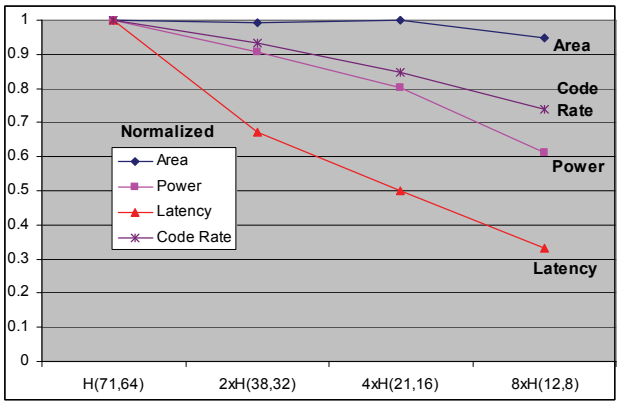

Figure 2. The effect of partitioning on the area, power, latency, and code rate of Hamming codes for a 64-bit link.

Figure 3 further investigates the effects of partitioning on the power, area, latency, and code rate of the codec of Hamming codes for the links of sizes 32 to 256 bits.

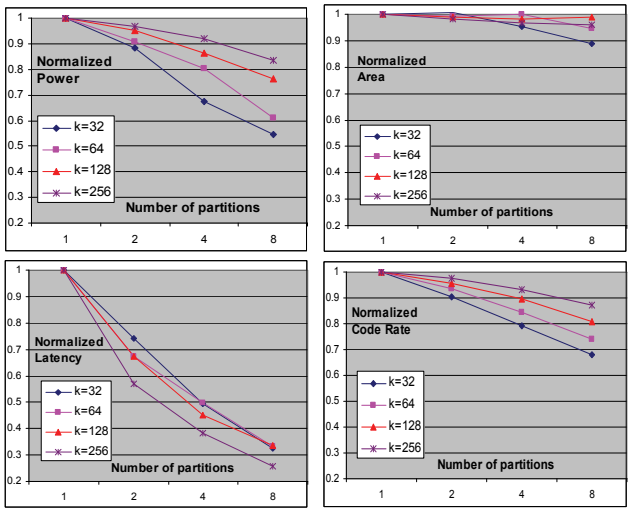

Figure 3. The effects of partitioning on the power, area, latency, and code rate of Hamming codes for links of sizes *k*=32 to 256 bits.

In addition to the reduction in power and latency for the encoder/decoder, another benefit of partitioning is the extra correction capability. H(71,64) provides a single error correction capability on a 64-bit link; however, four H(21,16)'s offer upto four bit corrections as long as the erroneous bits occur in different partitions. By interleaving the four partitions, 4xH(21,16) provides 4-bit burst error correction capability. As illustrated in Figure 4, if there is a burst error of 4-bit wide, these four erroneous bits will fall into 4 different partitions, and hence will be corrected by Hamming code. In Figure 4, different colors represent different partitions.

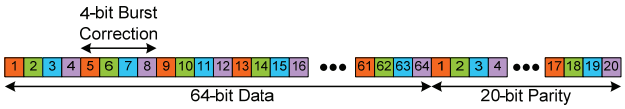

Figure 4. Four interleaved SEC Hamming(21,16) provides 4-bit burst error correction. Different colors represent different partitions.

4. Error Correcting Codes

The error correcting code for NoC should cover both random and burst errors in order to address all types of errors discussed in Section 2.

In [30], we showed that the most efficient code to cover both random and burst errors is the one specifically designed for the required size and the required level of reliability in terms of number of global and local error corrections. This code, named error-locality-aware code, has originally been suggested for on-chip memories [30]. In this paper, we propose to apply this code for e2e reliability of on-chip communications.

4.1. Error-Locality-Aware Codes

If both random and burst errors are common, it would be more effective to use a code which is specifically designed to deal with both types of errors. It has been shown that the most optimized code for a given requirement of random and burst error correction can be calculated using a Boolean satisfiability approach [30].

Figure 5 compares the number of error syndromes for different number of random and burst error corrections. The top-most curve, *A(g)*, shows the number of syndromes for *g* global error detection, while the most bottom curve, *B(l)*, represents the number of syndromes for *l* local error corrections. All intermediate curves, such as *B(g,l)*, represent the number of syndromes for *g* global error and *l* local error corrections. Figure 5 suggests that, for example, instead of using double error correcting (DEC) BCH(26,16), we can improve the code rate by 25% and use BurstCode(26,20) which corrects two adjacent errors. Another alternative is to use 2G4L(26,16) which has the same code rate as BCH(26,16), but on top of double random error correction, it corrects up to four local (i.e. adjacent) errors. 2G4L is a type of error-locality-aware code which targets corrections for errors which are more likely to happen.

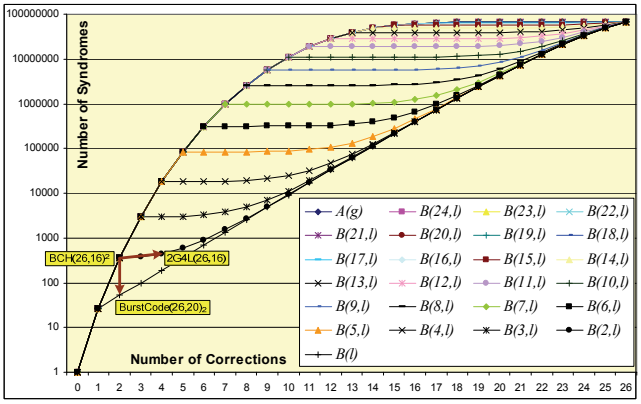

Figure 5. Error-Locality-Aware codes compared with random codes and burst codes for a 26-bit codeword [30].

Tables III and IV show the design of the parity check matrices for 2G4L(26,16) and BCH(26,16) respectively. Figures 6 and 7 show the hardware implementation of the encoder and decoder of 2G4L(26,16). BCH(26,16) can also be implemented in a similar way. We have implemented both of these codes in a 180nm standard cell library and compared their post-synthesis results using Synopsis Design Compiler in Table II. The results show that these two codes incur similar costs in terms of area, power consumption, and latency. Since the 2G4L(26,16) provides more reliability against burst errors, we suggest to use this code instead of the famous BCH.

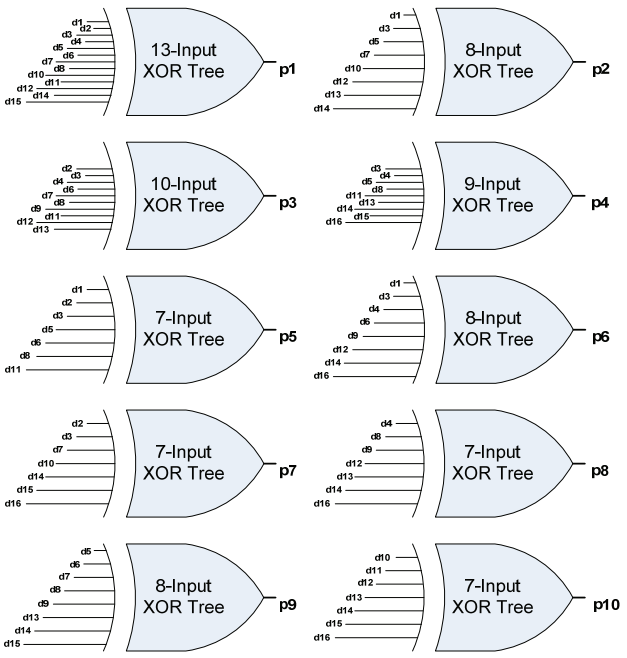

Figure 6. The implementation of the 2G4L(26,16) encoder.

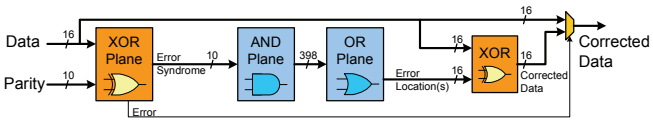

Figure 7. The implementation of the 2G4L(26,16) decoder.

Table II: Comparing area, power, and latency of the 2G4L(26,16) and BCH(26,16) in 180nm technology.

	Area (um^2)		Power (mW)		Latency (ns)	
	Encoder	Decoder	Encoder	Decoder	Encoder	Decoder
BCH(26,16)	2872	21043	2.5107	13.758	0.9	3.4
2G4L(26,16)	2744	22173	2.2365	14.0756	0.78	3.35

4.2. The Proposed E2E ECC

Although the e2e coding is in the packet level, it doesn't necessarily mean that the code takes the entire packet as a single codeword. A packet could be relatively large and using the entire packet as a single codeword, as discussed in Section 3.3, could make the encoder and decoder too large, power hungry, and very slow. We can break the packet into smaller blocks to address this problem. Moreover, packets could have different lengths and so it would be more efficient to partition the packet into fixed-size smaller pieces and run the ECC on each of them, rather than running the ECC on the entire packet.

For the e2e reliability of a network with 64-bit wide links, we propose using 2G4L(26,16) for each one of the four interleaved 16-bit data of a flit. The parity bits (10 parity bits for each 16 data bits) are included as a part of the header of the packet in an interleaved fashion. Interleaving increases the burst error correction capability of the code. In this case, each 2G4L code corrects a burst of size four; by interleaving four of them, the total error correction will improve to a burst error of size 16. Figure 8 shows the proposed e2e ECC scheme for a packet that originally was eight flits long (i.e. seven data flits and one header flit). With the added redundancy, the size of the packet increase to 13 flits.

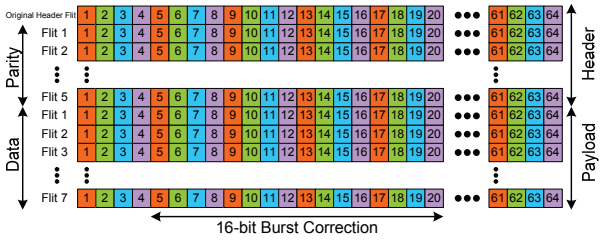

Figure 8. The proposed e2e ECC scheme: four interleaved 2G4L(26,16) for each flit; total of 32 2G4L(26,16) per packet.

Table III: The design of the 2G4L(26,16).

Bit Position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
Endoded Data Bits		p1	p2	p3	p4	p5	p6	p7	p8	d1	d2	d3	d4	p9	d5	d6	d7	d8	d9	p10	d10	d11	d12	d13	d14	d15	d16
Parity bit coverage	p1	1	0	0	0	0	0	0	0	1	1	1	1	0	1	1	1	1	0	0	1	1	1	0	1	1	0
	p2	0	1	0	0	0	0	0	0	1	0	1	0	0	1	0	1	0	0	0	1	0	1	1	1	0	0
	p3	0	0	1	0	0	0	0	0	0	1	1	1	0	0	1	1	1	0	0	0	1	1	1	0	0	0
	p4	0	0	0	1	0	0	0	0	0	0	1	1	0	1	0	0	1	0	0	0	1	0	1	1	1	1
	p5	0	0	0	0	1	0	0	0	1	1	1	0	0	1	1	0	1	0	0	0	1	0	0	0	0	0
	p6	0	0	0	0	0	1	0	0	1	0	1	1	0	0	1	0	0	1	0	0	0	1	0	1	0	1
	p7	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	1	0	0	0	1	0	0	0	1	1	1
	p8	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	1	0	0	0	1	1	1	0	1
	p9	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	1	1	1	0
	p10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
Decimal		1	2	4	8	16	32	64	128	51	85	127	173	256	283	309	327	413	420	512	579	541	679	910	1003	841	744

Table IV: Parallel implementation of BCH(26,16) adapted from [20].

Bit Position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
Endoded Data Bits		p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10	d11	d12	d13	d14	d15	d16
Parity bit coverage	p1	1	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	0	1	1	1	1	0	0	1	0	0
	p2	0	1	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	0	1	1	1	1	0	0	1	0
	p3	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	0	1	1	1	1	0	0	1
	p4	0	0	0	1	0	0	0	0	0	0	1	1	0	0	1	1	1	1	0	1	0	1	1	0	0	0
	p5	0	0	0	0	1	0	0	0	0	0	0	1	1	0	0	1	1	1	1	0	1	0	1	1	0	0
	p6	0	0	0	0	0	1	0	0	0	0	1	1	1	0	0	1	1	0	0	0	1	1	0	0	1	0
	p7	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	1	1	0	1	1	1	1	1	1	0	1
	p8	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	1	1	0	1	1	1	1	1	1	0
	p9	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	0	0	0	1	0	1	1	0	1	1
	p10	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1	0	1	1	1	1	0	0	1	0	0	1
Decimal		1	2	4	8	16	32	64	128	256	512	873	443	886	389	778	381	762	669	595	975	247	494	988	209	418	836

5. Online Diagnosis

Diagnosis of defective wires enables us to replace them by spare wires. This way we can avoid degradation of the error correction capability of the embedded ECC method. Diagnosis can be done at s2s or e2e levels.

Switch-to-switch diagnosis: If there is a s2s ECC to locate the faulty wire, the only extra effort to diagnose defects is to distinguish between transient and permanent errors. This can be done by tracking the wires which have been recently faulty. If a wire is frequently faulty, it is probably permanently faulty. A wire replacement protocol will be triggered after locating a permanently faulty wire.

Although s2s diagnosis is rather trivial, it requires a s2s ECC which is more expensive that e2e ECC. In this paper, we address the diagnosis problem based on the e2e information.

5.1. End-to-End Diagnosis

e2e diagnosis is not as trivial as s2s. We propose a probabilistic methodology to locate the defective wire without relying on s2s error correction or detection. Let's explain the idea through an example.

Example 1: Figure 9.a shows a 4-by-4 mesh with three packets, R(Red), G (Green), and B (Blue), routed with an XY routing scheme. We number the links of the mesh from 1 to 48. Each s2s link is 64-bit wide in each direction. Using an e2e ECC, which was explained in Section 4.2, the network interface of the destination node observes the error pattern on each packet. Figure 9.b shows an exemplary error pattern observed on packet G. In this example, packet G is composed of 13 flits and has seven erroneous bits. Two of the errors are randomly scattered, but the other five errors are under the same column (i.e. same bit position). This suggests that probably there is a faulty wire at bit position 12 in a link along the path that packet G has traveled. A faulty wire with a *SA0* behavior introduces errors only when the wire carries a logic '1'. Therefore, when the number of erroneous bits under the same column is greater than a threshold *b*, there is a good chance of having a permanent error on a wire at the corresponding position along the path.

The value of *b* affects the e2e diagnosis accuracy in two ways: If *b* is too small (e.g. *b*=2), to the probability of misdiagnosis of a transient error as a permanent fault would be high. If *b* is too large (e.g. larger than 50% of the number of total flits transferred), the chance of missing a permanent fault would be non-trivial. In the next section, we will discuss the effects of e2e mis-diagnosis on the accuracy of system diagnosis.

Whenever a destination node observes a potential permanent error on a bit position, it sends a control message to the host processor indicating the bit position of the error(s) and the source and destination of the erroneous packet. In an XY routing, the source and destination of a packet uniquely determine the route that the packet has taken. In this example, packet G had passed through links 7, 8, 9, and 23.

The host processor is one of the cores of the mesh that acts as a master and dispatches tasks to other cores. In the proposed scheme, the host processor keeps a scoreboard of the suspected wires and runs the diagnosis task in software. For all wires of the mesh (i.e. 3072 wires in this example: 48 links times 64 wires per link), there is a number in the scoreboard that shows the number of times that a wire was reported as a potentially defective wire. When the host receives a message about packet G, indicating that the 12th wire along the route is defective, it increments the values in the scoreboard for links 7, 8, 9, and 23 at bit position 12.

Figure 9.c shows the suspected bit positions observed at the destination of the each of the three packets in this example. Figure 9.d shows the contents of the scoreboard after these three erroneous packet transfers. Using the scoreboard, the host can observe that wire 12 of link 8 is more likely to be defective because it was a suspected wire twice. Using this voting strategy, after several packets pass through the system, the host is able to diagnose the defective wire with a reasonably high accuracy. Then, the host initiates a command to both nodes connected to the defective wire to further test the suspected wire in the active diagnosis mode and finally replace the defective wire with a spare wire. Note that similar to [15] and [18], we incorporate several spare wires for each link of the mesh.

One of the decisions that the host has to make is when to switch from passive diagnosis to active diagnosis. This decision should be made based on the accuracy of the diagnosis with respect to the utilization of the network; this will be studied later in the paper.

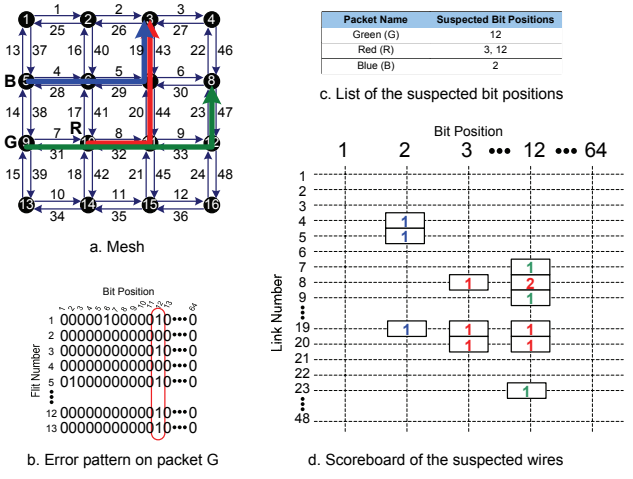

Figure 9. (a) Three packet transfers on a 4-by-4 mesh; (b) Error pattern observed on packet G; (c) The list of the suspected bit positions; (d) A scoreboard that keeps track of the suspected wires.

The maximum number in the scoreboard points to the most suspicious wire which is most likely a defective wire. The other suspected wires on the scoreboard, which don't get enough votes to be concluded as defective, are assumed to be defect-free. The error reported on them was either due to a transient error or a permanent error on another wire along the path that they shared. To alleviate the accumulation of the transient errors on the scoreboard, and to avoid confusing them with permanent errors, the scoreboard supports aging, meaning that once in a while periodically the host decrements all the values in the scoreboard by one.

If there are more than one defective wires on the network, the scoreboard finds one of them first and initiates the replacement command. Later, the host will be able to diagnose the second defective wire for replacement.

In this example, for simplicity, for each observed defective packet, we increase the values of the suspected wires by one. However, it is more accurate to increase the values of suspected wires by *1/d*, where *d* is the length of the packet transfer. If a defective packet only passes through one link, then that link is definitely defective; however, if a defective packet comes from a route of length five, then each of the five links along that route has a 20% chance of being defective.

Note that for most of the packet transfers, none of the wires will be suspected as a defective wire, and so there would be no need to send a message to host or to update the scoreboard. These control message transfers and scoreboard updates happen very rarely for a healthy network and doesn't cause a noticeable performance degradation on the on-chip network.

In order to collect e2e reliability data properly, the control messages need to stay error-free in the presence of defective wires. To satisfy this requirement, each control message is being sent three times using multi-path routing [17]. Then the host can construct the error-free message by voting.

So far, based on our discussion, the proposed diagnosis approach requires a deterministic routing like XY routing. XY routing is a widely suggested routing algorithm for NoCs; however, to avoid congestion and to provide a better quality of service and reliable communications, the packet may take a detour or use a different routing scheme [34-36].

If the routing is not 100% deterministic, in case of observing an error, the host doesn't know the exact route that the packet has taken from source to destination to properly update the scoreboard. In the following, we provide a probabilistic study to prove that even if the actual route is unknown, we can still use the collected data to accurately diagnose defective wires.

If the route from the source to the destination is unknown, the best bet is that the packet has taken one of the many minimum-length routes. We assume that the chance of taking any of the minimum routes is equal. We name this the *Min-Route* algorithm.

In Min-Route, we don't know the actual route, but we know the probability that a link has been used in a packet transfer. We call this the *usage probability* of the link. When a permanent error is observed at a packet destination and being reported to the host, using the usage probability of each link, and the length of the packet transfer, the host can calculate the probability of each link being the cause of the observed error. Figure 10 shows an example of an erroneous packet transfer in Min-Route algorithm.

Example 2: In the 4-by-4 mesh of Figure 10.a, there are six minimum routes from source *s* to destination *d*. Figure 10.a shows the number of routes that passes through each link. If we divide these numbers by the number of routes, we will have the usage probability of each link (shown in Figure 10.b). Assuming that link *AB* is defective, the chance of observing an error at destination is equal to the usage probability of *AB* which is 33%. When *d* observes the error, the chance of each link being the cause of the error is equal to the usage probability of the link divided by the length of the packet transfer. Therefore, the expected value for each link in the scoreboard is equal to the chance of detection (i.e. 33%) multiplied by the usage probability of the link divided by the length of the packet transfer (i.e. four in this example). Figure 10.c shows the expected values of the scoreboard for each of the links.

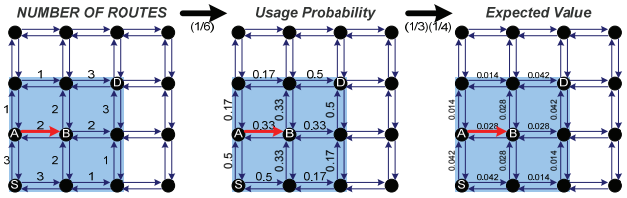

Figure 10. A packet transfer from *S* to *D* using a Min-Route algorithm given that link *AB* is defective; (a) The number of routes that passes through each link; (b) The usage probability of each link; (c) The expected values of the scoreboard for each link.

Usage probability of the links of a mesh can be calculated for a random packet transfer (i.e. a packet from a random source to a random destination) according to the routing algorithm [37]. Figure 11 shows the usage probability of the horizontal links of an 8-by-8 mesh for the XY-Route and Min-Route algorithms. Using the usage probability of the links, we can calculate the expected values of the scoreboard for any given defective link. Figure 12 shows the expected values of the links of an 8-by-8 mesh in Min-Route assuming that the link *AB* is defective. According to this analysis, the expected value of *AB* is significantly higher than those of the other links which means that the host is able to successfully diagnose *AB* as the defective link. Figure 13 gives a 3-D view of the expected values for the right-going and up-going links of the mesh of Figure 12. Figure 14 plots the expected values of the links of the same 8-by-8 mesh, for XY routing.

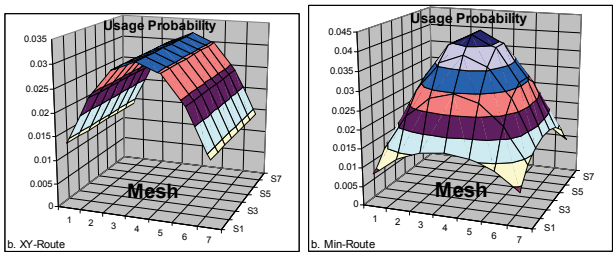

Figure 11. The average usage probability of the horizontal links of an 8-by-8 mesh for the XY-Route and Min-Route adapted from [37].

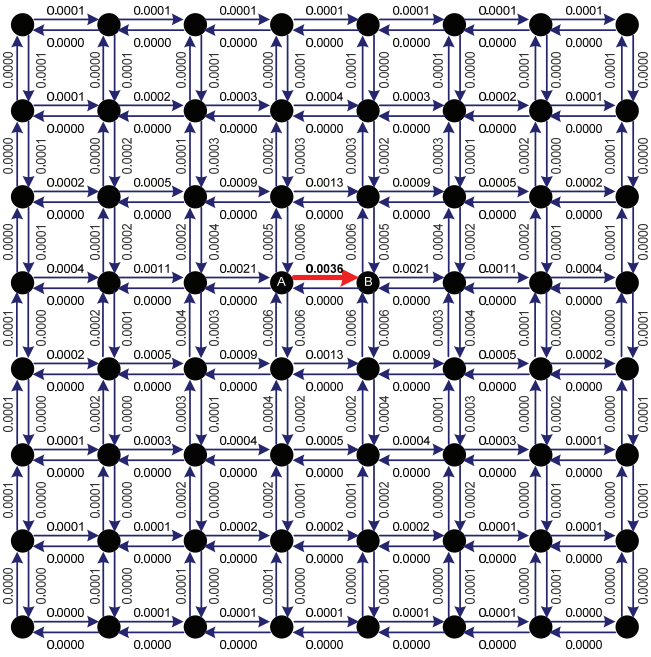

Figure 12. The expected values of the scoreboard for the links of an 8-by-8 mesh given that link *AB* is defective for Min-Route.

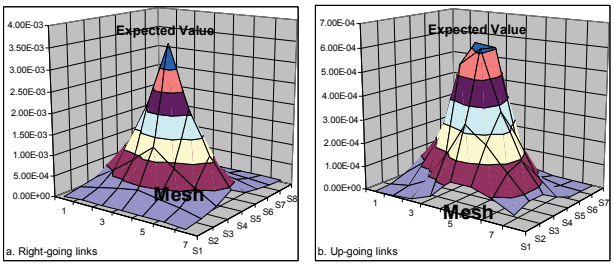

Figure 13. The expected values of the scoreboard for the links of an 8-by-8 mesh given that link *AB* (in Figure 12) is defective for the Min-Route algorithm; (a) expected values of right-going links; (b) expected values of up-going links.

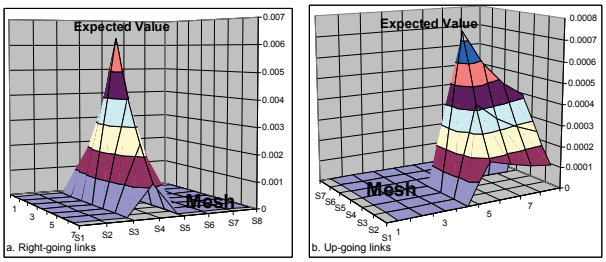

Figure 14. The expected values of the scoreboard for the links of an 8-by-8 mesh given that link *AB* (in Figure 12) is defective for XY routing; (a) expected values of right-going links; (b) expected values of up-going links.

5.2. Accuracy of Diagnosis

In the previous section, we illustrated the feasibility of diagnosis even when the routing is not deterministic. It was done through calculation of expected values in the scoreboard. In this section, we show Monte Carlo simulation results which measure the diagnosis accuracy with respect to parameters such as the routing algorithm, network utilization, transient error rates, and etcetera.

Figure 15 shows the average accuracy of diagnosis with respect to the number of packets that pass through the network for three different routing strategies: the XY-Route, the Min-Route, and a Hybrid-Route in which the packets are routed using the XY-Route for 50% of the time and a non-deterministic Min-Route for the other 50% of the time. As this figure depicts, after only 500 packets, the average accuracy of diagnosis exceeds 95% for all three routing schemes. Note that 500 is the total number of packets that pass through the network, many of which don't pass through the defective link.

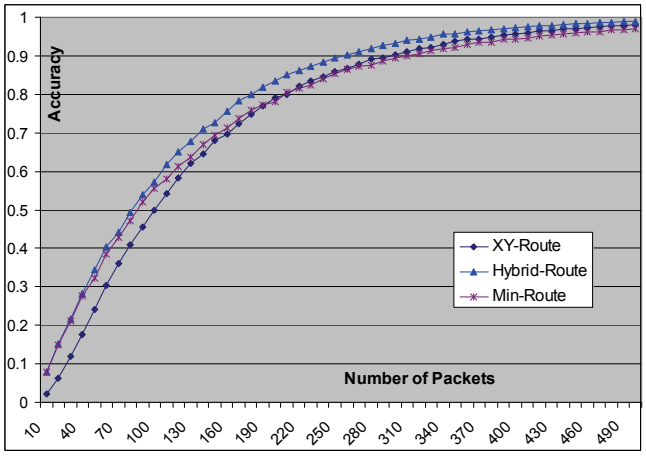

Figure 15. The accuracy of diagnosis increases as more packets pass through the network.

Since the usage probability of the links depends on the location of the link [37], the diagnosis accuracy also varies with respect to the location of the defective link. Figure 16 shows the diagnosis accuracy of the right-going links of an 8-by-8 mesh for the XY-Route and the Min-Route algorithms.

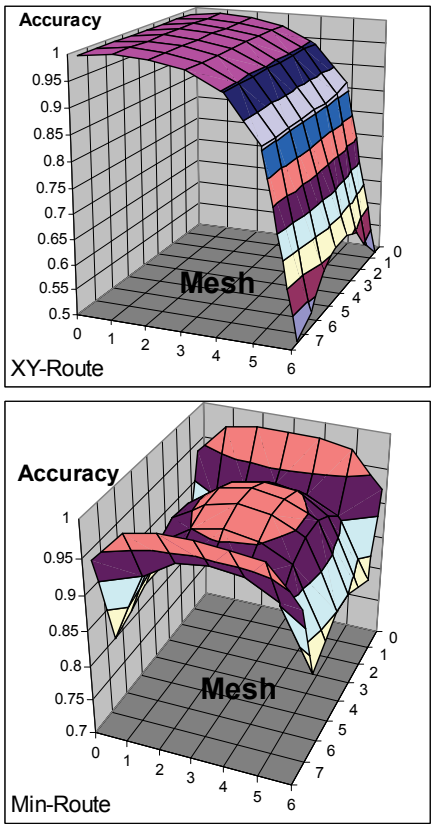

Figure 16. The diagnosis accuracy depends on the location of the defective wire on the mesh.

Another parameter that affects diagnosis accuracy is the chance of missing a defect when observing the packet at destination. This is the escape rate of the e2e defect observation and it depends on the value of *b*, defined in the previous section.

Finally, there are a group of other factors that act as *noises* and degrade the diagnosis accuracy. One source of *noise* could be transient or intermittent errors which are mis-diagnosed as defects, or it could be a defective router that corrupts the packets as they pass through. Another source of noises could be due to the fact that the packet passes via a route other than any of the minimum routes. Although these noises might degrade the diagnosis accuracy, their effect is not significant because our approach is probabilistic in nature. Figure 17 shows an experiment on an 8-by-8 mesh with the XY routing in which we study the effects of noises and the escape rate on the diagnosis accuracy. In this experiment we increase the noise to 50% meaning that half of the information that we collect in the scoreboard are caused by the noises which therefore point us to a wrong direction. As this figure shows, even with noise as high as 50%, there is no significant effect on the accuracy. The reason is that the noisy data are scattered on different bit positions of the 64-bit links, while the actual noise-free data are all collected on the same bit-position.

Figure 17 also shows that with upto a 40% escape rate for e2e error observation, the accuracy is still at about 90%.

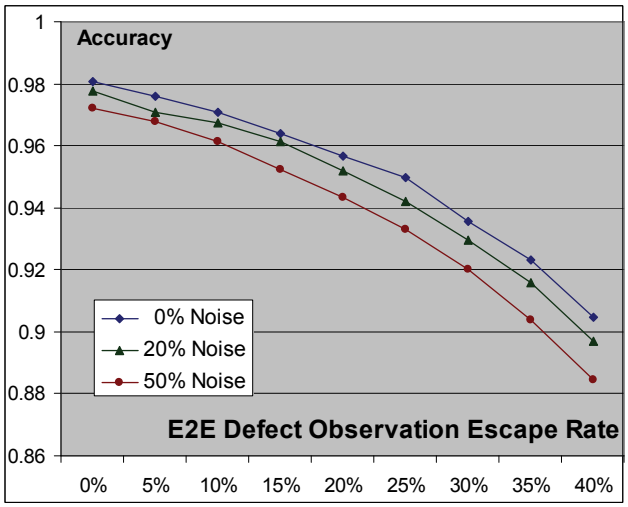

Figure 17. The accuracy of diagnosis w.r.t. the escape rate and noise.

6. Conclusion

In this paper, we propose a comprehensive end-to-end solution for error correction, data collection, and defect diagnosis and replacement for on-chip networks.

For e2e error correction, we propose to use four interleaved 2G4L(26,16)'s which provide two random and upto 16 adjacent-bit error corrections per flit. With the same code rate and cost as BCH(26,16), 2G4L(26,16) provides higher reliability against burst errors.

Furthermore, we show that the error pattern information that each packet destination observes can be gathered in a centralized software on the host processor and be used for diagnosis of defective wires. Our analytical and experimental studies show that under heavy noise, high escape rate, uncertainty about routing, and many other harmful effects, the collected data are still accurate enough for the purpose of passive diagnosis.

The collected data can also be used for other purposes such as diagnosis of defective routers, locating the intermittent faults, and many other interesting system observations.

7. Acknowledgement

The authors acknowledge the support of the National Science Foundation Center for Domain-Specific Computing (CDSC) and the Gigascale Systems Research Center (GSRC), one of six research centers funded under the Focus Center Research Program (FCRP), a Semiconductor Research Corporation entity.

8. References

[1] S.K. Moore, "Top 11 technologies of the decade; #5: Multicore CPUs", *IEEE Spectrum*, vol. 48, no. 1, pp. 40-42, Jan. 2011.

[2] V. Agarwal, M.S. Hrishikesh, S.W. Keckler, and D. Burger, "Clock rate versus IPC: the end of the road for conventional microarchitecture," *Proc. IEEE International Symposium on Computer Architecture (ISCA)*, pp. 248-259, 2000.

[3] K. Olukotun and L. Hammond, "The future of microprocessors," *ACM Queue Magazine*, vol. 3, no. 7, Sep. 2005.

[4] R. Kumar, D.M. Tullsen, N.P. Jouppi, and P. Ranganathan, "Heterogeneous chip multiprocessors," *IEEE Computer*, vol. 38, no. 11, pp. 32-38, Nov. 2005.

[5] D. Pham *et al*., "The design and implementation of a first-generation CELL processor – A multi-core SoC," *Proc. IEEE International Conference on Integrated Circuit and Technology,* pp. 49-52, 2005.

[6] S. Vangal *et al.*, "An 80-tile 1.28TFLOPS network-on-chip in 65nm CMOS," *Proc. IEEE Int. Solid-State Circuits Conference*, pp. 98-99, 589, 2007.

[7] NVIDIA Quadro FX 5600. http://www.nvidia.com/docs/IO/40049/quadro_fx_5600_datasheet.pdf

[8] L. Benini and G. De Micheli, "Networks on chips: a new SoC paradigm," *IEEE Computer*, vol. 35, pp. 70-78, 2002.

[9] G. De Micheli and L. Benini, *Networks on Chips.* Morgan Kaufmann Publishers, 2006.

[10] International Technology Roadmap for Semiconductors (I.T.R.S), 2009 Edition, Test and Test Equipment, 2009.

[11] 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 of Computers*, vol. 22, no. 5, pp. 434-442, 2005.

[12] R. Marculescu, "Networks-on-chip: the quest for on-chip fault-tolerant communication," *Proc. of the Symposium on VLSI*, pp. 8-12, 2003.

[13] T. Dumitras, S. Kerner, and R. Marculescu, "Towards on-chip fault-tolerant communication," *Proc. of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 225-232, 2003.

[14] D. Park, C. Nicopoulos, J. Kim, N. Vijaykrishnan, and C.R. Das "Exploring fault-tolerant network-on-chip architectures," *Proc. of the International Conference on Dependable Systems and Networks (DSN)*, pp. 93-104, June 2006.

[15] T. Lehtonen, P. Liljeberg, and J. Plosila, "Self-timed NoC links using combinations of fault tolerance methods," *Proc. IEEE Design Automation and Test in Europe,* 2007.

[16] M.C. Neuenhahn, D. Lemmer, H. Blume, and T.G. Noll, "Quantitative cost modeling of error protection for network-on-chip," *Proc. ProRISK Workshop*, pp. 331-337, 2007.

[17] Y. Jiao, Y. Yang, M. He, M. Yang, and Y. Jiang, "Multi-path routing for mesh/torus-based NoCs," *Proc. IEEE Int. Conference on Information Technology, ITNG*, pp. 734-742, 2007.

[18] S. Shamshiri and K.-T. Cheng, "Yield and cost analysis of a reliable NoC," *Proc. IEEE VLSI Test Symposium*, pp. 173-178, 2009.

[19] R. Naseer and J. Draper, "Parallel double error correcting code design to mitigate multi-bit upsets in SRAMs," *Proc. IEEE European Solid-State Circuits Conference (ESSCIRC)*, pp. 222-225, 2008.

[20] R. Naseer and J. Draper, "DEC ECC design to improve memory reliability in sub-100nm technologies," *Proc. IEEE International Conference on Electronics, Circuits and Systems, (ICECS)*, pp. 586-589, 2008.

[21] M.A. Bajura *et al.*, "Models and algorithmic limits for an ECC-based approach to hardening sub-100-nm SRAMs," *IEEE Transactions on Nuclear Science*, vol. 54, no. 4, pp. 935-945, 2007.

[22] N.M. Abramson, "A class of systematic codes for non-independent errors," *IRE Transactions on Information Theory*, vol. IT-5, pp. 150-157, Dec. 1959.

[23] S.H. Reiger, "Codes for the correction of clustered errors," *IRE Transactions on Information Theory*, pp 16-21, Mar. 1960.

[24] E.N. Gilbert, "A problem in binary encoding," *Proc. Applied Math Symposium*, pp. 291-297, 1960.

[25] I.S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300-304, June 1960.

[26] R.M. Roth and G. Seroussi, "Reduced-redundancy product codes for burst error correction," in *IEEE Transactions on Information Theory*, vol. 44, no. 4, pp. 1395-1406, July 1998.

[27] D. Raphaeli, "The burst error correcting capabilities of a simple array code," *IEEE Transactions on Information Theory*, vol. 51, no. 2, pp. 722-728, Feb. 2005.

[28] M. Blaum, "A family of efficient burst-correcting array codes," *IEEE Transactions on Information Theory*, vol. 36, no. 3, pp. 671-675, May 1990.

[29] P.G. Farrell, "A survey of array error control codes," *European Transactions on Telecommunications*, vol. 3, no. 5, pp. 441-454, Sep.-Oct. 1992.

[30] S. Shamshiri and K.-T. Cheng, "Error-locality-aware linear coding to correct multi-bit upsets in SRAMs," *Proc. IEEE International Test Conference (ITC)*, Nov. 2010.

[31] C.W. Slayman, "Cache and memory error detection, correction, and reduction techniques for terrestrial servers and workstations," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 397–404, Sep. 2005.

[32] J. Maiz, S. Hareland, K. Zhang, and P. Armstrong, "Characterization of multi-bit soft error events in advanced SRAMs," *Proc. IEEE International Electron Devices Meeting (IEDM)*, pp. 519-522, 2003.

[33] D. Radaelli, H. Puchner, S. Wong, and S. Daniel, "Investigation of multi-bit upsets in a 150 nm technology SRAM device," *IEEE Transactions on Nuclear Science*, vol. 52, no. 6, pp. 2433–2437, Dec. 2005.

[34] A. Kohler and M. Radetzki, "Fault-tolerant architecture and deflection routing for degradable NoC switches," *Proc. ACM/IEEE International Symposium on Networks-on-Chip*, pp. 22-31, May 2009.

[35] P. Bogdan, T. Dumitras, and R. Marculescu, "Stochastic communication: A new paradigm for fault-tolerant networks-on-chip," *Proc. VLSI Design*, 2007.

[36] M. Majer, C. Bobda, A. Ahmadinia, and J. Teich, "Packet routing in dynamically changing networks on chip," *Proc. IEEE International Parallel and Distributed Processing Symposium*, 2005.

[37] S. Shamshiri and K.-T. Cheng, "Modeling yield, cost, and quality of an NoC with uniformly and non-uniformly distributed redundancy," *Proc. IEEE VLSI Test Symposium (VTS)*, pp. 194-199, Apr. 2010.