Adaptive Stochastic Routing in Fault-tolerant On-chip Networks*

Wei Song[1], Doug Edwards[1], José Luis Nuñez-Yañez[2], and Sohini Dasgupta[1]
[1]School of Computer Science, University of Manchester, Manchester, M13 9PL UK
[2]Department of Electrical and Electronic Engineering, Bristol University, Bristol, BS8 1UB UK
{songw,doug,shinid}@cs.man.ac.uk, j.l.nunez-yanez@bristol.ac.uk

Abstract

Due to shrinking transistor geometries, on-chip circuits are becoming vulnerable to errors, but at the same time on-chip networks are required to provide reliable services over unreliable physical interconnects. A connection oriented stochastic routing (COSR) algorithm has been used on one NoC platform that provides excellent fault-tolerance and dynamic reconfiguration capability. A probability model has been built to analyze the COSR algorithm. According to the model, the performance may be improved by implementing a self learning mechanism in each router. Thus a new adaptive stochastic routing (ASR) algorithm is proposed whereby each router learns the network status from acknowledgement flits and stores the outcomes in a routing table. Simulation of both algorithms reveals that the ASR algorithm shows a higher path reservation success rate and a larger maximal accepted traffic than the COSR algorithm. The simulations also show that the learning procedures are accurate and that both algorithms are fault-tolerant to intermittent/permanent errors.

1 Introduction

As transistor dimensions continue to shrink, on-chip circuits are becoming vulnerable to transient, intermittent and permanent errors [4], especially on long interconnects [9]. The paradigm shift in current multiprocessor system-on-chip (SoC) designs, that replaces the SoC buses with a NoC communication fabric, introduces the challenge of providing reliable communication on unreliable physical interconnects.

Normally there are two ways for a network-on-chip (NoC) to recover from transient errors: retransmission and error correction. Considering the low error rate for current circuit design technology and the large overhead of error detection/correction circuits, retransmission is more power efficient than error correction [2]. Therefore, short-term transient errors could be handled in the transport layer (end-to-end level) and routers only need to provide a reliable path in the presence of permanent/longer term intermittent errors. Adaptive routing algorithms have long been utilized in NoCs with on-line defective interconnects. However, the turn-model is constrained to avoid deadlocks. In some extreme cases, an existed path is prohibited by the routing algorithm. In constrast to adaptive routing algorithms, the stochastic routing algorithms are always able to find an available path in the presence of errors.

The stochastic routing algorithm was used in a probabilistic flooding scheme which demonstrated a high message arrival rate with the cost of high energy consumption and long transmission latency [5, 3]. A tradoff to reduce the energy consumption could be made by constraining the extra messages flooded, such as the redundant random walk and directed flooding algorithms [8]. We have proposed a fault-tolerant and dynamically reconfigurable network-on-chip (NoRC) platform [7] based on a further constrained random walk routing algorithm, namely connection oriented stochastic routing (COSR) algorithm. Combining the reconfiguration technology and the COSR routing algorithm, this platform can a) map tasks onto network nodes at run-time, including moving a task from one node to another, b) provide delay guaranteed services, c) maintain data integrity through the frame level error detection and retransmission scheme and d) cope with intermittent/permanent errors.

However, the COSR algorithm tends to reserve long paths, constraining traffic, increasing the probability of rejection and consuming unnecessary channel resources. A new adaptive stochastic routing (ASR) algorithm is proposed. By adding a routing table in each router, the router can learn from random walks and then gradually direct random walks to shorter paths. This claim is derived from a probability model and is supported by transaction level simulations. Simulation results based on faulty networks also show that both algorithms are fault-tolerant to intermittent/permanent errors.

The remainder of this paper is organized as follows: section 2 redefines the fault-tolerant dynamically reconfigurable network-on-chip platform and the COSR algorithm described in [7], section 3 analyzes the COSR algorithm using a probability model and the new ASR algorithm is proposed, section 4 compares the performance of these two al-

*This work is supported by EPSRC grant EP/E06065X/1 and EP/E062164/1.

gorithms on a fault-free network, section 5 demonstrates the performance when intermittent and permanent errors occur, and the paper is concluded in section 6.

2 The NoRC Platform

Figure 1 shows a configured 4x4 mesh NoRC with nine tasks running on the chip. Every network node comprises a processor element, a network interface and a router. Applications are divided into several parallel tasks. One processor element can run one or multiple tasks according to the run-time configuration. For simplicity, we only configure one task per processor element in this paper. The network interface provides its local processor element a bidirectional channel to the on-chip network. The router is responsible for forwarding messages to its adjacent routers or the local network interface and reserve/release the channels according to the COSR algorithm. Shown in Figure 1, one task may run on only one node, e.g. tasks $\{1,2,4,5,8,9\}$, or run on multiple nodes, e.g. tasks $\{3,6,7\}$. Multiple processor elements may be configured with the same task if heavy computation loads are required or the task is important and a backup is necessary. Nodes identified as task 0 are idle nodes which are waiting for a configuration.

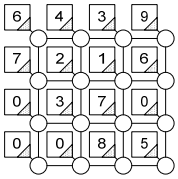


Figure 1. A 4x4 NoRC

The communications on this platform are connection oriented and function oriented. The target of a message is addressed by a function identifier (FID) denoting the task that can consume this message. When a master processor element sends out a message, namely a frame, the network interface divides the frame into a sequence of flits. It firstly encapsulates the FID into a *request* flit and sends it into the network. Routers in the network stochastically forward the *request* flit and reserve the path until a node matches the FID or no channel is available in a certain router. An *ok-ack* flit is sent back by the network interface of the target slave or a *false-ack* flit is bounced back to inform the failure and release the path. According to the flit received, the master node sends out all *data* flits to the reserved path (success), or re-sends the *request* flit again after a retry interval (failed). After all *data* flits have been received by the slave node, the communication procedure is terminated by a *false-ack* flit bounced back by the slave to release the path and inform the master.

Suppose the bandwidth of one channel is enough for all applications, the router in NoRC can be structured as Figure 2. Except for the routers on the boundaries, each router is connected with four adjacent nodes (south, west, north and east) and its local network interface by an input channel and an output channel. Every input channel is connected with a flit size input buffer. All input channels and output channels are fully connected through the crossbar dynamically configured by the arbiter, which is responsible for analyzing the flits arrived. When the crossbar is configured, the input buffer is able to transmit all received flits directly to the output channel without interrupting the arbiter. The paths configured in the crossbar are released automatically when a *false-ack* is transmitted.

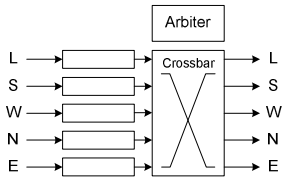


Figure 2. A simplified router for NoRC

The configuration procedure of the crossbar has two stages: the forward setup and the backward setup. When a *request* flit arrives at the input buffer, the arbiter tries to connect the forward path to an idle output channel. Because of congestion, the arbiter may fail to find an available output channel and a *false-ack* flit is bounced back. When a *false-ack*/*ok-ack* flit arrives from the backward channel, the arbiter then configures the crossbar to connect the backward channel. The network interface also has a timeout mechanism. When no further flit is received by a certain time interval, both the master and the slave node can send a *false-ack* flit to release the path and to avoid deadlocks. Therefore, when one permanent or intermittent error occurs on a reserved path, the path is released due to the timeout mechanism. This transport layer protocol cannot handle the case when two or more errors happen nearly simultaneously on a path. An error detection scheme running in the routers is required but this is not within the scope of this paper.

3 The Adaptive Stochastic Routing Algorithm

3.1 Analyzing the random walks

We intend to build a simple model to analyze the probability with which a master reserves a path to its slave in a fault-free and idle mesh in a limited number of hops. We assume a mesh network with infinite dimensions to eliminate the impact of boundaries, channels are exclusively allocated and only one active master is assumed to be using the network. A task may run on multiple nodes but this significantly complicates the calculation. Therefore, we only consider the single slave case.

According to the COSR algorithm, a *request* flit is forwarded to all available ports with an equal probability. For the ith router on a path, the port $\hat{p}_i$ is selected. For a possible path that traverses L routers (hop count L), the event S

Table 1. The success rates when the slave is less than 4 hops away

(Δ_x, Δ_y)	(1, 0)	(1, 1)	(2, 0)	(2, 1)	(3, 0)	(2, 2)	(3, 1)	(4, 0)
$P = P\{L \leq 16\}$	42.07%	32.32%	24.93%	21.22%	14.98%	16.43%	14.17%	9.60%
$P_p = P\{L \leq 16 \mid \hat{p} \in \mathcal{P}\}$	100.00%	48.39%	49.00%	31.10%	28.53%	24.15%	20.34%	17.94%
$P_{\bar{p}} = P\{L \leq 16 \mid \hat{p} \notin \mathcal{P}\}$	22.76%	16.25%	16.91%	11.35%	10.46%	8.71%	8.00%	6.81%

denotes that the path is selected by the random walk. Shown in (1), S occurs when all the routers on the path select the corresponding port $\hat{p}_i$.

$$P\{S\} = P\{\bigcap_{i=1}^{L} \hat{p}_i\} = \prod_{i=1}^{L} \frac{1}{a_i} \tag{1}$$

where a_i denotes the number of available output ports in the ith router.

Suppose there are N_m possible paths for a maximal m hops. The probability with which a request flit reaches the target slave node within the maximal hop count is:

$$P\{L \leq m\} = \sum_{i=1}^{N_m} P\{S_i\} = \sum_{i=1}^{N_m} \prod_{n=1}^{L_i} \frac{1}{a_n} \tag{2}$$

Since the inifinite mesh network is symmetrical, it is sufficient to describe all location cases by using the relative address between masters and slaves. The ports in a router are classified into two sets: $\mathcal{P}$, the ports that may lead to a shortest path, and $\bar{\mathcal{P}}$, the ports that definitely lead to a non-shortest path.

Table 1 demonstrates the results of all the possible slave locations within a 4 hop neighborhood, which include: P, the probability of reaching the target slave within m hops; P_p, the conditional probability of reaching the target when the port $\hat{p}$ selected by the first router may direct to a shortest path; $P_{\bar{p}}$, the conditional probability of reaching the target when the port $\hat{p}$ selected by the first router always points to a non-shortest path. Here the maximal hop count is set to 16 because $P\{L > 16\}$ is comparatively small and a path that connects two nodes 4 hop away with more than 16 hops is too expensive and inviable. Shown in table 1, P decreases with the master and slave distance. Therefore, reserving a path gets more difficult when the match slave is far away. The results also show that, when the selected port $\hat{p} \in \mathcal{P}$, the NoC has a better chance to build a path than the ports in $\bar{\mathcal{P}}$.

3.2 Learning from the acknowledgements

Obviously, the COSR algorithm reserves long paths even if most of the network is idle. This consumes extra resources and saturates the network at a low network load. However, if routers can learn from random walks to identify the port set $\mathcal{P}$, sending *request* flits to them has a better P and may reduce the occupied channel resources. Fortunately, routers indeed can recognize the $\mathcal{P}$-FID relationship by analyzing the acknowledgement flits.

Let a master $(x+\delta_x, y+\delta_y)$ has its slave running on node (x, y) and all nodes (a, b) that $|a-x|+|b-y| < |\delta_x|+|\delta_y|$ are idle, from exhaustive searches we find

$$P_p \geq P_{\bar{p}} \tag{3}$$

is true for all $L \leq 16$ cases. The *request* flit has a better chance if it is sent by port set $\mathcal{P}$ rather than $\bar{\mathcal{P}}$.

Considering a router that receives an *ok-ack* flit with FID t from port $\hat{p}$, we define port $\hat{p}$ as the estimated port that belongs to the port set $\mathcal{P}$ for FID t. If a router has repeatedly and successfully reserved a path for FID t through $\hat{p}$ for $(n-1)$ times, we define P_n as the probability of successfully reserving a path for the nth time. Furthermore, C_n is defined as the probability that the estimated $\hat{p}$ indeed belongs to $\mathcal{P}$ when the nth reservation succeeds, namely the confidence of $\hat{p}$. Because of $P_p \geq P_{\bar{p}}$, we can prove that the confidence C_n increases accumulatively with the time n. Re-sending *request* flits to the port that reserved a path the last time accumulatively increases the success rate P_n.

$$C_n \geq C_{n-1}, P_n \geq P_{n-1} \tag{4}$$

3.3 The implementation of the ASR algorithm

Although the results in section 3.2 assume an idle and infinite mesh network, the trend should be similar on heavily loaded, finite and non-regular networks. Channels that are busy in transmission or suffer from intermittent/permanent defects are treated as network boundaries to a *request* flit in the COSR algorithm. Therefore, in a heavily loaded, finite and non-regular network, the $\hat{p}$ estimated by the learning procedure should point to a shortest path under the current network status rather than the shortest paths in an idle mesh network. The learning procedure can adapt to the network status.

The new adaptive stochastic routing (ASR) algorithm can be easily implemented in the router structure shown in Figure 2. All that is required is that the COSR algorithm running in the arbiter is changed into the ASR algorithm. A routing table is required in every router to record the estimated $\mathcal{P}$-FID relationship and its confidence. Shown in Table 2, a routing table RT has n entries when n different tasks coexist in the network. Each FID has an table entry. The $\hat{p}$ field records the estimated port and the C field records the confidence to this estimation. Since $\hat{p}$ is denoted by (0..4), C is converted into a 5-bit integer to record each table entry

in one byte. $C = 31$ denotes the strongest confidence to $\hat{p} \in \mathcal{P}$ while 0 indicates that the router has no knowledge about this FID. Therefore, all of the confidence fields in RT are set to 0 after the initialization stage.

Table 2. A routing table after initialization

	$\hat{p}$ (0..4)	C (0..31)
FID_1	0	0
FID_2	0	0
–	0	0
FID_n	0	0

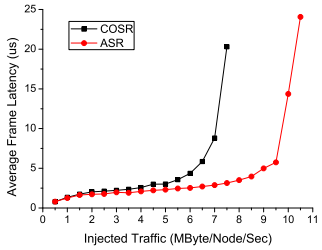
The ASR algorithm is an improved version of the COSR algorithm. When a *request* flit arrives, the router sends the flit to the estimated $\hat{p}$ with probability $C/32$ or to a random port with probability $(1 - C/32)$. Therefore, the router is likely to use the estimated port with a strong confidence while also exploring the network when not so confident. The estimated $\hat{p}$ and confidence C are updated by acknowledgement flits. C increases when an *ok-ack* flit is received from port $\hat{p}$ and decreases when a *false-ack* flit is received from $\hat{p}$. When the confidence to $\hat{p}$ drops to 0, $\hat{p}$ is set to the new incoming port of the next *ok-ack* flit. The new ASR algorithm is still a stochastic routing algorithm. Since all routing tables are set to 0 after initialization, the routers use the COSR algorithm at first and then update routing tables according to acknowledgement flits. Due to this learning procedure, the ASR algorithm is adaptable to status changes and run-time errors.

4 Performance Comparisons on Error Free NoCs

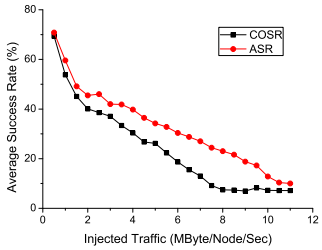
Both the COSR and the ASR algorithms have been implemented using timed SystemC transaction level models (TLM). We intend to implement the routers by fully asynchronous circuits to reduce the dynamic power consumption and the channel latency. Routers and network interfaces are connected by Chain style [1] asynchronous channels. The channel latency is set to 6.24 ns/byte to simulate a 3800 μm Chain channel in a 0.18 μm technology [6]. The router arbitration and internal crossbar latency is set to 5 ns, as it is set in [7]. For each test case, the simulation runs for 4 ms with a warm up time of 500 μs. A uniform random traffic pattern is used in all simulation cases. The configured processor elements periodically and randomly generate frames to all FIDs except the local FID and the idle FID 0 (see section 2). Each frame has a 64 byte data field which is divided into 16 *data* flits.

4.1 Performance comparisons of COSR and ASR

Figure 3 shows the comparison results of the COSR and the ASR algorithms on a 4x4 NoC. A total number of nine



(a) The average frame latency



(b) The average success rate

Figure 3. Comparison results on a 4x4 NoC

tasks running on this NoC are mapped as shown in Figure 1. The average frame latency in Figure 3(a) denotes the time interval since a frame being generated until the end-of-frame *false-ack* flit of this frame being received by the master node. The average success rate in Figure 3(b) is calculated by

$$\frac{N_{ok-ack}}{N_{ok-ack} + N_{false-ack}}$$

where $N_{false-ack}$ dose not include end-of-frame *false-ack* flits.

Shown in Figure 3(b). The ASR algorithm demonstrates a 15.59% higher success rate when the network load is heavy. Moreover, this 15.59% higher success rate alleviates the network contention and decreases the path length, which together push the maximal accepted traffic from 7.5 MByte/Node/Sec to 10.5 MByte/Node/Sec, providing 40.0% improvement.

4.2 Results of the learning procedure

To demonstrate the accuracy of the learning procedure, a 4x4 NoC mapped as shown in Figure 1 has been simulated for 4 ms and all estimations in routing tables are extracted in the end. Since nodes and channels may be reserved for a long time and these reservations could affect the network status, an extremely light traffic 1 MByte/Node/Sec is injected into the network to avoid this effect and simplify the explanations. By analyzing the extracted data, Figure 4 shows the estimated results in a graphic way. The estimated ports are denoted by corresponding arrows (south: $\downarrow$, west: $\leftarrow$, north: $\uparrow$, east: $\rightarrow$ and local: $\circ$). Ports with comparatively strong confidences ($C \geq 16$) are drawn by solid and

bold arrows while those with lower confidences ($C < 16$) are drawn with dash and slim arrows.

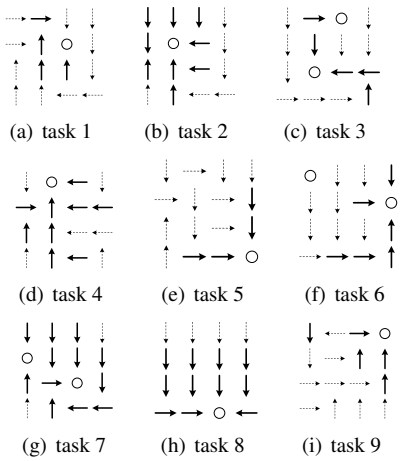


Figure 4. The estimated directions

For all tasks in Figure 4, nearly all slaves for certain functions, denoted by ○, are surround by strongly estimated ports pointed to them except the top left node (0, 0) in Figure 4(f). However, the estimations around this node are not wrong and actually explain how the network status can affect the estimated results. The node (0, 0) with FID 6 should be reserved at the time, since all strong estimations are pointing to the alternative node (1, 3) and the estimations around node (0, 0) are weak and are directing frames to the alternative. The routing tables indeed adjust themselves to the network status.

The majority of port estimations in routing tables are correct. For this test case, there are a total number of 144 estimated ports and confidence fields. Of these, 29 ports do not point to the nearest slave, which leads to an error rate 20.14%. Note that some erroneous estimations are affected by the changes of the network status and in fact point to the right directions. The actual error rate is lower than 20.14%. Furthermore, only the strong estimations significantly change the random walks of the COSR algorithm. The error rate for strong estimations is just 6.58%. Therefore, the port learning procedure of the ASR algorithm successfully learns the network status from the acknowledgement flits.

5 Performance on Faulty NoCs

As mentioned, the errors in network could be classified into three categories: transient errors, intermittent errors and permanent errors. On the NoRC platform, transient errors are handled by the error detection and retransmission mechanism in the transport level. Therefore, we only demonstrate the network performance in presence of permanent/intermittent errors. For asynchronous channels like Chain [1], any intermittent/permanent error causes a line to be stuck to a certain voltage, which then stops the whole channel. Therefore, unlike synchronous buses where intermittent/permanent errors give erroneous data on a bus, asynchronous channels are stopped when these errors occur. In the following simulations, routers detect the intermittent/permanent errors and drop all flits heading to the defective channels, and the performance of the the ASR algorithm is evaluated.

Figure 5 shows an example of a NoC with intermittent errors. During the simulation time 1 ms to 3 ms, the west channel of router (1, 2) is defective. During the simulation time 2 ms to 4 ms, the whole router (1, 0) is defective. Figure 6 demonstrates the simulation results of the variations on the accepted traffic when intermittent errors occur. The network is loaded with a heavy load of 8 MByte/Node/Sec. The accepted traffic of the ASR algorithm does not have a significant variance during the occurrence of intermittent errors. The accepted traffic of the COSR algorithm has a drop during 1 ms to 3 ms. The single error on the the west channel of router (1, 2) has a more severe impact than the fully defective router (1, 0). Note that the network has two nodes running FID 7 but only one for FID 1, the single error of router (1, 2) reduces the reachability of the rare resource FID 1. The results show that the ASR algorithm has a larger accepted traffic than the COSR algorithm during errors and the errors on rare resources have a more severe impact than redundant resources. Duplicating the resources can alleviate the impact of errors.

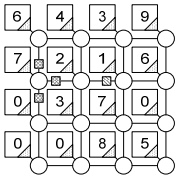


Figure 5. NoC with intermittent errors

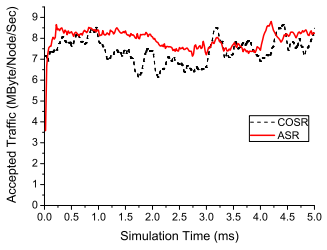


Figure 6. Accepted traffic with intermittent errors

Regarding the permanent errors, the NoC in Figure 7 suffers two permanent errors. After 1 ms simulation, the router (0, 0) is permanently defective and isolates the node (0, 0)

from the network. Later at 2 ms, the router (0, 3) is also permanently defective and isolates the only node for FID 9. Without node (0, 3), FID 9 is unavailable for the network, which would deadlock the network since all nodes are stuck when requiring FID 9. According to the NoRC platform, the system level reconfiguration mechanism can detect the failure and search an idle node to reconfigure it to FID 9. In this example, the selected idle node is node (2, 0), which is reconfigured to FID 9 after 2.5 ms.

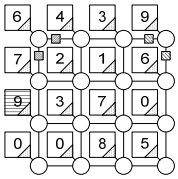


Figure 7. NoC with permanent errors

Figure 8 demonstrates the simulation results of the variations on the accepted traffic when permanent errors occur. The network is also loaded with a heavy load 8 MByte/Node/Sec. Both the COSR and the ASR algorithms show a slight drop when node (0, 0) is isolated from the network and drop to zero when the router (0, 3) is defective. After the system level reconfiguration mechanism reconfigures the node (2, 0) to FID 9, they all return back to normal after a period of instability. The ASR algorithm shows a significant "bounce" after the reconfiguration and it saturates the network at a larger injected traffic than the COSR algorithm, therefore the ASR algorithm can send out the frame stuck by failures in burst while the COSR algorithm is always running at the maximal accepted traffic.

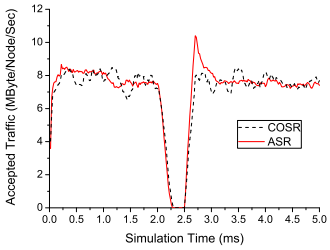


Figure 8. Accepted traffic with permanent errors

In this section, the simulations demonstrate the fault-tolerant capability of the COSR and the ASR algorithms. Especially in the second simulation case, both algorithms recover when an alternative node is configured, which is a feature that is not easily supported by non-stochastic routing algorithms. When intermittent/permanent errors occur, the ASR algorithm shows better accepted traffic performance than COSR.

6 Conclusion

This paper has proposed a new adaptive stochastic routing (ASR) algorithm running on the NoRC platform [7]. The ASR algorithm supports function oriented routing, provides strong fault-tolerance to intermittent/permanent errors and demonstrates the better path reservation success rate, average frame latency, maximal acceptable traffic performance than the COSR algorithm.

We have designed a simple probability model to analyze the success rate of the previous COSR algorithm. According to the analyses, we claim that sending a frame in the direction which has successfully reserved a path can improve the success rate. Both the COSR and the ASR algorithms have been modeled and simulated using TLMs. The simulation results on error free networks show that: the ASR algorithm has a 15.59% higher success rate and increases the maximal acceptable traffic by 40.0%, compared with the COSR algorithm on a 4x4 NoC; the routing tables successfully adapt themselves to the network status through the ASR algorithm. On faulty networks, both the COSR and the ASR algorithms demonstrate fault-tolerance to intermittent/permanent errors.

References

[1] J. Bainbridge and S. Furber. Chain: a delay-insensitive chip area interconnect. *IEEE Micro*, 22:16–23, 2002.

[2] D. Bertozzi, L. Benini, and G. D. Micheli. Error control schemes for on-chip communication links: the energy-reliability tradeoff. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(6):818–831, June 2005.

[3] P. Bogdan and R. Marculescu. A theoretical framework for on-chip stochastic communication analysis. In *Proceedings of the International Conference on Nano-Networks and Workshops*, 2006.

[4] C. Constantinescu. Trends and challenges in VLSI circuit reliability. *IEEE Micro*, 23(4):14–19, July 2003.

[5] T. Dumitras and R. Marculescu. On-chip stochastic communication. In *Proc of DATE*, 2003.

[6] S. Hollis and S. W. Moore. RasP: an area-efficient, on-chip network. In *Proc of ICCD*, pages 63–69, October 2006.

[7] J. L. Nunez-Yanez, D. Edwards, and A. M. Coppola. Adaptive routing strategies for fault-tolerant on-chip networks in dynamically reconfigurable systems. *IET Computers & Digital Techniques*, 2(3):184–198, May 2008.

[8] 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 *Proceedings of the IEEE Computer society Annual Symposium on VLSI*, 2004.

[9] 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 of CODES+ISSS*, pages 188–193, October 2003.