From Online Fault Detection to Fault Management in Network-on-Chips: A Ground-Up Approach

Siavoosh Payandeh Azad*, Behrad Niazmand*, Karl Janson*, Nevin George*, Adeboye Stephen Oyeniran*, Tsotne Putkaradze*, Apneet Kaur*, Jaan Raik*, Gert Jervan*, Raimund Ubar*, Thomas Hollstein*†
Department of Computer Systems
*Tallinn University of Technology, †Frankfurt University of Applied Sciences
email: {siavoosh, bniazmand, karl.janson, nevin, steph, tsotne, akaur, jaan.raik, gert.jervan, raiub, thomas}@ati.ttu.ee

***Abstract*—Due to the ongoing miniaturization of silicon technology beyond the sub-micron domain and the trend of integrating ever more components on a single chip, the Network-on-Chip (NoC) paradigm has emerged to address the scalability and performance shortcomings of bus-based interconnects. As the feature size shrinks, the system gets much more susceptible to faults caused by wear-out and environmental effects. Thus, in order to increase the reliability, creates the need for having mechanisms embedded into such a system that could detect and manage the faults in run-time.**

In this paper, a ground-up approach from fault detection to fault management for such a NoC-based system on chip is proposed that utilizes both local fault management for fast reaction to faults and a global fault management mechanisms for triggering a large-scale reconfiguration of the NoC. Also, detailed description of strategies for fault detection, localization, classification and propagation to a global fault management unit are provided and methods for local fault management are elaborated.

***Keywords*—Fault Detection, Checkers, Fault Classification, Fault Localization, Fault management, Reconfiguration, Network-on-Chip.**

I. Introduction

Network-on-Chip (NoC) has emerged as a paradigm to address the scalability and performance shortcomings of traditional bus-based architectures [1], [2]. The trend of nano-scale electronics shrinking in size, makes them more susceptible to wear-out and environmental effects. This necessitates the detection and management of faults occurring at the run-time of the system, in order to provide higher reliability.

This work addresses a reliable NoC framework, which is maintained as an open-source project named Bonfire [3]. It provides support for fault detection and localization, local fault management, local fault classification, and fault information propagation to a global system health monitoring unit. In a NoC-based System-on-Chip, routers are responsible for transmitting data between the Processing Elements (PEs).

In Network-on-Chip, a router is composed of a data-path and a control part. The packets are transmitted via the data-path, while the control part directs the flow of data and the path the data should take when being transmitted between routers. Thus, both for the data-path and the control part, fault tolerance is of utmost importance for a reliable communication.

For the data-path, error detection and/or error correction techniques (such as single parity and Hamming encoding [4]) can be used. However, due to the area overhead of error correction techniques such as Hamming, the focus of this work is on single bit parity for the detection of faults in the data-path (inter-router links and data-path components of the routers).

On the other hand, faults in the control part of NoC routers should be handled. One way is to detect them via concurrent online checkers (for instance via the approaches proposed in [5], [6]) due to their low fault detection latency. There are also other methods such as Built-In-Self-Test (BIST) [7]. However, they interrupt the normal operation of the system for testing upon a fault occurrence. Thus, in the scope of this work, we focus on concurrent online detection of faults for the control part of routers. It is important to note that the checker outputs also facilitate fault localization [8], pinpointing the defective part in the circuit. Additionally, higher abstract deductions can be made based on them, such as existence of defect in turns in a router (a path from an input port to an output port). Such information can be used for reconfiguration of the routing algorithm or re-mapping of the tasks by units in charge of application mapping and scheduling. Works such as [9] have addressed multi-layer fault diagnosis and combining checkers at different levels of abstraction, however, they impose high latency. Furthermore, they have not addressed any mechanism for classification of faults and fault management, which are considered in our work.

In this work a ground-up approach from fault detection to management for NoC-based System-on-chips is proposed. Strategies for fault detection, localization, classification and propagation to a global fault management unit are described. Furthermore, in order to improve the reaction time to faults, methods for local fault management are elaborated.

The rest of this paper is organized as follows: in section II the basics of the Bonfire framework, including the NoC and router architecture are discussed. Section III describes the fault model used in this work and the method of fault injection on the links. In section IV different fault detection mechanisms for control and data path of the routers are discussed. Section V describes methods of fault localization. Sections VI describe the the process of fault classification and Section VII provides methods of handling faulty packets at the router level. Section VIII details fault information propagation to system health monitoring unit and finally, section X concludes the paper.

II. Bonfire framework

A. Bonfire NoC Architecture

The aim of the Bonfire project is to create a fault-tolerant framework for testing dependability mechanisms in a NoC-based System-on-Chip(SoC). The targeted NoC is using a 2D mesh topology where each tile of the network consists of a wormhole switching router equipped with fault tolerance

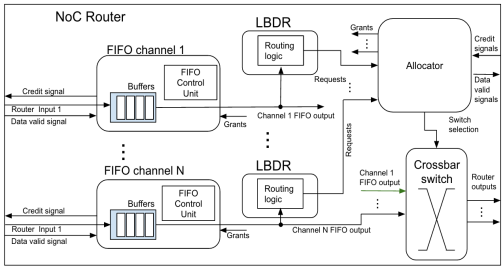

Fig. 1. Overview of the architecture of baseline credit-based flow control NoC router used in Bonfire network

mechanisms and a Processing Element (PE) connected to it via a Network Interface (NI). Each PE comprises a Plasma core [10], which is a 32-bit MIPS-I based open-source processor with three pipeline stages, along with 8 KB of RAM (as local memory). Details of the components of the framework are described in the following subsections. Bonfire is maintained as an open-source project, available at [3] .

B. Bonfire Router

The Bonfire network described in this paper utilizes 32 bit credit-based wormhole switching in the routers. Fig. 1 shows an overview of the baseline router used in the Bonfire network, without any fault-tolerance mechanism. The router comprises of an input buffer (implemented as First-In-First-Out (FIFO)), routing computation unit (implemented using Logic-Based Distributed Routing (LBDR) mechanism [11]), switch allocator (prioritizing multiple requests to the same output port based on Round-Robin policy) and crossbar switch.

We have opted for LBDR [11], since it is scalable compared to table-based routing in NoCs. Furthermore, LBDR describes the topology and the routing algorithm in a 2D NoC in terms of a fixed number of configuration bits, *i.e.* connectivity and routing bits. This makes it possible to use the connectivity bits for the indication of links in the 4 main directions as healthy or faulty, by setting the corresponding connectivity bit to zero (faulty) or one (healthy). Routing algorithm re-configuration (if necessary) can be done by changing the routing bits.

C. System Health Monitoring Unit (SHMU)

The Bonfire project targets a holistic system health monitoring and management solution. To implement this, a dedicated unit, called System Health Monitoring Unit (SHMU) [12], [13], is proposed which handles fault information collection and system-scale fault management and reconfiguration.

In Bonfire project SHMU runs as software on one of the Processing Elements (PEs) in the network. And if the processor fails, the SHMU tasks can be mapped on another node. Details about functionality and implementation of SHMU is beyond the scope of this paper.

III. Fault Model

In this work, we focus on single stuck-at fault model [14], which means in each router module only one fault can occur at a time. For data-path related modules, including the links, only one bit can get faulty at a time on the specific link. The same applies to the control part related modules. Thus, separate control part modules and data links from different ports can get faulty at the same time, but only one fault in each of them at a time. Transient faults are modeled as single stuck-at faults which last one clock cycle. Intermittent faults are modeled as bursts of transient faults in short periods. Permanent faults are modeled as a moving from transient fault to intermittent state and then finally with a permanent stuck-at fault.

In this work, fault injection is done using force command of ModelSim from Mentor Graphics [15]. The injection points are links between routers and also internal signals of the individual modules inside the router.

IV. Fault Detection

The Bonfire framework uses different methods for detection of faults in data-path and in the control part of the network.

A. Data-path Fault Detection

Since this work focuses on a single stuck-at-fault model, a simple parity checker module is used to cover all single-bit faults on the input ports of the router. Upon receiving a faulty flit, the router starts a fault classification process and also manages the fault locally in order to prevent network congestion (for more information, please refer to section VII).

B. Control Part Fault Detection

Concurrent Online checkers are utilized to detect faults in the control part of the NoC routers. A *checker* is a concurrent online fault detection module [5], [6]. It detects faults occurring at inputs and outputs of fan-out free regions [16] of the circuit with low latency. Since checkers provide fault information required for fault localization, this method is preferable to Double or Triple Modular Redundancy (DMR and TMR) schemes. The use of concurrent checkers for online fault detection in control part of NoC routers are described in more detail in [5], [6], [17], [18]. It is worth noting that the complete set of checkers for the control part of Bonfire NoC are available at [3], which covers the control part of FIFO, routing logic (LBDR) and allocator unit (allocator) shown in Fig. 1.

V. Fault Localization

As the number of checkers can grow very large (in the order of hundreds per router), it is not feasible to send the fault detection information from all these checkers to SHMU. Also, in case of a NoC router, for example, flipping of a bit in a register in one of the router's internal modules will not provide valuable information to the SHMU in the application layer. However, if the outputs of the checkers connected to this module are combined, it is possible to translate the output of the checkers into more meaningful abstracted information.

By combining the checkers for the control part of the router, it is possible to report faults at a more abstract level. For instance, in [8], a fault localization method is introduced which groups sets of checkers, making an assertion vector, facilitating finding fault location at different granularity levels in the control part of a NoC router. This can also be used when signaling higher levels in the architecture, such as the application level about the occurrence of faults.

Works such as [19], model faults in the control part as a complete node failure. In [20], illegal turns in the routers are detected, however, each router depends on the information

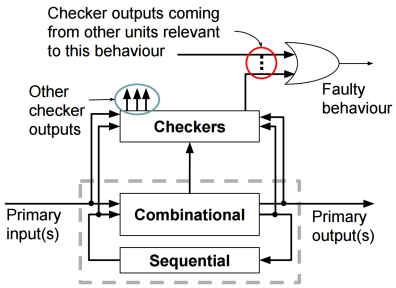

Fig. 2. General structure of the fault detection, grouping and classification mechanisms

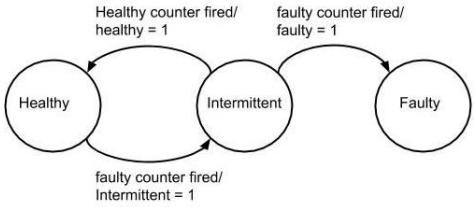

Fig. 3. Finite State Machine (FSM) for the fault classifier unit

its neighbor routers for online fault detection. On the other hand, in our work, we combine checker outputs (as shown in Fig. 2) for the control part of a router. Further, this can be translated into detection of a *turn fault*. Unlike [20], we use the checker outputs in the current router to model turn faults, and there is no need for collecting information from neighbor routers. A turn fault is defined as a fault occurring in one of the components on the path from an input port to an output port of the router (*e.g.* a West to North turn fault or a straight path). This information can be passed to SHMU to the application layer. Later, if required, the SHMU can initiate re-configuration of the routing algorithm or re-mapping of the tasks on the nodes based on the fault information received from the lower (hardware) level.

VI. Fault Classification

With the growing number of transient faults, it would be impractical to send a separate notification to SHMU for each occurring fault. Not only would this impose additional latency by sending a notification from hardware to application layer, but it will also incur a significant power overhead.

To overcome this problem, faults are classified locally in the routers as *permanent*, *intermittent* or *transient*. The classified fault information is transmitted to SHMU if the fault is classified as intermittent or permanent. In [21], [22], an online fault classification mechanism is introduced as part of a cross-layer fault management framework, however no details regarding the implementation of the fault classifier is provided. Whereas, in our work, a fault classification method based on [23], [24] is implemented; where a set of counters are used to count the healthy and faulty packets going through a network link. Each of the counters are compared with a threshold value. When a counter reaches its threshold, a signal is issued which is used by a control Finite State Machine (FSM) in charge of health making decision. Fig. 3 illustrates the FSM Diagram of the classifier unit. Every time the faulty packet counter

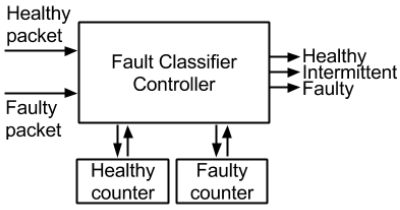

Fig. 4. Fault classifier block diagram

reaches its threshold, the FSM moves one step closer to the Faulty state. Every time the a counter reaches its threshold, both counters would be reset. It is noteworthy to mention that there could be different variations of state diagram models implemented for classification. The current state diagram as described in the Fig. 3 implements a scheme where there is no recycling of once faulty links. In contrast to [23], [24], since no error correction method has been used in this method, only two four-bit counters are utilized (see Fig. 4).

VII. Local Fault Management

Once a fault has been detected in the system, if it is classified as intermittent or permanent, the SHMU is notified. After obtaining the fault information, processing and making a decision, the SHMU can issue a command regarding that particular fault. But, during this time the effect of the fault has already propagated to other parts of the system and containment of the effects would be difficult if not impossible. So even though SHMU is responsible for fault management in the system, it can only manage the faults (in global scale) and some more detailed, distributed, mechanisms are needed for management of faults locally. This problem can be solved by implementing local fault management at each router. In this section two solutions for local management of the faults are provided.

A. Packet Dropping Mechanism

One of the important cases to be addressed is appearance of faulty flit at the input port of a router, where the following situations might happen:

- **Fault in the flit type:** in this case, it is usually not possible to identify the flit type and it (and also subsequent flits belonging to the same packet) cannot be routed. If this is not taken care of, eventually the input buffer (FIFO) of the router will get full, which can, in turn, leads to network congestion.
- **Fault in the payload data:** this type of fault does not have any effect on the network performance. However, since the packet data is corrupt, the fault will manifest itself in the application layer.
- **Fault in the destination address field:** the routing module might not be able to route the packet or the packet gets forwarded to a wrong destination. This might also result in network congestion if it is not properly taken care of.

One of the approaches to bypass the problem of having faulty flits is to use error-correction techniques, such as Hamming coding (single bit error correction, double bit error detection) for all flits. By comparing the overhead of these

TABLE I
AREA AND FLIT SIZE OVERHEAD COMPARISON OF SINGLE BIT PARITY AND HAMMING DECODER

Unit name	Unit area (μm^2)	Overhead in flit size (bits)
32-bit single bit parity checker	663	1
32-bit Hamming decoder	7050	8

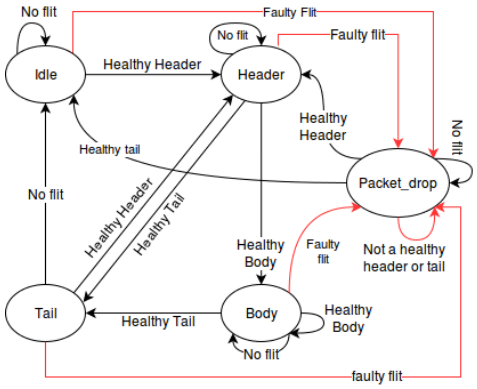

Fig. 5. Finite State Machine (FSM) diagram for the packet dropping mechanism

methods (shown in Table I) using AMS 0.18 μm CMOS technology library [25], it becomes clear that those methods impose additional area overhead to the correction circuit and also increase the flit (due to the additional bits needed for error correction). This, will affect the size of other network parameters, such as the physical link width which, in turn, also increases the size of the input buffer (FIFO) and crossbar switch.

In order to handle the above-mentioned situations, a *packet dropping* mechanism is incorporated in Bonfire framework. However, while using wormhole switching, in some cases, dropping the packet is not possible, for instance, when packet's header flit has already left the router. In such case, it is possible to cut the packet from the current position and attach a fake tail to it and forward it, while dropping the rest of the packet. This will not affect the network's operation. The results of such measures would manifest themselves in the application layer by comparing the packet length with the information in the header flit or as corrupt data. In our router architecture, the packet dropping mechanism is handled by a Finite State Machine (FSM), as is shown in Fig. 5. Additionally, the packet dropping mechanism has to generate fake credits for the upstream router in order not to interrupt the flow of the traffic.

In the Bonfire router, the packet dropping mechanism is improved even further by adding the flit saving functionality – a capability to detect position of the error in flits. In case the error is in the payload part, the packet will still be transmitted to its destination, thus making the application layer to handle the data errors. This will avoid re-transmissions in non-critical applications (for example many multimedia applications).

B. Routing Management

Once a link is classified as faulty, the router automatically sends this information to upstream router to update its LBDR connectivity bits (these bits can be over-written by SHMU later). If the change in connectivity bits happens when a packet is being processed, it might result in the packet being divided or mis-routed. In order to avoid this problem, the new connectivity bits should be stored in a register and routing module should wait until a new header flit arrives. The same approach is applied to routing bits of LBDR reconfiguration. The reconfiguration command is issued by the processing element at each node (once the reconfiguration message is issued by SHMU).

Another important point is to take care of faults occurring in the FIFOs which will be propagated to LBDR modules. In this case, to prevent congestion and network failure, the routing module (LBDR) should manipulate the FIFO modules in order to drop the packet. This is performed with a secondary and much simpler packet dropping mechanism, which generates fake grant signal to the FIFO when a faulty header is detected using a simple parity unit. Since LBDR is only sensitive to the header flit when making routing decisions, there is no need for support for cutting the packet and attaching fake tail. It is assumed that the dropped packets would be handled at the application layer.

VIII. FAULT INFORMATION PROPAGATION

The process of information transmission to the SHMU is also crucial. This can be done either (1) via reusing the existing network, or (2) by using an additional infrastructure working in parallel with the main NoC. There have been many proposals for fault information propagation to a global fault management unit. Some of the proposals, such as iJTAG [21], [22], [26], use scan chains. However, using an infrastructure like iJTAG requires single (or very limited) number of access points which limits the mapping possibilities for the SHMU on nodes since SHMU must have direct access to iJTAG access point. In addition, in approaches such as [21], [22], [26], the Instrument Manager (IM), which works as the iJTAG network controller and knows the structure of the instrumentation network, can become a single point of failure.

Some of the previous works in the literature have taken advantage of dual NoC architectures, such as [27]–[34]. In [34], in addition to the main network, a checker network is used (which is assumed to be 100% reliable) in order to deliver data to its destination in case of a fault occurrence in the main network. In [33], in parallel to the main NoC (which is used for transmitting the data), an additional control network is considered which is used for sending reconfiguration data for updating the connectivity and routing bits of LBDR in the network routers. The control network is used to inform a global manager node regarding faults occurring in different nodes. Despite the advantages these works might bring, they all incur additional area and power overhead. Moreover, if the area of the augmented circuitry for transmitting the fault information is not negligible, it can increase the probability of faults occurring in the additional network itself as well.

In this work, the classified fault information is propagated to SHMU via the NoC itself. The information would be bypassed to the Network Interface (NI). The NI will check the address of the SHMU and will pass the info to the node if SHMU is mapped on the same node (self diagnostic) or will generate

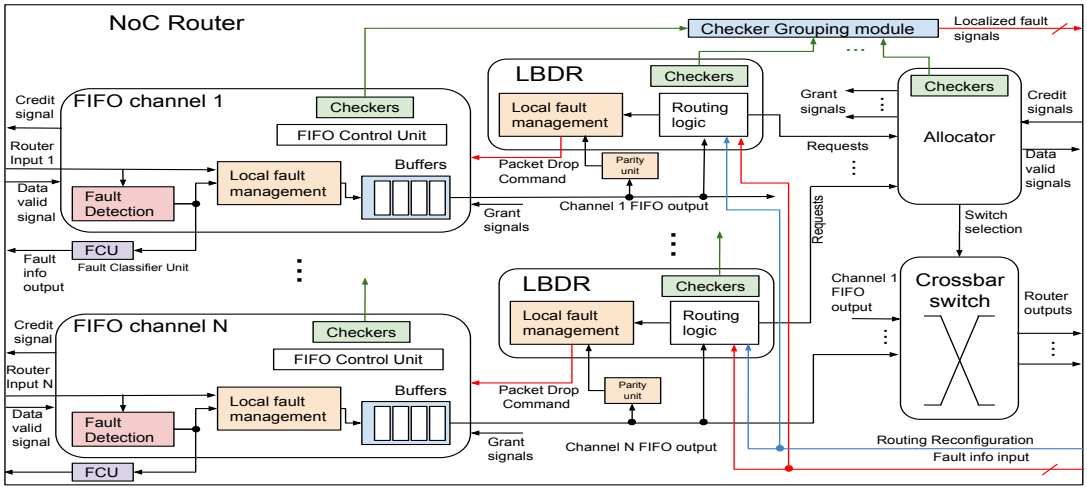

Fig. 6. Overview of the architecture of baseline credit-based equipped with fault tolerance mechanisms

TABLE II
AREA AND AVERAGE PACKET AND FLIT DROP FOR DIFFERENT PACKET DROPPING MECHANISMS.

Unit name	Unit area (μm^2)	Area overhead%	Average packet drop	Average flit drop
Original FIFO	14357	–	–	–
FIFO with packet dropping	16045	11.7%	≃1%	3.3%
FIFO with flit saving	16042	11.7%	≃1%	1.14%

TABLE III
AREA OVERHEAD RESULTS OF SELF-UPDATING LBDR OVER BASELINE LBDR

Unit name	Unit area (μm^2)	Increase in LBDR size	Increase in baseline router size
LBDR	1744	–	–
Self updating LBDR	2940	68.5%	5.9%

and send packets through the network to SHMU as soon as it finds idle time.

As mentioned in the previous section, after the local classification of the faults the information is sent to SHMU, which updates the system health map and can also trigger global re-configuration of the system in order to compensate for the faults. The reconfiguration packets will be sent to each node from SHMU and the node will send the reconfiguration information through the NI to the router. However, if the main NoC is used for transmission of the fault information and reconfiguration packets, under the running routing algorithm, the faults that should be reported, might also themselves prevent the messages to be correctly transmitted to the SHMU.

IX. RESULTS

Table II shows the area overhead of solutions for FIFO described in section VII (obtained using AMS 0.18 μm CMOS technology library [25] and synthesized via Synopsys Design Compiler [35]) along with the average flit and packet dropping ratio with random single stuck-at-fault injection on the network links with average rate of 5×10^6 faults per second. As it can be seen, when comparing the packet dropping approach to flit saving, the average full packet drop rate is not changing. This is due to the faults occurring in the header flit. However, the amount of flit drops is reduced by half, since the flits with the faulty payload will not be dropped in case of flit saving.

Table III shows the area overhead of the self updating LBDR unit. Both the area overhead of the self updating LBDR over the original LBDR (around 68%) and also its area overhead with respect to the baseline router without any fault tolerant mechanism (around 6%) are assessed.

By putting together all the mechanisms described in previous sections (fault detection, localization, classification and local management as shown in Fig. 6), the router grows 60.7% in area which is still lower than duplicate/triplicate-based methods such as DMR and TMR, while it also provides fault localization, management and system reconfiguration support at the same time. Moreover, the instantaneous detection of faults in the control part via the concurrent online checkers and combining them facilitates inferring more abstract and high level fault information (such as turn faults). Two main reasons for using such abstraction of the information are: (1) there is no advantage in transmitting very detailed fault information to the SHMU, since in order to make high-level decisions, SHMU has to abstract the information into turn faults. (2) Additionally, it reduces the amount of information to be transferred to SHMU through the NoC, thus, reducing the network latency and power consumption.

X. CONCLUSION

In this paper, a ground-up approach from fault detection to fault management for a Network-on-Chip based system is proposed. Concurrent online checkers are utilized to detect the faults in the control path and single parity check is used for the data-path. Fault localization and abstraction (into turn faults) are achieved by grouping information gathered from the control part checkers. Moreover, methods of local fault management at the hardware level using different packet dropping mechanisms are introduced and compared. To reduce the overhead of fault information propagation to application layer and its additional processing load, local fault classification mechanism is implemented which generates minimal, classified fault information for propagation.

Additionally, the necessity of having a relocatable System Health Monitoring Unit (SHMU) at the software layer is elaborated. SHMU utilizes the NoC itself for transmitting fault information after classification, thus avoiding a dual-NoC architecture and results in lower area overhead. The experimental results show the overall cost of applying such mechanisms, having 60.7% area overhead, which still makes it a better option than DMR/TMR-based approaches.

ACKNOWLEDGMENTS

The work has been supported by EU's FP7 STREP BASTION, H2020 RIA IMMORTAL, H2020 Twinning TUTORIAL, Estonian institutional research grant IUT 19-1, by the Estonian Center of Excellence in IT EXCITE funded by the European Regional Development Fund, and supported by Estonian IT Academy programme.

REFERENCES

[1] W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Design Automation Conference, 2001. Proceedings*, 2001, pp. 684–689.

[2] L. Benini and G. D. Micheli, "Networks on chips: a new soc paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, Jan 2002.

[3] "Project bonfire network-on-chip," https://github.com/Project-Bonfire, 2015.

[4] S. Ghosh, N. A. Touba, and S. Basu, "Synthesis of low power ced circuits based on parity codes," in *23rd IEEE VLSI Test Symposium (VTS'05)*, May 2005, pp. 315–320.

[5] A. Prodromou, A. Panteli, C. Nicopoulos, and Y. Sazeides, "NoCAlert: An On-Line and Real-Time Fault Detection Mechanism for Network-on-Chip Architectures," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-45. IEEE Computer Society, 2012, pp. 60–71.

[6] P. Saltarelli, B. Niazmand, J. Raik, V. Govind, T. Hollstein, G. Jervan, and R. Hariharan, "A framework for combining concurrent checking and on-line embedded test for low-latency fault detection in noc routers," in *Proceedings of the 9th International Symposium on Networks-on-Chip*, ser. NOCS '15. New York, NY, USA: ACM, 2015, pp. 6:1–6:8.

[7] R. Sharma and K. K. Saluja, "An implementation and analysis of a concurrent built-in self-test technique," in *Fault-Tolerant Computing, 1988. FTCS-18, Digest of Papers., Eighteenth International Symposium on*, June 1988, pp. 164–169.

[8] K. Chrysanthou, P. Englezakis, A. Prodromou, A. Panteli, C. Nicopoulos, Y. Sazeides, and G. Dimitrakopoulos, "An online and real-time fault detection and localization mechanism for network-on-chip architectures," *ACM Trans. Archit. Code Optim.*, vol. 13, no. 2, pp. 22:1–22:26, Jun. 2016.

[9] G. Schley, A. Dalirsani, M. Eggenberger, N. Hatami, H. J. Wunderlich, and M. Radetzki, "Multi-layer diagnosis for fault-tolerant networks-on-chip," *IEEE Transactions on Computers*, vol. PP, no. 99, pp. 1–1, 2016.

[10] Plasma CPU. http://plasmacpu.no-ip.org.

[11] J. Flich and J. Duato, "Logic-based distributed routing for nocs," *IEEE Computer Architecture Letters*, vol. 7, no. 1, pp. 13–16, Jan 2008.

[12] S. P. Azad, B. Niazmand, P. Ellervee, J. Raik, G. Jervan, and T. Hollstein, "Socdep2: A framework for dependable task deployment on many-core systems under mixed-criticality constraints," in *2016 11th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, June 2016, pp. 1–6.

[13] S. P. Azad, B. Niazmand, J. Raik, G. Jervan, and T. Hollstein, "Holistic approach for fault-tolerant network-on-chip based many-core systems," *CoRR*, vol. abs/1601.07089, 2016. [Online]. Available: http://arxiv.org/abs/1601.07089

[14] A. Dalirsani, "Self-diagnosis in network-on-chips," PhD Thesis, Institut fr Technische Informatik der Universitt Stuttgart, July 2015.

[15] "ModelSim, Mentor Graphics," https://www.mentor.com/products/fv/modelsim/, 2017.

[16] R. Ubar, J. Raik, and H. Vierhaus, *Design and Test Technology for Dependable Systems-on-chip*, ser. Premier reference source. Information Science Reference, 2010. [Online]. Available: https://books.google.ee/books?id=_1zzPfTZND8C

[17] P. Saltarelli, B. Niazmand, R. Hariharan, J. Raik, G. Jervan, and T. Hollstein, "Automated minimization of concurrent online checkers for network-on-chips," in *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2015 10th International Symposium on*, June 2015, pp. 1–8.

[18] P. Saltarelli, B. Niazmand, J. Raik, R. Hariharan, G. Jervan, and T. Hollstein, "A framework for comprehensive automated evaluation of concurrent online checkers," in *Digital System Design (DSD), 2015 Euromicro Conference on*, Aug 2015, pp. 288–292.

[19] Y. Jojima and M. Fukushi, "A fault-tolerant routing method for 2d-mesh network-on-chips based on components of a router," in *2016 IEEE 5th Global Conference on Consumer Electronics*, Oct 2016, pp. 1–2.

[20] L. Huang, X. Zhang, M. Ebrahimi, and G. Li, "Tolerating transient illegal turn faults in nocs," *Microprocessors and Microsystems*, vol. 43, pp. 104 – 115, 2016, many-Core System-on-Chip Architectures and Applications (PDP 15). [Online]. Available: //www.sciencedirect.com/science/article/pii/S0141933116000284

[21] K. Shibin, S. Devadze, and A. Jutman, "On-line fault classification and handling in ieee1687 based fault management system for complex socs," in *2016 17th Latin-American Test Symposium (LATS)*, April 2016, pp. 69–74.

[22] A. Jutman, K. Shibin, and S. Devadze, "Reliable health monitoring and fault management infrastructure based on embedded instrumentation and ieee 1687," in *2016 IEEE AUTOTESTCON*, Sept 2016, pp. 1–10.

[23] J. Silveira, M. Bodin, J. M. Ferreira, A. C. Pinheiro, T. Webber, and C. Marcon, "A fault prediction module for a fault tolerant noc operation," in *Sixteenth International Symposium on Quality Electronic Design*, March 2015, pp. 284–288.

[24] J. Silveira, C. Marcon, P. Cortez, G. Barroso, J. a. M. Ferreira, and R. Mota, "Scenario preprocessing approach for the reconfiguration of fault-tolerant noc-based mpsocs," *Microprocess. Microsyst.*, vol. 40, no. C, pp. 137–153, Feb. 2016. [Online]. Available: http://dx.doi.org/10.1016/j.micpro.2015.08.005

[25] (2016) AMS 0.18um CMOS process. http://ams.com/eng/Products/Full-Service-Foundry/Process-Technology/CMOS/0.18-m-CMOS-process/.

[26] "Ieee approved draft standard for access and control of instrumentation embedded within a semiconductor device," *IEEE P1687/D1.71, March 2014*, pp. 1–347, Nov 2014.

[27] A. K. Abousamra, R. G. Melhem, and A. K. Jones, "Deja vu switching for multiplane nocs," in *Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on*, May 2012, pp. 11–18.

[28] R. Das, S. Narayanasamy, S. K. Satpathy, and R. G. Dreslinski, "Catnap: Energy proportional multiple network-on-chip," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13. New York, NY, USA: ACM, 2013, pp. 320–331. [Online]. Available: http://doi.acm.org/10.1145/2485922.2485950

[29] A. Flores, J. L. Aragon, and M. E. Acacio, "Heterogeneous interconnects for energy-efficient message management in cmps," *IEEE Transactions on Computers*, vol. 59, no. 1, pp. 16–28, Jan 2010.

[30] S. Volos, C. Seiculescu, B. Grot, N. K. Pour, B. Falsafi, and G. D. Micheli, "Ccnoc: Specializing on-chip interconnects for energy efficiency in cache-coherent servers," in *Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on*, May 2012, pp. 67–74.

[31] N. E. Jerger, A. Kannan, Z. Li, and G. H. Loh, "Noc architectures for silicon interposer systems: Why pay for more wires when you can get them (from your interposer) for free?" in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec 2014, pp. 458–470.

[32] J. Balfour and W. J. Dally, "Design tradeoffs for tiled cmp on-chip networks," in *Proceedings of the 20th Annual International Conference on Supercomputing*, ser. ICS '06. New York, NY, USA: ACM, 2006, pp. 187–198. [Online]. Available: http://doi.acm.org/10.1145/1183401.1183430

[33] A. Strano, D. Bertozzi, F. Trivino, J. Sanchez, F. Alfaro, and J. Flich, "Osr-lite: Fast and deadlock-free noc reconfiguration framework," in *Embedded Computer Systems (SAMOS), 2012 International Conference on*, July 2012, pp. 86–95.

[34] R. Parikh and V. Bertacco, "Formally enhanced runtime verification to ensure noc functional correctness," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-44. ACM, 2011, pp. 410–419.

[35] (1994) Synopsys design compiler. http://www.synopsys.com.