An Integrated Method for Implementing Online Fault Detection in NoC-based MPSoCs

Vinicius Fochi, Eduardo Wächter, Augusto Erichsen, Alexandre M. Amory, Fernando G. Moraes
FACIN - PUCRS – Av. Ipiranga 6681, 90619-900, Porto Alegre, Brazil
{vinicius.fochi, eduardo.wachter, augusto.erichsen}@acad.pucrs.br, {alexandre.amory, fernando.moraes}@pucrs.br

Abstract— **The continuing development of the silicon technology leads to systems with hundreds of processors interconnected by a network on chip (NoC-based MPSoCs). On one hand, the nanotechnology enables to develop such complex systems, but, on the other hand, the vulnerability to faults increases. The literature presents partial fault-tolerant approaches, targeting specific parts of the system, as high-level methods, router level, link level, and routing algorithms. There is an important gap in the literature, with an integrated method, from the fault detection at the router level up to the fault recovery and correct execution of applications in a real MPSoC. This is the goal of the present work, to present a method with fault-tolerant techniques from the physical to the transport layers. The MPSoC is modeled at the RTL level, using VHDL. A fault campaign injection (5 simultaneous injected faults) resulted in 2,000 simulated scenarios. Results demonstrated the effectiveness of the proposal, with most of the scenarios working correctly with routers operating in degraded mode, with an impact on the execution time below 1%.**

Keywords*— online fault detection; fault-tolerant NoC-based MPSoCs*

I. Introduction and Related Work

The reduction in the physical size of the transistors resulted in the reduction of the interconnections' length. Despite this reduction appears beneficial for performance, the fault probability grows. The global interconnections become relatively longer and narrower, being able to suffer effects as electromigration or short-circuit. Short channel transistors are more susceptible to permanent and transient faults. Permanent faults can occur in the process of chip manufacturing or during the circuit lifetime due to a wear-out of a component. Transient faults can occur from external interference as radiation (as SEUs) or an internal noise generated by the circuit itself (as crosstalk). Thus, fault-tolerant design has a key role in current nanometric technologies, leading to research on fault mitigation techniques for NoCs and MPSoCs. Next paragraphs evaluate different works, each one focusing at a specific level, as high-level (e.g. task migration), router, link, and routing algorithms.

Meloni et al. [1] propose *system level* adaptive and fault tolerant techniques for NoC-based MPSoCs in the context of the MADNESS project. This work searches to reduce the performance loss by using dynamic remapping (task migration) of faulty PEs at execution time.

Fick et. al. [2] propose the Vicis NoC, with redundancy at the *router level* to ensure a correct communication in the presence of faults. This proposal comprises four steps: fault detection, fault diagnosis, system reconfiguration, and system recovery. Error detection mechanisms inform that a new fault was detected, using BIST circuits to detect faults at runtime in a given router component. Then, the diagnosis is executed to determine the fault location. The system reconfiguration disables the faulty component and determines to the functional components how they will work in the new configuration. The proposal inserts multiplexers at the input ports to enable port swapping, and a bypass bus enables to connect the input ports to output ports if the internal crossbar fails. System recovery detects that a component left the fault state, restoring the router component to the original state.

Concatto et al. [3] adopt three fault-tolerant techniques at the *router level*. The first technique enables to share input buffers with adjacent buffers. If a given buffer fails, it may use buffer slots from adjacent buffers. The second technique adopts Hamming codes to protect the channels. Finally, TMR (*Triple Modular Redundancy*) protects the state-machines responsible for controlling the input buffers. The method assumes that the NoC passes by an off-line test to detect and identify faulty buffers. Each router has an extra flip-flop for each input port, to report which buffer presents a fault.

Veiga et. al. [4] and Silva et. al. [5] propose techniques at the *link* level. Their goal is to protect the link against transient faults (*crosstalk*) by using CRC or Hamming codes. Vitkovskiy [6] propose a methodology enabling partially faulty links to continue the transfer of information, albeit at a gracefully degraded mode, in order to maintain network connectivity.

Several Authors also present fault-tolerant routing algorithms [7][8][9][10]. Wachter et al. [10] propose a fault-tolerant routing technique, assuming a BIST circuit to detect faults. The routing technique finds a faulty-free path, with four important features: (*1*). *topology-agnostic*, i.e., it may be applied to regular and irregular NoC topologies; (*2*) *complete reachability*, i.e., if there is a path between a given source-target pair, the method finds it; (3) *fault-independent*, i.e., multiple faults may exist in any router and/or links at any moment; (4) *scalable* in terms of silicon area, i.e. the router area overhead is independent of the NoC size.

State-of-the-art proposals present partial solutions to cope with faults in many-core systems. An integrated method, from the physical layer to the application layer is a gap to be fulfilled.

The *goal* of the present work is to propose a fault-tolerant stack for online fault detection in NoC-based MPSoCs. The method detects faults at the NoC level, reconfigures the router or triggers the search of a faulty-free path, and then ensures the correct communication between processing elements.

The main *contributions* of the paper include:

- Router reconfiguration to operate in *degraded* mode. The advantages to a router to operate in degraded mode are: (1) avoid reconfiguration at higher layers (routing algorithm, operating system); (2) as a fault may be transient, the router may detect the end of the fault and return to the normal mode (system recovery is out of the scope of the present work).
- Fault-tolerant communication protocol. The fault-tolerant protocol ensures the correct communication between PEs in the presence of one or multiple faults in the communication path.

II. Fault-Tolerant Method

Table 1 presents a simplified view of the OSI model (without presentation and session layers) and for each layer the relevant architectural features adopted in the present work (second column) and the added fault-tolerant features (third column). Note that the upper layer, the application layer, does not require modifications in the user code, as code annotation (e.g. checkpoints). The same code designed for the original platform may be used in the fault-tolerant platform.

Table 1 – Simplified view of the OSI model with the added features for fault-tolerance.

Layer	Architectural Features	Added fault-tolerant features
Physical	Duplicated physical channels per link (16-bit filt)	CRC added to data links; CRC in routers
Data Link	Synchronous credit-based flow control	Test wrappers; packet discarding
Network	Adaptive Routing	Auxiliary NoC; control to operate in degraded mode; faulty-free path search
Transport	Message passing (OS level)	Fault-Tolerant communication protocol; packet retransmission
Application	Communication API	Not changed

A. Physical Layer

The present work adopts a NoC with *duplicated physical channels per link*, with support for deterministic XY routing and adaptive source routing. The physical layer adds for each channel 4 CRC bits. The CRC computation uses the polynomial $g = 1 + x + x^4$, ensuring a fault coverage of 93.75% for stuck-at and bit-flip fault models.

Figure 1 presents the router architecture for one port (input channels 0 and 1 and output channels 0 and 1). The gray rectangles represent the CRC modules:

- CRC decoders before input buffers (CRC1) – enable to detect faults in the channels;
- CRC decoders after input buffers (CRC2) – enable to detect faults in the buffers;
- CRC decoders after the crossbar (CRC3) – enable to detect faults in the internal router circuitry, such as the crossbar.
- CRC encoder at the output port (CRC4) – due to the use of source routing, the flit contents may change because the router consumes part of the header. For this reason, it is necessary to re-encode the header flits.

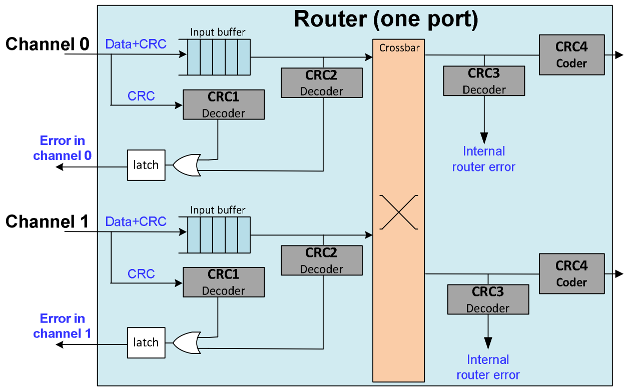

Figure 1 - Router architecture, with CRC modules, for one router port.

When modules CRC1 or CRC2 detect a fault, the channel is *disabled*. The data link layer is responsible for isolating this channel through test wrapper cells described in section II.B. When module CRC3 detects a fault, the entire router is disabled because the internal router circuitry is defective.

Note that CRC modules do not ensure complete fault coverage at the router level. The state-machine to control the input buffers and the module controlling the crossbar require protection. These modules may be protected using approaches like the ones proposed by [3] (i.e. TMR) or with periodic scan-based structural test.

B. Data Link Layer

Figure 2 presents a simplified view of two routers (only CRC1 and CRC4), with the test wrappers cells (gray TW rectangles). The main function of this layer is to discard corrupted packets.

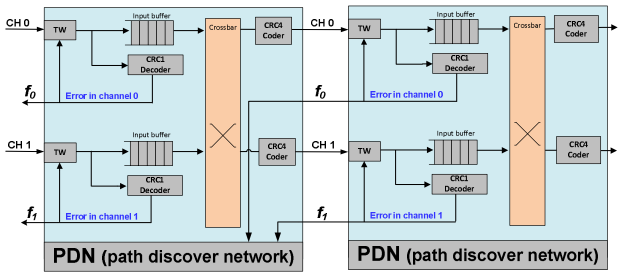

Figure 2 - Test wrappers (TW) cells added at the input ports of the router to isolate fault ports. Simplified view of the router showing only the CRC1 modules.

When CRC1 or CRC2 modules detect a fault, the upstream router (that sent the packet) detects the fault by a side-band signal (f_0 and f_1 in Figure 2). The network layer executes actions enabling the fault recovering.

The function of the TW cells is twofold. First, the TWs assert a signal of the flow control protocol (credit based in the present work), in such a way the upstream may drain all remaining flits of the packet. In practice, this corresponds to a *packet discarding* action. The second function of the TW cell is to block outgoing corrupted flits to not affect healthy routers, avoiding Byzantine faults.

Note that if the fault is detected in the middle of a packet, the wrapper blocks the packet tail, but part of the packet continues to travel to its destination. The processing element at the destination (PE_D) may easily detect an incomplete packet because in wormhole packet-switching flits arrive in sequence, within an interval of few clock cycles between flits. Thus, the network interface or the operating system (OS) of the PE_D may discard an incoming packet when the flow of flits is disrupted for a given number of clock cycles.

C. Network Layer

This layer uses a secondary network, *Path Discover Network* – PDN [11]. The main function of the PDN is to broadcast a message (in practice a single word) from a given router to a selected processing element. The PDN is a small and parallel network, not interfering with the NoC traffic.

Each fault signal *f* in Figure 2 enters in the PDN. For each link two situations may arise:

- 1 faulty channel – in this case the connection between routers is degraded, but still operational. In this case, two actions are executed. First, the module controlling the crossbar is reconfigured, in such a way to use only the healthy channel. Next, the PDN requests the packet retransmission (*seek_resend* service presented next) to the source PE.
- 2 faulty channels – in this case the connection between routers is broken. From this moment, the router control logic signalizes the link as faulty. Any attempt to use a faulty link makes the PDN request a new path to the source and to request the packet retransmission (*seek_unreachable* service).

D. Transport Layer

This layer implements the fault-tolerant communication protocol at the task level, the control of the PDN, and the packet retransmission.

The communication API adopts two MPI-like primitives: a non-blocking *send()* and a blocking *receive()*. To implement the *send()* primitive, a dedicated memory space in the OS, named *pipe*, stores each message written by tasks. Figure 3 details the communication protocol. The execution of a *send()* transfers the contents of the message to a free *pipe* position, assigning its status to *USED*. When task B (assumed mapped in PE_2) executes a *receive()*, a packet with a *message request* is sent to PE_1 resulting in two actions. First, the pipe status changes to *WAITING_ACK* (1 in Figure 3). Then, the message is delivered to PE_2 (2 in Figure 3), and it is transferred to the task B memory space. When PE_1 receives a new message request (3 in Figure 3), the pipe position with a *WAITING_ACK* is released (4 in Figure 3). If a pipe position has a status equal to USED, a new message may be transferred.

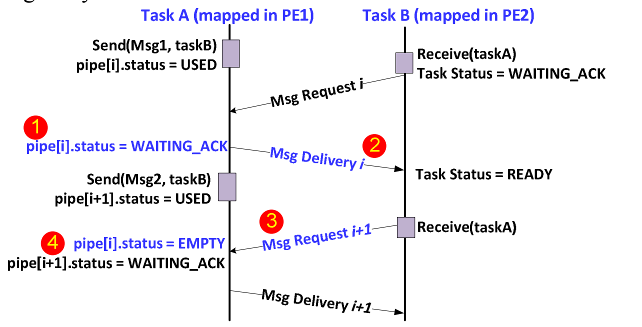

Figure 3 - Communication protocol to exchange data packets.

The main features making this communication protocol fault tolerant include:

- *Implicit acknowledgment* – the request for a new packet ensures that the previous packet was correctly received;
- *Packet sequence number* – avoids the reception of the same packet twice, due to packet retransmission.

When a fault occurs, the OS may receive two messages from the PDN: *seek_resend* or *seek_unreachable.* In the first case, the OS extracts from the PDN message the address to the destination PE, and searches in the *pipe* area the packet to be retransmitted. Note that there is no additional memory cost to implement the packet retransmission, since the packet stays stored in the *pipe* area until a new message request (implicit acknowledgment).

The second message, *seek_unreachable*, the PE requires the search for a new path. The OS configures the PDN to broadcast a path to the destination PE (PE_D) and waits the reception of a message from PE_D with the new path (embedded in a *backtrack_path* message presented later). The NI stores the new path, and all communications between the PE and PE_D will use this new path, transmitted with the use of source routing. The packet retransmission occurs after the definition of the new path.

III. Online Fault Detection And Recovery

The previous section detailed the function of each layer involved in the online fault detection and recovery process. This section shows in Figure 4 three possible scenarios where all layers work together.

In Figure 4(a) a fault in one of the south channels of R6 is detected by R3. The faulty channel in R6 is disabled, and R3 sends a *seek_resend* message to R1. The PE connected to R1 retransmits the message to R9, using the healthy channel between R3 and R6 (observe the disabled channel in the first MPSoC of Figure 4(b), red line).

In Figure 4(b) a second fault arises in the healthy south channel of R6. In this case, the PDN is used to establish a new path between R1 and R9 (R1→R2→R5→R8→R9). When PE_1 (connected to R1) receives the backtrack packet with the new path, it computes the physical channels to be used to avoid deadlocks, storing the new path and physical channels information at the NI. After this process, the message is retransmitted. Note that XY is the default routing algorithm. Only when path computation is required source routing is used.

The example in Figure 4(c) is similar to the one in Figure 4(a), excepting that now source routing is being used, instead of XY. An XY packet contains in its header the destination address. A source routing packet contains in its header the turns to arrive at the destination router. This difference induces a different behavior to retransmit a message. In Figure 4(a), PE_1 extracts from the *seek_resend* packet the address of PE_D, retransmitting the correct packet. On the other hand, in Figure 4(c) it is not possible to know which message was blocked due a fault, because the *seek resend* does not contain the destination address. In this case, all messages with a state equal to "*waiting ack*" are retransmitted. This issue does not represent a problem since the OS discards messages received with the same sequence number.

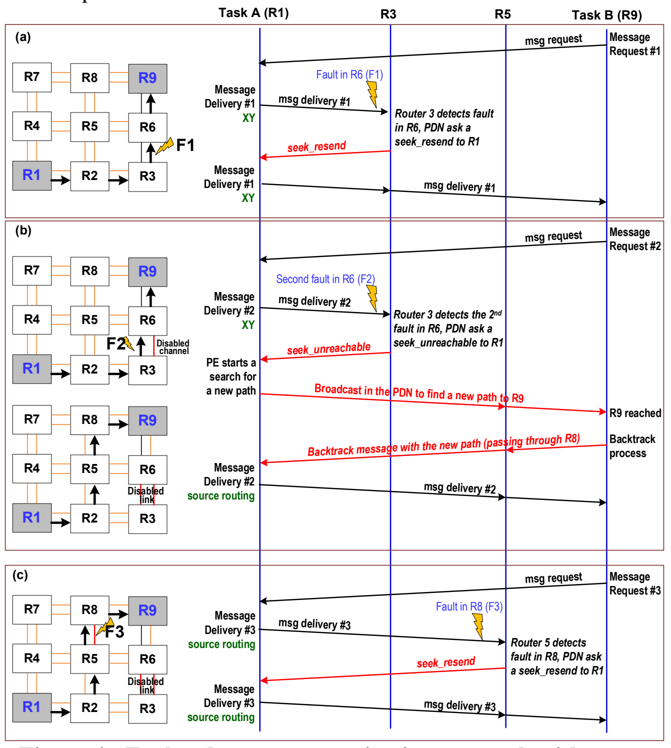

Figure 4 - Fault-tolerant communication protocol, with router reconfiguration and new path computation.

IV. Results

A. Evaluation of the Fault Detection Protocol

Figure 5 presents the behavior of the communication in a given router (R15), assuming packets injected by router R1 (4x4 MPSoC size). Without faults, packets are received at south port 0 (S0) and transferred to the local port 0 (L0).

The first fault occurs at time *t8* (the 8th clock cycle in the Figure 5), at S0 (disabling the S0 channel of R15). Note that L0 partially consumes the packet, discarding it. The packet is retransmitted at time *t13*, using now the healthy south channel (S1). Such experiment corresponds to Figure 4(a) and Figure 4(c), i.e., degraded mode. In this experiment, the total time between sending the packet, fault detection and packet retransmission was 1,173 clock cycles. The largest portion of this time is consumed by the source processor, which executes the packet retransmission in the kernel.

Next, at time *t20*, a second fault is injected at S1, disabling the S1 channel of R15 and disabling the entire south link. Such experiment corresponds to Figure 4(b). At time *t25*, the packet is retransmitted, entering in R15 using the west port 0, and consumed at the local port 1 (L1). The local port changed due to the use of source routing in this particular example. In this experiment, the total time, between sending the packet at the source and receiving the packet at destination, was 2,420 clock cycles. The difference observed in the second experiment comes from: (*i*) path search and backtracking methods (in hardware); (*ii*) new path computation (in software).

Besides the shorter time required reestablishing the communication in case of a fault, the advantages of using a router in degraded mode include:

a) Packets that would use the faulty channel are automatically redirected to the healthy channel. Only the first packet using the

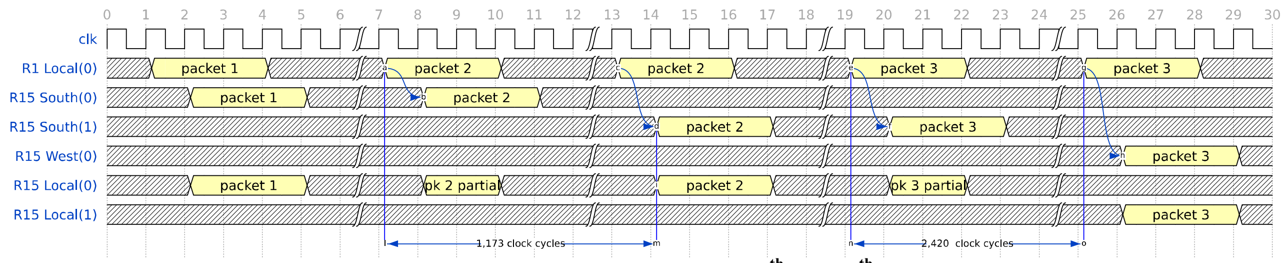

Figure 5 – Scenario with faults injected at the 8th and 20th clock cycles.

faulty channel is retransmitted after the fault detection. Next packets automatically bypass the faulty channel. Thus, even in the presence of a fault in a given port, the router operates as a fault-free circuit.

b) The incidence of the path search, backtrack, and path computation are drastically reduced (experiments presented in the next section), also reducing the need for routing tables to store source routing paths.

c) A finer grain fault recovery enables the system to support more faults, with smoother performance degradation.

B. *Fault injection campaign*

Two applications are used as benchmarks: *synthetic* (6 tasks) and an *MPEG* decoder (5 tasks). A random fault insertion campaign is used to validate different fault configurations. Stuck-at faults are injected into the router ports, excepting the local port because this paper assumes NoC faults only. An automatic fault analysis flow was created, which includes the MPSoC generation, the creation of fault scenarios, simulation of all scenarios, and performance evaluation. For each application, 1,000 scenarios are simulated (MPSoC modeled at the RTL level with VHDL). Five faults are injected at each scenario at the same moment, at random ports. The simulations demonstrate that the method supports several simultaneous faulty ports, as long as the number of faults does not physically split the network into disjoint graphs, i.e. there is no path from source to destination.

With the *synthetic* application, 246 scenarios were not affected by the faults while 754 require the execution of the proposed fault-tolerant method. Among the 754 faulty scenarios, 725 required only packet retransmission (96.13%). In the 725 scenarios requiring packet retransmission, 434(59.9%)/234(32.3%)/49(6.8%)/7(1.1%) scenarios required 1/2/3/4 retransmissions, respectively. As mentioned in the advantages of operating in degraded mode, the number of retransmissions is small, since after router reconfiguration, the router operates as a faulty-free circuit. The number of searches for new paths was 29, corresponding to 3.87% of the faulty cases. This result corroborates the statement related to the small incidence of new path computation.

Similar behavior was observed with the *MPEG* application. With the *MPEG* application, 391 scenarios were not affected by the faults while 609 require the execution of the proposed fault-tolerant method. Among the 609 faulty scenarios, 608 required only packet retransmission (99.83%). In the 608 scenarios requiring packet retransmission, 404(66.4%)/175(28.8%)/26(4.3%)/3(0.5%) scenarios required 1/2/3/4 retransmissions, respectively. Only 1(0.16%) scenario required path computation.

The execution time of applications is in practice not affected by the fault injection. The average execution time of the synthetic application, without faults, is 2.925 ms, while with faults 2.948 ms (+ 0.7%). The average execution time of the *MPEG* application, without faults, is 4.591 ms, while with faults 4.611 ms (+ 0.4%).

C. *Area Overhead*

The baseline PE received CRC modules and larger router buffers (4 extra bits for CRC). Targeting a Virtex5 FPGA the number of LUTs and FFs was 6298/2785 and 6678/2790 for the baseline and modified PE, respectively. This corresponds to an area overhead of 6.03%

V. CONCLUSIONS AND FUTURE WORKS

This work presented an integrated method for online fault detection in NoC-based MPSoCs. The proposal added fault tolerant features from the physical to the transport layer, integrating hardware modules and software functions implemented at the operating system level. Scalability is an important feature of the proposal since it does not employ tables to store the system status, and the size of the auxiliary NoC (PDN) does not increase with the system size.

An important fault-tolerant feature at the hardware level is the adoption of duplicated physical channel. These duplicate physical channels enable routers to operate in degraded mode. Most of the faulty simulated scenarios required only one or two packet retransmissions, (92.1% and 95.2% for the synthetic and MPEG applications respectively) with five simultaneous injected faults. The impact of the method at the applications' execution time is negligible (below 1%).

Directions for future works include: (1) periodically send test packets to faulty channels to distinguish permanent to transient faults to recover faulty channels; (2) migrate the affected tasks to other PEs when it becomes unreachable due to faults in its surrounding links; (3) reduce the hardware overhead at the NI. The network interface is responsible for storing the paths used by source routing. Such paths can be managed by the operating system, without affecting the performance; (4) extend the method to cope with faults at processors.

ACKNOWLEDGMENTS

The Author Fernando Moraes is supported by CNPq - projects 472126/2013-0 and 302625/2012-7, and FAPERGS - project 2242-2551/14-8. The Author Alexandre Amory is supported by CNPq – project 460205/2014-5.

REFERENCES

[1] Meloni, P.; et. al. *System Adaptivity and Fault-tolerance in NoC-based MPSoCs: the MADNESS Project Approach*. In: DSD, 2012, pp. 517-524.0.

[2] Fick, D.; et. al. *Vicis: A Reliable Network for Unreliable Silicon*. In: DAC, 2009, pp. 812-8170.

[3] Concatto, C.; Matos, D.; Carro, L.; Kastensmidt, F.; Susin, A.; Cota, E.; Kreutz. M. *Fault Tolerant Mechanism to Improve Yield in NoCs using a Reconfigurable Router*. In: SBCCI, 2009, 6p.

[4] Veiga, F.; Zeferino, C. *Implementation of Techniques for Fault Tolerance in a Network-on-Chip*. In: WSCAD-SCC, 2010, pp. 80-87.

[5] Lucas, A.; Moraes, F. *Crosstalk Fault Tolerant NoC - Design and Evaluation*. In: IFIP-VLSI-SOC, 2009, pp. 115-120.

[6] Vitkovskiy, A.; Soteriou, V.; Nicopoulos, C. A *Dynamically Adjusting Gracefully Degrading Link-Level Fault-Tolerant Mechanism for NoCs*. IEEE Trans. on CAD of ICs and Systems, v.31(8), pp. 1235-1248.

[7] DeOrio, A.; et al. *A Reliable Routing Architecture and Algorithm for NoCs*. IEEE Transactions on CAD, v.31(5), 2012, pp. 726-739.

[8] Fick, D. et. al. *A Highly Resilient Routing Algorithm for Fault-tolerant NoCs*. In: DATE 2009, pp. 21-26.

[9] Flich, J.; et al. *A Survey and Evaluation of Topology-Agnostic Deterministic Routing Algorithms*. IEEE Transactions on Parallel and Distributed Systems, v.23(3), 2012, pp. 405-425.

[10] Wachter, E.; Erichsen, A.; Amory, A.; Moraes, F. *Topology-Agnostic Fault-Tolerant NoC Routing Method*. In: DATE, 2013, pp. 1595-1600.

[11] Wachter, E.; Erichsen, A.; Juracy, L.; Amory, A.; Moraes, F. *A Fast Runtime Fault Recovery Approach for NoC-Based MPSoCS for Performance Constrained Applications*. In: SBCCI, 2014, 6p.