A Layered Approach for Fault Tolerant NoC-Based MPSoCs

— Special Session: Dependable MPSoCs —

Eduardo Wachter, Francisco Barreto, Vinicuis Fochi, Alexandre M. Amory, Fernando G. Moraes
FACIN - PUCRS – Av. Ipiranga 6681, 90619-900, Porto Alegre, Brazil
eduardo.wachter@acad.pucrs.br, fernando.moraes@pucrs.br

Abstract—Fault tolerant design has a key role in current nanometric technologies, leading to research on fault mitigation techniques for NoC-based MPSoCs. Most of the state-of-the-art papers present partial solutions to design a fault tolerant MPSoC, i.e., they present fault tolerant mechanisms for either NoCs or processing elements (PEs). The goal of this paper is to propose a comprehensive set of recovery mechanisms, organized in a layered stack, ensuring the correct execution of applications in the presence of transient or permanent faults, for both NoC and PEs. Faults injected into the NoC may induce it to operate in degraded mode or require the search of fault-free paths. In both cases, the communication is reestablished in less than 50 microseconds, using an end-to-end recovery mechanism. Faults injected into the PEs fire a lightweight and fast task relocation protocol, which executes in less than one millisecond.

Keywords—Fault tolerant MPSoC; fault recovery; fault tolerant communication; NoC.

I. FAULT TOLERANT MPSOC MODEL

The proposed layered approach works on an MPSoC model described as follows. An MPSoC consists of a set of PEs interconnected by a given network topology. Figure 1(a) illustrates the MPSoC model. Multiple PEs, called Manager Processors (local–LMP, global–GMP), are responsible to initialize the applications and to manage the system, improving system scalability and avoiding a single point of failure. The GMP has the same functionalities of the LMP, plus the coordination of the access to the task repository memory. Slave processors (SP) execute user tasks. The hardware of the SPs, LMPs, and GMP is the same. The difference is in the operating system (micro-kernel) executing at each PE class. The task repository is an external memory that stores all object codes of user tasks.

This paper adopts common features found in MPSoCs [1]:

- each PE (Figure 1(b)) contains one processor (32 bits MIPS), a true dual-port private memory (64 KBytes), DMA controller, Network Interface (NI), and the router;
- homogeneous processing, i.e., all PEs have the same architecture. The software part of the fault tolerant approach is entirely written in C, thus, it can be compiled to a different target processor, supporting heterogeneous MPSoCs;
- applications are modelled as task graphs, using message-passing as the communication method;
- the communication model is message passing (*send*() and *recv*() communication primitives);
- each SP can execute more than one task simultaneously (multi-tasking OS). There is no shared memory in the adopted MPSoC. Each local memory acts as a scratchpad memory.

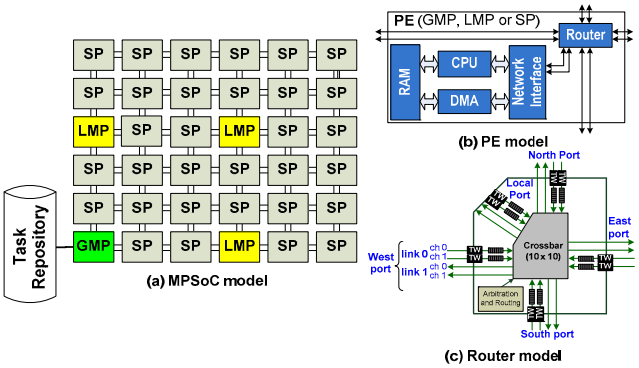

Figure 1 – MPSoC, PE and Router Models.

Figure 1(c) shows a simplified view of the NoC router. This work adopts the following common features found in NoCs [2]:

- 2D mesh topology – topology found in most NoCs [3] because its physical implementation is straightforward. Agarwal et al. [2] suggest that 2D meshes are more efficient in terms of latency, power consumption when compared to most other topologies;
- input buffering for temporary flit storage;
- wormhole packet-switching;credit-based flow control between adjacent routers;
- 5 bi-directional ports, 2 unidirectional links per port.

Important NoC features related to fault tolerance include:

- simultaneous support for distributed and source routing;
- test wrapper cells at the NoC's input ports;
- two 16-bit physical channels per link (no virtual channels),

978-1-5090-1331-9/16/$31.00 ©2016 IEEE

with two priority levels. Duplicated physical channels increase the router bandwidth without incurring the silicon cost of virtual channels [4];

- adoption of an auxiliary NoC, named Path Discover Network (PDN), used for fault tolerance purposes [5];
- adoption of CRC as error detection technique.

At the NoC level, the occurrence of a fault detection event isolates the faulty region to the neighbor routers. This feature is supported because each input port has wrapper cells (i.e. conventional test wrapper cells), as illustrates Figure 1(c). Each test wrapper (TW) cell can be individually controlled, isolating an input channel from the rest of the network in the presence of a fault, avoiding fault propagation. The isolation method requires minimal extra hardware and can be adapted to other router designs. Moreover, it enables a graceful reduction of the network performance as the number of faults increases since it is possible to shut off each router channel individually. Additionally, the *fault detection method is decoupled* from the rest of the fault recovery approach. A different test method can be used by activating the fault mode of the TW when it detects a defective router channel.

II. PROPOSED FAULT TOLERANT LAYERED APPROACH

This session briefly introduces to the proposed layered MPSoC services required to provide fault tolerance, illustrated in Figure 2. The detailed description is presented in the next sections. The Figure 2 is split into two main parts (hardware and software) representing the services embedded in the NoC hardware and the services implemented in software, in the micro-kernel. The MPSoC application designer creates the message passing tasks by using the reliable *send()* and *recv*() services provided by the operating system (transport layer). The system implements the FT features hidden from the programmer. Therefore, the programmer is not aware of the fault tolerance features used by *send*() and *recv*().

The tasks exchange packets using the NoC and, eventually, one message might not be delivered due to a defective router. The lower layer in Figure 2 represents the NoC *physical layer*, where the NoC *fault_detection*() service is implemented. In this case we adopt CRC to detect faults. The CRC module generates a fault signal to the upper layers.

After the fault detection, the *data link* layer immediately isolates the affected NoC channel. Any packet addressed to the faulty channel is drained (*packet_drain*() service) and discarded to avoid fault propagation by corrupted data. Depending on the number of faults in the router, the fault might be resolved locally by the data link layer, without affecting the rest of the system. This feature is possible because the network has two physical channels per link. Thus, if the fault is restricted to a single channel, the data link layer redirects the packets to the healthy channel (*replace_channel*() service). Otherwise, if the fault affects both channels of the same link, the fault signal must be propagated to the network layer.

The *network layer* is activated only when at least one entire link is affected by faults. In this case, it is necessary to find a new path for the packet, avoiding the defective link. The network layer replies to the packet source address informing that the packet has been drained. The packet source node receives this packet in the *transport layer* and requests a *path_search*() service to the network layer. The network layer returns the new path (if one exists) but the NoC hardware does not guarantee that this new path is deadlock-free. This function is implemented in the *resolve_deadlock*() service, located in the transport layer.

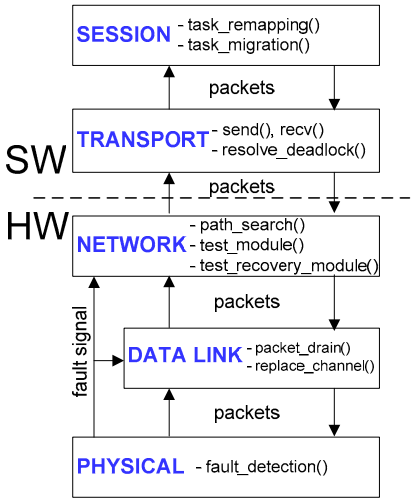

Figure 2 – Overview of the proposed layered MPSoC services to provide fault tolerance.

Faults might also affect the SPs. It is assumed that the SPs have some test procedure, which can notify a fault detection to its local manager. The local manager, which knows the tasks running in its SPs, invoke a *task_remapping*() service (in the manager's *session layer*) to find a new suitable place to the tasks which were running in the defective SP. The *task_remapping*() is an online procedure that takes into account the communication volume and the dependency between tasks to optimize task mapping onto PEs. The *task_remapping*() service returns new healthy SPs, and the *task_migration*() service is called to send the task's object code to the new SPs. Finally, the manager commands the SPs to restart the affected applications.

III. PHYSICAL AND DATA LINK LAYERS

The ***physical*** layer detects faults in the NoC and activates different recovery strategies. Fault detection occurs at the flit level. Each 16-bit flit receives four extra bits for the CRC field. Related works at this level include [6][7][8][9].

Figure 3 presents the router architecture for one port (input channels 0 and 1 and output channels 0 and 1). The CRC modules have the following roles:

- CRC decoders before input buffers (CRC1) – detects faults in the channels (e.g. caused by crosstalk or electromigration);

- CRC decoders after input buffers (CRC2) – detects faults in the buffers (e.g. caused by SEUs and wearout);
- CRC decoders after the crossbar (CRC3) – detects faults in the internal router circuitry, such as the crossbar.
- CRC coder at the output port (CRC4) – the use of source routing might require changing the content of header flits inside the router. Therefore, it is necessary to re-encode these flits.

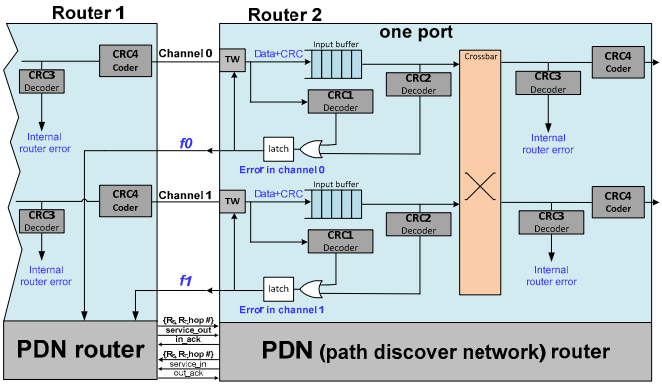

Figure 3 - Router architecture for one router port, with CRC and TW modules. The PDN router is presented with its external interface.

When CRC1 or CRC2 modules detect a fault, a fault signal is asserted (f_0 or f_1 in Figure 3). This signal disables the channel affected by the fault, and it is propagated to the upstream router (which sent the packet – router 1 in Figure 3) in such a way to notify the auxiliary NoC, PDN (Section IV). When module CRC3 detects a fault, the entire router is disabled because the internal shared router circuitry is defective. In this case, all fault signals are simultaneously asserted, isolating the entire PE from the NoC.

The main function of the ***data*** link layer is to *discard* corrupted packets. The function of the TW cells is twofold. First, the TWs assert a signal of the flow control protocol (credit based), in such a way as the upstream drains all remaining flits of the packet. In practice, this corresponds to a *packet draining* action. The second function of the TWs is to block corrupted flits, avoiding its transmission to healthy routers, avoiding Byzantine faults.

If a fault is detected in the middle of the packet, the TW blocks the packet tail, but part of the packet continues to travel to its destination. The PE 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 NI or the micro-kernel of the PE_D discards an incoming packet when the flow of flits is disrupted for a given number of clock cycles.

IV. NETWORK LAYER

This section details the auxiliary NoC, PDN (IV.A), the NoC operation in degraded mode (IV.B), and the path reconfiguration via source routing (IV.C). In the absence of faults, PEs communicate using the deterministic XY routing algorithm. The routing algorithm only changes to source routing if there are permanent faults in the path.

Related works at this level include [10][11][12][13][14][15].

A. *Path Discover Network (PDN)*

The main goal of the PDN [5] is to provide the *path_search*() service (in Figure 2) by propagating a *wave* from a source to a target PE, visiting all PEs of the MPSoC, equivalent to a broadcast communication. The main *innovation* compared to a conventional broadcast on a packet-switching network is that PDN is specially designed and dedicated network for this communication pattern.

The PDN is a parallel network, loosely coupled to the router. The PDN has the same number of ports as the main network, but it has wider channels and packets have a single flit, easing the broadcast and simplifying the hardware design (e.g. no buffers are required at the input ports). It contains a small memory equivalent to 4 flits to store the current broadcasts. For the sake of scalability, this memory size is fixed and independent of the network size. This memory temporally stores the path taken by the wave until it reaches the destination.

The service types supported by the PDN include:

- *seek_resend* - generates a wave from a given PE asking a packet retransmission;
- *seek_unreachable* - generates a wave from a given PE reporting that the R_T is unreachable due to a fault in the path R_S to R_T;
- *seek_new_path* - generates a wave used to find a fault-free path from R_S to R_T;
- *clean_pdn* - generates a wave responsible for cleaning all PDN memories with entries equal to {R_S, R_T} (source and target of the communication).

B. *Operation in Degraded Mode*

The adoption of two physical channels per link enables the NoC to operate in a degraded mode. This corresponds to the *replace_channel*() service in Figure 2, which switches channels bypassing the faulty one. The advantages to operate in degraded mode are: (1) avoid reconfiguration at higher layers (routing algorithm, micro-kernel system) because the recovery has higher cost; (2) as a fault may be transient, the router may detect the end of the fault and return to the normal mode; (3) graceful network performance degradation.

Each fault signal *f* in Figure 3 enters in the PDN. If there is only one faulty channel, the connection between routers is degraded, but still operational. In this case, two actions are executed. First, the module controlling the crossbar of the router is reconfigured, in such a way to use only the healthy channel. Next, the PDN requests a packet retransmission (*seek_resend* service) to the source PE.

Figure 4(a) presents a fault-free communication between R_S and R_T, represented by the yellow squares in the links. A fault is detected at R2 by the CRC modules, the TW cells of the affected south channel are activated, the crossbar of R1 is reconfigured to avoid the affected output north channel, and the PDN router of R1 starts a wave to R_S with the service *seek_resend* (Figure 4(b)). Note that by activating the TW cells, all in-fly flits are discarded, avoiding stalls in the routers' buffers. When R_S receives the *seek_resend* wave, two actions are executed: the packet affected by the fault is retransmitted, and a second wave is propagated, with the service *clean_pdn*. The retransmitted packet and all the next packets to the same destination use the healthy channel (Figure 4(c)).

The detected faults may be permanent or transient. In both cases, as shown in Figure 4, the channel is disabled. A *non-intrusive online test method* is proposed to recover the router in the presence of transient faults. The method relies on two hardware modules: *test module* (TM) and *test recovery module* (TRM). The method is completely implemented in hardware, transparent to the software layer. Each router has one TM. The TM executes the following actions:

- Detects a faulty channel. A side-band signal (f_0 and f_1 in Figure 3) notifies a given router that some input channel of a neighbor router failed.
- Starts a timer to inject the test packet. The execution of the test occurs after the effect of the transient fault.
- Injects the test packet in the faulty channel. This test packet may be safely injected into the faulty channel since the control logic of the router has disabled this channel to transmit data packets.
- At the end of the test packet, the same side-band signal f_x signalizes if the channel may be restored (f_x='0') or if the fault is permanent (f_x='1').

C. Path Search

The previous section assumed a fault in only one channel of the link. If both channels of the link are faulty, the path between R_S and R_T must change to preserve the correct communication between PEs.

Figure 5 summarizes the routing method. Figure 5(a) presents the fault detection, blocking the path between R1 and R2. R1 notifies to the source PE that the path is broken through a wave with *seek_unreachable* service. The source PE cleans the first wave and starts a new wave with the service *seek_new_path* (Figure 5(b)). When the *seek_new_path* wave reaches the target router, a backtrack process starts, creating the new path between R_S and R_T (Figure 5(c)).

Once R_T is reached, the backtrack process starts. In this second step, the PDN of R_T injects into the main NoC a packet that will contain the new path. This packet is called *backtrack packet*.

The third step of the method starts once R_S receives the packet containing the backtrack path. It includes two actions: propagate a wave with the service *clean_pdn* and the new path evaluation. The new path may present turns leading to deadlocks. To avoid deadlocks the NoC in divided into two disjoint networks, each one implementing a partial adaptive routing algorithm, west-first and east-first.

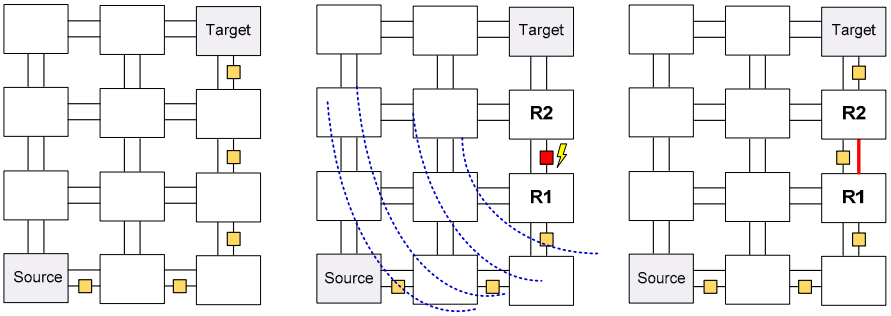

Figure 4 – Communication in degraded mode with a single faulty channel.

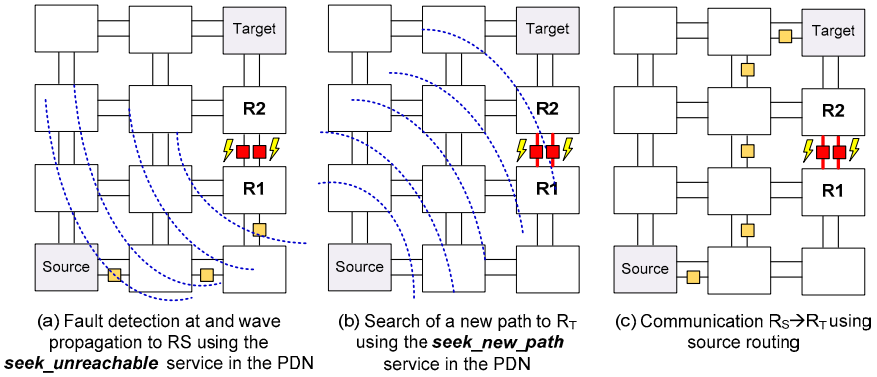

Figure 5 - Communication in the presence of permanent faults in both channels.

V. TRANSPORT LAYER

Related works at this level include [16][17][18]. 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 a non-blocking *Send()*, a dedicated memory space in the OS, named *pipe*, stores messages written by tasks.:

Figure 6 details the communication protocol. The execution of a *send()* on task A (PE_1) 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 *message request* packet is sent to PE_1 resulting in two actions. First, the pipe status changes to *WAITING_ACK* (1 in Figure 6). Then, the message is delivered to PE_2 (2 in Figure 6), and it is transferred to the task B memory space. When PE_1 receives a new message request (3 in Figure 6), the pipe position with a *WAITING_ACK* status is released (4 in Figure 6). If a pipe position has a status equal to USED, a new message may be transferred.

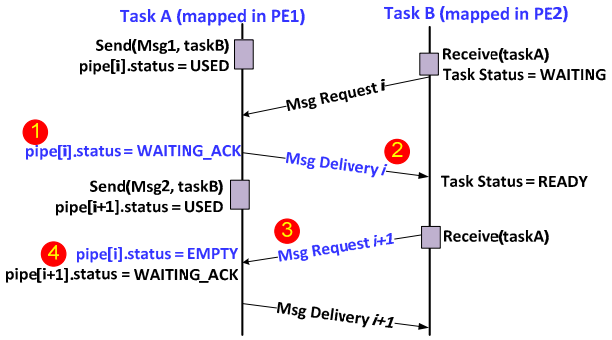

Figure 6 - Communication protocol to exchange data packets [19].

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 micro-kernel may receive two messages from the PDN: *seek_resend* (Figure 4) or *seek_unreachable* (Figure 5). In the first case, the micro-kernel extracts from the PDN message the address to the destination PE, and searches in the *pipe* area the packet to be retransmitted. The second message is *seek_unreachable*, where the PE requires the search for a new path. The micro-kernel configures the PDN to broadcast a path to the target PE (PE_T) and waits the reception of a message from PE_T with the new path (embedded in the *backtrack_path* message). The PE stores the new path, and all communications between the PE and PE_T will use this new path, transmitted with the use of source routing. The packet retransmission occurs after the definition of the new path.

VI. SESSION LAYER

Related works at this level include [20][21][22][23].

The micro-kernel provides services to the session layer. Those services include dynamic application mapping, load balancing, monitoring, QoS support, among others. The proposed application recovery protocol is implemented at the micro-kernel, as OS services. The application recovery proposal required [24] four new services:

- *Fault notification message*: message sent from the fault detection mechanism to its local manager to inform the detection of a defective SP.
- *Freeze task*: message sent from the local manager to a set of PEs so that they can put certain tasks in freeze state until the recovery is complete. A task in a freeze state cannot be scheduled by the micro-kernel and cannot send/receive messages.
- *Unfreeze task*: message sent from the local manager to a set of PEs, changing the task status from freeze to the ready state. The unfreezed task initially cleans up old data context and starts the task execution from the beginning. It works similarly to an application reboot where any previous data is lost, and the application is ready to process new data.
- T*ask relocation*, notifies the network address of a task to the other tasks of the same application.

The protocol works as follows, with all actions executed in a manager PE (PE_M), LMP or GMP:

1) PE_M receives a fault notification message, indicating a fault at PE_F.
2) PE_M constructs a set $TF = \{t_1, t_2, ..., t_n\}$ representing the affected tasks running in the defective PE_F.
3) PE_M identifies for each t_i in TF the applications affected the fault, constructing a new set $TA = \{A_1, A_2, ..., A_m\}$, being A_i an application to be frozen.
4) PE_M sends a *freeze message* for all tasks of each application in TA.
5) PE_M relocates only the tasks in TF to healthy PEs. All other unaffected tasks stay in the same location. After the dynamic task mapping, the unaffected tasks receive the new addresses for the relocated tasks.
6) PE_M sends the *relocation* and the *unfreeze messages* for all tasks of each application in TA.

VII. CONCLUSIONS AND FUTURE WORK

This paper presented a layered approach for runtime fault recovery in MPSoCs. The method has contributions in all layers, being the first proposal to addresses a comprehensive fault tolerant approach.

The proposal has limitations, which are reference for future works:

1) Manager PEs are assumed fault-free;
2) The external memory is also assumed fault-free, and the interface with this memory is a single point of failure. Replicating the external memory and the PEs with access to it could mitigate this limitation;
3) The FT communication protocol was designed for message-passing communication. Shared-memory communication requires adaptation in the proposed protocol.

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] Javaid, H.; Parameswaran, S. “Pipelined Multiprocessor System-on-Chip for Multimedia”. Springer, 169p., 2014.

[2] Agarwal, A.; Iskander, C.; Shankar, R. “Survey of Network on Chip (NoC) Architectures & Contributions”. Journal of Engineering, Computing and Architecture, v.3(1), 15p., 2009.

[3] Salminen, E.; Kulmala, A.; Hämäläinen, T. “Survey of Network-on-chip Proposals”. White Paper, OCP-IP, 13p., 2008.

[4] Carara, E.; Calazans, N.; Moraes, F. “Router architecture for high-performance NoCs”. In: SBCCI, pp. 111-116, 2007.

[5] Wachter, E.; Erichsen, A.; Amory, A.; Moraes, F. “Topology-agnostic fault-tolerant NoC routing method”. In: DATE, pp. 1595-1600, 2013.

[6] Cota, E.; Amory, A; Lubaszewski, M. “Reliability, Availability and Serviceability of Networks-on-Chip”. Springer, 2012, 209p.

[7] Fick, D.; DeOrio, A.; Chen, G.; Bertacco, V.; Sylvester, D.; Blaauw, D. “A Highly Resilient Routing Algorithm for Fault-Tolerant NoCs”. In: DATE, pp. 21-26, 2009.

[8] Concatto, C.; et al.. “Fault Tolerant Mechanism to Improve Yield in NoCs using a Reconfigurable Router”. In: SBCCI, 2009, 6p.

[9] Vitkovskiy, A.; Soteriou, V.; Nicopoulos, C. “A Dynamically Adjusting Gracefully Degrading Link-Level Fault-Tolerant Mechanism for NoCs”. IEEE Transactions on CAD of Integrated Circuits and Systems, v.31(8), pp.1235-1248, 2012.

[10] Rodrigo, S.; et al. “Cost-Efficient On-Chip Routing Implementations for CMP and MPSoC Systems”. IEEE Transactions on CAD of Integrated Circuits and Systems, v.30(4), pp. 534-547, 2011.

[11] Sem-Jacobsen, F.; Rodrigo, S.; Skeie, T. “iFDOR: Dynamic Rerouting on-Chip”. In: International Workshop on Interconnection Network Architecture: On-Chip, Multi-Chip, pp 11-14, 2011.

[12] Feng, C.; Lu, Z.; Jantsch, A.; Li, J.; Zhang, M. “A Reconfigurable Fault-Tolerant Deflection Routing Algorithm Based on Reinforcement Learning for Network-on-Chip”. In: NoCArc, pp. 11-16, 2010.

[13] Flich, J.; Mejia, A.; Lopez, P.; Duato, J. “Region-Based Routing: An Efficient Routing Mechanism to Tackle Unreliable Hardware in Network on Chips”. In: NOCS, pp. 183-194, 2007.

[14] DeOrio, A.; et al. “A Reliable Routing Architecture and Algorithm for NoCs”. IEEE Transactions on CAD of Integrated Circuits and Systems, v.31(5), pp. 726-739, 2012.

[15] Fick, D.; DeOrio, A.; Jin Hu; Bertacco, V.; Blaauw, D.; Sylvester, D. “Vicis: A Reliable Network for Unreliable Silicon”. In: DAC, 2009, pp. 812-817.

[16] Aulwes, R.T.; et al. “Architecture of LA-MPI, a Network-Fault-Tolerant MPI”. In: PDPS, pp. 15- 26, 2004.

[17] Kariniemi, H.; Nurmi, J. “Fault-Tolerant Communication over Micronmesh NOC with Micron Message-Passing Protocol”. In: SoC, pp. 5–12, 2009.

[18] Zhu, X.; Qin, W. “Prototyping a Fault-Tolerant Multiprocessor SoC with Run-Time Fault Recovery”. In: DAC, pp. 53–56, 2006.

[19] Fochi, V.; Wachter, E.; Erichsen, A.; Amory, A.; Moraes, F. “An Integrated Method for Implementing Online Fault Detection in NoC-based MPSoCs”. In: ISCAS, 2015.

[20] Gizopoulos, D.; Psarakis, M.; Adve, S.V.; Ramachandran, P.; Hari, S.K.S.; Sorin, D.; Meixner, A.; Biswas, A.; Vera, X. “Architectures for online error detection and recovery in multicore processors”. In: DATE, 6p., 2011.

[21] Bolchini, C.; Carminati, M.; Miele, A. “Self-adaptive fault tolerance in multi-/many-core systems”. Journal of Electronic Testing, v.29(2), pp. 159-175, 2013.

[22] Hébert, N.; Almeida, G.M.; Benoit, P.; Sassatelli, G.; Torres, L. “Evaluation of a Distributed Fault Handler Method for MPSoC”. In: ISCAS, pp. 2329–2332, 2011.

[23] Ter Braak, T.D.; Burgess, S.T.; Hurskainen, H.; Kerkhoff, H.G.; Vermeulen, B.; Zhang, X. “On-line dependability enhancement of multiprocessor SoCs by resource management”. In: SoC, pp. 103-110, 2010.

[24] Barreto, F.F.S., Amory, A.M., Moraes F.G: “Fault recovery protocol for distributed memory MPSoCs”. In: ISCAS, pp. 421-424, 2015.