Phoenix NoC: A Distributed Fault Tolerant Architecture

César Marcon, Alexandre Amory, Thais Webber, Thomas Volpato, Letícia B. Poehls

Pontifical Catholic University of Rio Grande do Sul
Av. Ipiranga 6681, Porto Alegre, Brazil

cesar.marcon@pucrs.br

Abstract—**The advances in deep submicron technology have made the development of large Multiprocessor Systems-on-Chip (MPSoC) possible and Networks-on-Chip (NoCs) have been recognized to provide an efficient communication architecture for such systems. With the positive effects on the device's integration some drawbacks arise, such as the increase of fault susceptibility during the MPSoC manufacturing and operation. This work presents Phoenix, which is a direct mesh NoC that implements fault tolerant mechanisms in order to enable end-to-end communication when some links fail. Phoenix implements a distributed fault tolerant mechanism in software (i.e. in each processor) and in hardware (i.e. in each router). Experimental results show that Phoenix is scalable and allows the MPSoC operation even in the presence of several faulty links.**

Keywords - Fault tolerance, NoC, MPSoC.

I. INTRODUCTION

Recent submicron technologies allow to integrate billion transistors into a single chip, creating high performance Systems-on-Chip (SoCs). These technologies are increasingly susceptible to faults due to the increasing complexity of the submicron processes. Since the presence of faults detected after the SoC's manufacturing (or during the SoC's operation) may hinder the commercialization, industry and academia spend much research effort on issues regarding fault tolerance.

High performance SoCs, commonly called Multiprocessor SoCs (MPSoCs), have many Processing Elements (PEs), which operate in parallel in order to achieve the application's functionality. It is essential that the MPSoC's communication architecture is efficient regarding performance including low latency and high throughput. Networks-on-Chip's (NoCs) architectures are typically designed to meet such communication features [1]. Furthermore, the parallelism of NoC links are designed to provide redundant communication among resources. If a link fails, another route can be employed with parallelism reduction penalty, which is likely to reduce NoC flow without impact on the application's functionality.

NoCs tolerance to *operating faults* (faults that occur during the MPSoC operation) is designed to offer mechanisms to detect and recover from these faults. The efficiency of these mechanisms determines whether the system is not only able to withstand the detected faults, but also the delay needed to recover the system. This paper describes the design of a 2D mesh NoC named Phoenix, which represents a fault-tolerant NoC with source routing tables and mechanisms to detect faults and to efficiently distribute the information to all PEs. The proposed Phoenix NoC is able to deal with both manufacturing and operating faults. The main issues addressed in this paper are: (i) the NoC architecture as well as its impact on the silicon area, (ii) the mechanism implemented in order to detect faults (i.e. link analysis at idle moments), and (iii) the distributed algorithm proposed to report faults. Although the routing algorithm is undoubtedly important, it is not specifically addressed in this paper.

The paper is organized as follows: Section II describes the hardware and software of the monitoring mechanism of Phoenix NoC. Section III explains the fault tolerant mechanisms implemented on Phoenix. Section IV presents and discusses some related works, while Section V shows experimental results of the proposed mechanisms and their implementation. Finally, Section VI concludes this work.

II. PHOENIX NOC ARCHITECTURE

Figure 1 shows the Phoenix distributed mechanism, which places the hardware part (i.e. *HwPhoenix*) on each router and the software part (i.e. *OsPhoenix*) on the Operating System of each PE connected to each router.

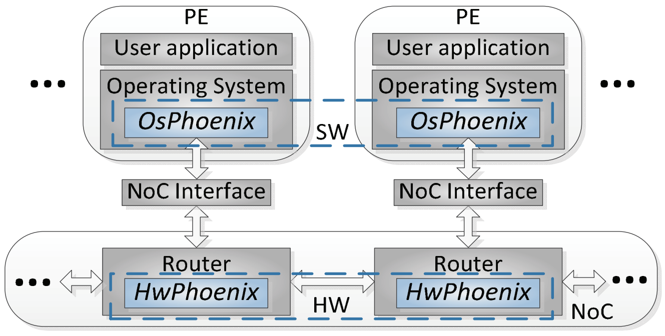

Figure 1 – Phoenix's distributed architecture.

A. Phoenix NoC Fundaments

Phoenix NoC has direct 2D mesh topology, consisting of $m \times n$ routers using bidirectional links to interconnect PEs placed alongside with them. The NoC employs routing tables for source routing decisions and the *OsPhoenix* performs routing algorithms to fill the routing table according to the PE position and the faulty links. Further, Phoenix NoC implements wormhole switching, which divides packets into flits (the flit size of Phoenix is equal to the phit size), needing only small buffers for the necessary data storing. Additionally, the Phoenix NoC uses a credit-based flow control to reduce the number of clocks needed for flits' transmission.

B. Phoenix Router Architecture

Figure 2 shows the Phoenix router architecture. The router architecture is based on a routing table with an extra module, which implements the fault tolerance mechanism. The basic router architecture of Phoenix encompasses four components described in the following: (i) Four bidirectional ports, dedicated to interconnect routers (NORTH, SOUTH, EAST and WEST), and a bidirectional port that enables the communication between the router and its local PE (LOCAL). The input link contains configurable buffers used when other packets congest the routing path. (ii) A *Crossbar switch* that establish unblocking connections between input and output ports, and (iii) a *Routing Table* associating ranges of NoC addressing to output ports. At last, (iv) a *Switch Control* circuit that performs packets routing and arbitration according to the *Packet Header* and to the *Routing Table*. The arbitration follows a dynamic rotating policy to ensure that all incoming requests are processed, avoiding the so-called starvation phenomenon.

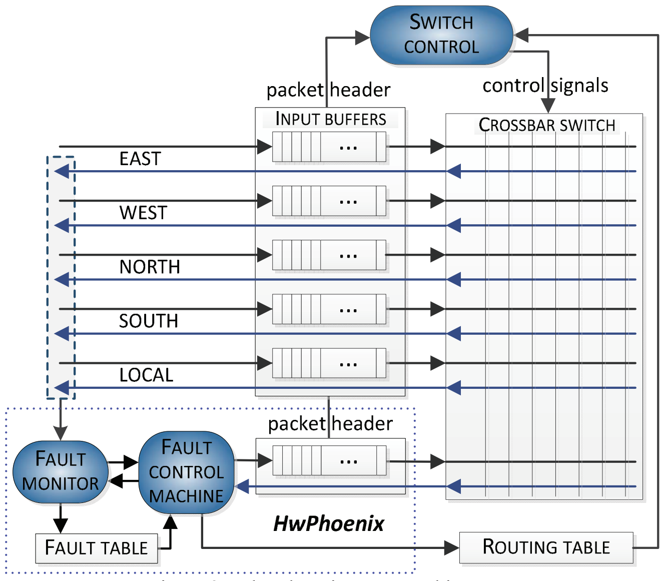

Figure 2 - The Phoenix router architecture.

The *HwPhoenix* module basically includes two parts: (i) the *Fault Control Machine*, which searches for control packets in all input ports, and takes decisions according to the command code (for details refer to Section II.D) and (ii) the *Fault Monitor*, which detects faulty output links and sets the links' status on the *Fault Table*, which is a 4-field vector. Each field stores the operation status of the NORTH, SOUTH, EAST and WEST output links, containing two bits to inform, weather the link is (i) not verified, (ii) faulty or (iii) operating properly.

C. Fault Tolerant Routing Algorithm

Phoenix is a source routing NoC, where routes are computed according to the *Routing Table*, which is initialized by XY routing. Nonetheless, depending on the occurrence of faults, this table of *OsPhoenix* can modify the *Routing Table* by adding new deadlock-free routes. *OsPhoenix* presents a routing algorithm that is similar to Region Based Routing [9], which groups target addresses into regions in order to reduce the Routing table size. In addition, the *Routing Table* provides several paths, even in the presence of faults, with a minimum of four regions. If the *Routing Table* size increases, Phoenix may provide an alternative for the minimum path using e.g. heuristic algorithms, which are not the focus of this work.

D. Packet Format

Each field of a Phoenix packet has a length of exactly 1 flit and the number of flits in a packet is limited to $2^{(\text{flit size in bits})}$. Phoenix employs two types of packets: (i) the *Data Packet*, which carries the PE messages; and (ii) the *Control Packet*, which is employed by the router control mechanisms.

Figure 3 illustrates Phoenix's packets format. The header encloses flag_address and size fields. The flag_address is the first flit of the header, composed of (i) a 1-bit flag to define the packet type. A flag set to zero defines Data Packets, whereas in Control Packets the flag is set to one; and (ii) the XY target address, whose addressing capacity depends on the flit size. Moreover, Phoenix operation requires minimum flit length of 8-bits for supporting the flag_address field, the control flag and 64 PE addresses. The size field is the second flit of the header, containing the quantity of flits that compose the packet payload. Whereas data packet payload is entirely transparent to NoC operation, the first flit of the control packet payload is a command code used to control the routers' operation.

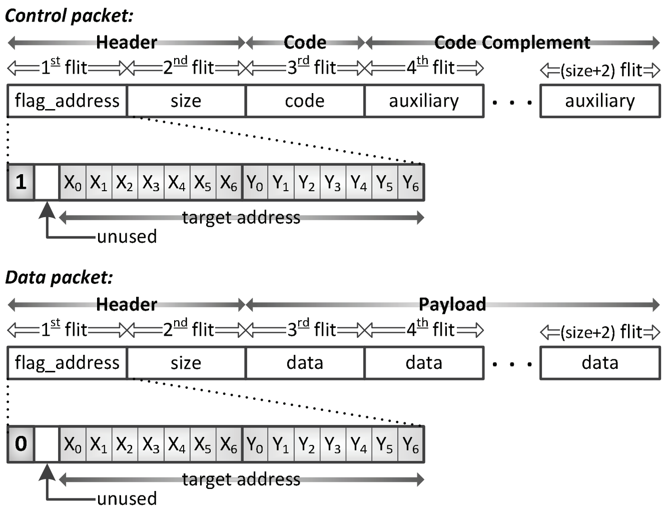

Figure 3 – Packets of Phoenix NoC considering flits of 16-bits length.

Phoenix fault tolerance mechanism implements the next commands: TEST_LINKS, TR_ROUT_TAB, RD_FAULT_TAB, WR_FAULT_TAB, RD_ROUT_TAB, WR_ROUT_TAB, RST_FAULT_TAB, TR_FAULT_TAB and RST_ALL_FAULTS, which are detailed in Section III.

III. FAULT TOLERANCE MECHANISMS

This section describes the joint operation of *OsPhoenix* and *HwPhoenix* performing Phoenix fault tolerance mechanisms.

A. OsPhoenix Description

OsPhoenix is a small software layer placed into the PE's Operating System, which contains drivers for high-level operation and routines, which implement the distributed fault tolerant mechanisms.

OsPhoenix comprises a *Global Fault Table, which* stores the status of all NoC links, informing if a given link was or was not tested and, once tested, if it is operating properly. Employing command, *OsPhoenix* makes four fault tolerant mechanisms possible, all four, being: (i) *Fault Detection*; (ii) *Fault Notification*; (iii) *Fault Re-evaluation*; and (iv) *Packet Drop*, are described in detailed below.

B. Fault Detection Mechanism

The Phoenix fault model considers only faults on output ports of inter router links, which encompasses links on ports NORTH, SOUTH, EAST and WEST. Figure 4 shows a diagram for the flow of the *Fault Detection Mechanism*, starting with *OsPhoenix* sending the TEST_LINKS command to *HwPhoenix*.

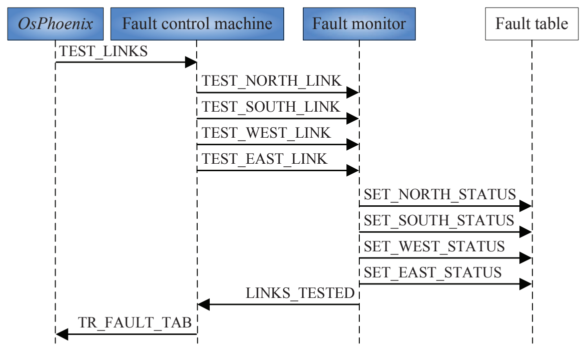

Figure 4 – Flow Diagram of the Fault Detection Mechanisms.

The *Fault Control Machine* interprets the TEST_LINKS command sending a predefined test packet to each output port. At same time, the *Fault Control Machine* sends an internal command (e.g. TEST_EAST_LINK) to the *Fault Monitor* informing that it has to analyze the link's quality. When the neighbour router receives a test packet, it loops back a packet with the same information. Consequently, the *Fault Monitor* detects faulty links in the case that one of the following conditions occurs: (i) the low level control protocol fails; (ii) the test packet is not replied; or (iii) the test packet is replied but with altered content. Otherwise, the link is considered tested and operating properly.

When the *Fault Monitor* finishes testing all link, it sends the internal LINKS_TESTED command to the *Fault Control Machine*, which itself sends the TR_FAULT_TAB command to the *OsPhoenix* containing the *Faulty Table* enclosed in the *Code Complement* packet field (as seen in Figure 3).

C. Fault Notification Mechanism

The fault notification is a distributed mechanism, which provides the status of all NoC links to *OsPhoenix*. This mechanism starts whenever an *OsPhoenix* receives a *Fault Table* whose values differ from the ones previously stored in the *Global Fault Table*. In a next step, the *OsPhoenix* performs its *Fault Notification Mechanism* propagating fault information to all the *OsPhoenix* of its neighbour routers until it does reach the stabilization condition. Normally, the *Fault Notification Mechanism* starts after the execution of the *Fault Detection Mechanism*, due to the need of the information from the completed *Fault Table*.

Figure 5 illustrates the *Notification Mechanism*, which comprehends the following steps:

1. Whenever *OsPhoenix* receives a control packet with the *Fault Table's* content (i.e. throughout the TR_FAULT_TAB command), it verifies its contents against the previous *Fault Table* stored in the *Global Fault Table*. If at least one value presents alterations, *OsPhoenix* updates the *Global Fault Table* and sends new packets with the same content to all other neighbour PEs, otherwise *OsPhoenix* discards the received packet.
2. At the beginning of *Fault Notification Mechanism's* operation, the OsPhoenix starts a timer, which operates in the same clock cycle as the NoC. The timer is used to check the *Maximum Stabilization Time Period* (MSTP), which is the condition of the fault notification's stabilization mechanism. When the timer reaches the value of MSTP, the *OsPhoenix* considers that all the faults were passed to all the other *OsPhoenix* in order to compute the routing algorithm.
3. When the distributed notification mechanism finishes, the *OsPhoenix* sends the command WR_ROUT_TAB, which contains the *Routing Table* enclosed in the *Code Complement* field (as depicted in Figure 3) to the *Fault Control Machine*.

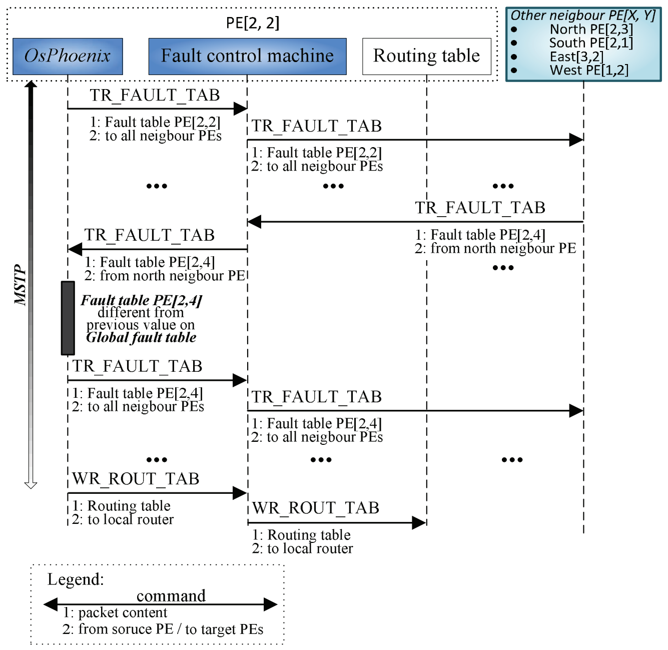

Figure 5 – Exemplified partial operation of the Fault Notification Mechanism / PE placed on coordinates X=2, Y=2).

Figure 6 shows an example of fault notification propagation in a 6×5 Mesh NoC.

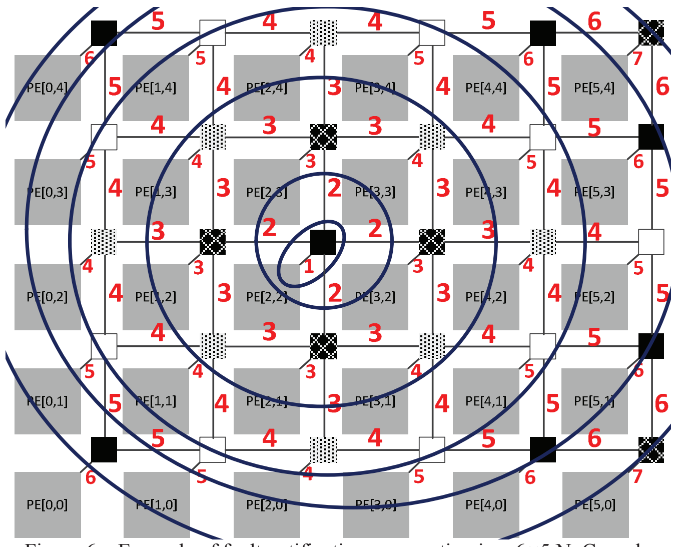

Figure 6 – Example of fault notification propagation in a 6×5 NoC mesh.

The algorithm propagates fault notifications from the center to the borders in a wave format. MSTP is proportional to the maximum NoC length, which is dependent on the NoC size and on the quantity and positioning of faulty links. Since these faults are not known during design, we use here MSTP as the maximum NoC length delay, which is a worst-case condition achieved by a packet passing through all routers.

D. Fault Re-Evaluation Mechanism

The *Fault Notification Mechanism* support only permanent faults. This approach facilitates the notification mechanism to stabilize quickly. However, the NoC supports reassessment of the faulty status of all NoC links, which is required by any *OsPhoenix* in a distributed way using the RST_ALL_FAULTS command. Figure 7 illustrates this mechanism, which guarantees that the fault tolerance mechanism of Phoenix does not accidentally detect a transient fault as a permanent one.

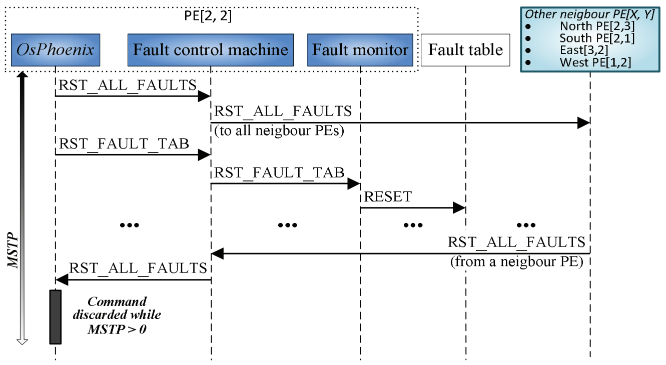

Figure 7 – Fault re-evaluation mechanism.

The *reset* command is sent to all neighbouring PEs that upon receiving the command: (i) reset their *Global Fault Table*; (ii) request to their local routers to reset the *Fault Table* using the RST_FAULT_TAB command; and (iii) retransmit the command RST_ALL_FAULTS to their respective PE neighbours. During MSTP, when the NoC stops running, occurs the propagation of system's reboot message. After, the operation returns to normal as in the start-up, running a *Fault Detection Mechanism* followed by the *Fault Notification Mechanism*.

E. Packet Drop Mechanism

Some NoC links may fail during Phoenix's operation as described in the previous two sections. These faults are taken into account similarly to the initial NoC operation. However, routers, which have not been updated with this fault information, consider that the fault does not exist and the router may consider transmitting packets along the failed path.

In order to avoid packets trapped inside the NoC due to a faulty link, a *Packet Drop Mechanism* has been implemented. The flits that already passed through the faulty link compose an incomplete packet, which is propagated through all routers until it reaches the targeted PE. Therefore, *OsPhoenix* discards the flits, since they compose an invalid packet. Additionally, when packet size does not match the quantity of flits, the output port recognizes it as a faulty packet. Thus, the internal control registers are updated to enable the transmission of further packets. The router eliminates the remaining flits, one flit per clock cycle. When a packet is dropped, its content is lost and it has to be sent again at a higher software levels.

F. Debugging Commands

For debugging reasons, the fault tolerance mechanism of Phoenix encompasses the commands of Table I, enabling the *Faulty Table* and the *Routing Table*.

TABLE I – DEBUGGING COMMANDS OF PHOENIX.

Command	Description
RD_FAULT_TAB	*OsPhoenix* employs command RD_FAULT_TAB in order to read the content of the *Fault Table*. When the *Fault Control Machine* receives this command, it replies with the command TR_FAULT_TAB, containing the *Fault Table* codified into the 2nd flit of the *Code Complement* field
WR_FAULT_TAB	*OsPhoenix* uses the command WR_FAULT_TAB in order to set the status of the links NORTH, SOUTH, EAST and WEST, inside the *Fault Table* of the local router
RD_ROUT_TAB	*OsPhoenix* sends the command RD_ROUT_TAB to the *Fault Control Machine* in order to read the *Routing Table*
TR_ROUT_TAB	The *Fault Control Machine* replies the command RD_ROUT_TAB by inserting the *Routing Table* into the *Code Complement* field. Notice that the *Routing Table* size, and consequently the packet size, depend on the number of regions defined by the Routing Mechanism

IV. RELATED WORKS

NoC-based MPSoCs present superior performance in terms of bandwidth and scalability. Their use provides a great opportunity for the research on fault tolerance methods, mainly because they may be implemented in hardware, software or a combination of both.

In literature different fault tolerance methods based on redundancy are described as being applied to NoCs. Among these are: ECC, spare wires, spare routers and backup NoC paths [2], which all apply extra redundancy to the NoC. Similarly, methods that exploit the natural path redundancy existent in a variety of network topologies may possibly reduce the hardware overhead. The importance of the latter approaches is that multiple paths are still not sufficient to build resilient systems. For instance, a NoC will typically have multiple possible routes between end-to-end communications. However, a single fault is sufficient to disturb the entire system, if no proper methods for NoC fault detection, diagnose, and recover are used. In the following works related to this topic are summarized and compared in Table II.

Vicis NoC [3] is able to preserve the functionality of the system based on the inherent redundancy found in most networks and even reducing the hardware overhead. The authors compare this approach with implementations based on Triple Modular Redundancy (TMR). Each router has a built-in Self-Test to diagnose faults and reconfigure the hardware in order to bypass faulty regions. The method is implemented in hardware with 42% of area overhead and a greater fault tolerance when compared to TMR methods.

Most fault-tolerant routing algorithms avoid faulty regions but apply restrictions that may reduce the NoC performance. Moreover, most works require virtual channels used only for fault tolerance, then increasing the silicon area. Fick et al. [4] present a routing algorithm to configure the NoC in order to maintain functionality with faulty components. The proposed method, based on routing tables and without any virtual channel, requires about 300 logic gates for each router regardless NoC size and can support 10% of faulty links.

TABLE II - RELATED WORK SUMMARY.

Work	Fault location	Implementation	Base approach	Fault duration	Means to dependability	Area scalability	Quantity of faults	Area overhead
[3]	router	hardware	bypass, BIST, ECC	permanent	diagnose, reconfiguration	-	50% of routers	42% of router
[4]	link	hardware	routing table	permanent	reconfiguration	yes	10% of links	300 gates/router
[5]	router, region	hardware	turn-based routing algorithm	permanent	reconfiguration	yes	1 faulty router	8% of router
[6]	router	hardware	routing table	permanent	reconfiguration	no	-	> 100% of router
[7]	message	hardware	heartbeat messages	permanent, transient	detection	yes	-	small
[8]	link, router	hardware	hierarchical routing algorithm	permanent, transient	detection, reconfiguration	yes	10% (links, routers)	< 3% of router
This	link	hardware/software	routing table	permanent, transient	detection, reconfiguration	yes	large	> 35% of router

Zhang et al. [5] propose a method to avoid deadlock. This approach adopts a turn-based fault tolerant approach to avoid routing cycles. It has been proven that the approach is deadlock free for any one-faulty-router and the silicon cost is 8% compared to the base router. It is using two NoCs with each node connected to two routers; case a single link is recognized as faulty, the adjacent routers are entirely disabled.

Feng et al. [6] proposes a fault-tolerant solution for a bufferless NoC including the detection of both transient and permanent faults. This paper proposes the reconfiguration of a routing table during packet transmission through the Reconfigurable Fault-Tolerant Deflection Routing (FTDR) algorithm in order to tolerate permanent faults without deadlock and livelock. In addition, the work presents a hierarchical FTDR (FTDR-H) algorithm, in order to reduce the area overhead of the FTDR router. The experimental results show that FTDR and FTDR-H are implemented with reliable bufferless routers, which can protect against any fault distribution pattern considering that the NoC is not split into two or more disconnected sub-networks.

Garbade et al. [7] investigate the message overhead for fault detection monitoring with decentralized fault detection units considering unified 2D mesh NoCs. Here, timed fault detection messages called heartbeats to continuously monitor the health state of several cores on the chip, each one executing instances of this software-based unit is used. Moreover, the approach investigates routing algorithms for different message types and demonstrates the reduction of the impact of fault detection messages on application messages.

Neishaburi and Zilic [8] present a fault-tolerant NoC router, which proposes no-deadlock interconnection of subnets in hierarchical architectures. The work presents an enhanced flow control mechanism which purpose is to mitigate the effects of both transient and permanent errors. Experimental results show improvements on the operation of NoC applications as well as the decrease in the average latency and energy consumption.

V. EXPERIMENTAL RESULTS

This section analyses and compares the area occupation of a Phoenix router compared to different other NoC routers. In addition, the efficiency of the distributed fault notification mechanism is demonstrated.

A. *Area of a Central Router*

The Phoenix NoC was implemented based on a Hermes NoC and by adding the fault tolerance mechanisms to the hardware. In order to evaluate the impact of these mechanisms, the routers of both communication architectures were synthesized to a 65 nm gate length of STMicroelectronics CMOS process using a Cadence RTL-Compiler. Since the size of Phoenix NoC is depending on the *Routing Table*, three different table sizes are explored during the comparison. It is important to remark that, as Phoenix is a region-based routing NoC [9], the *Routing Table* grows linearly proportional to the number of regions.

TABLE III – ROUTER'S AREA COMPARISON.

Router		Cells (um^2)	Area increase	Net (um^2)	Area increase
Hermes		78,6	*(reference)*	69,5	*(reference)*
Phoenix	4 regions	104,9	33,3 %	102,5	47,5 %
	8 regions	115,7	46,7 %	102,460	50,4 %
	16 regions	124,6	58,4 %	102,460	53,7 %

The results described on Table III demonstrate that the presented fault tolerance approach does not significantly increase the router area. Additionally, both cell and net area grow linearly with *Routing Table* size, which emphasizes that most optimized routing paths may be explored without significant costs.

B. *Stabilization Time of Fault Notification Mechanism*

This section demonstrates the stabilization time of the *Fault Notification Mechanism*. Figure 8 shows the setup of experimental results that were conducted running 48 simulations, representing scenarios of four NoC sizes (4×4, 5×5, 8×8, 10×10) versus 12 faulty link configurations, which totalizes 48 simulations. The faulty link configurations are: (i) four quantities of simultaneous faulty links (1, 4, 8, 16), and (ii) the method to place faulty links onto the NoCs (*random*, *region* and *long path*). While in the *random* method, the faulty link's positions are arbitrarily chosen, in the *region* method, scenarios where faults are concentrated in a given NoC region are explored. In the *long path* method the faulty links are placed in such a way that their placement increases the quantity of hops of an arbitrarily but distant communication.

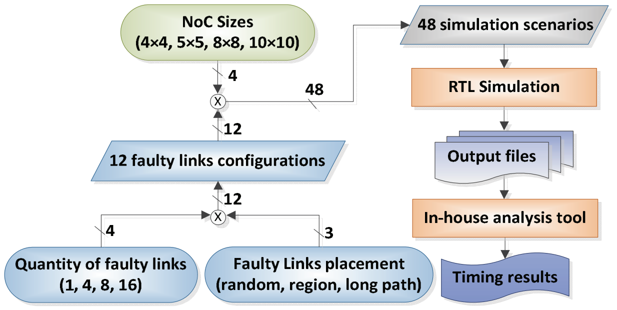

Figure 8 – Setup of experimental results.

All scenarios were simulated with a clock cycle accurate simulator, generating a set of results stored into output simulation files. Following, an in-house tool that correlates some parameters (e.g. abstracting the average of all results of the same NoC size, and with the same quantity of faults, but different method of links placement) timing results are extracted. The regarding results are following presented.

Figure 9 illustrates the total stabilization time, in clock cycles, for a quantity of faults and NoCs of different sizes. The results point out that the number of faults typically has a small effect on the stabilization time. The exception is when the number of faults is so large that most of the NoC links are faulty. For instance, in a 4×4 NoC, which has 24 inter-router links, 16 faulty links represent 67% of the total quantity of links. In this situation the stabilization time decreases since the propagation process has to evaluate fewer links.

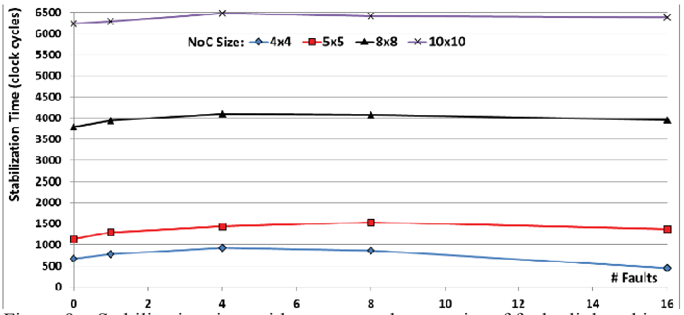

Figure 9 – Stabilization time with respect to the quantity of faulty links taking into account four NoC sizes.

Figure 10 highlights the previous mentioned effect since, in this chart, the number of NoC routers normalizes the stabilization time. For instance, if the total stabilization time is of 700 clock cycles on a 4×4 NoC, then 700/16 is the normalized value. This chart shows that the proposed method is more efficient with more faults and the stabilization time per router is about 60 clock cycles for NoCs of different sizes.

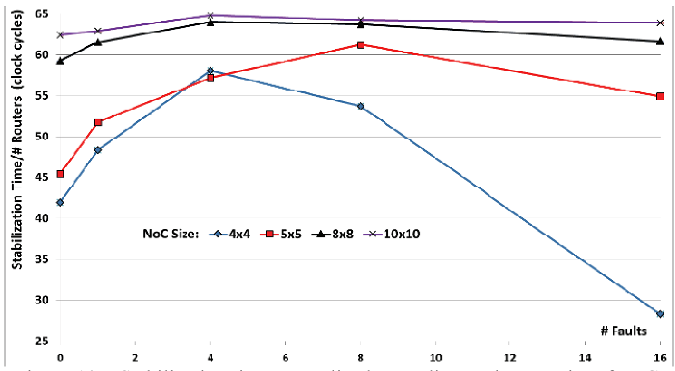

Figure 10 – Stabilization time normalized according to the quantity of NoC routers w.r.t. the quantity of faulty links.

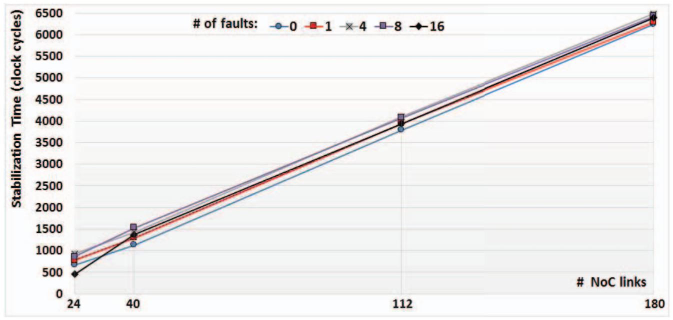

Figure 11 – Stabilization time versus the quantity of NoC links.

Figure 11 illustrates the scalability of the proposed approach. It shows that the stabilization time is linearly proportional to the amount of NoC links, and the variation of the faults quantity has small effect.

VI. Conclusion

Fault diagnosis represents a crucial tool for MPSoCs architectures based on NoCs, mainly to guarantee system recovery and minimized latency in case of faults. Moreover, more and more applications present several requirements at runtime besides large MPSoCs. Fault tolerant mechanisms can be also applied to discover alternative routes to overcome delays caused by faulty links. Phoenix, which is a fault tolerant 2D mesh NoC that enables properly communication in case of fault discovery during execution as well as in the case of manufacturing faults. The proposed mechanism is based on fault propagations from the NoC center to its borders in a distributed process. Each adjacent router relays the fault notification to other routers until reaching the maximum period stabilization time, which is proportional to the maximum NoC length. By doing so, the presented approach copes satisfactorily with runtime faults, since it even allows the MPSoC operation in the presence of several faults. In addition, experimental results show that Phoenix is scalable in terms of silicon area.

Acknowledgment

This work is funded by FAPERGS PqG 12/1777-4 and Docfix SPI n. 2843-25.51/12-3. Additional financial support also granted by CNPQ/FAPESP to the INCT-SEC (National Institute of Science and Technology Embedded Critical Systems Brazil), processes 573963/2008-8 and 08/57870-9.

References

[1] A. Jantsch and H. Tenhunen. Network on Chip. Kluwer Academic Publishers, 312p., Jan. 2003.

[2] É. Cota, A. Amory and M. Lubaszewski. **Reliability, Availability and Serviceability of Networks-on-Chip**. *Springer*, 209p., 2012.

[3] D. Fick, A. DeOrio, J. Hu, V. Bertacco, D. Blaauw and D. Sylvester. **Vicis: a reliable network for unreliable silicon**. *Design Automation Conference*, pp.812-817, 2009.

[4] D. Fick, A. DeOrio, G. Chen, V. Bertacco, D. Sylvester, and D. Blaauw. **A highly resilient routing algorithm for fault-tolerant NoCs**. *Design, Automation, and Test in Europe*, pp.21-26, 2009.

[5] Z. Zhang, A. Greiner and S. Taktak, **A reconfigurable routing algorithm for a fault-tolerant 2D-Mesh Network-on-Chip**. *Design Automation Conference*, pp.441-446, 2008

[6] C. Feng, Z. Lu, A. Jantsch, M. Zhang and Z. Xing. **Addressing Transient and Permanent Faults in NoC With Efficient Fault-Tolerant Deflection Router.** *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, v.21, n.6, p.1053-1066, Jun. 2013.

[7] Garbade, S. Weis, S. Schlingmann, B. Fechner and T. Ungerer. **Impact of Message Based Fault Detectors on Applications Messages in a Network on Chip**. *Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, p.470-477, 2013.

[8] M. H. Neishaburi and Z. Zilic. **NISHA: A fault-tolerant NoC router enabling deadlock-free Interconnection of Subnets in Hierarchical Architectures.** *Journal of Systems Architecture*, v.59, n.7, p.551-569, Aug. 2013.

[9] A. Mejia, M. Palesi, J. Flich, S. Kumar, P. Lopez, R. Holsmark and J. Duato, **Region-Based Routing: A Mechanism to Support Efficient Routing Algorithms in NoCs**. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, v.17, n.3, pp.356-369, Mar. 2009.