Topology-Agnostic Fault-Tolerant NoC Routing Method

Eduardo Wachter, Augusto Erichsen, Alexandre Amory, Fernando Moraes
FACIN – PUCRS – Av. Ipiranga 6681– Porto Alegre – RS – Brazil
{eduardo.wachter, augusto.erichsen}@acad.pucrs.br, {alexandre.amory, fernando.moraes}@pucrs.br

***Abstract*-Routing algorithms for NoCs were extensively studied in the last 12 years, and proposals for algorithms targeting some cost function, as latency reduction or congestion avoidance, abound in the literature. Fault-tolerant routing algorithms were also proposed, being the table-based approach the most adopted method. Considering SoCs with hundred of cores in a near future, features as scalability, reachability, and fault assumptions should be considered in the fault-tolerant routing methods. However, the current proposals some have some limitations: (1) increasing cost related to the NoC size, compromising scalability; (2) some healthy routers may not be reached even if there is a source-target path; (3) some algorithms restricts the number of faults and their location to operate correctly. The present work presents a method, inspired in VLSI routing algorithms, to search the path between source-target pairs where the network topology is abstracted. Results present the routing path for different topologies (mesh, torus, Spidergon and Hierarchical-Spidergon) in the presence of faulty routers. The silicon area overhead and total execution time of the path computation is small, demonstrating that the proposed method may be adopted in NoC designs.**

I. INTRODUCTION

Large SoCs, containing dozens of cores interconnected through a NoC, start to be used in embedded devices. Such devices can deliver the expected performance, allied with low power consumption. However, due to fabrication process problems or device aging, some of these cores or NoC routers may fail. To increase the yield and the lifetime of such devices, fault tolerant design techniques must be applied.

NoC-based MPSoC are a particular example of SoCs with natural hardware redundancy, with several identical processors interconnected to several identical routers. Therefore, if some processors or routers fail, tasks assigned to these processors may be migrated to other functional processors, and if a router fails, a new path in the NoC may be used.

Considering this scenario, the goal of this work is to present an original routing method for NoCs targeting fault tolerance. Routing algorithms were extensively studied since the NoCs' beginnings. However, *generic* methods targeting fault tolerance is a gap in the literature. The proposed routing method has the following features:

- *topology-agnostic*, the proposed routing method may be applied to regular and irregular NoC topologies;
- *complete reachability*, i.e., if there is a path between a given source-target pair, the method is able to find it;
- *fault-independent*, i.e., faults may exist in any router and/or links at any moment;
- *scalable area*, i.e. the router area overhead is independent of the NoC size.

This paper is organized as follows. Section II reviews the state-of-the-art of fault tolerant routing methods for NoCs. Section III presents the proposed method. Section IV evaluates the proposed method, and Section V concludes this paper.

II. RELATED WORKS

Common approaches for fault-tolerant NoC can be divided in two categories: (*i*) add hardware redundancy to the NoC to deal with faults. Recently, Cota et al. [1] surveyed this category which includes methods such as ECC and CRC codes, data retransmission, spare wires, TMR, backup path, and spare routers; (*ii*) add logic to enable the NoC using its natural path redundancy (presented in most NoC topologies) to find fault-free paths. The first category is typically NoC-dependent, adds more hardware than the second category, and it does not comply with all features presented in the previous section. The second category typically complies with most of the features, except for topology abstraction. For instance, several works were developed for topology-dependent routing algorithms [2]-[7], while few topology-agnostic approaches were proposed [8]. These characteristics of both fault tolerant categories motivated the research of fault tolerant routing, with focus on all the features presented before.

Rodrigo et al. [2] present the uLBDR. The first version of the algorithm, LBDR, stores the connectivity status of the neighbors' routers ($C_{east,}C_{west}$, C_{north} and C_{south}) and the turns ($R_{east\text{-}west}$, $R_{east\text{-}north}$, etc), leading to minimal paths. If this information is not sufficient to reach the destination, a deroute process (algorithm named $LBDR_{dr}$) is executed, trying to route the packet in a non-minimal path. However, the deroute process does not ensure reachability. Then, the Authors propose the replication of the packets in two output ports - uLBDR, ensuring complete reachability. The drawback is that packet replication potentially increases the network congestion and a virtual cut through (VCT) approach is applied to avoid deadlocks. VCT requires larger buffers to store the entire packet and has longer network latency compared to wormhole. Compared to LBDR, the uLBDR increases 44.7% the silicon area. The Authors present results for 2D irregular meshes, not exploring other topologies.

Sem-Jacobsen et al. [3] propose two modifications in the original FDOR algorithm [9]: (*i*) create a boundary around a failed router; (*ii*) reroute packets around the failed router. The boundary around the failed router is made reconfiguring its neighbor routers' as boundary routers. Two virtual channels are used to ensure deadlock freedom by prohibiting one turn. The results have shown an increase of 279% in terms of area for a mesh NoC. The Authors do not discuss the reachability feature and the number of maximum faults that the algorithm may support.

Schonwald et al. [8] propose an algorithm using routing tables at each router. The Force-Directed Wormhole Routing (FDWR) stores the number of hops to each destination for each port. This information enables to compute all possible paths to all destinations, not only the shortest one. This table is updated each time a packet is sent. Before send any packet, the router asks to all its neighbors the number of hops to the destination. Then, it chooses the neighbor with the minimum hop number and updates its table. This process is made at each router. Due to this process, the average time to send packets varies between 50 and 100 cycles, which is too long compared to non-fault tolerant NoCs (2-5 clock cycles). Results are presented for 2D mesh and torus topologies. Due to the use of tables, any topology may be supported. The drawback of the approach is scalability, since tables' sizes increases with the NoC size.

Feng et al. [4] present a fault-tolerant routing algorithm, named FTDR, based on the Q-routing approach [10], taking into account the hop count to route packets in the network. This table-based routing algorithm was modified to divide the network in regions, resulting in the FTDR-H algorithm. Compared to FTDR router, FTDR-H router reduces up to 27% the NoC area. When compared to a non-fault tolerant router, for a 4x4 NoC, the FTDR-H area overhead reaches 100%. Moreover, the number of tables increases with the NoC size. The Authors do not discuss the number of maximum faults that the algorithm supports, the supported topologies and reachability.

A similar approach to reduce the tables' size, by dividing the NoC into virtual regions, was taken by Flich et al. [5]. According to Rodrigo et al. [2], reachability may be achieved with a higher number of regions, but the performance decreases with the number of regions, since this parameter affects the router critical path.

DeOrio et al. [6] and Fick et al. [7] also propose fault-tolerant routing algorithms using tables. All routers contain a table to all router's destinations. The algorithm is divided in two steps. First, each router selects the direction to route a given destination based on its neighbors, updating the entry in its own table. Second, the router applies a set of rules to avoid deadlock. These rules disable turns without requiring virtual channels or duplicated physical links. On the other hand, this approach does not ensure full reachability. Presented results for mesh and torus topologies guarantee that at least 99,99% of paths are deadlock free with up to 10% of faulty links.

Table 1 compares the reviewed approaches, positioning our method to the evaluated features. The main drawback of table-based approaches is their scalability and the fact they do not ensure reachability. The closest approach is the uLBDR method, however the duplication of packets impacts in the traffic on the network, increasing with the NoC size. The fault-tolerant routing algorithms are very dependable to the NoC topology, as mesh and torus. Flich et al. [11] evaluate topology-agnostic fault tolerant routing algorithms, not specifically designed for NoCs, but the Authors claim that they may be used for NoCs. They present results regarding average routing distance, computing complexity, average packet latency, and throughput for mesh and torus topologies, with or without failures. However, results concerning how much time the algorithm takes to find the path are not presented. Results about full reachability and silicon area are not presented too.

III. Proposed Routing Method

This section describes the architectural model where the proposed method can be applied (Section A). Next, it describes the proposed method (Section B) and its hardware design (Section C).

A. Architectural Model

The proposed method is designed for NoC-based MPSoCs. Each Processing Element (PE), Figure 1(a), contains an embedded processor, memory, Network Interface (NI), the NoC router, and some User Defined Logic (UDL). The router can have any number of ports connecting its neighbor PEs via bidirectional NoC links, resulting in regular or irregular topologies. Neighbor PEs are expected to be very close to each other such that the inter-PE (number 2 in Figure 1(a)) links are much shorter than the intra-PE links (number 3 in Figure 1(a)).

PEs must be tested for hardware defects after manufacturing process and periodically during its lifetime. The actual test of PEs is out of the scope of this work, but there are several recent proposals that could be used such as BIST, functional test or a mix of different test approaches [1]. Despite of the test approach used, there must be a test wrapper, such as IEEE 1500 or other test wrapper variations [1], to isolate the PE under test from the rest of the design. Moreover, the proposed method focuses on *permanent faults* detected at runtime. Transient faults can be managed combining the proposed method with error detecting/correction codes and packet retransmission [1].

TABLE I. State-of-the-art comparison.

Proposal	References	Routing Type	Fault Model	Area overhead	Scalable	Topology agnostic?	Full Reachability	max # faults
uLBDR	**[2] 2011**	distributed	links/routers	46% (compared to the baseline router)	yes	irregular mesh	yes	any
iFDOR	**[3] 2011**	distributed	router	~279% compared to FDOR	yes	irregular mesh	N/A	N/A
FDWR	**[8] 2007**	table	links/routers	N/A	no	yes	N/A	N/A
FTDR-H	**[4] 2010**	region based tables	links/routers	~100% in 12x12 NoC	no	N/A	N/A	N/A
Flich et al.	**[5] 2007**	region based tables	links/routers	240 gates in 8x8 NoC	no	N/A	yes (practically unfeasible)	N/A
Vicis	**[6] 2012 [7] 2009**	table	links/routers	300 gates per router (4x4), 330 (12x12)	no	irregular mesh /torus	no	N/A
Proposed Work		*path search*	links/routers	42% in LUTS/ 58% in FFs (compared to the baseline router)	yes	yes	yes	any

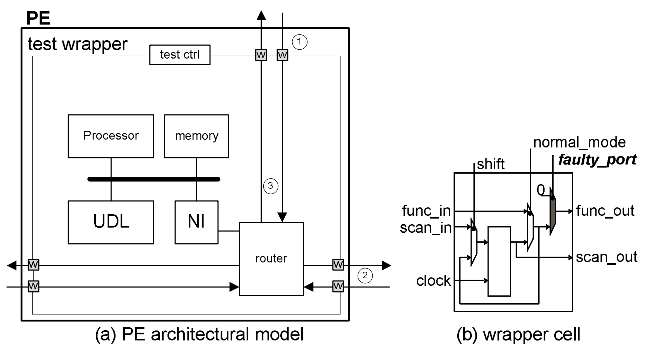

Figure 1 – Conceptual architecture model.

These test runs can identify defective PEs or NoC links, and isolate them using the wrapper cells of the defective logic. For, instance, by activating the *faulty_port* signal of the wrapper cell (Figure 1(b)), a defective port cannot generate Byzantine faults, disturbing the functional part of the design. In fact, activating the *faulty_port* of all wrapper cells disable an entire faulty PE.

Applications that need QoS typically use NoCs with virtual or replicated physical channels to provide the required performance guarantees [13]. In this paper, we use these *existing channels* to avoid deadlock cycles created by the presence of faulty links and routers. The number of virtual or replicated channels required to avoid deadlocks depends on the network topology. For example, two virtual or replicated channels are sufficient to avoid deadlocks in a mesh topology [12]. Other topologies such as 2D-Torus and hypercube need additional channels [12]. In summary, we assume that the network has a sufficient number of virtual or replicated physical channels for both performance and fault tolerance reasons.

The NoC must also support source routing such that, depending on the fault scenario, the PE need to apply different routing algorithms to circumvent faulty regions. The path to circumvent faulty regions may require a turn that could lead to deadlock. For this reason, the NoC needs to have replicated channels, each one allowing a different deadlock free routing algorithm. For example, in a mesh topology, a common solution to avoid deadlocks in fully-adaptive routing algorithms is to divide the NoC in two disjoint networks, each one implementing a partial adaptive routing algorithm, for example, west-first and east-first [13].

B. Behavioral Description of the Routing Method

At startup, the network is assumed fault free and packets are normally sent from the source PE to the target PE. Once the packet reaches its destination, the target PE must deliver an acknowledgment packet to the source PE. If the *ack* packet is not delivered in a given time, the source PE assumes the original path is faulty, and triggers the proposed path search method. This method, executed at runtime, comprises three steps, as illustrated in Figure 2: (*i*) seek new path; (*ii*) backtrack the new path; (*iii*) clear the seek structures and compute the new path. At the end of the described process, the source PE receives a valid path to the target PE. This path is stored in the PE memory and the following packet transfers (from this source to the target) use this path. Thus, the proposed method is executed only once for each missed *ack* packet, without requiring the whole seek process again for each packet transfer.

In the first step of the method, ***seek step***, *if* the source PE identifies that it is not able to transfer packets to a target PE, this PE generates a *seek request*, asking for a fault-free path to the target PE. The router stores this request into an internal table containing the fields **S/T/P/#**, meaning source router, target router, incoming port, and hop counter. Each router checks if it is the target router. If not, it increments the hop counter and broadcast the *seek request* to its neighbors (except for the port which originated the request), via a separate network optimized for broadcast. The next routers store the *seek request* in their own internal tables, checking if they are the target router and repeat the process until the target router is reached or a timeout mechanism at the source PE is fired. In this case, the source PE broadcasts a clear command to free the seek tables and it tries another seek a random time latter. The target router is declared unreachable after a given number of unsuccessful seek requests.

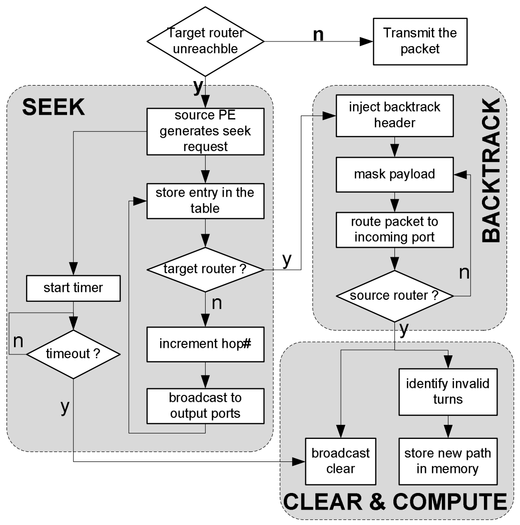

Figure 2 – Flowchart of the routing method.

The table size is *constant* for the NoC design, regardless the NoC size, ensuring scalability. The number of the table entries is only related to the maximum number of simultaneous seeks, and its size is not proportional to the NoC size, as in the table-based routing methods. If the number of simultaneous seeks is larger than the table size, the last seek waits the end of a seek to restart the process. If the designer knows that it is unlikely to detect multiple faults at the exact same time, then it is possible to reduce the table size to one. In other words, the designer can tradeoff between the time to determine a new fault-free path and the silicon area overhead.

Figure 3 illustrates a scenario with four faulty routers, one source, and one target router. Initially, the source PE requests a seek to its router, creating an entry in its local table with value S/T/L/0, meaning that the source request came from the local port (L) and it is hop 0 (Figure 3(a)). Since this is not the target router, it broadcasts the seek request to the north port. The second router (Figure 3(b)) stores the entry S/T/S/1 in its local table, meaning it received the request from the south port and it is the hop number 1. The broadcast continues until it reaches the target router (Figure 3(d)) via the east port with 8 hops.

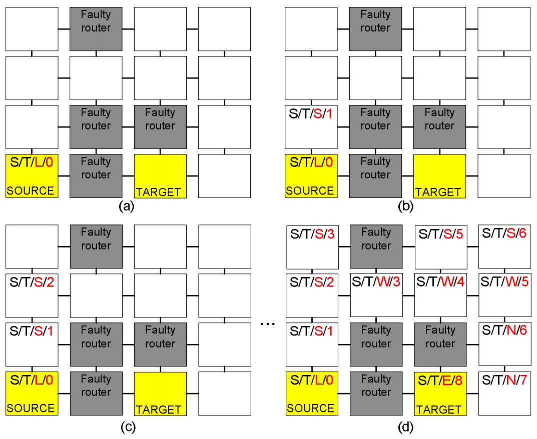

Figure 3 –Seek steps of the method in a 4x4 mesh, with four faulty routers.

The second step, ***backtrack step*** is executed only once, and starts when the target router is reached by the first received seek. All subsequent seeks are discarded. The target router injects a *backtrack packet* into the NoC, where its payload correspond to the ports selected by each router to reach the target router. The *backtrack packet* uses source routing, following the reverse path followed by the seek step. The backtrack ends when the source router is reached and the backtrack packet is sent to the source embedded processor to check the validity of the path.

Figure 4 shows the backtrack process, where the target router creates the backtrack packet. Its payload is initially filled with don't care data. At the first router (the target router), it inserts its seek incoming port (E) into the payload. Then, the packet follows the direction stored into the internal table (E), reaching the next router. The next router also inserts its incoming port (W) into the payload. Then, the packet follows the direction stored into the internal table (S), reaching the next router via its south port. In this way, the backtrack packet follows the reverse seek path, until it reaches the source router where the payload is now fully filled with the path to the target router. The path in this example is [N N E E E S S W E].

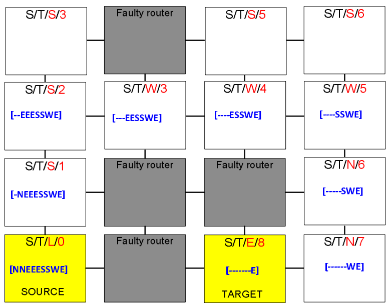

Figure 4– Backtracking process.

The backtrack packet may not be received in two cases: the target router is unreachable, or the number of maximal simultaneous seeks is exceeded. In the second case, the source router can initiate a new seek after a random time. If all the seek requests failed to receive backtrack, then the target router is declared unreachable.

The third step is named ***compute path and clear seek***. The path followed by the backtrack packet might create a deadlock. For this reason, the source embedded processor checks the path, for instance [N N E E E S S W E], for invalid turns. When there is an invalid turn, it forces the packet to change the physical channel (or virtual channel), breaking the dependency cycle. For instance, the path presented above has a turn from south to west. This path should be changed from channel 0 to 1 because it is a prohibited turn in the west-first routing algorithm, but allowed by east-first one. Thus, the corrected path to the example target router must be [N0 N0 W0 W0 W0 S0 S0 W1 E1]. Once this path is computed, it is stored in the PE main memory for future packet transmissions.

In parallel to the path computation, the source PE requests to clear the entries in the seek table labeled with the source router address. This clear command is broadcasted to the entire NoC, similarly to the seek process.

Note that the path computation depends on the network topology and its routing algorithms. For this reason, this part is implemented in software such that it can be easily modified. The hardware structure, presented next, is not dependent on the network topology and the routing algorithms.

C. *Hardware Structure*

Broadcasting messages in the NoC would create a congested scenario, disturbing the NoC performance. For this reason, a seek routing module was created, separating the NoC from the "seek network". Figure 5 illustrates how the seek routing module is connected to its neighbor routers (in this example they are N, S, W, E) creating two separated networks. The conventional NoC links are represented by dashed lines (number 2 in Figure 5) and the seek links are the continuous lines (number 1), consisting of 26 wires in this example. These seek links are used during the seek/clear steps to broadcast the seek/clear commands without disturbing the NoC traffic.

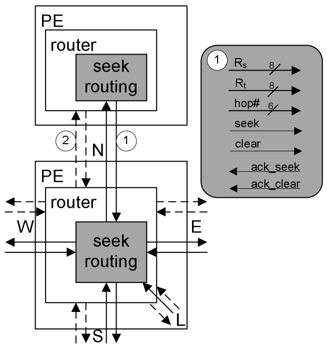

Figure 5 – Inter-router connection.

Note that the local port in Figure 5 has only the incoming seek link because, according to Figure 2, when a source PE detects an unreachable PE, it starts the seek process. The outgoing seek link is not required because the target PE does not receive data from the seek module.

Finally, the extra wires required by the seek module introduce a negligible wire area overhead when connecting the PEs since, as specified in Section III.A, the NoC wires among PEs are much shorter than the NoC wires inside the PE. Moreover, there are several metal layers to route these short global wires since there is no logic among PEs, only wires. On the other hand, internally, the PE typically has some hard IP blocks, such as memory and processor, limiting the number of metal layer for internal routing.

Figure 6 details a generic router with 2 physical channels in the local input port. It also presents the logic required by the proposed method (gray area) and its intra-router signals. The inter-router links, presented in Figure 5, are not presented here for sake of simplicity. The routing and arbitration module is slightly modified to interface with the seek routing module. At the beginning of the backtrack step, the routing and arbitration module identifies a backtrack packet (1) and asks the seek module to search its internal table (2) for the corresponding output port for the current seek request. The seek module answers with the appropriate output port (3). For the backtrack step, the payload receives the path information stored in the seek table, through the added mux (4 and 5).

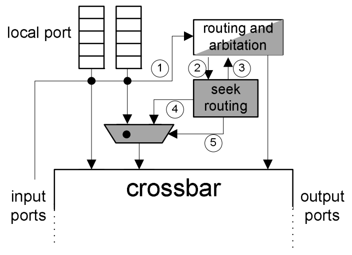

Figure 6 – Intra-router connection.

IV. Experimental Results

According to Salminen et al. [14], from 66 NoC designs reviewed, the most common NoC topologies are mesh and torus (56%), bus and crossbar (14%), custom (12%), fat-tree (11%), and ring-based (7%). Since bus, crossbar, and fat-tree typically do not provide alternative paths in the presence of faults, these topologies were not evaluated.

The Spidergon STNoC topology is a regular [15], point-to-point topology similar to a simple bidirectional ring, except that each node has, in addition to links to its clockwise and counter-clockwise neighbor nodes, a direct bidirectional link to its diagonally opposite neighbor (Figure 7).

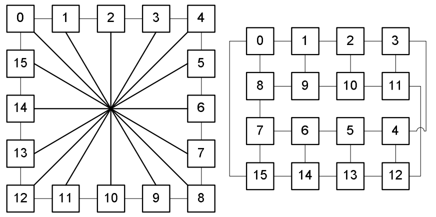

Figure 7 – Spidergon STNoC, topology and physical view.

The Spidergon STNoC uses the Across-First routing algorithm. This algorithm moves packets along the ring, in the proper direction, to reach nodes which are closer to the source node, and use the across link only once at the beginning for destinations that are far away [15]. For example, a given packet from routers 0 to 5, would take the path 0→8→7→6→5. The Across-First routing requires two virtual channels to be deadlock free. The Hierarchical-Spidergon topology uses two Spidergon networks layers, where each router communicates with its network and, additionally, with the lower/upper layer.

This section evaluates the proposed routing method with four topologies: 2D-mesh, 2D-Torus, Spidergon and Hierarchical-Spidergon. All NoCs are configured with 100 routers, with four routers pairs starting the seek process. Figure 8 shows a configuration for the 2D-mesh and 2D-torus topologies.

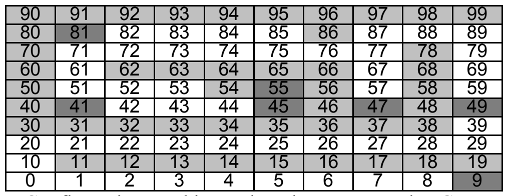

Figure 8 – Configuration used in mesh and torus networks. Gray routers are faulty routers and the dark gray routers are communicating router pairs. Four communicating pairs are illustrated: 81→9, 45→47, 41→49, 9→55.

For instance, let us assume the communication from routers 81 to 9 using the torus topology. As illustrated in Figure 9, the proposed method was able to find shortest paths using the wraparound links 29 to 20 and 0 to 9, resulting in the path *SEEEEEENEESSSSSSESSW* (steps 1 and 2). The method was also able to find minimal paths in the Spidergon NoCs, assuming the same four communicating pairs used for mesh and torus. The Figure illustrating the fault scenario for the Spidergon NoCs was omitted due to space limitations.

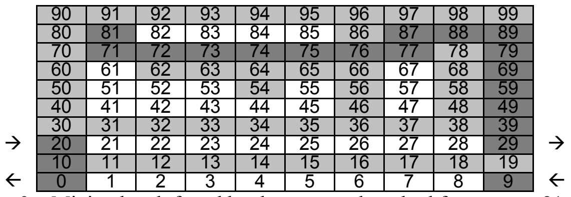

Figure 9 – Minimal path found by the proposed method from router 81 to 09 (dark gray), in a torus topology (gray routers are faulty node).

A. *Time to Find Alternative Paths*

Table 2 shows the time to find paths in the presence of faults for each topology, in terms of clock cycles, for each step of the method. The values presented for *step 3* include only the clean process, not the path computation. To account for the path computation time, we executed the path computation algorithm assuming a mesh topology and a MIPS processor running at the same frequency of the NoC. The result is that the processor takes, on average, 200 clock cycles per hop to evaluate the turns of the path. The complexity of this algorithm is O(n), where n is the number of hops in the path.

Next, we evaluate the average results from data presented in Table 2. The average number of clock cycles per hop (AH_{cc}), assuming all topologies, is 16 and 15.5 for the seek step and the backtrack step, respectively. It is not required to account for the time of step 3 (clean) since it happens in parallel with the path computation. Therefore, the AH_{cc} for step 1 and 2 is 31.3 (standard deviation of 2.6). Using these average values, we can estimate that, for instance, a path of 10 hops would take 2,313 clock cycles, being 313 clock cycles for steps 1 and 2, and 2,000 clock cycles to compute the path. To put this result into perspective, 2,313 clock cycles is a value inferior to a typical time slice in a multi-task operating system. Therefore, the impact to compute a new path represents a very small impact in the overall performance of an MPSoC system. Moreover, recall that, due to the locality of tasks in a MPSoC, two communicating tasks are expected to be close to each other. Thus, the number of hops is typically smaller than 10 hops.

TABLE 2 – TIME TO EXECUTE THE PROPOSED ROUTING METHOD, IN CLOCK CYCLES, FOR DIFFERENT TOPOLOGIES.

Topology	Rs→Rt	Number of hops	Step 1 *seek*	Step 2 *backtrack*	Step 3 *clean*
Mesh	09→55	43	688	548	635
	41→49	16	256	221	245
	45→47	16	256	221	620
	81→09	36	576	463	530
Torus	09→55	27	432	354	395
	41→49	16	256	221	245
	45→47	16	256	221	440
	81→09	20	320	269	350
Spidergon	09→55	5	80	89	500
	41→49	10	160	149	155
	45→47	4	64	77	65
	81→09	29	464	378	440
Hierarchical-Spidergon	09→55	5	80	89	246
	41→49	10	160	154	245
	45→47	4	64	81	230
	81→09	5	80	93	245

Once the path is determined by the proposed method, packets are transmitted as standard wormhole packets, with the same router latency compared to the base router [13], i.e. the proposed method does not affect the network latency.

The reviewed works do not present similar analysis for path computation, so we can only perform a rough comparison. For instance, the Vicis NoC [6] takes around 1,000 cycles to reconfigure the network, without accounting the execution of the routing algorithm. In [8] the average time between the sending of two consecutive packets is between 50 and 100 cycles, which increase latency. In our method this limitation does not exist, since the path search is executed once.

B. *Area Overhead*

Table 3 presents the area overhead in terms of look up tables (LUTs) and flip-flops (FFs) compared to the baseline router (with a seek table with 4 entries). Regarding the LUTs occupation, the overhead is 42%, and for FFs 58%. The router was also synthesized using Cadence Encounter RTL Compiler, targeting a 65 nm technology. The router with the seek module required 6,204 cells, occupying 43,049 μm².

TABLE 3 – AREA OVERHEAD FOR XILINX XC5VLX330TFF1738-2 DEVICE.

	Area Occupation		
sub module	LUTs	FFs	module
Switch Control	351	97	baseline router
TOTAL	1610	433	
Seek module	370	184	baseline router modified + seek module
Switch Control	339	157	
TOTAL	2293	682	
Overhead	42%	58%	

A silicon area comparison with previous works is not straightforward because the target router architecture might be considerably different and the technology library can also be different. The best match is uLBDR [2] router, which is similar to the evaluated router and uses the same technology. The uLBDR [2] router consumed approximately 62,050 μm². Considering that the router represents less than 10% of the PE area, an increase of 50% in the router area represents less than 5% in the total area overhead of one PE, which is an acceptable cost considering full reachability even under severe faults scenarios, scalability in terms of NoC size, no impact on the communication performance after the path calculation, and topology independency.

V. CONCLUSIONS AND FUTURE WORKS

The paper presented an original method for fault-tolerant NoC routing, compliant with the following features: topology-agnostic, complete reachability, fault-independent, scalability. Moreover, the total path search and computation time is small compared to the literature, with an acceptable area cost. Due to this characteristics the proposed routing method may easily integrated into another NoC design that require fault tolerance.

As future works, we can enumerate: (*i*) add the wrapper logic to isolate faulty PEs; (*ii*) treat packets that were blocked due to faults, avoid NoC stalling; (*iii*) evaluate the performance degradation of real applications in the presence of faults.

ACKNOWLEDGMENTS

Fernando Moraes is supported by CNPq, FAPERGS, and CAPES, projects 301599/2009-2, 10/0814-9, 708/11, respectively. The authors acknowledge the support granted by CNPQ and 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] Cota, E.; Amory, A; Lubaszewski, M. “Reliability, Availability and Serviceability of Networks-on-Chip”. Springer, 2012, 209p.

[2] 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), 2011, pp. 534-547.

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

[4] 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, 2010, pp. 11-16.

[5] 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 2007, pp. 183-194.

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

[7] Fick, D. et. al. “A Highly Resilient Routing Algorithm for Fault-Tolerant NoCs”. In: DATE 2009, pp. 21-26.

[8] Schonwald, T.; et. al. “Fully Adaptive Fault-Tolerant Routing Algorithm for Network-on-Chip Architectures”. In: Euromicro, 2007, pp. 527-534.

[9] Skeie, T.; et. al. “Flexible DOR routing for virtualization of multicore chips”. In: SOC, 2009. pp. 73-76.

[10] Boyan, J.; Littman, M. “Packet Routing in Dynamically Changing Networks: a Reinforcement Learning Approach”. In: Advances in Neural Information Processing Systems, 1993, pp. 671-678.

[11] 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.

[12] Linder, D.H. and Harden, J.C. “An Adaptive and Fault Tolerant Wormhole Routing Strategy for k-ary n-cubes”. IEEE Trans. Comput. v. 40(1), 1991, pp. 2-12.

[13] Carara, E.; Moraes, F. “Flow Oriented Routing for NOCS”. In: SOCC 2010, pp. 367-370.

[14] Salminen, E,; et. al. “Survey of Network-on-Chip Proposals”. White Paper, OCP-IP, mar. 2008.

[15] Coppola, M.; et al. “Design of Cost-Efficient Interconnect Processing Units: Spidergon STNoC”. CRC Press, 2008, 288 p.