MAZENOC: NOVEL APPROACH FOR FAULT-TOLERANT NOC ROUTING

Eduardo Weber Wächter, Fernando Gehm Moraes
FACIN – PUCRS – Av. Ipiranga 6681– Porto Alegre – RS – Brazil
eduardo.wachter@acad.pucrs.br, fernando.moraes@pucrs.br

ABSTRACT

This paper presents an original approach to define a path between two routers in a NoC with faulty routers. Current state-of-the art adopts non-scalable solutions, using tables to store paths, or distributed approaches that keep the status of neighbor routers. The proposed approach searched its foundations in the firsts routing algorithms for VLSI circuits, using a three-step process: seek new path, backtrack the path, store the new path. Results demonstrate the effectiveness of the approach, with the algorithm being able to find the path between routers in complex scenarios, with a small area overhead over a baseline router.

I. INTRODUCTION

Innovations on integrated circuit fabrication continuously reduce the size of transistors. From one side, it becomes possible to implement complex MPSoCs in a single die. From the other side, the reliability of these systems is compromised due to increased fault probability in deep submicron technologies [1].

In the MPSoC design arena, NoCs are becoming widely used to interconnect processing elements (PEs) due to their scalability and performance, compared to buses. In addition, also compared to buses, NoCs provide more opportunities to implement fault tolerant techniques for intra-chip communication. For instance, a NoC has multiple paths for any pair of modules, which can be exploited to improve the fault tolerance of the communication by using fault-tolerant routing algorithms. A fault-tolerant routing algorithm, besides being deadlock and livelock free, should present the following features:

i. *generic*, being possible to adapt it to different NoCs, including different NoC topologies;
ii. *complete reachability*, i.e., if a path between a given source-target pair exists, the routing algorithm must find it;
iii. *independence of the fault location and the moment it occurs*, i.e., faults may reach routers and/or links at any moment. Some routing algorithms restrict the number of simultaneous faults, or the location of the faults, or if the fault occurs in links or in routers;
iv. *scalable*, its cost must be independent of the NoC size. Table-based approaches (discussed in the state-of-the-art) have their area cost linked to the NoC size;
v. *local path computation*, i.e., algorithms with a PE responsible to store and compute all paths must be avoided due to the large size of actual SoCs (feature also related to scalability). Each router or PE must be able to compute the new path if a fault happens, without interacting with a central manager;
vi. *acceptable cost*, i.e., the implementation of the proposal routing algorithm in a reference NoC should not severely impact the silicon area neither its performance (frequency).

The third feature is related to the fault model. Transient faults, as crosstalk in links, may be treated with CRC or other coding schemes [2]. The proposed work assumes permanent faults where routers or links may fail due to manufacturing problems or aging reasons.

The goal of this paper is to present an original approach for fault-tolerant NoC routing, compliant with the previous features. The proposed approach searched its foundations in the firsts routing algorithms for VLSI circuits, as the Lee one [3].

This paper is organized as follows. Section II reviews the state-of-the art in fault-tolerant NoC routing approaches. Section III presents the proposed routing approach. Section IV evaluates the proposed algorithm and Section V concludes the paper.

II. RELATED WORK

Common approaches for fault-tolerant routing include: distributed routing [4][5], table-based routing [6][7][8][9][10] or extra hardware added in

the NoC to help a given routing algorithm to route packets when faults happens [11].

Rodrigo et. al. [4] 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 with this information the algorithm cannot reach the destination, a deroute process (algorithm named $LBDR_{dr}$) is executed, trying to route the packet in a non-minimal path, to avoid deadlocks. The problem is that even with the deroute process, the complete reachability is not achieved. Then, the authors propose the replication of the message in two output ports (third version of the algorithm, named uLBDR), assuring complete reachability. The drawback is that one message should be killed, and a virtual cut through methodology should be applied to avoid deadlocks. The Authors present latency, frequency and power estimation for a 65 nm technology. Compared to LBDR (algorithm that does not guarantee complete reachability), the uLBDR shown a 44.7% increase in terms of area.

Sem-Jacobsen et al. [5] propose two modifications in the original FDOR algorithm [12]: (*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. For example, if a router R1 has a faulty router in the north, R1 is reconfigured as north edge router. To assure deadlock freedom, two virtual channels are used. Each VC prohibits one turn. The VC 0 prohibits the turn north-west and VC 1 north-east. The authors do not discuss the reachability feature.

Schonwald et. al. [6] propose an algorithm using routing tables at each router. The algorithm is named Force-Directed Wormhole Routing (FDWR). 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 between sending packets is between 50 and 100 cycles, which is too high compared to non-fault tolerant NoCs (2-5 clock cycles). There are no results regarding area occupation.

Feng et al. [7] present a fault-tolerant routing algorithm, named FTDR, based on the Q-routing approach [13], 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. For example, when a packet reaches a router, the router first checks if the destination is in the same region as the current router. In an affirmative case, the routing decision is based on the local routing table. If not, the router decision is based on the region routing table. Compared to FTDR router, FTDR-H router reduces up to 27% the area. When compared to a non-fault tolerant router, for a 4x4 NoC, the FTDR-H area overhead reaches 100%. It is important to mention that, even if the routing-algorithm decreases the area occupation, compared to FTDR, the number of tables increases with the NoC size.

A similar approach to reduce the tables' size was taken by Flich et. al. [8]. According to Rodrigo et al. [4], the drawback of such mechanism is that, even with a great number of regions, full reachability is not achieved, and the performance is penalized due to a longest critical path.

DeOrio et. al. [9] and Fick et. al. [10] also propose fault-tolerant routing algorithms based on 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 or links without requiring virtual channels or duplicated physical links. In 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.

Tsai et. al. in [11] propose a fault-tolerant NoC scheme using bidirectional links, named Bidirectional Fault-Tolerant NoC (BFT-NoC). This mechanism is devised to mitigate potential performance degradation due to faulty links through dynamic sharing of other non-faulty links. For example, an extra hardware is inserted to the TX and RX links for each direction of the network. If one of the links fails, the non-faulty link can share the bus,

Table 1: State-of-the-art comparison (N/A not available).

Proposal	References	Type	Fault Model	Area overhead	Scalable	Topology agnostic?	Reachability
uLBDR	**[4] 2011**	distributed	links/routers	46%	yes	yes	yes
iFDOR	**[5] 2011**	distributed	router	~279% compared to FDOR	yes	yes	N/A
FDWR	**[6] 2007**	table	links/routers	N/A	no	yes	N/A
FTDR-H	**[7] 2010**	table	links/routers	~100% in 12x12 NoC	no	yes	N/A
Flich et. al.	**[8] 2007**	region based tables	links/routers	240 gates in 8x8 NoC	no	N/A	not
Vicis	**[9] 2012 [10] 2009**	table	links/routers	300 gates per router (4x4) 330 (12x12)	no	Only mesh and torus.	no
BFT-NoC	**[11] 2011**	extra hardware for links	links/routers	4%	yes	yes	no
Proposed Work		*path search*	links/routers	42% in LUTS/ 58% in FFs	yes	yes	yes

enabling TX and RX in the same bus. The problem is that this approach does not solve the problem if the TX and the RX links are failed. The Author proposed a integration with the routing algorithm proposed in [10], showing a 99.94% of reachability for a 8x8 NoC with up to 7 faulty channels. The overhead in terms of gates is around 4% compared to a NoC without bidirectional channel. These results do not include the routing algorithm, only the bidirectional channel.

Table 1 compares the approaches evaluated in this paper. The main drawback of table-base approaches is their scalability and the fact they not ensure reachability. The closest approach in terms of desired coverage of features is uLBDR approach. The uLBDR results also shown that the duplication of packets impacts in the traffic on the network, and this traffic is not scalable, i.e. the traffic increases when the NoC size increases. Our approach is compliant with the features presented in the Introduction, and its type is named path search, detailed in the next Section.

III. OVERVIEW OF THE ROUTING ALGORITHM

The proposed routing algorithm comprises 3 steps: (*i*) seek new path; (*ii*) backtrack the new path; (*iii*) clear the seek structures and store the new path.

A. Seek New Path

The first step of the algorithm requires a hardware module, added to each router, named *seek module*. The *seek module* communicates with other routers through the interface illustrated in Figure 1. The *seek module* adds, in the present implementation, 26 wires at each link: R_S, R_T, *hop number*, *seek request*, *clear request*, *seek and clear acknowledgements*.

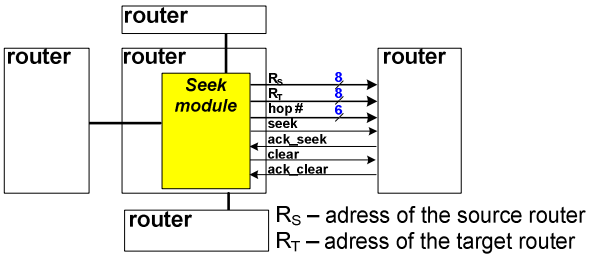

Figure 1 – External interface of the *seek module*.

The *seek module* contains a finite state machine and a 4-entry table. Each entry of the table contains 5 fields: *slot availability*, R_S, R_T, *input port*, *hop counter*. This represents a constant cost of 100 bits, regardless the NoC size. The table has 4 entries to support up to 4 simultaneous seek requests (this is a parameterizable value).

If a 5th seek request arrives at the *seek module* the module will delay the answer until the availability of a free entry. We consider this situation unlikely to occur, because communicating tasks are supposed to be mapped closer to each other. Due to the locality of the communication, the probability of more than 4 simultaneous faults seeking a path in the same NoC region is very low.

Figure 2 details the first step of the algorithm. The source router starts the algorithm to the target router. It is out of the scope of the present work to detect faults. It is assumed that a set of routers or links is faulty. In the example presented in Figure 2, we assume only faulty routers for sake of simplicity. The source router starts the new path computation by receiving a request from the IP connected to it.

When the algorithm starts, Figure 2(a), one entry of the table receives the values **S/T/L/0**, meaning R_S, R_T, local port (input port), zero (hop counter). These values are broadcasted to all neighbor seek modules, Figure 2(b). The seek module con-

nected to the north port of R_S stores in its table **S/T/S/1**, writing S (south) in the input port field, meaning that the seek request came from this direction; and 1 in the hop counter field (this value is obtained by adding 1 to the value in the *hop counter* signal). This broadcast process continues until reaching R_T, as illustrated in Figure 2(c) and Figure 2(d).

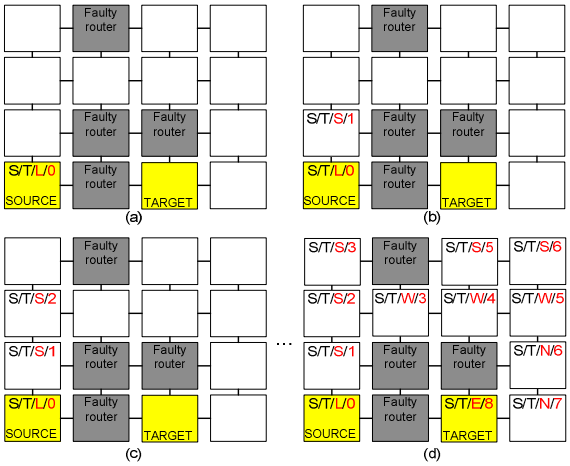

Figure 2 – Seek steps of the algorithm in a 4x4 mesh, with four faulty routers.

The broadcast process continues, even if R_T is reached. As shown in Figure 2(d), R_T was reached by its east port, with a hop counter equal to 8. Note that it possible to reach a given R_T by the more than one port, since two or more routers can activate the *seek* request to R_T.

B. Backtrack Step

Once R_T is reached, the backtrack process starts. In this second step, the *seek* module of R_T injects into the NoC a packet that will contain the new path. Note that in the first step the seek process does not interfere with the NoC traffic. In this second step, the NoC is used to transmit to R_S the new path through a new packet type: *backtrack packet*. The backtrack packet header contains an identifier signalizing it as a backtrack packet and R_S. The payload size if a function of the hop counter (HC).

One possible implementation of this second step include two adaptations in the original NoC router: (*i*) multiplexing the local port with the *seek module*, making this module to act as a second IP connected to the router; (*ii*) modify the router control to accept packets with the output port coded in the packet.

When the backtrack packet enters into the router, the output port is selected by the seek module. This information is obtained by reading the *input port* field of the seek table, in the position indexed by R_S.

The initial payload of the backtrack packet contains the input port from where the seek request came. This information is inserted in the payload position indexed by HC+1. In the example of Figure 2, R_T was reached by the East port, with a number of eight hops. In Figure 3, the initial payload corresponds to eight don´t cares and the inserted direction ([--------E]).

The payload in the *second* and *succeeding* routers in the path receive the port index from which the backtrack packet came. In the example, the payload transmitted by the router connect on the south of R_T is [-------WE].

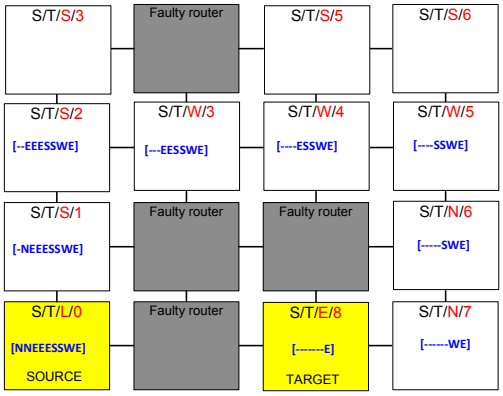

Figure 3 – Backtracking process.

The second step finishes when the packet arrives at R_S, since the backtrack packet has R_S in its header. This is the process used to identify that the source router was reached. The new path, in the example, is [N N E E E S S W E]. Note that the last part of the path seems invalid, because the packet arriving in the East port should be transmitted to the East port. Such invalid directions are used to detect the target router in the new path.

C. Clear and compute path

Once R_S is reached with the packet containing the path, the third step of the method starts. It includes two actions: broadcast the clear signal to the seek modules and store the new path. The cleaning process is very similar to the seek step, resetting the *availability* bit of the seek table entry containing R_S.

The new path is transmitted to the IP con-

nected to the router. In the present approach the NoC adopts source routing and duplicated physical channels [14] (smaller area compared to virtual channels). The new path may present turns leading to deadlocks. One solution to avoid deadlocks in 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. Therefore, one of the physical channels transmits packets using the west-first routing algorithm, and the other one the east-first. Such method ensures that no deadlocks will occur.

The IP connected to the source router evaluates the path, adding for each hop one of the channels, according to one of the routing algorithms. The chosen channel is a function of the turns taken in the path. The packet changes the channel when an invalid turn of the current routing algorithm should be taken. In our case, the west-first routing algorithm is used in channel 0 and the east-first in the channel 1. For example, if a packet starts the path in the channel 0 and there is a turn from south to west, the path should be changed from the channel 0 to 1 because it is a turn prohibited by west-first, but allowed by east-first. In the example, the last turn must change the channel. The path with the right channel, generated by the IP is: [N0 N0 W0 W0 W0 S0 S0 W1 E1]. From this moment, the IP connected to R_S can send messages to R_T with source routing, using the computed path.

Concluding, if there is a path from a given source to a given target, the proposed approach finds a path in a NoC with faulty routers or links. To transmit packets using this path, deadlock free routing algorithms should be used, as previously mentioned. Other adaptive routing algorithms, as well as other NoC features could be adopted. It is important to mention that the proposed method can be applied to other topologies, since the *seek* broadcast does not require a regular structure.

IV.RESULTS

Our approach adopts as reference the router proposed in [15], which has the following features: (*i*) Hamiltonian routing algorithm, (*ii*) 16-bit wide flit, (*iii*) duplicated physical channel, with high and low priorities; (*iv*) circuit switching over packet switching. The main reason to adopt this NoC is the replication of physical channels, feature required to to avoid deadlock when using adaptive source routing. Three main modifications were performed in the reference router: (*i*) addition of the *seek module*; (*ii*) modification of the router control (named *switch control*), allowing to support source routing and reading the table of the seek module; (*iii*) addition of logic to enable the seek module to inject packets in the NoC.

A. Experimental Setup

This section evaluates the proposed approach with four routers simultaneously asking a new path. As experimental scenario, we consider a 10x10 2-D Mesh, with a set of faulty routers, as presented in Figure 4. This scenario contains one bottleneck for the seek request at router 42, and at least 2 channel changes after the seek completion.

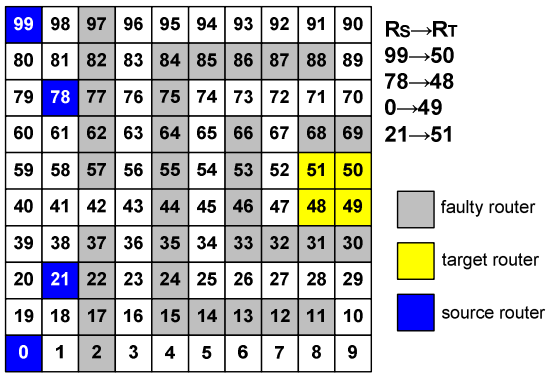

Figure 4 – Evaluation scenario of a NoC 10x10.

This algorithm returned the expected paths for each pair $R_S \rightarrow R_T$. For example, the path returned for 0→49 was:

[NNNNEEENNNNNEEEEEESSWWSSSEEW]

To avoid deadlock, assuming west-first and east-first for channel 0 and 1, respectively, the IP connected to the router must change the channel in the 20st direction from 0 to 1 and in the 25st from 1 to 0. The following string presents the path with the channel:

[N0 N0 N0 N0 E0 E0 E0 N0 N0 N0 N0 N0 E0 E0 E0 E0 E0 E0 S0 S0 W1 W1 S1 S1 S1 E0 E0 W0]

B. Evaluation

Our approach was evaluated in terms of clock cycles for each step of the algorithm. Table 2 shows the number of clock cycles for each step for the four pairs of RS→RT. The whole process takes in average 38 clock cycles per hop to find the path (last column of the Table). We consider this an

excellent result, since: (*i*) as the number of hops between tasks is frequently small due to the mapping process, few tasks will be mapped far from each other (task migration may mitigate this problem); (*ii*) few hundreds of clock cycles represents a smaller amount of time compared to a dedicated algorithm executed in a given PEs. For example, the Vicis NoC [9] takes around 1000 cycles only to reconfigure the network when a fault happens.

Table 2 – Results in terms of the number of clock cycles for each step of the approach.

Rt→Rs	hops	# cycles step 1	# cycles step 2	# cycles step 3	all process	Cycles per hop
99→50	28	390	350	332	1072	38,3
78→48	25	353	308	246	907	36,2
00→49	28	390	350	293	1033	36,9
21→51	24	320	292	285	897	37,3

Table 3 presents the area overhead in terms of look up tables (LUTs) and flip-flops (FFs) compared to the baseline router. Regarding the LUTs occupation, the overhead is 42%. The overhead in terms of FFs is higher, 58%. This is due to the fact that the VHDL code was not optimized for FPGAs (LUTRAMs). As a consequence, the operating frequency was reduced 34%, from 193 MHz to 128 MHz.

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%	

The router was synthesized using Cadence Encounter RTL Compiler, targeting a 65 nm technology library from STMicroelectronics. The router with the seek module required 6,204 cells, occupying 43,049 um². For comparative purposes, the uLBDR [4] router, with the same technology, consumed approximately 62,050 um².

V.CONCLUSION

Our main contribution is to present an original approach for fault-tolerant routing in NoCs. The approach is generic (can be ported to other topologies); presents complete reachability; is independent of where and when the fault occurs; and scalability is achieved by a constant table size, differently from the traditional table-based approaches.

As future work, the implementation should be optimized to reduce the number of FFs and increase the operating frequency. Also, the proposed scheme will be integrated in an MPSoC platform, enabling the evaluation of the approach at the application layer. The proposed method can also consider the NoC congestion if the seek process is weighted with the NoC traffic status.

ACKNOLEDGEMENTS

Fernando Moraes is supported by CNPq, FAPERGS, and CAPES, projects 301599/2009-2, 10/0814-9, 708/11, respectively.

REFERENCES

[1] Zhang, M.; Yu Q. and Ampadu, P. “Fine-Grained Splitting Methods to Address Permanent Errors in Network-on-Chip Links”. In: ISCAS 2012, pp. 2717-2720.

[2] Zimmer H.; Jantsch A. “A Fault Model Notation and Error-Control Scheme for Switch-to-Switch Buses in a Network-on-Chip”. In: CODES+ISSS 2003. pp. 188-193.

[3] Lee, C. Y. “An Algorithm for Path Connections and Its Applications”. IRE Transactions on Electronic Computers, Sep. 1961. pp. 346-365.

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

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

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

[7] Feng, C. et. al. “A reconfigurable fault-tolerant deflection routing algorithm based on reinforcement learning for network-on-chip”. In: NoCArc 2010, pp. 11-16.

[8] Flich, J. et. al. “Region-Based Routing: An Efficient Routing Mechanism to Tackle Unreliable Hardware in Network on Chips”. In: NOCS 2007. pp. 183-194.

[9] DeOrio, A. et. al. “A Reliable Routing Architecture and Algorithm for NoCs”. IEEE Transactions on CAD of Integrated Circuits and Systems, May 2012, pp. 726-739.

[10] Fick, D. et. al. “A highly resilient routing algorithm for fault-tolerant NoCs” In: DATE 2009. pp. 21-26.

[11] Tsai, W. et. al. ”A Fault-Tolerant NoC Scheme using bidirectional channel”. In: DAC 2011. pp. 918-923.

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

[13] Boyan, J. and Littman, M. “Packet routing in dynamically changing networks: a reinforcement learning approach”. In: Advances in Neural Information Processing Systems 1993. pp. 671-678.

[14] Carara, E.; Moraes, F. G.; “Flow oriented routing for NOCS”. In: SOCC 2010. pp. 367-370.

[15] Carara, E. et. al. “Achieving Composability in NoC-Based MPSoCs through QoS Management at Software Level”. In: DATE 2011. pp. 407-412.