OSR-Lite: Fast and Deadlock-Free NoC Reconfiguration Framework

Alessandro Strano, Davide Bertozzi
University of Ferrara
ENDIF Department,
Via Savonarola 9,
44122 Ferrara, Italy
Emails: {strlsn1, brtdvd}@unife.it

Francisco Triviño, José L. Sánchez
Francisco J. Alfaro
Universidad de Castilla-La Mancha
Computing Systems Department s/n,
02071, Albacete, Spain
Emails: {ftrivino,jsanchez,falfaro}@dsi.uclm.es

José Flich
Universitat Politècnica de València
Department of Computer Engineering
Camino de Vera s/n,
46071, Valencia, Spain
Email: jflich@disca.upv.es

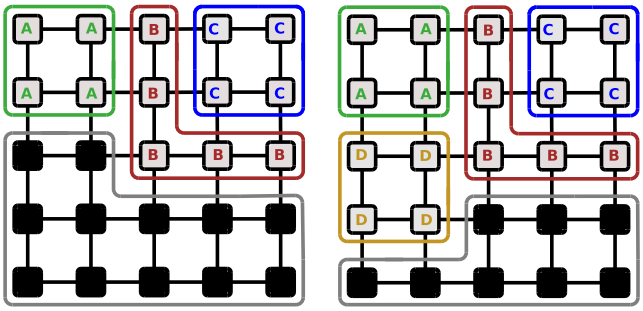
***Abstract*—Current and future on-chip networks will feature an enhanced degree of reconfigurability. Power management and virtualization strategies as well as the need to survive to the progressive onset of wear-out faults are root causes for that. In all these cases, a non-intrusive and efficient reconfiguration method is needed to allow the network to function uninterruptedly over the course of the reconfiguration process while remaining deadlock-free. This paper is inspired by the overlapped static reconfiguration (OSR) protocol developed for off-chip networks. However, in its native form its implementation in NoCs is out-of-reach. Therefore, we provide a careful engineering of the NoC switch architecture and of the system-level infrastructure to support a cost-effective, complete and transparent reconfiguration process. Performance during the reconfiguration process is not affected and implementation costs (critical path and area overhead) are proved to be fully affordable for a constrained system. Less than 250 cycles are needed for the reconfiguration process of an 8x8 2D mesh with marginal impact on system performance.**

I. Introduction

The new mobile usage models that are coming about require the execution of multiple use cases on the same device while optimizing resource consumption for each of them. On the other hand, the hardware and software design convergence in todays complex embedded systems call for an upgrade of architecture building blocks in the direction of runtime reconfigurability and adaptivity. As a result, applications should be able to frequently reconfigure the underlying hardware platform on-the-fly and in a cost-effective way, thus selecting the most convenient operating point that suits their needs and allows an efficient use of system resources.

Modern multi-core integrated systems achieve scalable computation horsepower and power efficiency by integrating a large number of processing cores on the same silicon die. This trend is unmistakable since current products already include tens and even hundreds of processing cores, such as the Tilera multicore processor [1]. In this context, on-chip interconnection networks (networks-on-chip, NoCs) are typically used to provide communication parallelism and the reference integration infrastructure for the whole system.

To address the new functionalities, the NoC must be enriched with an efficient reconfiguration process which enables the smooth and transparent transition between system configurations. For instance, Figure 1 shows two different configurations of a multicore system over time. In the first one (configuration A) different applications are mapped to the NoC nodes and execute concurrently, while other resources are powered down. Later, the resource manager may trigger a chip reconfiguration to power on unused resources and thus activate a new application (configuration B).



(a) Configuration A (b) Configuration B

Fig. 1. Two NoC configurations where the routing algorithm needs to be adapted.

The transition between configurations needs a careful design of the NoC routing algorithm, which establishes the paths for every packet in the network. At each configuration a different routing algorithm is needed. In both cases, the algorithm must be deadlock-free (should not introduce cycles in its channel dependency graph). However, in the transition between configurations, both algorithms can induce extra dependencies that lead to deadlock.

Therefore, in order to migrate from one configuration to the other, one possible approach is to drain the network, then changing the routing algorithm to the new one and finally resuming traffic injection with the new algorithm. This is the case of the so called traditional static reconfiguration (TSR). In this case system performance is likely to be heavily impacted by the reconfiguration process due to the temporarily low resource utilization. Alternatively, the network can be dynamically reconfigured, in the sense that traffic is not stopped during the reconfiguration process, but an effort is needed to avoid deadlock situations. This is typically achieved by devoting extra resources to the network. We refer to this

case as the dynamic reconfiguration.

In this paper we advance state-of-the-art in reconfiguration frameworks for NoC-based systems. However, instead of designing a brand new reconfiguration mechanism, we recognize the large amount of bibliography and proposals made for reconfiguration mechanisms in high-performance off-chip networks. In this sense, we pick the approach that better suits the NoC domain and the tight resource budgets of the on-chip environment.

The Overlapping Static Reconfiguration process (OSR) in [2] enables a transparent system reconfiguration process. However, in [2] only the protocol was described while at the same time highlighting the key architectural requirements to properly support it (namely virtual channels, routing tables, event notification, involvement of end-nodes in the reconfiguration process). Unfortunately, no practical implementation insights were provided, thus raising the reader's skepticism on the applicability of OSR to an on-chip setting.

In this paper we report on the first-time implementation of the native OSR protocol in an on-chip network, proving that the needed network over-provisioning is such to make the protocol not viable in practice. As a consequence, the paper targets the modification of OSR to better match the requirements of the resource-constrained NoC setting, thus resulting into the OSR-Lite framework. Such modifications concern both selected protocol features (without giving up the goodness of the underlying idea) and relevant implementation techniques.

With OSR-Lite in place, it is possible to reconfigure a whole 64-node network in a few hundreds of cycles, enabling the entire and transparent transition between any pair of independent and unrelated configurations. Moreover, this is achieved with no impact on network latency and with no impact on switch delay. The reconfiguration performance of OSR-Lite makes it the enabling tool for planned reconfigurations in multicore systems. The following specific scenarios can be therefore materialized by the outcome of this work:

- *Virtualization of the system*. Our method enables the runtime division of the entire network into sets of virtual regions for assignment to different applications running concurrently.
- *Power management*. The reconfiguration mechanism can be exploited for powering down unused resources; such functionality becomes compulsory to keep power consumption levels to reasonable bounds.
- *Reliability*. When a NoC is augmented with transient fault tolerance, then this kind of faults can be tolerated without any loss of information. However, intermittent faults are likely to be an indicator of the gradual onset of a permanent fault (typically, a wear-out fault). In this case, OSR-Lite can be used to reconfigure the network so to exclude the affected link/switch component, before the permanent fault shows up and causes packet loss.

The rest of the paper is organized as follows. In Section II related work for reconfiguration is described. Then, in Section III the OSR technique is briefly described. Section IV focuses on the OSR-Lite proposal, whereas Section V deals with implementation issues. Then, Sections VI and VII deal with high-level results and synthesis results, respectively. The paper is concluded in Section VIII with conclusions and future work.

II. Previous Work

During the last two decades, a large number of proposals have been presented about resilient routing for both off-chip and on-chip networks. These approaches are either nonreconfigurable fault-tolerant routing strategies which tolerate a limited number of faults [3]–[6], or reconfigurable routing mechanisms that allow unlimited changes to the network. We focus on schemes of the second category, in particular on those based on reconfiguration processes that consider such changes to the network structure to obtain new routing paths replacing the previous ones.

In off-chip networks, such as those used in clusters, during a reconfiguration process, the topology resulting from the connection/disconnection or failure of network components is discovered by a central node, which runs the reconfiguration algorithm in software. The management software computes new routing tables and distributes them to each node. Detecting the new topology and communicating the new routing tables can be completed with or without traffic into the network. Static reconfiguration first stops and drains all user traffic from the network before completing the reconfiguration process [7], [8]. This reconfiguration method is unable to provide real-time and quality-of-service support needed by some applications. On the contrary, dynamic reconfiguration updates routing tables without stopping user traffic. In this case, the main challenge is to guarantee deadlock freedom as old and new routing functions are simultaneously active [2], [9]–[15].

In the context of networks on chip, new techniques have been proposed and other retain some features of the above. The Vicis NoC architecture [16] uses the turn routing model during fault-free operation, and a heuristic solution that makes exceptions to that routing model to maximize connectivity. Reconfiguration process rewrites the routing tables based on the information from built-in-self-test units in each router. When large number of faults occur, exceptions sometimes result in deadlocked routing paths.

A reconfigurable fault-tolerant deflection routing algorithm based on reinforcement learning for NoC has been proposed in [17]. The algorithm reconfigures the routing tables through reinforcement learning based on 2-hop fault information. In [18], a reconfigurable routing algorithm for a 2D-mesh NoC is presented. This algorithm introduces low hardware cost but can only be used in one faulty router or regular region topology. Other proposals can deal with irregular fault regions. A mechanism to tolerate failures in networks for parallel computers is described in [19]. It tolerates any number of failures regardless of their spatial and temporal distributions. Immunet is limited by the network connectivity and results in high area overhead because it requires three routing tables per router. In [20], a region-based routing has been proposed to handle

irregular networks. This algorithm groups destinations into regions to make routing decision. However, it does not provide a reconfiguration method to migrate from one configuration to another.

Finally, [21] presented Ariadne, an agnostic reconfiguration algorithm for NoCs, capable of circumventing large numbers of simultaneous faults, and able to handle unreliable future silicon technologies. Ariadne utilizes up*/down* for high performance and deadlock-free routing in irregular network topologies that result from large numbers of faults.

Ariadne is implemented in a fully distributed mode. Thus it results in very simple hardware and low complexity although it comes with suboptimal solutions for lack of global view. The up*/down* routing will not perform optimally under certain configurations, specially in the absence of failures (in a 2D mesh). In addition, up*/down* routing is encoded in routing tables at switches. Unfortunately, the Ariadne latency badly scales with network size (the configuration latency increases with the square of the nodes number). This latter property has a severe impact on the network performance especially because Ariadne does not guarantee a transparent transition between configurations. The flits have to freeze in the network pipelines and the throughput drops to zero during reconfiguration. Even when the communication resumes, a high contention due to the fullness of injection queues strongly degraded the network performance for a long period.

As opposed to these solutions, OSR-Lite does not use routing tables at switches, allows coding any efficient routing algorithm (even DOR routing) and requires lightweight switch support to enable truly fast dynamic reconfiguration. Moreover its latency smoothly increases with network size, and the configuration transition is transparent, ultimately preserving the throughput of the system.

III. Native OSR technique

Typically, a routing algorithm is deadlock-free when its channel dependency graph (CDG) is acyclic (we do not consider fully adaptive routing algorithms). The CDG is set by representing the resources of the network by vertices (mainly the buffers associated with the ports of each switch) and the dependencies between two resources by arcs. There is a dependency between two resources r_1 and r_2 if a message can use r_1 and request r_2.

Two routing algorithms R_1 and R_2 are deadlock-free when they have an acyclic channel dependency graph. However, when using both algorithms at the same time new extra dependencies are induced potentially leading to deadlock. This can be seen in Figure 2 where a cycle is formed when using two routing algorithms (XY and YX) at the same time. During a reconfiguration process we refer to R_{old} as the old routing function and R_{new} as the new routing function. Similarly, packets routed with R_{old} will be referred to as old packets and packets routed with R_{new} will be referred to as new packets.

The native OSR method is based on the fact that those cycles are created only when old packets using R_{old} are routed after new messages using R_{new}. If old packets are guaranteed to never go behind new packets the extra dependencies do not occur in practice and then no deadlock can be formed. Indeed, in a static reconfiguration process the entire network is drained thus guaranteeing old packets will never go behind new ones.

OSR is a static reconfiguration process but localized at link/router level, and not at network level. Indeed, it guarantees that new packets are only forwarded via links that have been drained from old packets. This is achieved by triggering a token that separates old packets from new packets. The token is triggered by all the end nodes and tokens advance through the network hop by hop. Indeed, tokens follow the CDG of the old routing function, draining the network from old packets. However, in contrast with static reconfiguration, the new packets can enter the network at routers where the token already passed. Figure 3 shows the complete native OSR mechanism, involving a central manager. In a first step, a reconfiguration action is triggered, either by the detection of a malfunctioning component or by a higher level manager in the system stack requiring a reconfiguration, e.g. a new application is admitted. In any case, when needed the central manager may receive event notifications through the network (step 1). Then, in step 2, the new algorithm for the new configuration is computed by the central manager. The resulting information is disseminated to all the switches in step 3. In step 4 the end nodes trigger the token and the OSR reconfiguration spreads

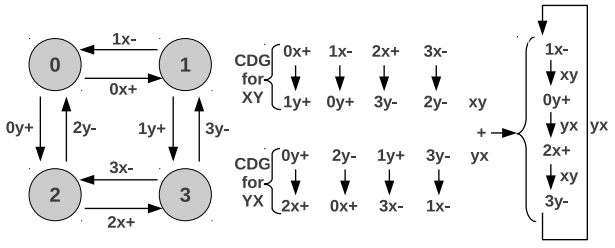


Fig. 2. Channel dependency graph for two routing algorithms and the combination of both.

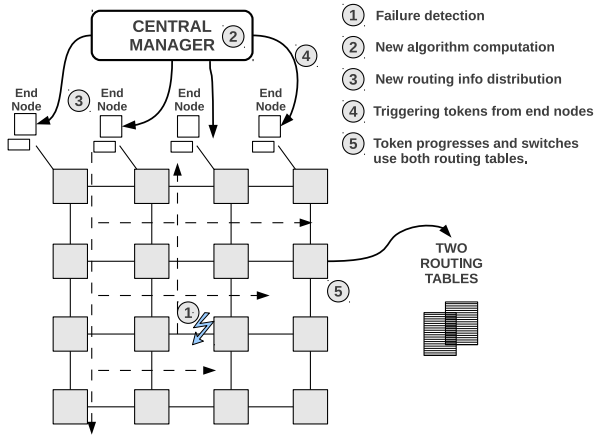


Fig. 3. Reconfiguration steps performed in an OSR environment.

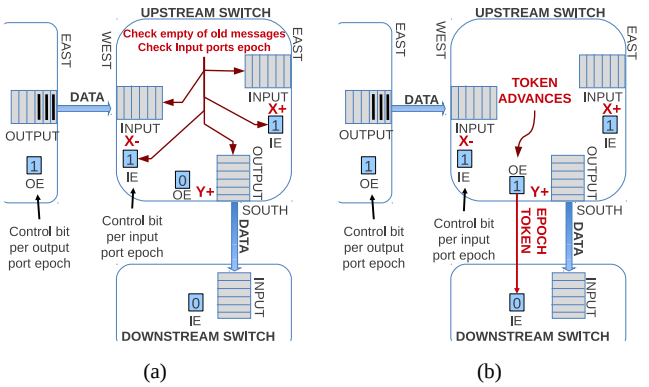


Fig. 4. Token advance in a network: (a) check for absence of old messages and input ports epoch, (b) token signal propagation. The token separates old traffic from new traffic.

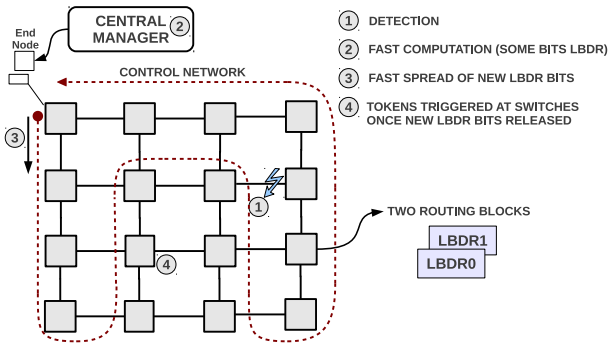


Fig. 5. Reconfiguration steps performed in an OSR-Lite environment.

throughout the network (step 5).

Figure 4 shows how tokens advance in a network. At a given output port, a token is triggered to the next downstream router indicating the output port has been drained from old packets. This is guaranteed when the token has been received through all the input ports of the switch that have old (R_{old}) output dependencies with the output port. These port dependencies can be extracted from the R_{old} routing algorithm. However, how to perform this is not explained in [2], although it is key to obtaining an efficient implementation.

Notice that the token divides two epochs in the network, the old epoch (when packets are routed with the R_{old} routing function) and the new epoch (when packets are routed with the R_{new} routing function).

IV. OSR-Lite

The OSR mechanism needs to be modified in order to better suit the NoC environment so to become an efficient and plausible mechanism for planned reconfigurations. Indeed, the main issues addressed in this paper are the following:

- *Codification of the routing information.* During the reconfiguration process both routing algorithms coexist at the same time at routers. This means resources need to be sized for both algorithms. In OSR, routing tables were used to store the routing info. In NoCs, however, routing tables are an expensive resource in terms of access time, area, and power consumption. Therefore, hosting two routing tables per switch input port does not appear to be a cost-effective solution for OSR-Lite.
- *Control virtual channel (VC) used in OSR.* Different actions (sending routing information to routers, triggering the reconfiguration process) are performed during the OSR reconfiguration which imply the exchange of information between a central manager and the routers or the endnodes. In [2] this was implemented by means of a control VC. Unfortunately, using VCs only for that purpose has a large impact on router implementation (will be seen later) and is not fully justified in an on-chip.
- *Reliable control VC assumed in OSR.* A different (spanning-tree) algorithm is assumed in OSR to effectively route control packets through the control VC.
- *Involvement of end nodes in the reconfiguration process.* In OSR the end nodes were notified to trigger the reconfiguration. This is done by end nodes injecting the token directly as a new packet. In NoCs, reaching the end nodes via dedicated packets from the central manager would be a time-consuming course of action. In order to cut down on the reconfiguration latency, involving only switches and not endnodes in the reconfiguration would be an appealing property in a NoC setting.

In order to address all these issues, we propose the OSR-Lite approach. Figure 5 shows all the steps and the main modifications performed. In particular, we exploit a control network through which routers can inform about expected topology changes (e.g., an output link is having frequent transient failures and is going to fail soon, or a region of the NoC is overheated and needs to be powered down). The control network collects all the notification events and sends them to a central manager (step 1). If the reconfiguration is instead initiated by a resource manager in the context of power management or virtualization strategies, step 1 can be skipped. The central manager then computes the new configuration (step 2) and disseminates the new routing information to the switches (step 3). Then, every switch starts the OSR-Lite reconfiguration process in step 4. Notice that end nodes are not involved in the reconfiguration process.

The control network can be used also in step 3 for routing bit dissemination to the switches. In previous work in [22] we have presented the design of a dual network for switch-to-global manager bidirectional signaling, thus offloading critical control tasks from the main data network. In that work, the dual network was used to notify diagnosis information to the manager following the main NoC testing phase, and to notify configuration bits of the routing mechanism to the switches. The same network could be reused for other purposes, such as congestion management, deadlock recovery and software debugging. In [22] it is showed to be a cost-effective solution for control signaling, which can be easily and effectively made reliable through a combination of fault-tolerant and online testing strategies. For this reason, this work relies on such a fault-tolerant dual network to convey control information

of the reconfiguration process. Furthermore, [22] also reports an efficient computation algorithm that comes up with the routing configuration bits of a new network partitioning or topology shape. This is the algorithm the controller runs in step 2. Given that the control network and the computation algorithm are covered by previous work, from now on we focus on the core reconfiguration process of the network and on the microarchitectural support for that. The reader should keep in mind that all these mechanisms will work together in the complete reconfiguration framework. In the next section we describe the router implementation in more detail.

V. OSR-Lite implementation

Without lack of generality, we use the xpipesLite switch architecture [23] to prove viability of our OSR-Lite mechanism. The switch implements both input and output buffering and relies on wormhole switching. The crossing latency is 1 cycle in the link and 1 cycle in the switch itself. The switch relies on a stall/go flow control protocol. It requires two control wires: one going forward and flagging data availability (”valid”) and one going backward and signaling either a condition of buffer filled (”stall”) or of buffer free (”go”). We assume the following parameter values in the architecture: 32 bit flit width, 6 flit output buffers and 2 flit input buffers. To note that different flit width and input/output buffer depth could be assumed while preserving the OSR-Lite mechanism implementation.

The switch architecture is extremely modular. A port-arbiter, a crossbar multiplexer and an output buffer are instantiated for each output port, while a routing module is cascaded to the buffer stage of each input port. We implement logic-based distributed routing (LBDR) [24]: instead of relying on routing tables, each switch has simple combinational logic that computes target output ports from packet destinations. The support for different routing algorithms and topology shapes is achieved by means of 16 configuration bits for the routing mechanism of the switch (hereafter denoted as LBDR bits). LBDR bits carry the routing algorithm information (expressed in terms of routing restrictions), the connectivity information of switch output ports and special detour bits. See [24] for more details. Such bits make LBDR a flexible routing mechanism while at the same time significantly cutting down on the memory requirements of routing tables. LBDR bits are computed by a central NoC manager and disseminated to the switch input ports through the dual control network. Indeed, two sets of LBDR bits are allocated at each router for OSR-Lite. Upon receiving the new routing bits, a router triggers the reconfiguration process by auto-generating initial tokens at its local input port (port connected to an end node) and processing the tokens accordingly.

The logic enabling the OSR-Lite mechanism was integrated into the above mentioned baseline switch taking care to preserve its modularity together with its performance. Thus, the OSR-Lite logic was designed in new modules plugged into the switch without affecting the existing blocks. Moreover, the new modules were instantiated for each switch port following the modularity of the baseline blocks (the OSR-Lite mechanism can be extended for switches of every arity by means of simple logic replication).

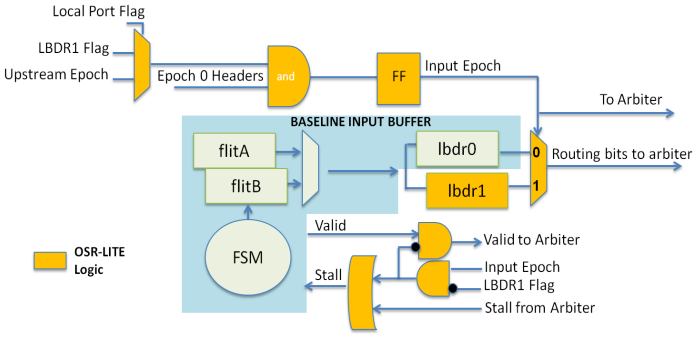


Fig. 6. Switch input buffer enhanced with the OSR-Lite logic and a new set of routing mechanism.

A. OSR-Lite at the Input Ports

As a first step, the baseline switch was enhanced with a second routing logic unit (LBDR1) collecting the new routing info coming from the central manager. This unit is connected to the input buffer as the baseline LBDR0 block (see Figure 6) although is used exclusively for routing packets in the new epoch (new packets). The switch arbiters need to select the routing info from the appropriate routing logic block (either LBDR0 or LBDR1). This is obtained from a multiplexer configured by the current epoch of the input port (in a flip-flop). In order to reduce the reconfiguration latency, the input port evolves to the new epoch as soon as there are no stored header flits at the input port with the epoch bit set to zero (*Epoch 0 headers* signal) and the token has been received from the upstream switch (*upstream epoch* signal). Notice that in the case of the ports connected to end node (*local* port; *local port* flag), the token is assumed to arrive with the arrival of the new configuration bits (*LBDR1 flag*). In this case, the header flits located in the buffers are considered of the new epoch when the new configuration bits have arrived and the routing mechanism (LBDR1) is set. Notice that local ports do not introduce dependencies between channels that may lead to deadlocks, therefore is safe to assume all the injected flits as belonging to the new routing function. To notice that the token propagation will always start from local ports at switches, not involving end nodes.

The number of flit headers to be routed by LBDR0 and stored in the buffer is detected by a 2 bits counter monitoring the incoming and outgoing headers of the input buffer module. The counter increases its value when a header is accepted and the incoming token is low and decreases its value when a header is sent. In order to preserve the max performance of the baseline switch, sequential logic stages were exploited to avoid impacting the critical path in the OSR-Lite mechanism.

Notice that the implementation prevents possible race conditions from occurring. For instance, a token may be received from the upstream switch before the new routing bits are received. In that case, the header flits in the input buffers are

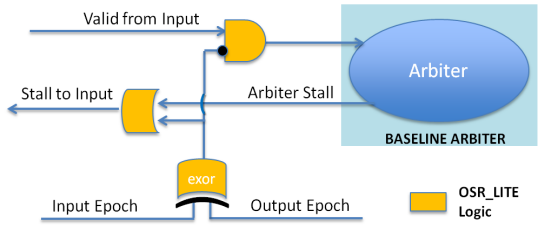


Fig. 7. Switch arbiter enhanced with the OSR-Lite logic.

stalled and declared not valid to the internal switch logic until LBDR1 is set.

B. OSR-Lite at the Arbiters

OSR-Lite requires a lightweight new module plugged around the baseline arbiters. The logic is reported in Figure 7. Basically, a set of AND/OR logic blocks together with a set of $EXOR$ blocks allow the arbiter to process an incoming header exclusively when the epoch of the switch input port is the same as the one of the destination output port. On the contrary, a packet residing in an input port with the new epoch is stalled until the output port evolves to the new epoch (guaranteeing old packets go first and then new packets).

C. OSR-Lite at the Output Ports

Concerning the output port, an output port evolves to the new epoch when all the input ports with output dependencies to this output port have evolved to the new epoch. In order to efficiently deal with the dependencies, OSR-Lite takes profit of the routing bits used in LBDR. Routing bits indicate the routing restrictions that exist at neighboring switches. Therefore, they can be seen also as channel dependencies. If the R_{xy} bit is set it means that there is a link dependency between the output port x and the output port y at the next switch. On the contrary, if the bit is reset it means there is no dependency and in that case we can safely assume no packets will come through the port x requesting output port y. Therefore, the output port needs to receive both the epochs of the input ports and the routing restrictions located at the neighboring switches. The mechanism is enabled by a set of OR blocks (each of them belonging to a different input port) followed by an AND block, as represented in Figure 8.

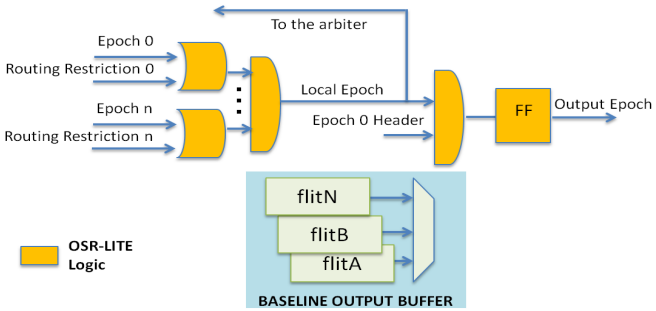


Fig. 8. Switch output buffer enhanced with the OSR-Lite logic.

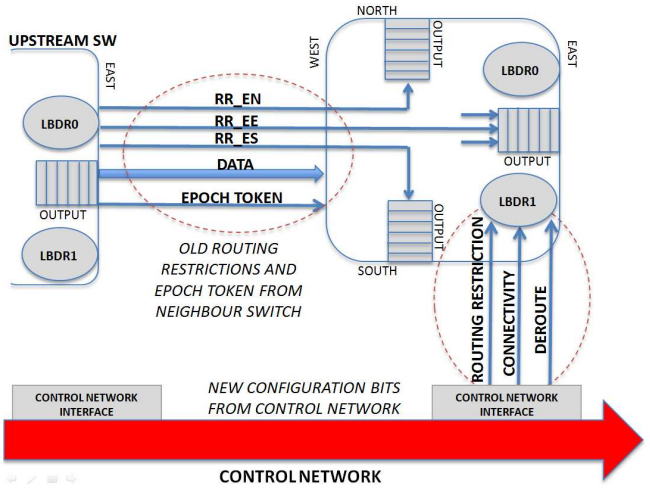


Fig. 9. Configuration information from neighbor switches and control network

In contrast with the baseline OSR technique (where the routing restriction information was saved in the routing table), the OSR-Lite mechanism needs to obtain channel dependencies from the routing logic located at neighbor switches. As a result, three additional routing bits are sent by the LBDR0 logic of the upstream switch together with the token bit. To note that LBDR0 received its routing bits information through the control network in an earlier configuration stage. In addition, the input port needs to send the incoming routing restriction signals to the appropriate output ports. Thus every link is extended by 4 additional wires (i.e. 1 token wire + 3 routing restriction wires). See Figure 9.

Finally, the token is sent by the output port to the downstream switch when all the input ports with dependencies with the output port have evolved to the new epoch, meaning all these input ports have drained all the old packets from their buffers (see the $LocalEpoch$ signal in Figure 8).

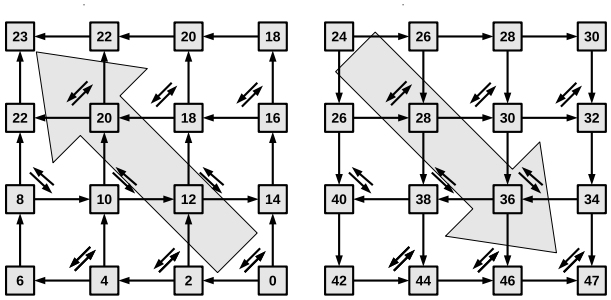
Once the network has completely migrated to Epoch 1, the central manager can safely fill LBDR0 bits with a copy of LBDR1 bits, and instruct all the switches to safely swap to Epoch0 again. This allows for the system to be ready in few cycles for a new reconfiguration process.

VI. System-Level Evaluation

In this section, we evaluate OSR-Lite. First, we show how the OSR-Lite propagates over the network. Then, we evaluate the reconfiguration time overhead under different injection rates using synthetic traffic. Moreover, we compare the proposed reconfiguration with a static reconfiguration process in terms of network latency. Finally, we provide performance results by running real applications in a full system simulator environment.

A. Propagation

In order to simulate the reconfiguration process, we have modeled the OSR-Lite scheme in our event-driven cycle-accurate network simulator. A 8×8 mesh is used with wormhole switching (although the proposed method also works for



(a) (b)

Fig. 10. OSR-Lite propagation over a 4×4 2D mesh topology: (a) scrolling up, and (b) scrolling down.

virtual cut-through switching). Flit size is set to 4 byte and messages are 5-flit long. For the transient state, 50K messages are assumed and results are collected after 50K messages are received.

Figure 10 shows how OSR-Lite tokens propagate over a mesh when there is no traffic traveling through the network. The diagonal arrows represent the bidirectional restrictions imposed by the routing algorithm (Segment-Based routing [25] in this case). In this figure, the numbers inside the switches represent the cycle when the token signal is propagated to its neighbors. Moreover, the arrows among switches depict the direction of the token signal propagations. As we can see, the token signals propagate among switches throughout the network in the order of the routing channel dependency graph, where Figure 10.(a) follows a scrolling up zig-zag direction, and Figure 10.(b) follows a scrolling down zig-zag direction.

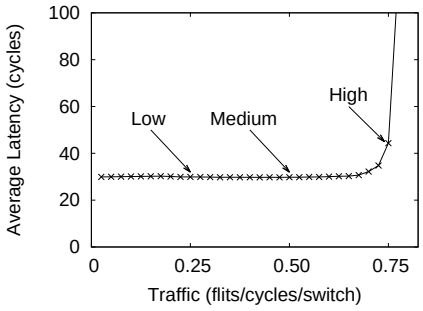
When no messages are traveling through the network and a regular 2D mesh is considered then the number of clock cycles required for the OSR-Lite reconfiguration process is modeled by the following formula:

$$PropagationTime = (4xDx(D-1)) - 1 \quad (1)$$

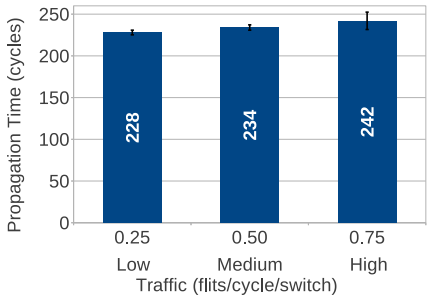
where D represents the mesh dimension. As we can see, it is a very fast process as the protocol uses only 223 cycles when a 8×8 mesh is considered. The high speed of the OSR-Lite reconfiguration process allows to perform frequent planned reconfigurations without affecting the integrity of the system operations. However, when there are messages traveling through the network the switches must drain the input queues of old messages before propagating the token signal as explained in Section III. This fact delays the OSR-Lite propagation depending on the network load. In the following, we analyze this effect taking into account different injection rates.

B. Time Overhead

In order to analyze the impact of the network load over the OSR-Lite reconfiguration framework, we have performed different simulations varying the injection rate. For each rate, we assume a constant packet generation rate for all end nodes. Moreover, in order to ensure that start-up instabilities do not affect our evaluation results, reconfiguration is not invoked until the network is completely stabilized. Figure 11.(a) shows the performance obtained in a 8×8 2D mesh network under



(a)



(b)

Fig. 11. (a) Average message latency at different injection rates for SR routing on 8×8 2D mesh (b) OSR-Lite propagation over a 8×8 2D mesh topology at different injection rates.

uniform traffic when no reconfiguration process is triggered. The figure indicates the three network injection rates that are used in the simulations. In what follows, the three rates are referred to as Low, Medium, and High, respectively.

Figure 11.(b) shows the number of cycles involved in the propagation of the OSR-Lite process taking into account the three different injection rates. Each bar depicts the mean of 30 simulations varying the seed. Moreover, we show the error bars that represent the 95% confidence interval. As we can see, the propagation time does not exceed 242 cycles for the High injection rate. Moreover, the difference between both the minimum and the maximum network loads is only 14 cycles, and therefore, the network traffic condition has a minimal effect on the OSR-Lite token propagation.

Finally, the contribution in terms of cycles for event notification (A), algorithm computation (B), configuration bits delivery (C) and OSR-Lite propagation (D) should be taken into account to determine the total latency for a complete reconfiguration process. In particular, (A) and (C) latencies depend on the position of the components to reconfigure with respect to the central manager. On the other hand, (B) and (D) latencies are related to the number of components to reconfigure and the traffic injection rate respectively. As an example, when we consider a 8×8 mesh then at most 66 cycles are required to cross the control network. Moreover, if 7 switches need to be reconfigured (i.e. the scenario of Figure 1) then 195 cycles are required by the computation algorithm in [22]. Finally, 242 cycles are spent by (D) in a High injection rate scenario. Summing up, the total amount of cycles for a

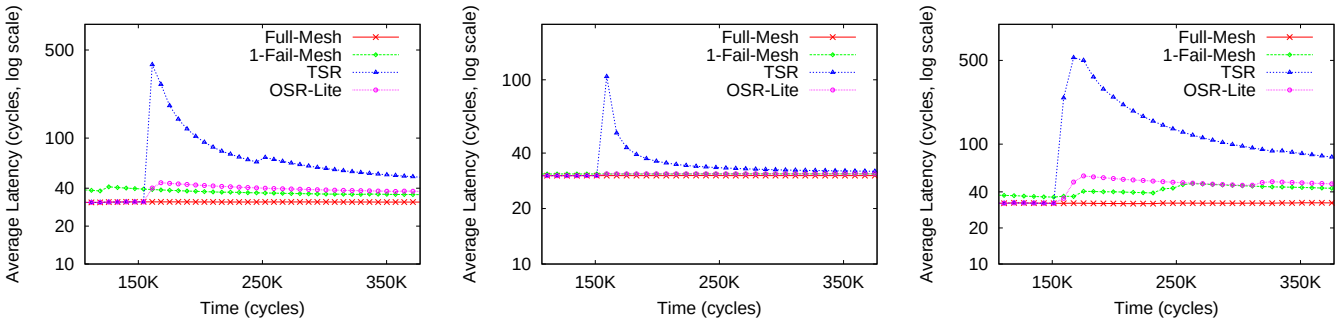


Fig. 12. Average message latency with (a) hotspot traffic and uniform traffic ((b) medium network load and (c) high network load).

complete reconfiguration process is the following:

$$66(A) + 195(B) + 66(C) + 242(D) = 569\ Cycles \qquad (2)$$

For dissemination of new LBDR bits to the switches, the dual network has to carry 17 bits per switch. However, not all switches need to be reconfigured, since the algorithm in [22] is able to evolve a system configuration into a new one while updating the minimum amount of LBDR configuration bits.

C. Comparison

In this section we compare the OSR-Lite protocol and the traditional static reconfiguration process (TSR). Figures 12.(a), 12.(b), and 12.(c) represent the average network latency respectively under hotspot traffic and uniform traffic with Medium and High injection rates, where both reconfiguration processes (OSR-Lite and TSR) are invoked after 150K cycles. Moreover, we have plotted two additional lines: the average message latency for the full mesh (Full-Mesh), and the average message latency for the mesh which has one link disabled from the beginning of the simulation (1-Fail-Mesh). Notice the y-axis is in logarithmic scale. Moreover, we have selected a random link in the 8×8 mesh as faulty. Under hotspot traffic pattern, 5 nodes are randomly chosen as hot spots which receive an extra proportion of traffic (30%) in addition to the regular uniform traffic.

The first observation is that both Full-Mesh and 1-Fail-Mesh obtain a different message latency. This is normal because the 1-Fail-Mesh suffers a latency degradation due to the disabled link. On the other hand, the two reconfiguration processes (OSR-Lite and TSR) start at the same time at the 150K cycle. At this point, the reconfiguration process moves from the Full-Mesh to the 1-Fail-Mesh topology. This effect can be estimated by the figures as the latency evolves from the latency obtained for the Full-Mesh to the latency obtained for the 1-Fail-Mesh. However, an important result based on the figures is that OSR-Lite performs the reconfiguration without degrading the obtained performance. In this case, the obtained latency grows up to the 1-Fail-Mesh line. Therefore, the latency is always near the maximum obtained with the 1-Fail-Mesh topology. In the TSR case, on the contrary, the latency is degraded due to the reconfiguration process overhead (need to drain the network). In the three cases, the latency grows above the 1-Fail-Mesh latency until it stabilizes. Specifically, in the Figure 12.(c) the latency of the TSR line grows to more than 500 cycles, and then stabilizes after 350K cycles. In this period of time, the TSR reconfiguration is degrading the obtained latency more than the link failure degradation produces. On the other hand, the OSR-Lite latency is upper bounded by the 1-Fail-Mesh latency.

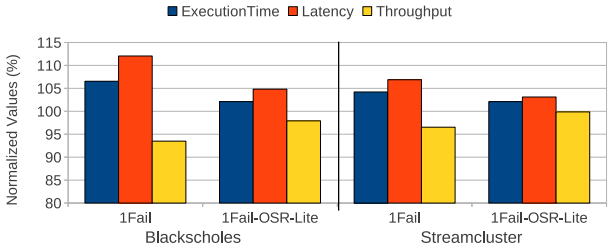


Fig. 13. Performance with real applications.

Interestingly, the hotspot traffic and the uniform traffic with a High load have similar reconfiguration performance. Then, we can observe that the OSR-Lite has no impact on the message generation while the TSR process does. In fact, the TSR process increases considerably the obtained latency for all the cases. The main reason is that TSR queues all the messages at end nodes during reconfiguration while that need disappears in the OSR-Lite scheme.

D. Performance with Real Applications

In the following, we present performance results when real applications are used. In this case, we use a full-system simulator based on Simics-GEMS [26], [27]. Regarding the on-chip network, we have used the same configuration as the previously detailed in Section VI. Messages are 8-flit long for control messages and 72-flit long for data messages. As workload, we have used the PARSEC v2.1 benchmark suite [28]. Although we have used all the applications from the PARSEC v2.1 benchmark, due to the lack of space, we only show results for two applications: Blackscholes, and Streamcluster. In all cases, Simsmall input set has been used.

Figure 13 shows the execution time, the network latency, and the network throughput. All the results are shown in normalized terms with respect to the results of a fully connected 4×4 mesh without link failures. The x-axis depicts the 1Fail topology (a 4×4 mesh topology with 1 link failure), and 1Fail-OSR-Lite that represents the same topology as 1Fail but, in this

case, the OSR-Lite reconfiguration process is triggered in the middle of the application execution. Therefore, for this latter case, we pass from the fully connected topology to the 1Fail topology using the OSR-Lite reconfiguration. These results are shown for both Blackscholes, and Streamcluster PARSEC applications.

As we can see, the performance degradation is minimal for both cases (1Fail and 1Fail-OSR). Regarding the 1Fail-OSR case, obviously the degradation is lower because the link failure only affects half of the application execution, and the execution time degradation does not exceeds 3% of execution time overhead in any application.

VII. Synthesis results

The implementation of a switch enhanced with the OSR-Lite mechanism has been compared in terms of area and routing delay with a switch based on the native OSR mechanism described in Section III and the baseline xpipesLite switch architecture [23]. The evaluation will demonstrate the infeasibility of the native OSR mechanism for an on-chip setting because of the need for VCs and the low scalability of routing tables.

For the experiments, an industrial memory compiler for a 40nm process technology was used to generate the memory macros required by the routing tables of the OSR mechanism. The switches together with their reconfiguration mechanisms were synthesized for the same 40nm industrial library.

A. Area Comparison

The description of the OSR mechanism in [2] focuses on the protocol details and it lacks of practical implementation details. Thus we exploited the information provided in [2] to model the OSR mechanism at RTL level and evaluate this latter solution in an on-chip constrained system. Especially, the OSR mechanism relies on 1 data VC supported by an additional control VC, and it adopts routing tables. As a result, we implemented the OSR mechanism into a 5x5 switch augmented with VCs by following the design techniques for area efficiency in [29] and we enhanced the switch with the 40nm memory macros to model the routing tables.

The 8×8 mesh topology of Section VI was considered. Thus, 64 end-nodes are the total number of destinations in the system. When routing tables are used for distributed routing, each switch input port has a memory module with a number of words equal to the amount of destinations. Every word is composed of 3 bits, matching the switch radix. Given a destination ID, the switch selects the target output port based on look-up table. The minimum word width that the memory compiler, at the 40nm technology node, can generate is 4 bits. As a result, above all the available memory cuts, a single-port low-power RAM with 64 words of 4 bits was the memory cut showing the lowest routing delay and area footprint.

Finally, Figure 14.(a) shows the area footprint of this latter solution (the $OSR\ SW$) with respect to a baseline switch and our proposed solution ($OSR\text{-}LITE\ SW$). In particular, the OSR-Lite area overhead takes into account also the contribution of the control network carrying the information from the global manger to the routing mechanisms. For this purpose, we exploited the fault-tolerant control network proposed in [22].

The OSR-Lite reconfiguration mechanism requires a 14% of area overhead with respect to the baseline switch. This result is mainly due to the additional LBDR routing mechanism (+12%) contribute. On the other hand, the area overhead of the remaining reconfiguration logic (detailed in Section V) is negligible when integrated into the switch.

Interestingly the OSR-Lite switch outperforms the baseline OSR switch: this latter requires approximately two times larger area than the counterpart solution. This result is mainly due to the severe area penalty introduced by the VCs and the 65% area saving achieved by the LBDR mechanism with respect to the routing table.

As a last consideration, the routing mechanism of the OSR-Lite solution scales with network size. In fact, while the memory macro suffers from increasing area and delay penalties, the logic complexity of the distributed routing algorithms does not depend on the number of destinations, hence it stays constant. Indeed, the distributed routing algorithms just grow with the switch radix.

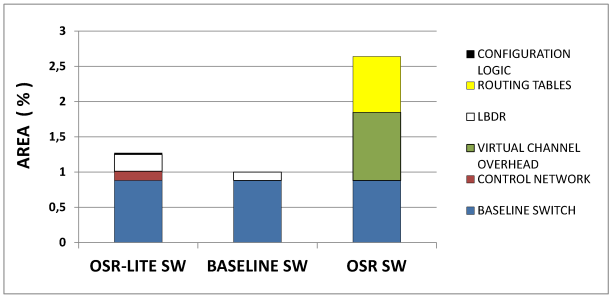
B. Routing Delay Comparison

In order to evaluate the effects of the OSR-Lite mechanism on the switch routing delay, we performed the 5x5 switch synthesis for maximum performance. The same experiment was repeated for both the baseline switch and the switch augmented with the baseline OSR mechanism. The $OSR\text{-}LITE$ switch and the baseline switch achieved a similar maximum operating speed of 750 MHz. As described in Section V, the reconfiguration scheme was designed to avoid long critical path and preserve the baseline switch performance. Our OSR-Lite-enabled switch is thus capable of an at-speed reconfiguration.

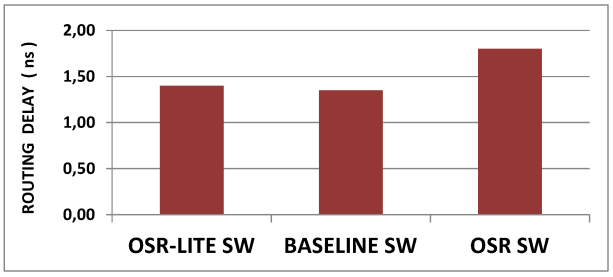
On the other hand, the OSR switch is the 35% slower than our proposed solution as showed by Figure 14.(b). This result is mainly due to the intrinsic complexity added by the VC logic and the delay required to access the 64 words RAM routing tables.

VIII. Conclusion

In this paper we have proposed the implementation of a fast and transparent reconfiguration mechanism in NoCs. The strict constraints found in on-chip networks demand for efficient and compact mechanisms that do not excess in area and latency demands. The native OSR reconfiguration mechanism has been proven to be unsuitable for this purpose although proposing an interesting intuition. Therefore, in this paper we have engineered the proper protocol and implementation modifications to fit reasonable area budgets, with no impact on performance while retaining the same underlying principle for fast reconfiguration. The final mechanism, OSR-Lite, is able to support a transparent NoC reconfiguration with as little as less than 250 cycles.



(a)



(b)

Fig. 14. 5x5 switch (a) area and (b) routing delay comparison.

ACKNOWLEDGMENT

This work was supported by the Spanish MEC and MICINN, as well as European Comission FEDER funds, under Grants CSD2006-00046 and TIN2009-14475-C04. It was also partly supported by JCCM under Grant POII10-0289-3724, by the project NaNoC (project label 248972) and by the project Virtical which is funded by the European Commission within the FP7 Research Programme.

REFERENCES

[1] "Tilera Tile-Gx Product Brief," 2011. Available: http://www.tilera.com/products/processors/TILE-Gx_Family

[2] O. Lysne, J. Montanana, J. Flich, J. Duato, T. Pinkston, and T. Skeie, "An efficient and deadlock-free network reconfiguration protocol," *IEEE Transactions of Computers*, vol. 57, no. 6, pp. 762–779, 2008.

[3] W. Dally, L. Dennison, D. Harris, K. Kan, and T. Xanthopoulus, "The reliable router: A reliable and high-performance communication substrate for parallel computers," in *Proceedings of the Workshop on Parallel Computer Routing and Communication (PCRCW)*, May 1994, pp. 241–255.

[4] C. Glass and L. Ni, "Fault-tolerant wormhole routing in meshes without virtual channels," *IEEE Transactions Parallel and Distributed Systems*, vol. 7, no. 6, 1996.

[5] M. Gómez, J. Duato, J. Flich, P. López, A. Robles, N. Nordbotten, O. Lysne, and T. Skeie, "An efficient fault-tolerant routing methodology for meshes and tori," *Computer Architecture Letters*, vol. 3, no. 1, pp. 3–3, January-December 2004.

[6] C.-T. Ho and L. Stockmeyer, "A new approach to fault-tolerant wormhole routing for mesh-connected parallel computers," *IEEE Transactions on Computers*, vol. 53, no. 4, pp. 427–439, 2004.

[7] K. M. et al., "Fibre channel switch fabric-2 (fc-sw-2)," NCITS 321-200x T11/Project 1305-D/Rev 4. 3 Specification, Tech. Rep., March 2000.

[8] M. Schroeder, A. Birrell, M. Burrows, H. Murray, R. Needham, T. Rodeheffer, E. Satterthwaite, and C. Thacker, "Autonet: a high-speed, self-configuring local area network using point-to-point links," *IEEE Journal on Selected Areas in Communicartions*, vol. 9, no. 8, pp. 1318–1335, October 1991.

[9] R. Casado, A. Bermúdez, , J. Duato, F. Quiles, and J. Sánchez, "A protocol for deadlock-free dynamic reconfiguration in high-speed local area networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 2, pp. 115–132, February 2001.

[10] O. Lysne and J. Duato, "Fast dynamic reconfiguration in irregular networks," in *Proceedings of the 2000 International Conference of Parallel Processing (ICPP)*. IEEE Computer Society, 2000, pp. 449–458.

[11] T. Pinkston, R. Pang, and J. Duato, "Deadlock-free dynamic reconfiguration schemes for increased network dependability," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 8, pp. 780–794, 2003.

[12] J. Duato, O. Lysne, R. Pang, and T. Pinkston, "Part I: A theory for deadlock-free dynamic network reconfiguration," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 5, pp. 412–427, May 2005.

[13] O. Lysne, T. Pinkston, and J. Duato, "Part II: A methodology for developing deadlock-free dynamic network reconfiguration processes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 5, pp. 428–443, May 2005.

[14] D. Avresky and N. Natchev, "Dynamic reconfiguration in computer clusters with irregular topologies in the presence of multiple node and link failures," *IEEE Transactions Computers*, vol. 54, no. 5, pp. 603–615, May 2005.

[15] J. Acosta and D. Avresky, "Intelligent dynamic network reconfiguration," in *Proceedings of the 21st International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE Computer Society, 2007, pp. 1–9.

[16] D. Fick, A. DeOrio, J.H., V. Bertacco, D. Blaauw, and D. Sylvester, "Vicis: A reliable network for unreliable silicon," in *Proceedings of the 46th Design Automation Conference (DAC)*, July 2009, pp. 812–817.

[17] C. Feng, Z. Lu, A. Jantsch, J. Li, and M. Zhang, "A reconfigurable fault-tolerant deflection routing algorithm based on reinforcement learning for Network-on-Chip," in *Proceedings of the International Workshop on Network on Chip Architectures (NocArc)*, 2010.

[18] Z. Zhang, A. Greiner, and S. Taktak, "A reconfigurable routing algorithm for a fault-tolerant 2D-mesh Network-on-Chip," in *Proceedings of the 46th Design Automation Conference (DAC)*. ACM, June 2008, pp. 441–446.

[19] V. Puente, J. Gregorio, F. Vallejo, and R. Beivide, "Immunet: A cheap and robust fault-tolerant packet routing mechanism," in *Proceedings of the 31th Annual International Symposium on Computer Architecture (ISCA)*, June 2004, pp. 198–209.

[20] A. Mejia, J. Flich, J. Duato, S.-A. Reinemo, and T. Skeie, "Segment-based routing: An efficient fault-tolerant routing algorithm for meshes and tori," in *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS)*, 2006, pp. 1–10.

[21] K. Aisopos, A. DeOrio, L.-S. Peh, and V. Bertacco, "Ariadne: Agnostic reconfiguration in a disconnected network environment," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2011.

[22] A. Ghiribaldi, D. Ludivici, F. Triviño, A. Strano, J. Flich, J. Sánchez, F. Alfaro, M. Favalli, and D. Bertozzi, "A complete self-testing and self-configuring noc infrastructure for cost-effective mpsocs," *ACM Transactions on Embedded Computing Systems*, 2011.

[23] S. Stergiou *et al.*, "Xpipes lite: a synthesis oriented design library for networks on chips," in *DAC*, 2005.

[24] S. Rodrigo, J. Flich, A. Roca, S. Medardoni, D. Bertozzi, J. Camacho, F. Silla, and J. Duato, "Addressing manufacturing challenges with cost-efficient fault tolerant routing," in *Proceedings of the 2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip*, ser. NOCS '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 25–32. Available: http://dx.doi.org/10.1109/NOCS.2010.12

[25] A. Mejia, J. Flich, and J. Duato, "On the potentials of segment-based routing for nocs," in *Parallel Processing, 2008. ICPP '08. 37th International Conference on*, sept. 2008, pp. 594 –603.

[26] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hållberg, J. Högberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A Full System Simulation Platform," *Computer*, vol. 35, no. 2, pp. 50–58, 2002.

[27] M. M. K. Martin, et al. , "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 92–99, 2005.

[28] C. Bienia and K. Li, "Parsec 2.0: A new benchmark suite for chip-multiprocessors," in *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, June 2009.

[29] F. Gilabert, M. E. Gmez, S. Medardoni, and D. Bertozzi, "Improved utilization of noc channel bandwidth by switch replication for cost-effective multi-processor systems-on-chip," in *Fourth ACM/IEEE International Symposium on Networks-on-Chip*, 2010, pp. 165–172.