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

F. Triviño[1], A. Strano[3], J. Flich[2], D. Bertozzi[3], J.L. Sánchez[1], and F.J. Alfaro[1]

Abstract— **Current and future on-chip networks will feature an enhanced degree of reconfigurability. On one hand, NoCs must survive to component failures occurring at runtime. On the other hand, power management strategies and virtualization support at network level will increasingly result into the need of planned and temporary disconnection of network components. 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 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.**

Keywords— **NoC, Virtualization, Fault-Tolerant, Reconfiguration.**

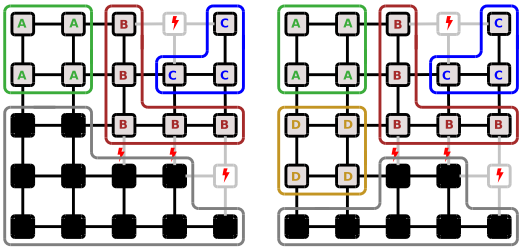
I. Introduction

As complexity keeps increasing, there emerge new requirements that affect how the Networks-on-Chip (NoC) is designed. New challenges arise like providing support for an ever increasing probability of manufacturing defects. Indeed, reliability is one of the most critical emerging challenges. Nanoscale fabrication processes inevitably result in defective components, which lead to permanent errors. Wearout components also impact the system configuration rendering the network affected.

Besides reliability concerns, there is also an interest in providing further functionality to the system. Virtualizing the entire chip into sets of virtual regions and assigning them to different applications running concurrently is appealing in those systems. Similarly, powering down unused resources during most of the time is becoming compulsory to keep power consumption levels to reasonable bounds.

To address these new functionalities, the NoC must be enriched with an efficient reconfiguration process which enables the smooth and transparent transition between configurations. For instance, Figure 1 shows two different configurations of a chip multiprocessor (CMP) system where in the first configuration (conf. A) different applications are mapped, some components are failed, and some resources are powered down. Suddenly, the chip needs to be reconfigured in order to allocate a new application and to power on new resources (conf. B).

In this paper we address this issue, providing an efficient reconfiguration mechanism to NoC-based systems. The Overlapping Static Reconfiguration process (OSR) in [1] enables a transparent reconfiguration process. However, in [1] 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). No practical implementation insights were provided therefore the applicability of OSR to an on-chip setting is highly questionable. 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 a resource-constrained NoC setting, thus resulting into the OSR-Lite framework. Such modifications concerns both some protocol details (without giving up the goodness of the underlying idea) and relevant implementation techniques.



(a) Configuration A (b) Configuration B

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

In particular, we provide a complete framework where the entire reconfiguration process is supported, starting from notification of an event inside the network (we develop a control network), the computation of the proper routing algorithm for the new configuration, notification of the new configuration (through the control network), and support for the OSR-Lite reconfiguration process.

With OSR-Lite in place, the system is able to reconfigure in a very short period of time, 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 rest of the paper is organized as follows. Section II describes the related work. 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. Finally, the paper is concluded in Section VIII.

[1]Univ. de Castilla-la Mancha, e-mails: {ftrivino, falfaro, jsanchez}@dsi.uclm.es.
[2]Univ. Politecnica de Valencia, email: jflich@disca.upv.es.
[3]Univ. Ferrara, e-mail: {strlsn1, brtdvd}@unife.it

II. Previous Work

In the context of networks on chip, new techniques have been proposed tackling the problem of resilient routing. The Vicis NoC architecture [2] 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 [3]. The algorithm reconfigures the routing tables through reinforcement learning based on 2-hop fault information. In [4], 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.

Finally, [5] presented Ariadne, an agnostic reconfiguration algorithm for NoCs. 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

Two routing algorithms R_{new} and R_{old}[1] 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. 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 traditional static reconfiguration (STR) process the entire network is drained thus guaranteeing old packets will never go behind new ones.

[1]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.

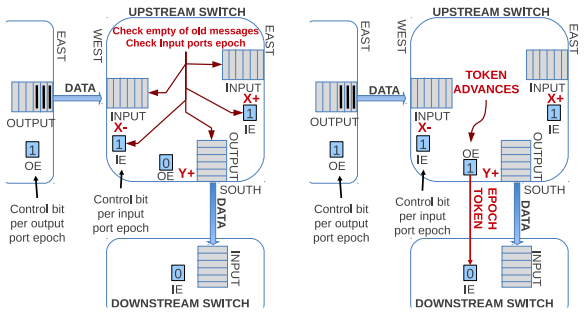


Fig. 2. Token advance in a network: (a) check for absence of old messages and input ports epoch, (b) token signal propagation. The token separates old traffic from new traffic.

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. In contrast with static reconfiguration, the new packets can enter the network at routers where the token already passed.

Figure 2 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 [1], although it is key to obtaining an efficient implementation.

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. In OSR, routing tables were used to store the routing info for both routing algorithms (R_{new} and R_{old}). 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. 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 algorithm is assumed in OSR to effectively route control packets through the control VC.

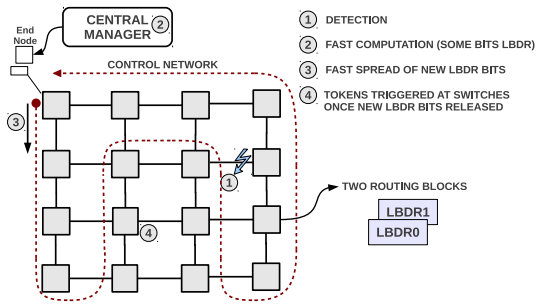


Fig. 3. Reconfiguration steps performed with OSR-Lite.

Involvement of end nodes in the reconfiguration process. 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 3 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. 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.

Given that the control network and the computation algorithm are covered by previous work [6], from now on we focus on the core reconfiguration process of the network and on the microarchitectural support. 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 [7] 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. 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) [8]: 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 18 configuration bits for the routing mechanism of the switch (hereafter denoted as LBDR bits). See [8] 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 autogenerating initial tokens at its local input port (port connected to an end node) and processing the tokens.

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 4.(a)) 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.

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.

B. OSR-Lite at the Arbiters

OSR-Lite requires a lightweight new module plugged around the baseline arbiters. The logic is reported in Figure 4.(b). 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

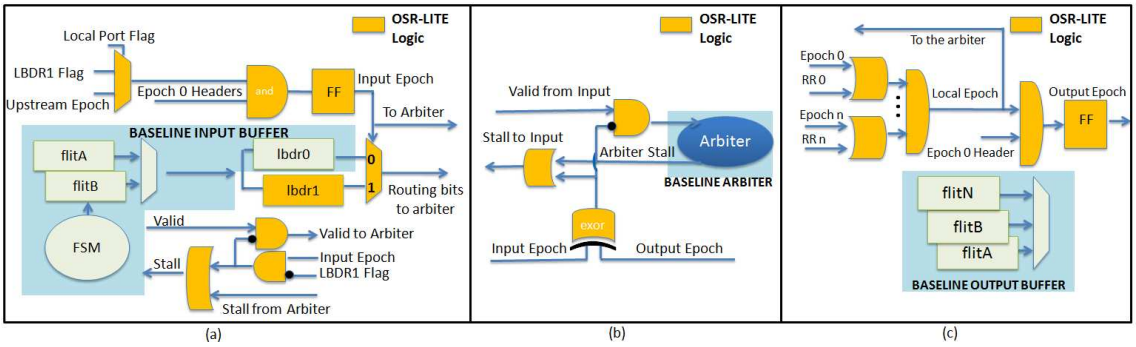


Fig. 4. (a) Switch input buffer enhanced with the OSR-Lite logic and a new set of routing mechanism, (b) switch arbiter enhanced with the OSR-Lite logic, and (c) switch output buffer enhanced with the OSR-Lite logic.

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

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 4.(c).

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 5.

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 4.(c)).

Once the network has completely migrated to Epoch 1, the central manager can safely fill LBDR0

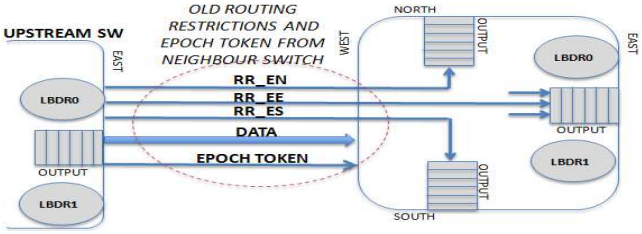


Fig. 5. Configuration information from neighbor switches and control network

bits with a copy of LBDR1 bits, and instruct all the switches to safely swap to Epoch 0 again. This allows for the system to be ready in few cycles for a new reconfiguration process.

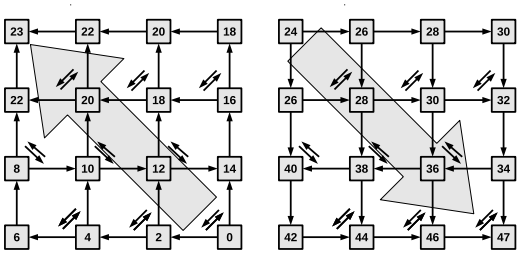
VI. System-Level Evaluation

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. Flit size is set to 4 byte and messages are 5-flit long. For the transient state, 50K messages are assumed before results are collected.

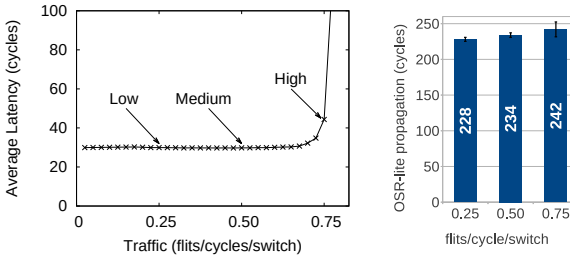
Figure 6 shows how OSR-Lite tokens propagate over a mesh when there is no traffic traveling through the network. The diagonal arrows represent the bidirectional restrictions imposed by the routing algorithm (Segment-Based routing [9]). In this figure, the numbers inside the switches represent the cycle 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 6.(a) follows a scrolling up zig-zag direction, and Figure 6.(b) follows a scrolling down zig-zag direction.

As we can see, it is a very fast process as the protocol uses only 47 cycles when a 4×4 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 de-



(a) (b)

Fig. 6. OSR-Lite propagation over a 4×4 2D mesh topology: (a) scrolling up, and (b) scrolling down.



(a) (b)

Fig. 7. (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 at different injection rates.

pending on the network load. In the following, we analyze this effect taking into account different injection rates.

B. Time Overhead

We have performed different simulations varying the injection rate. For each rate, we assume a constant packet generation rate for all end nodes. Figure 7.(a) shows the performance obtained in a 8×8 2D mesh network under uniform traffic when no reconfiguration process is triggered. The figure indicates the three network injection rates that are used in the simulations (Low, Medium, and High).

Figure 7.(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.

C. Comparison

In this section we compare the OSR-Lite protocol and the traditional static reconfiguration process (TSR). Figures 8.(a), 8.(b), and 8.(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 8.(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.

VII. Synthesis results

A. Area Comparison

The description of the OSR mechanism in [1] focuses on the protocol details and it lacks of practical implementation details. Thus we exploited the information provided in [1] 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 [10] 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

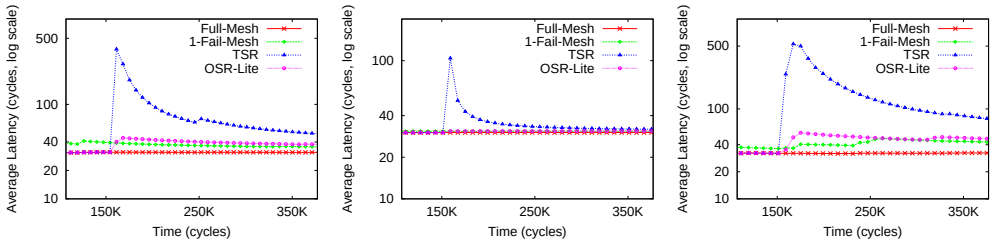


Fig. 8. Average message latency with (a) hotspot traffic and uniform traffic ((b) medium load and (c) high load).

low-power RAM with 64 words of 4 bits was the memory cut showing the lowest routing delay and area footprint.

Finally, Figure 9.(a) shows the area footprint of this latter solution (the *OSR SW*) with respect to a baseline switch and our proposed solution (*OSR-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 [6].

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-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 9.(b). This result is mainly due to the intrinsic

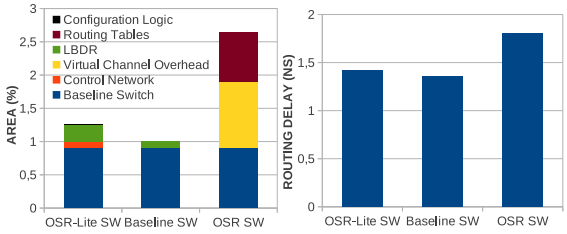


Fig. 9. 5x5 switch area and routing delay comparison.

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 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.

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 and by the project NaNoC (project label 248972).

Referencias

[1] O. Lysne, et al., "An efficient and deadlock-free network reconfiguration protocol," *TC*, 2008.
[2] D. Fick, et al., "Vicis: A reliable network for unreliable silicon," in *DAC*, 2009.
[3] C. Feng, et al., "A reconfigurable fault-tolerant deflection routing algorithm based on reinforcement learning for Network-on-Chip," in *NocArc)*, 2010.
[4] Z. Zhang, et al., "A reconfigurable routing algorithm for a fault-tolerant 2D-mesh Network-on-Chip," in *DAC*, 2008.
[5] K. Aisopos, et al., "Ariadne: Agnostic reconfiguration in a disconnected network environment," in *PACT*, 2011.
[6] A. Ghiribaldi, et al., "A complete self-testing and self-configuring noc infrastructure for cost-effective mpsocs," *TECS*, 2011.
[7] S. Stergiou, et al., "Xpipes lite: a synthesis oriented design library for networks on chips," in *DAC*, 2005.
[8] S. Rodrigo, et al., "Addressing manufacturing challenges with cost-efficient fault tolerant routing," in *NOCS*, 2010.
[9] A. Mejia, et al., "On the potentials of segment-based routing for nocs," in *ICPP*, 2008.
[10] F. Gilabert, et al., "Improved utilization of noc channel bandwidth by switch replication for cost-effective multi-processor systems-on-chip," in *NOCS*, 2010.