Cost-Efficient On-Chip Routing Implementations for CMP and MPSoC Systems

S. Rodrigo, J. Flich, A. Roca, S. Medardoni, D. Bertozzi, J. Camacho, F. Silla, and J. Duato

***Abstract*—The high-performance computing domain is enriching with the inclusion of networks-on-chip (NoCs) as a key component of many-core (CMPs or MPSoCs) architectures. NoCs face the communication scalability challenge while meeting tight power, area, and latency constraints. Designers must address new challenges that were not present before. Defective components, the enhancement of application-level parallelism, or power-aware techniques may break topology regularity, thus, efficient routing becomes a challenge. This paper presents universal logic-based distributed routing (uLBDR), an efficient logic-based mechanism that adapts to any irregular topology derived from 2-D meshes, instead of using routing tables. uLBDR requires a small set of configuration bits, thus being more practical than large routing tables implemented in memories. Several implementations of uLBDR are presented highlighting the tradeoff between routing cost and coverage. The alternatives span from the previously proposed LBDR approach (with 30% of coverage) to the uLBDR mechanism achieving full coverage. This comes with a small performance cost, thus exhibiting the tradeoff between fault tolerance and performance. Power consumption, area, and delay estimates are also provided highlighting the efficiency of the mechanism. To do this, different router models (one for CMPs and one for MPSoCs) have been designed as a proof concept.**

***Index Terms*—Fault-tolerance, logic design, networks-on-chip, routing.**

I. Introduction

MAIN MICROPROCESSOR manufacturers have shifted to chip multi-processors (CMPs) for their products. In CMPs many cores are put together in the same chip, and as technology advances more cores are included. Recently, Intel has announced a research chip with 48 cores, each being *x*86 compatible, under the Tera-Scale Computing Research Program [1]. Embedded systems are also shifting to multi-core solutions [multiprocessor system-on-chips (MPSoCs)]. A clear example of high-end SoCs are the products offered by Tilera [2] where multi-core chips provide support to a wide range of computing applications, including advanced networking, high-end digital multimedia, wireless infrastructure, and cloud computing. The most recent product by Tilera offers 100 cores in the same chip. It is well accepted in the community that processor performance will be increased in the following years by including more cores in the same die.

CMPs and high-end MPSoCs rely on an on-chip network (NoC) able to handle all the communication traffic between cores. Initial chip designs with few cores included buses and rings as the communication subsystem. Such solutions are well suited for chips with a small number of cores, like the Cell processor [3]. However, as the number of cores scales up, the bus structure suffers from lack of enough bandwidth. Thus, designers evolved to more scalable solutions.

On the other hand, in CMP and high-end MPSoCs, *tile design* is gaining momentum. The tile is designed in isolation and, once finished, the chip is built by replicating tiles. By doing this, the design effort to build a chip is drastically reduced. Tiled designs also advocate for regular network structures like 2-D meshes, as in Fig. 1. Typically, each tile incorporates a processor core, cache memory controllers, and a router to enable communication between tiles.

However, even if the design of a chip with a 2-D mesh network is correct, the on-chip network may face new challenges leading to non-regular heterogeneous topologies. Indeed, as technology advances, correct manufacturing becomes challenging and defective components will become frequent. As a consequence, a defective tile in the chip, if not addressed, will ruin the 2-D mesh structure of the network, leading to an irregular network that cannot be handled by the routing algorithm, thus rendering the chip useless. An example can be seen in Fig. 2, where the link between routers 6 and 7 has failed. Unless the routing mechanism is prepared to reroute a message around the defective link, it would be impossible for both routers to communicate.

Another challenge is the problem of not extracting enough parallelism from applications so to efficiently use tens and hundreds of cores in future chips. As a solution, the chip can be partitioned into multiple domains, each one running a

Manuscript received July 2, 2010; revised November 17, 2010; accepted November 17, 2010. Date of current version March 18, 2011. This work was supported by the Spanish MEC and MICINN, as well as by the European Commission FEDER funds, under Grants CSD2006-00046 and TIN2009-14475-C04. This work was supported in part by the Project NaNoC (Project Label 248972) which is funded by the European Commission within the Research Programme FP7. This paper was recommended by Associate Editor L. P. Carloni.

S. Rodrigo, J. Flich, A. Roca, J. Camacho, and F. Silla are with the Parallel Architectures Group, Technical University of Valencia, Valencia 46022, Spain (e-mail: srodrigo@gap.upv.es; jflich@disca.upv.es; anrope2@gap.upv.es; jecavil@gap.upv.es; fsilla@disca.upv.es).

S. Medardoni is with the Integrated Systems Laboratory of Minatec, Grenoble 38000, France (e-mail: mdrsmn@unife.it).

D. Bertozzi is with the Department of Engineering, University of Ferrara, Ferrara 44100, Italy (e-mail: brtdvd@unife.it).

J. Duato is with the Parallel Architectures Group, Technical University of Valencia, Valencia 46022, Spain, and also with the Simula Research Laboratory, Fornebu 1364, Oslo, Norway (e-mail: jduato@disca.upv.es).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2011.2119150

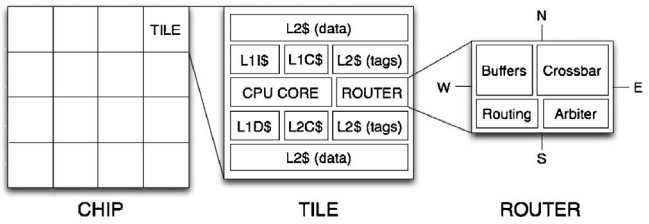

Fig. 1. Tile-based design.

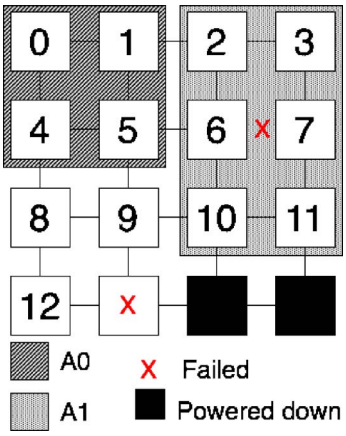

Fig. 2. Future challenges to be addressed in NoCs.

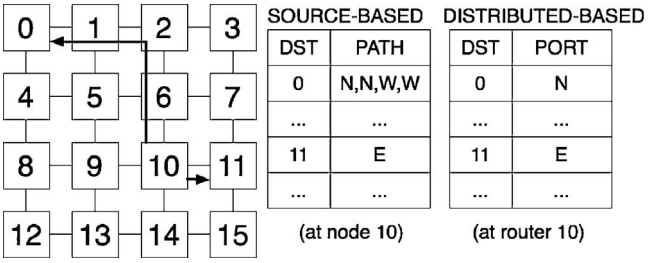

Fig. 3. Routing tables example.

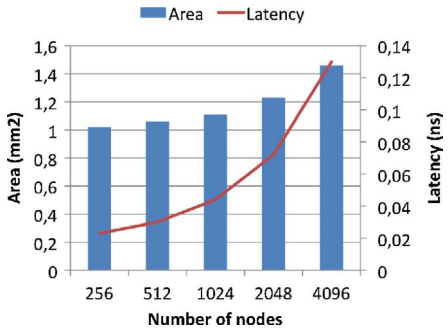

Fig. 4. Area and latency for memory macros/blocks as a function of the number of entries for a 90 nm technology node.

different application (or serving a different customer). In this scenario, and in order to fit as many applications as possible, the partition of the chip resources may lead to irregularly shaped domains. The way applications are usually mapped onto the chip uses a concept known as virtualization, where a real chip is partitioned into several smaller virtual chips. In Fig. 2, an example is shown. Two applications, $A0$ and $A1$, are already mapped on the chip using a different number of nodes. If a new application named $A2$ needs only three nodes to perform its task it could then be mapped on the nodes attached to routers 8, 9, and 12, grouping them into an irregular region. If such irregular mapping is not allowed, then $A2$ will need to wait the completion of $A0$ or $A1$. The objective is to minimize any fragmentation by allowing irregular patterns.

As a major challenge for chip design, there is a clear need to introduce efficient power saving methods. As the number of cores increases, probably many of the cores will remain unpowered (in sleep mode) most of the time, thus achieving large savings in power consumption. The same strategy should be applied to the on-chip network, which has been reported [4] to consume around 30% of the total chip power consumption. Powering routers off and on will lead to temporary irregularities in the topology. An example is present in Fig. 2. Tiles 14 and 15 are idle, and as a result of a power-aware technique implemented, they are powered down until requested by another computation task. This is translated into an irregular topology with the rest of nodes that are still active and need an effective routing layer.

Addressing all the previous challenges requires some effort at the on-chip network level, in particular, the most important thing to address is providing efficient support for irregular network topologies. Indeed, all the challenges mentioned above can be correctly addressed by deploying a routing mechanism able to deal with any of the derived topologies.

Current solutions that advocate for irregular topologies are based on routing tables. The basic mechanism consists either on implementing look-up tables at every end-node (when using source-based routing), or forwarding tables at every router (when using distributed-based routing). For every destination, there is a path or output port associated, respectively, to each kind of routing, that the message follows, or chooses, to arrive at its destination. Let us see an example in Fig. 3. In source-based routing, if a message had node 10 as a source and node 0 as destination, the path to follow, $N - N - W - W$ (north, north, west, west), would be coded on the message header. In distributed routing, there is no path coded at the message header, just the destination, so on every router, the routing table is accessed to see the output port the message has to take on its way to its destination. In the example, at router 10, for destination 0, the output port chosen is N (north).

The main advantage of table-based routing is that any topology and any routing algorithm can be used. However, as routing tables are implemented with memories, they do not scale in terms of latency, power consumption, and area, thus being impractical for large NoCs [5]. Indeed, a routing table with as many entries as the number of nodes and input ports are needed in the worst case, with the possible addition that every entry needs to store different output ports returned by the routing algorithm. Hence, the cost of this implementation is $N \times d \times d$, where N is the number of nodes and d is the number of ports. An example of poor scalability of tables can be seen in Fig. 4 which shows the synthesis of memory macros with 90 nm technology obtained with Memaker [6]. As can be seen in the figure, there is a direct increase of area with respect to the number of entries (table size). Latency results

have the same trend. In addition, the increase on router delay could change the critical path of the router leading to lower performance.

On the other hand, dimension-order routing (DOR, also called XY) [7] is based on the concept of minimal-path routing and is very simple: first route the message in one dimension (X) and then, in the other dimension (Y), when related to a 2-D mesh. DOR is an efficient solution (logic-based implementation) in terms of area, power, and delay overheads, but as opposed to a routing tables implementation, it lacks the associated flexibility and cannot support any of the challenges mentioned before.

So, it is imperative for current and future designs to face these challenges, benefiting from the flexibility of routing tables, while achieving important savings in three critical key aspects that influence every design at the nano-scale domain: area, latency (critical path) and power. Indeed, effective and efficient designs that accomplish to get overall savings in any of the aspects mentioned before will be a great contribution to the NoC field. Here we take on such a challenge. But, rather than addressing completely irregular topologies (more suitable for low-end MPSoC systems) we focus on irregular topologies derived from an initial 2-D mesh structure, where the following properties still remain: 1) a router is connected to at most four routers, each one in a different direction and dimension, and 2) a hop along a valid direction and dimension will not cross more than one row or column. If we assume that the previous two properties are guaranteed by the final topology, then the solutions to provide efficient routing in those topologies get simplified. The aim is to provide a step further and deliver a logic-based routing mechanism able to cover all the possible cases derived from a 2-D mesh, that is, with full support for any failure/virtualization/domain/region configuration. The mechanism, referred to as universal logic-based distributed routing (uLBDR), is an evolution of LBDR [5].

The remainder of this paper is organized as follows. In Section II, we describe the related work. In Section III, we describe the uLBDR mechanism. In Section IV, we describe two router implementations to support the uLBDR mechanism. Then, in Section V we evaluate uLBDR. Finally, in Section VI we present the conclusion and provide future directions.

II. Related Work

There are solutions from the off-chip network domain that could be applied to the NoC field. All these mechanisms do not fit properly in NoCs unless they are thoroughly redesigned. As an example, proposals for TCP/IP protocols [8] are not suitable for NoCs as they rely on message dropping, and would severely affect network performance. There are also techniques used in large parallel systems like the Blue Gene/L system [9] where entire sets of healthy nodes (lamb nodes) are switched off to keep topology and routing algorithm unchanged. Other mechanisms, focused on routing optimization [10], require the use of virtual channels (up to five in some cases) but they do not achieve 100% coverage practically. Also, these mechanisms rely on adaptive routing, and the network must deal with out-of-order delivery issues, a feature that could be difficult to implement in NoCs. Other examples are Interval Routing [11] and extensions [12], group sets of destinations requesting the same port, and are an initial attempt to compress the routing table. However, these techniques are not easily applicable to irregular networks.

Street-Sign Routing [13], a source-based routing implementation, compresses the message header so to minimize the impact on network bandwidth. Street-Sign Routing includes only the router ID of the next turn and the direction of the turn in the message header. Although message header is reduced it still consumes bandwidth. In addition, a table including the paths for every destination is required at every end-node.

Regarding distributed routing solutions, first we focus on region-based routing (RBR) [14] (and a similar proposal [15]). At each router RBR groups into a region different destinations that can be reached through a given output port. The main drawback of such mechanism is that, even with 16 regions defined, it still does not achieve 100% coverage and induces a long critical path [16].

Default-backup path [17] tries to keep healthy processing elements when the attached router fails. It consists of adding redundant wiring and buffers that connect output and input ports directly. However, it does not address routing in irregular topologies.

Adaptive stochastic routing [18] is a recently proposed algorithm (an improvement from the cooperative on-demand secure route algorithm [19]) that relies on a self-learning method to handle failures by assigning confidence fields to output ports for different tasks (or applications) running in the system. Thus, it requires a routing table at each router, suffering from the same scalability problems and routing costs related to routing tables.

In [20], the authors proposed an architecture based on deflection routing that attempts to detect fault errors by adding cyclic redundancy check modules at input and output ports for crossbar faults, and security codes for link faults with the support of routing matrices, one for each type of fault. The link fault matrix is $n \times n$, n being the dimension of the mesh. It uses a variant of deflection routing called delta XY with weighted priority. Deflection (or reflection) can lead to potential starvation solved with message dropping, thus impacting performance. Also, routing matrices exhibit poor scalability as the number of destinations increases.

Another recent proposal, whose objective is to minimize the size of routing tables, either at end-nodes or at routers, is described in [21]. Three techniques are proposed: turn-table, XY deviation table, and source routing deviation points. All of them consist of a table created by different routing algorithms to handle irregular cases in combination with routing strategies like XY (combined with YX), source routing, or the *don't turn* technique (meaning that a message must not change direction when traversing the router unless indicated). Deadlock-freedom, however, is not assured in all these strategies (e.g., when changing from XY to YX) unless virtual channels are used.

In [22], a compendium of state-of-the-art look-up tables implementations for routing purposes is proposed, shifting from

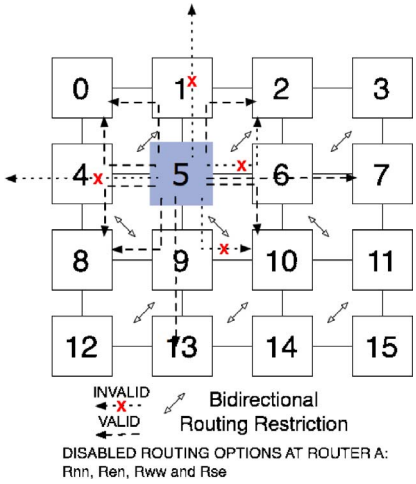

Fig. 5. SR as a set of routing restrictions and routing bits at router 5.

fully hardwired to partially or fully configurable solutions, depending on the degree of flexibility.

Novel implementations based on dimension-ordered routing, like FDOR [23], arise to provide coverage on irregular topologies. FDOR is based on the idea of dividing the dimensional mesh irregular topology into regular sub-meshes, a core mesh and one or more flask meshes. Depending on the division, at the core mesh, messages are routed with XY routing and on the flank meshes with YX, or vice versa. One bit per router is needed to configure XY or YX routing. FDOR provides a cheap and efficient routing solution to offer coverage on a set of irregular topologies that are restricted by certain conditions, but it does not offer full coverage.

As a summary of the related work on fault-tolerant unicast routing for NoCs, the proposals use routing tables (either at sources or at destinations) and/or rely on an excessive number of resources (virtual channels) to avoid the deadlock problem. Also, none of the solutions (except when using tables) is able to provide full coverage (all the possible failure cases) for a 2-D mesh. Thus, existing solutions are very expensive in terms of routing delay and/or required silicon area.

III. uLBDR Description

The description of uLBDR will be guided as an evolution from the basic mechanism, previously proposed, and with low coverage, to the most enhanced version, with full coverage. In each, example cases will be described so to motivate the need for the next version. As we focus on 2-D meshes we will refer to each router port by its direction, being N, E, W, and S for north, east, west, and south, respectively.

A. The Foundations

uLBDR is a mechanism that allows a compact and efficient implementation of routing algorithms in NoCs. Indeed, different routing algorithms can be implemented when using uLBDR. Therefore, uLBDR should not be seen as a routing algorithm by itself, rather as an implementation of routing algorithms.

In order to enable an efficient implementation of the routing algorithm, uLBDR relies on a compact representation of the routing algorithm, thus requiring a small silicon area and achieving low latency while providing flexibility, and this fact makes uLBDR different from other approaches. More precisely, the compact representation of the routing algorithm implemented with uLBDR is based on a set of routing restrictions. A routing restriction is defined among two consecutive network links if no message is allowed to cross them in a given order along its path to destination. Enforcing certain routing restrictions is required to ensure deadlock freedom.

Fig. 5 shows an example of the segment-based routing algorithm (SR) [24] represented by its set of routing restrictions. SR is a topology-agnostic routing algorithm since it can be implemented on any topology. Therefore, it is suitable for providing fault tolerance. With SR, many different instances of the routing algorithm can be obtained by simply placing routing restrictions in different locations (always ensuring deadlock freedom and connectivity). An example of valid/invalid paths is shown in the figure.

Representing a routing algorithm by a set of routing restrictions allows an efficient implementation of the algorithm. This is based on two sets of configuration bits: routing bits (R_{xy}), which define the set of routing restrictions, and connectivity bits (C_x), which define the availability of links. Those bits are computed for a given topology and the routing algorithm represented by its routing restrictions and are distributed over the routers in the network before starting normal operation.

Routing bits in a router represent the routing restrictions found at the neighbor routers. As we have four output ports labeled as N, E, W, and S, the R_{xy} bit at a given router indicates whether a message is allowed (by the routing algorithm) to leave the router through output port x and at the next router to take output y. For example, in Fig. 5, R_{ne} bit at router 5 is set because a message can be routed at router 5 through the N output port and at the router 1 through the E output port. R_{en} is, however, reset due to the existing routing restriction. With R_{xy} bits a 1-hop visibility of the allowed routing restrictions is provided. Note that four of those routing bits (R_{nn}, R_{ee}, R_{ww}, and R_{ss}) indicate whether a message can advance along the same direction or, on the contrary, either should take a non-minimal path, or has reached the mesh border.

A C_x bit defines the connectivity at the x output port. For example, if the C_n bit is set it means that there is a neighbor router connected through the N port. C_x bits act as a filter for connectivity and topology definition.

Routing and connectivity bits are a simple yet powerful mechanism to route messages in combination with the logic described later. Note that the reconfiguration of these bits is out the scope of this paper and is left for future work.

B. First Mechanism: LBDR

uLBDR is based on LBDR. LBDR required only eight routing bits (R_{ne}, R_{nw}, R_{en}, R_{es}, R_{wn}, R_{ws}, R_{se}, R_{sw}) and four connectivity bits (C_n, C_e, C_w, C_s) per router. In order to support non-minimal paths, the original LBDR module has been extended as shown in Fig. 6.

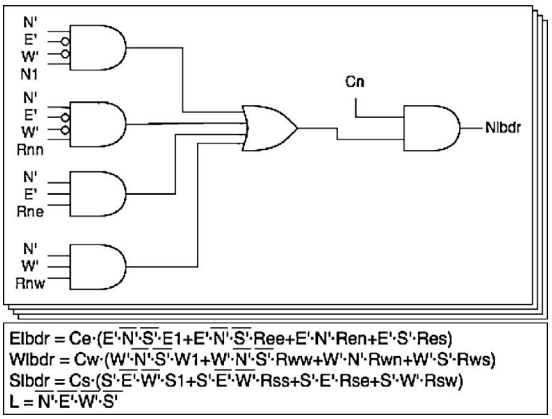

Fig. 6. LBDR logic as the core module in uLBDR.

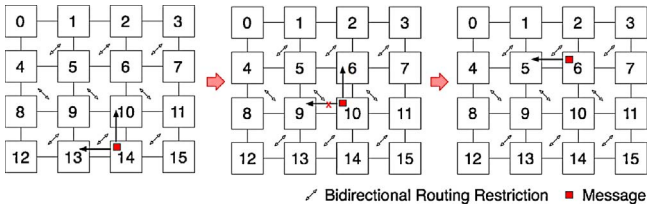

Fig. 7. Example of routing decisions in LBDR.

LBDR is based on the relative position in the mesh of the destination node and the current router. In order to compare both, LBDR uses a COMPARATOR module (see Fig. 11), which generates eight control signals. On one hand, signals N', E', W' and S' indicate the relative position of the final node. For example, in Fig. 5, if the current router is 14 and our destination is router 11, signals N' and E' would be activated. On the other hand, signals $N1$, $E1$, $W1$, and $S1$ indicate whether the final node is one hop away in each direction. In the case of Fig. 5 again, at router 14, $N1$ signal would be set if our destination is router 10. With these control signals, and using the R_{xy} and C_x bits, the LBDR module computes an initial set of routing decisions (Fig. 6).

Fig. 7 shows an example. Router 14 wants to send a message to router 5. At 14, signals N' and W' are activated as destination is on the NW quadrant. As N' and W' are active, and R_{nw} is set, and there is connectivity to the north (C_n is set), N becomes a valid choice. W port can also be valid, as R_{wn} is set, and there is connectivity to the west. Let us assume that the arbiter chooses N finally. At the next hop, at router 10, N' and W' are active, again. LBDR provides N direction as a valid choice, but now W is discarded, as R_{wn} bit is reset at router 10, which represents the routing restriction at router 9. Message is sent north to router 6. In this case, only W' signal is active, and after the routing process, W is the only routing option. The message is finally delivered to router 5, where it will be forwarded to the local port.

C. Second Mechanism: $LBDR_{dr}$

The LBDR mechanism relies on the use of minimal paths for every source-destination pair. This leads, as we will see in the evaluation, to a low percentage of topologies being supported. Indeed, in a 2-D mesh is easy to imagine sets of failed links/routers that require non-minimal paths for some source-destination pairs. A single hole in the network is a clear example. Fig. 8 shows a topology not supported by LBDR. The path from A to B is non-minimal. At router A, the possible directions to reach B are N and E, however, both links are missing, and therefore no possible way out.

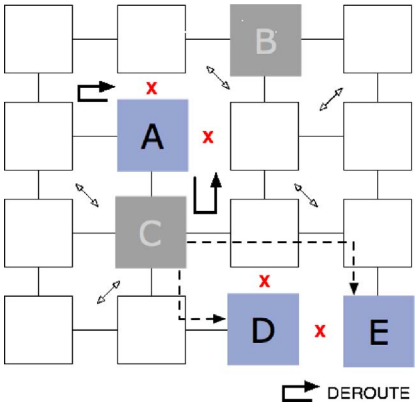

Fig. 8. Example of deroute and fork operations.

We need to provide non-minimal support in an efficient way (i.e., add the minimal set of logic resources to handle this). This motivates for the first extension for LBDR, the DEROUTES module, or $LBDR_{dr}$. Fig. 9 shows the additional logic. In particular, for every input port of the router a deroute option is provided. A set of two bits ($dr0$ and $dr1$) encode the deroute option that can be N, E, W, or S. Deroute options are computed in accordance to the routing algorithm, as these options must ensure deadlock-freedom property. The computation of these options are performed by an exhaustive search algorithm that tests all the paths in a recursive way for every source-destination pair. This algorithm, performed offline, tries all possible deroute options (in case LBDR offers no outputs), one per output port available but avoiding U turns and crossing routing restrictions. Whenever the previous LBDR logic is unable to provide a valid output port (NOR gate with four inputs) the deroute option is taken into account. The logic is replicated for every input port, therefore the deroute option used is the one associated with the corresponding input port.

Alternatively, the deroute option can be designed for the entire router, instead of having a deroute option per input port. However, this reduces flexibility and leads to non-supported topologies (this alternative will be analyzed later, called $LBDR_{1dr}$). Fig. 8 shows an example, where two different deroute options are required for two different input ports at router A. If going N, and the message comes from input port S, then a deroute is set to W. On the other hand, if the message is coming from W, and the intention is to go E, then a deroute is set to S. Note that the deroute option needs to be computed in accordance with the routing algorithm to avoid deadlocks. In Fig. 8, a deroute option at input port W at router B cannot be set to S as it would let messages crossing a routing restriction.

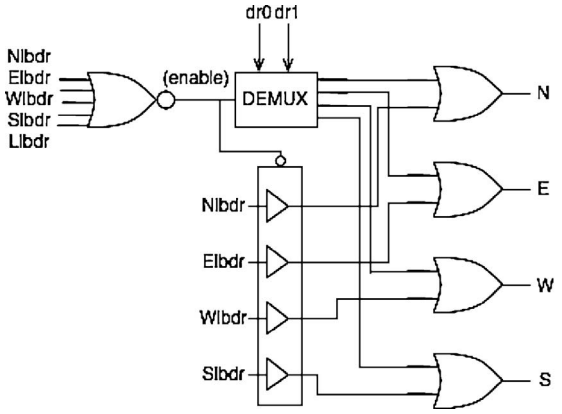

Fig. 9. Deroutes logic.

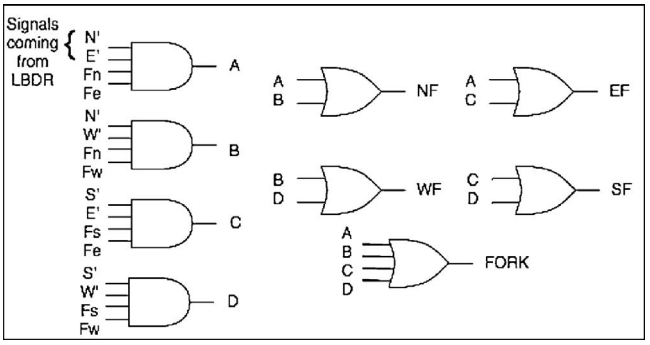

Fig. 10. Fork logic.

D. Third Mechanism: uLBDR

$LBDR_{dr}$ enhances greatly the coverage. However, there are subtle cases that are still not covered by $LBDR_{dr}$. Fig. 8 shows an example. The problem comes by the fact that for some destinations located at the same quadrant, at router C the routing engine should provide one port (S) for some destinations (router D) and another port (E) for other destinations (router E). As LBDR (or $LBDR_{dr}$) works in quadrants, there is no way to indicate the router which option should be given to the message.

To solve this, we introduce the FORKS module. The objective is to fork (replicate) the message through two output ports. The logic is shown in Fig. 10. As shown, it relies on four additional configuration bits (fork bits): F_n, F_e, F_w, and F_s. These bits are set to reflect the output ports that must be used to fork a packet. Whenever a packet comes and its destination is in the same quadrant defined by the fork bits, then the packet needs to be forked, e.g., packet is replicated through two output ports. The $FORK$ signal is set and forwarded to the arbiter to distinguish between two possible valid routing options that could come from the LBDR module and a fork operation. F_n, F_e, F_w, and F_s bits are set appropriately. If due to the irregularity in the network topology considered, at least one pair of end nodes could not communicate through one path, even in the presence of deroutes, the fork operations are considered. To do so, the bit computation algorithm tests any possible fork operation at routers where no deroute succeeded, again discarding any possible choice that leads to cross a routing restriction. Fork options are computed, offline, in a similar way as deroute options.

The fork operation leads, however, to important changes in the router design. The router arbiter needs to be changed to allow one message to compete for more than one output port at the same time. There are two alternatives. In the first one, the arbiter may consider a request from a message to two output ports as an indivisible request, granting or denying access to both outputs at the same time. This leads to a simpler design of the buffering at the input port, as only one read pointer is needed. In the second one, the arbiter may grant or deny access to one port regardless of the other port. This leads to a complex input buffering, as message forwarding may be shifted for both outputs, thus each requiring a read pointer. We assume the first option because of its simplicity (notice fork operations will be used in few cases).

Deadlock may occur in wormhole switching, since two fork messages may compete for the same set of resources. Although the routing algorithm used is deadlock-free, performing fork actions leads to deadlock. Imagine that a message m_1 gets the output port N at router X and requests output port E at router Y. However, message m_2 gets output port E at router Y and requests output port N at router X. None of the messages will advance since the input buffers will fill and the output ports will never be released.

The easiest solution is the use of virtual cut-through (VCT) switching, thus ensuring a packet[1] will fit always in a buffer. Thus, the output ports in the previous example will be released (the packet has been forwarded entirely) and the requests for the output port will be granted. Other options rely on performing flit-level VCT [25] and wormhole switching between routers. Basically, those solutions label each flit with identifiers so flits from different messages can be mixed in the same buffer. Internal tables are, however, required to keep flit identifiers. We opt for the first solution (VCT switching). Although VCT is seen as demanding much buffer space at routers, a careful design of the router can minimize this impact.

When using forks, one of the replica will reach the final destination while the other needs to be removed from the network. This will be easily achieved by silently destroying the replica packet to be removed at a router when gets no output direction and it is not at its destination (remember that forks are computed before normal operation so in this case, the removal is confirmed). In Section V we show area, power, and latency results for two routers designed for uLBDR, thus showing upfront the real impact of such router changes. Note also the buffer requirements in a VCT switch do not depend on the implementation of FORK operations. Indeed, the flow control is applied on a per-buffer basis, regardless of packets are forked or not.

The FORK extension, together with the DEROUTES extension and the LBDR module, forms the uLBDR mechanism. The uLBDR mechanism is shown in Fig. 11.

[1]In VCT we use the term packet since a message may be packetized.

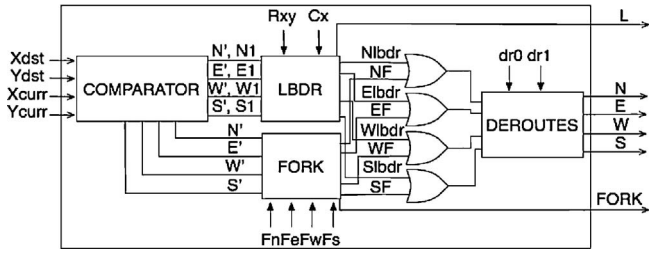

Fig. 11. uLBDR mechanism.

IV. Routers Description

In this section, we provide a brief description of the routers used for the evaluation. Two routers are provided to address two different areas: MPSoCs where unpipelined routers are usually deployed, and CMPs where pipelined routers are common. In both cases, an initial wormhole router design has been evolved to allow VCT switching (required for uLBDR). The most important changes are: 1) packetization of messages at the network interfaces; 2) buffering sizing to packet sizes; and 3) changes in flow control. To allow the replication of packets we need also to: 1) change the arbiter logic, and 2) remove stale copies of packets (being forked).

A. MPSoC Router Design

Typically, NoC building blocks for use in MPSoCs target lower operating speeds with respect to CMPs and are generally unpipelined [26]. The reference component that we consider to assess the feasibility of uLBDR is an input buffered router implementing wormhole switching (Fig. 12). Size of the input buffer is tunable and set to four slots. In 1 cycle, a flit covers the distance between two consecutive input buffers of connected routers through the inter-router link. The switch traversal inside the router is controlled by a modular arbiter round-robin arbiter (one per output port). A lightweight stall/go flow control policy is implemented. 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"). This latter signal is indicated as *flow control* in Fig. 12. The router design is implemented in a 65 nm industrial technology library.

$LBDR$ and $LBDR_{dr}$ mechanisms are implemented in a similar way, from an architecture viewpoint. The head flit contains destination coordinates which are read, after storage in the input buffer, by the routing logic. The output signals elaborated by the LBDR/$LBDR_{dr}$ module represent *match* signals sent to the arbiters. A *match* signal indicates that the packet from a given input port requires a specific output port. It is interesting to note that the LBDR/$LBDR_{dr}$ routing logic enables to preserve the modular design style of the router architecture (one routing module per input port).

This router was evolved to VCT switching to support fork operations (thus uLBDR). The signals used and the architecture schematic are the same of Fig. 12, just the meaning of flow control signals and the arbiter behavior change. First of all, we had to evolve the basic stall/go flow control protocol

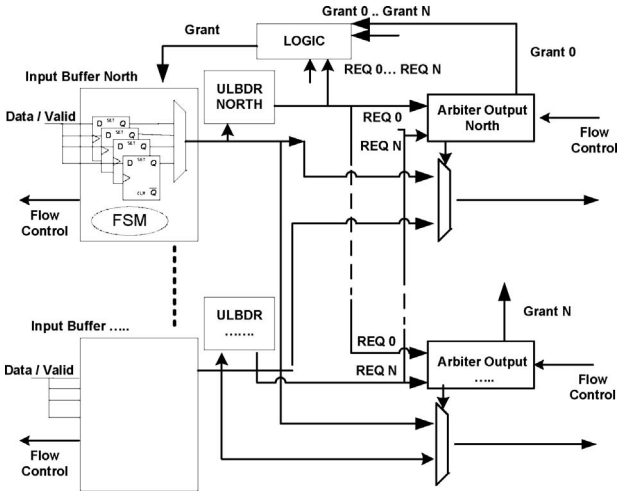

Fig. 12. MPSoC router schematic.

to credit-based flow control. In fact, stall/go would have been acceptable only in case all packets were of the same length. If packets exhibit variable length (e.g., reads versus writes, variable number of write/read burst beats, and so on), then the router arbiter needs to know the number of available slots in the downstream buffer before granting a new packet head. Therefore, we now use the *flowcontrol* signals in Fig. 12 as *credits*. An input buffer asserts a credit high when it has a grant from the arbiter AND it has valid flits to send.

The arbiter behavior had to be modified as well. A port arbiter (say for the N output port) performs round-robin arbitration among all inputs with *valid* asserted and presenting a *headflit*. Say that input N is the winner. Then, the arbiter compares its counter value (denoting the number of free slots in the downstream input buffer) with the packet length from the *Ninputport*. If it is larger, then *grant* is asserted enabling switch traversal to all the winning packet flits. If there is no space downstream for the entire packet, the grant is kept low.

In uLBDR, packets can be forked through two output ports. When this happens, the LBDR logic asserts two match signals heading to two different port arbiters. When both of them assert their *grant* signals, a unique *grant* is sent to the requesting input buffer, as illustrated in Fig. 12. One of the packets will reach destination. The other one will reach a router where the LBDR logic will not provide a valid match signal. In that situation, the *grant* signal is set by default to asserted, thus the packet will be forwarded to the crossbar which is not configured for the input port, thus the packet will be filtered. The input buffer is not aware that no arbitration has been performed for the forked packet, and the *grant* signal is kept asserted, thus will also correctly generate a *credit* to the upstream router, since buffer slots are cleared. This is the way the misrouted forked packet is silently discarded.

B. CMP Router Design

The CMP router is a pipelined input-buffered wormhole router with five stages: input buffer (IB), routing (RT), switch allocator (SW), crossbar (XB), and link traversal (LT). We used a simple router with no virtual channels and

five input/output ports. The input buffer size is set to four flits. The RT module has been implemented to support XY, $LBDR$, $LBDR_{dr}$, and $uLBDR$ routing implementations and the Stall/Go flow control. Finally, the SW module has been designed with a round-robin arbiter as in [27]. The router has been implemented using the 45 nm technology open source Nangate [28].

In order to adapt the basic CMP router to VCT we have performed the following changes. First, buffers at routers have been set to maximum packet size, in our case to four flits. In addition, packetization is performed at the interface nodes when required (notice that this is also needed for the MPSoC router, probably with a different packet size). Message sizes in CMPs (using a coherence protocol) are known beforehand. Usually, a short message contains a memory address and a coherence command and a long message also includes the cache line. In our case (in the evaluation), short messages are set to 8 bytes and long messages to 72 bytes (cache line size is 64 bytes). Assuming 8-byte flits, short messages are not packetized and long messages are packetized in 11 packets (taking into account packet header is replicated).

To efficiently forward packets in VCT we need to change the flow control mechanism (as in MPSoC router). In the CMP case where packet sizes are known, we opted for the Stall/&Go flow control at the packet level. That is, a stall or go signal is asserted per packet. Notice that we assume links with one cycle delay, thus round trip time is set to three cycles. Buffers of four flits are thus enough to avoid introducing bubbles. However, for messages with sizes lower than packet size (and round-trip time; e.g., one-flit packets) bubbles between packets are generated. To avoid bubbles we decided to pad short packets to four-flit packets. Obviously, this may affect performance. In the next section, we analyze the impact of padding and packetization.

SW is the most critical stage in our design. Thus, the arbiter modifications applied in the MPSoC arbiter are not affordable for the CMP router. To solve this we have implemented the arbiter shown in Fig. 13. This arbiter is the same used in the WH design but it adds a new module performed in parallel. This module arbitrates between fork requests. The grant signals of this module enable (or disable) the non-fork grants. Higher priority is given to fork requests. By doing this minimum impact on the SW latency is expected. Stale packets (generated by fork operations) are silently discarded in the same way as in the MPSoC router.

V. Evaluation

In this section, we provide several evaluation results to assess the impact of the different implementations. First, we provide a coverage analysis for each solution. Then, we analyze the overhead and power consumption of the mechanisms in the router designs, including the overhead for the VCT router support. Finally, we provide performance results with synthetic and real traffic (by running real applications in a full system simulator environment). We also test, in particular, if the packetization requirement by uLBDR affects final application's execution time.

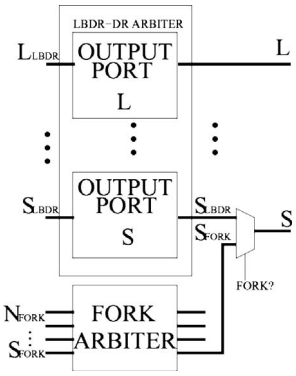

Fig. 13. New arbiter for the CMP router with fork requests.

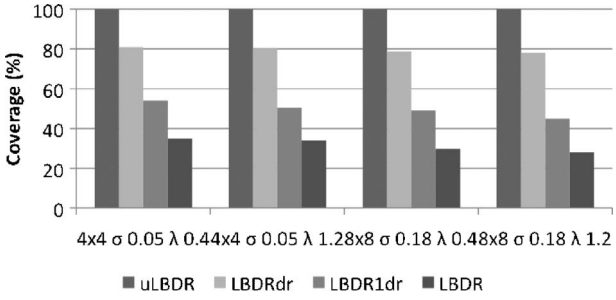

Fig. 14. Coverage of different mechanisms.

A. Coverage Analysis

In this section, we evaluate the coverage provided by different versions of the mechanism, from the original LBDR to the full uLBDR mechanism. Coverage is measured as the percentage of topologies supported from a pool of topologies. A topology is considered supported if every node in the network reaches all possible destinations.

A set of topologies derived from the link variability analysis provided in [16] has been used. In particular, different NoC operating frequency thresholds were set and links not reaching those thresholds (due to variability effects) were labelled as faulty. Chips were modeled on a real 65 nm implementation NoC layout where all cores are identical, and their size is $1\,\text{mm}^2$. Two different configurations were used, 4×4 and 8×8 NoCs with different values of spatial correlation (λ 0.4 and λ 1.2) and variance (σ 0.05 and σ 0.18). In total, 1423 topologies have been evaluated.

The evaluation comprehends four different scenarios, LBDR, LBDR with 1 global deroute ($LBDR_{1dr}$), $LBDR_{dr}$ and uLBDR. In Fig. 14, results show how the addition of the two enhancements, deroutes and forks, affects significantly the coverage. Although having one global deroute per router helps to increase coverage by 50%, further benefits are obtained for deroutes per each input port (5 per router), as coverage further increases to 80%. Finally, the fork mechanism is the one that guarantees full coverage.

B. MPSoC Router Overhead

We synthesized the MPSoC router with a 65 nm STMicroelectronics technology library and Synopsys Physical Compiler. Routers with all routing mechanisms (LBDR, $LBDR_{dr}$,

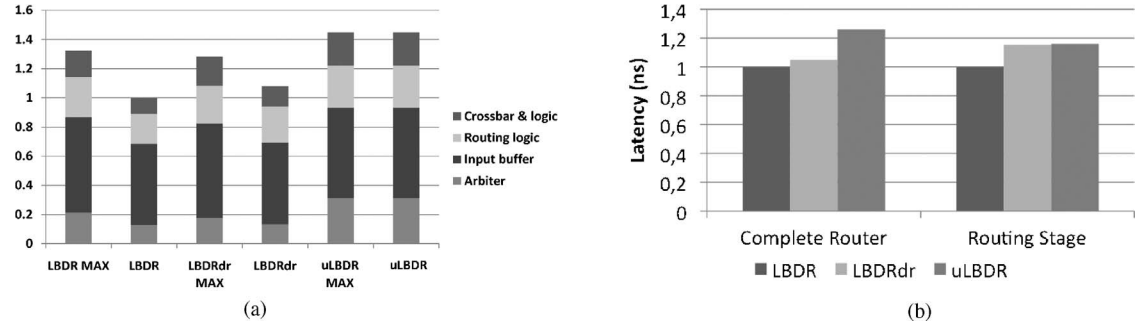

Fig. 15. MPSoC router, normalized results. (a) Area. (b) Latency.

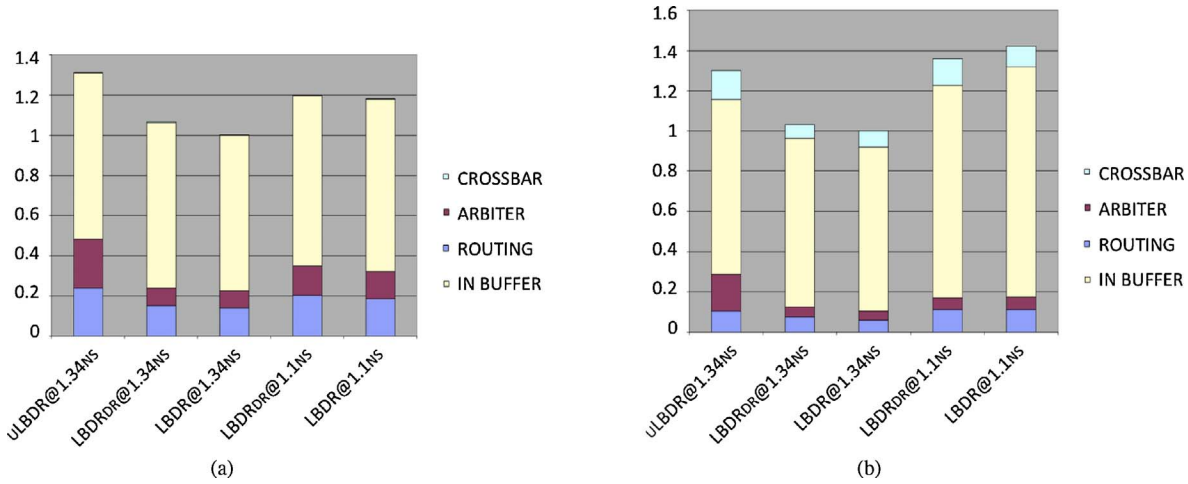

Fig. 16. MPSoC router power analysis, normalized results. (a) Idle power. (b) Active power.

and uLBDR) were synthesized both for maximum performance and for the same target speed (that of the slowest architecture, i.e., uLBDR). All routers implement the same amount of buffering (4 slots). The choice of a specific routing mechanism affects the maximum achievable speed by each router: 1 GHz for LBDR, 950 MHz for $LBDR_{dr}$, and 750 MHz for uLBDR.

Post-synthesis area results for the routers are illustrated in Fig. 15(a). By looking at the maximum performance figures, uLBDR is about 10% larger than LBDR, clearly due to the more complex port arbiters and to their need to handle true credit-based flow control. To make this relatively more complex circuit faster, the synthesis tool tried to speed up the crossbar at the cost of further increased area. We also observe that LBDR and $LBDR_{dr}$ feature approximately the same maximum area, except for hardly controllable specific optimizations that the synthesis tool applies to the two netlists. The take-away message here is that the logic complexity of these two routing mechanisms is pretty much equivalent.

When the three routers under test were re-synthesized to meet the performance of the slowest one, uLBDR, then of course the relaxation of the delay constraint for LBDR and $LBDR_{dr}$ allowed the synthesis tool to infer a more area efficient gate level netlist for them. As a consequence, the area efficiency gap with uLBDR became as large as 44.7%, the absolute worst-case.

We conclude that whenever the three routing schemes are employed at their maximum performance, the area gap is not significant (around 10%) while tremendously gaining in fault coverage. When the target speed is affordable for each of them and close to that of the slowest scheme, then the choice between the routers becomes a true area-coverage tradeoff decision. When the target frequency is very low [a few hundred MHz, not shown in Fig. 15(a)], then the gate level netlists of the three schemes can be almost equally optimized, resulting in almost the same area while keeping the coverage differences.

Fig. 15(b) shows the latency breakdown of the three mechanisms, LBDR, $LBDR_{dr}$, and uLBDR. As can be seen, uLBDR introduces 25% more delay compared to the basic system in the critical path of the router, while the gap of $LBDR_{dr}$ is around 3%. But the routing stage is not setting the critical path as shown in the figure. The latency for the routing stage for $LBDR_{dr}$ and uLBDR is almost the same. The latency gap introduced in uLBDR for the total router is due to the changes to support VCT switching and this is reflected in other router elements like the input buffers.

Power analysis has also been performed for the different routing implementations. Fig. 16(a) reports normalized total idle power of the routers with their respective breakdowns. Routers with LBDR and $LBDR_{dr}$ have been implemented and characterized twice: in the first variant they are synthesized at their common maximum speed (1.1 ns clock period), while in the other variant they are synthesized at the maximum speed achievable by uLBDR, thus resulting into a fair power comparison.

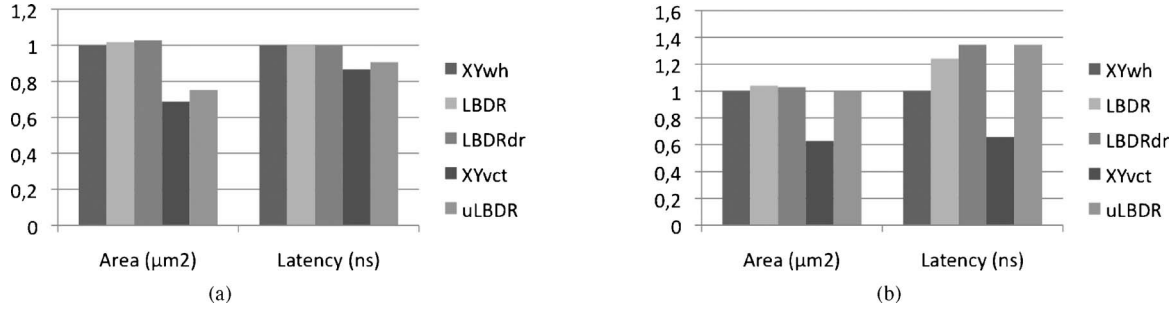

Fig. 17. CMP router, normalized results. (a) Complete router. (b) RT module.

Idle power is dominated in all cases by the buffer contribution, since clock gating has not been applied. Power of the routing block inside all switches is relevant, and even dominant with respect to the arbitration power. This is due not to the combinational logic computing the target output port, but rather to the registers storing the values of LBDR configuration bits. This is the price to pay to keep these bits potentially reprogrammable. We notice however that a clock gating technique here would be very effective, since these bits are not changed until a failure occurs in the network and routing for this latter has to be reconfigured.

$LBDR_{dr}$ and LBDR have an almost equivalent power consumption, due to the very similar switch structure with just minor relative modifications, which is consistently lower than that for uLBDR. This latter features more complex arbitration logic (with a larger number of state registers), routing logic, and configuration registers, thus giving rise to an almost 30% idle power overhead.

Fig. 16(b) reports total router power in active mode, as derived through the average power computing capability of Synopsys PrimeTime PX (50% switching activity). First of all, the crossbar contribution to total power now becomes evident as an effect of the switching activity. In spite of this, the input buffer is still by far the largest contributor. Interestingly, the gap between LBDR and $LBDR_{dr}$ powers at maximum performance and relaxed performance is now larger, due to the increased contribution of the combinational logic. The gap is such that by comparing only power of the routers synthesized at their respective maximum performance, uLBDR turns out to be the least consuming scheme. Again, when aiming at the same target speed, uLBDR proves almost 30% more power hungry than the other schemes.

C. CMP Router Overhead

Fig. 17(a) summarizes frequency and area results of the CMP router for different switching techniques and routing mechanisms. The first thing to highlight is the improvement of both area and frequency of the VCT router. The reason for this improvement is due to the use of buffers with the same size of packets. This has simplified the IB stage because in VCT the flit header of every packet is mapped always into the same buffer slot, thus simplifying read logic. Also, the logic to keep track the number of mapped flits in the buffer has also been simplified. Due to the per-packet flow control only a control signal is required. Although such simplifications can also be made in WH, bubbles would be introduced (known as atomic buffer allocation).

There is no difference in the operating frequency when using either XY, LBDR, or $LBDR_{dr}$, and only a marginal increase in area (differences fall within the uncertainties of the synthesis optimization process). This is due to the RT stage not setting the maximum frequency of the router. uLBDR experiences, however, a small impact in performance and area. This performance degradation is due to the overhead added to the SW stage.

Fig. 17(b) shows the area overhead and frequencies of the different routing modules, thus not considering the entire router. There are significative differences between the XY module for WH and for VCT. These differences are due to the different flow control mechanism used in both versions. Also, the different input buffer design affects the routing module. On the other hand, LBDR and $LBDR_{dr}$ mechanisms have a small impact on area but a large one in frequency. Also, the complexity of uLBDR has a large impact on both area and frequency. However, remember that this module is not the one setting the router frequency.

Power analysis for the CMP router architectures has also been performed: Wormhole routers with LBDR and $LBDR_{dr}$ routing mechanisms and the VCT router with uLBDR routing mechanism. Fig. 18(a) reports normalized total idle power of the routers. As same as the MPSoC counterparts, routers with LBDR and $LBDR_{dr}$ have been implemented classified in two variants: in the first one they are synthesized at their common maximum speed (in this case, 0.68 ns clock period), while in the second one they are synthesized at the maximum speed achievable by uLBDR (for CMP version, 0.75 ns clock period), for a fair power comparison.

Idle power is dominated in all cases by the buffer contribution, since clock gating has not been applied. Power of the routing block inside all switches is relevant, but as can be seen in Fig. 18(a) the routing module presents the lowest idle power consumption. As mentioned before, the main element that causes the power consumption in the routing module is the registers that store the values of LBDR configuration bits (routing, connectivity, deroute, and fork bits). Identically to the MPSoC router, $LBDR_{dr}$ and LBDR have an almost equivalent power consumption, due to the very similar switch structure with just minor relative modifications, which is consistently lower than that for uLBDR. Note that, when using uLBDR the routing module and the arbiter increase their power consumption. This increment is almost 20%.

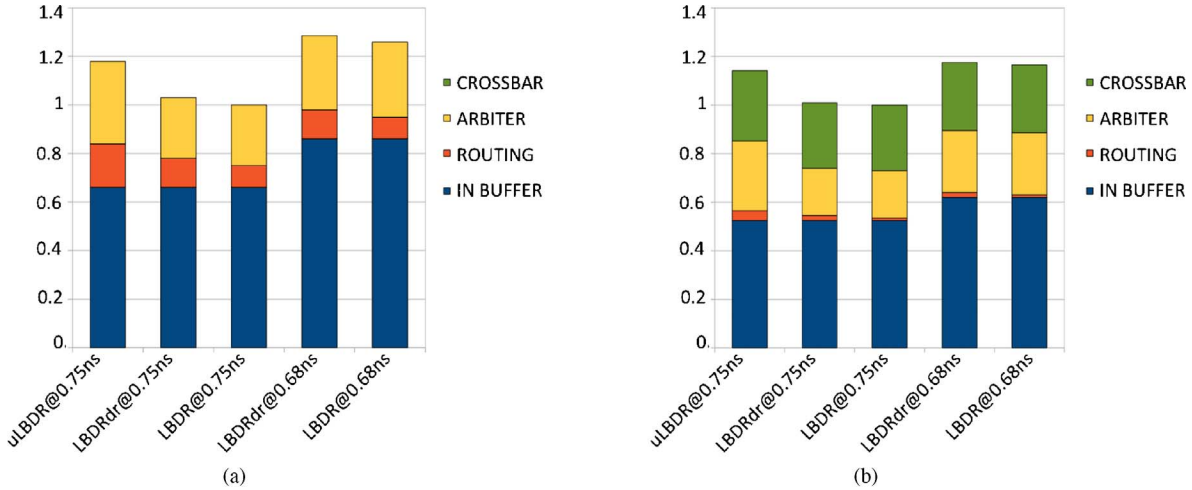

Fig. 18. CMP router power analysis, normalized results. (a) Idle power. (b) Active power.

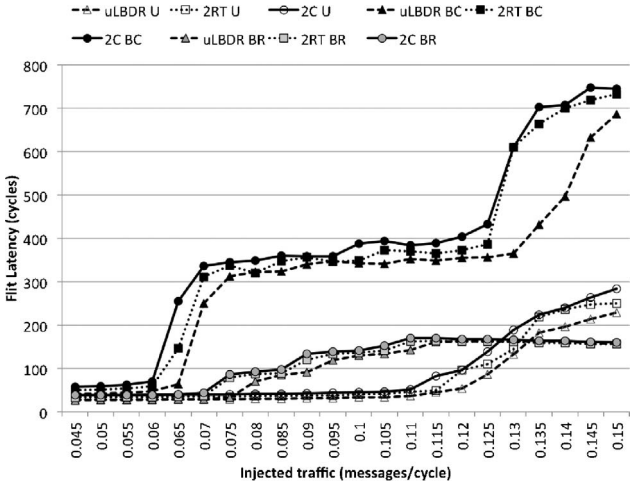

Fig. 19. Flit latency with different mechanisms in 8×8 mesh under synthetic traffic.

Fig. 18(b) reports total router power in active mode, performed on the same scenario that was set for the MPSoC variants. Note that router power presents similar results than idle power. However, the gap between uLBDR and $LBDR_{dr}$ with respect to LBDR is reduced.

To conclude the router's evaluation, it is worth mentioning LBDR versions emerge as a implementation solution that aims to be the minimum representation of a routing table with a logic-based design. Note that LBDR overhead does not grow with system size, as opposite to any routing table scheme (even if it is reduced). In [5], a direct comparison of LBDR-based approach versus routing tables is performed.

D. Performance Analysis

This section starts with the performance evaluation of several mechanisms: uLBDR, distributed routing tables assuming a two-cycle delay router (2C), and distributed routing tables assuming a two-cycle routing stage. We have developed, for such purpose, an in-house cycle accurate flit-level network simulator. The tests were made on 4×4 and 8×8 mesh topologies under three traffic load types: uniform, bit-reversal, and bit-complement. To force the use of deroutes and forks (thus, using full potential of uLBDR) three link failures were included randomly into five different topologies for each mesh size, respectively. 4×4 topologies required eight deroutes and two forks, and 8×8 topologies required 12 deroutes and three forks, on average.

In Fig. 19, average flit latencies for the 8×8 mesh case are shown (4×4 case results show the same behavior). We can see that uLBDR shows better latencies than any routing table implementation, specifically on the bit-complement scenario, where we can find an average gap of around 50 cycles, both in 4×4 and 8×8 case scenarios. In fact, routing tables are penalized due to extra latency on accessing the information in memory macros.

In Fig. 20, average flit throughput results are shown for the different traffic scenarios, again for the 8×8 mesh case (similar results for the 4×4 case). Similar conclusions can be extracted as the ones in the latency results, uLBDR performs equally or better than when using routing table implementations in any traffic scenario.

Fig. 21 shows the impact of fork operations on the overall traffic (number of hops of nonprofitable replica packets divided by the number of hops of all packets). Results show an average impact of 3.5% that belongs to packets that were replicated (only the replica packet that is wasted). This impact may be minimal regarding power costs derived from traffic, and depends on the tradeoff that designers want to achieve. uLBDR, again, offers 100% coverage with important savings in area and power on the hardware compared to routing tables, but some expenses have to be paid when compared to other logic-based mechanisms (which do not offer full coverage).

We have run also several analyses for performance using the GEMS/SIMICS platform [29] upgraded with the event driven cycle-accurate network simulator mentioned before. Several SPLASH-2 [30] applications (Barnes, FFT, LU, Radix) and Apache application have been run in the platform. Chip environment was configured with 16 cores spread in a 4×4 mesh. For this chip configuration, two cache coherency protocols

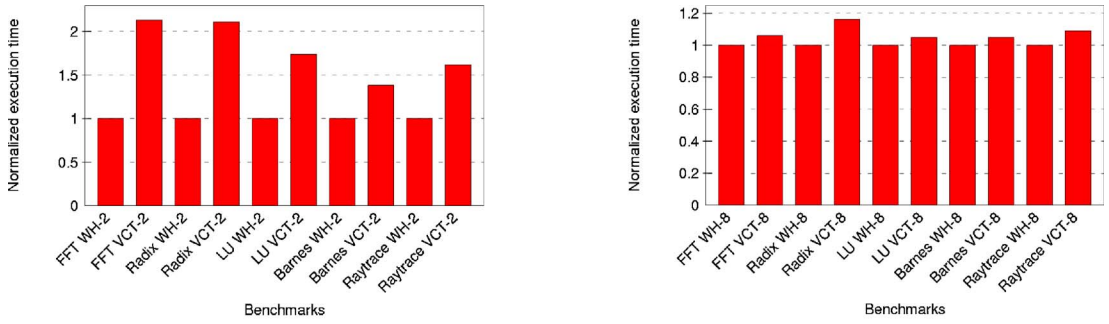

Fig. 24. Execution time of applications in a 4×4 CMP system where packetization is performed. (a) Flit size is set to 2 bytes. (b) Flit size is set to 8 bytes.

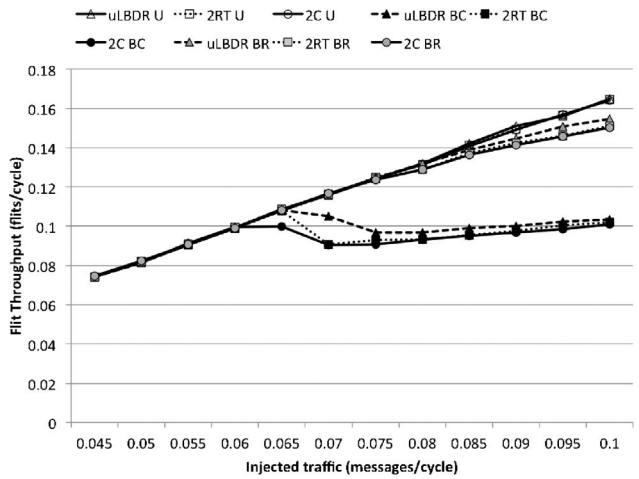

Fig. 20. Flit throughput with different mechanisms in 8×8 mesh under synthetic traffic.

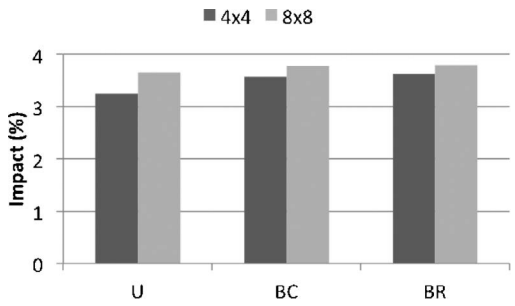

Fig. 21. Impact of nonprofitable hops of replica packets over the total number of hops made by all packets.

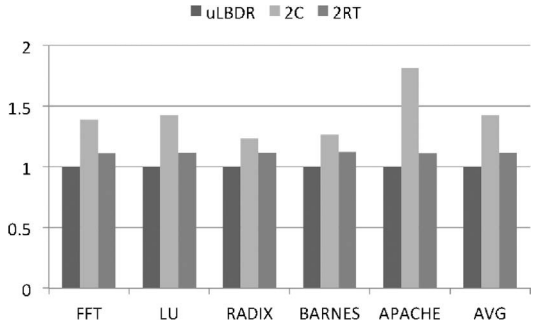

Fig. 22. Normalized execution time of applications, directory-based protocol.

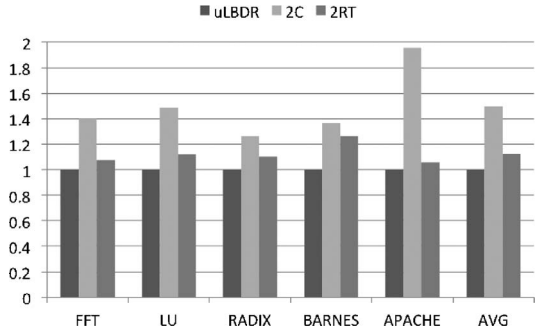

Fig. 23. Normalized execution time of applications, token-based protocol.

have been evaluated to keep coherency between private L1 caches and a shared (but distributed) L2 cache: directory-based and token-based. Virtual-cut-through switching is assumed and flit size is set to 3 bytes (performing packetization and padding when needed).

In Fig. 22, the results for performance evaluation in a directory-based protocol are shown. This figure shows the normalized execution time when using a router with distributed routing tables (two-cycle delay router) and a router using uLBDR mechanism (one-cycle delay router). As can be seen, a slow router (a two-cycle delay router with distributed routing tables), 2C results in figure, affects greatly the execution time of applications, in some cases, like Apache, almost doubling it. When the RT stage experiences two cycles (only for header flits) for accessing the routing table at a memory macro, notice that in this situation the overhead of routing tables is smaller, around 11%. Therefore, the designer needs to avoid the large latency of routing tables.

Performance evaluation on the token-based protocol is shown in Fig. 23. Again, for the three mechanisms, uLBDR performs better, with average gaps of 50% and 11%, being similar to the directory-based conclusions.

To conclude the evaluation, Fig. 24 shows performance on several SPLASH-2 applications (Barnes, FFT, LU, Radix, and Raytrace) results when packetization is used in a VCT router. Also, results assuming WH switching are included for comparison purposes. Fig. 24(a) shows the worst case for packetization when flit size is narrow (2 bytes). In that situation packetizing messages usually doubles execution time of applications. However, in Fig. 24(b) where flit size is widened (8 bytes) the situation changes and now the im-

pact is much lower. Notice that packetization overhead in execution time is lower than 20% in all the applications, being on average 5%. Anyway, there is an overhead in packetization.

VI. Conclusion

In this paper we presented uLBDR, a logic-based routing layer for on-chip networks to support any irregular topology derived from a 2-D mesh without using routing tables. The objective of the full mechanism, uLBDR, is to offer full coverage on this set of topologies, result of several challenges to be taken into account: fault-tolerance, chip virtualization, and power-aware techniques. This is achieved with a tradeoff between router design and coverage. The mechanism proposed spans from low coverage (30%) with no router overhead and no performance impact, to full coverage with a marginal impact on router design. In particular, uLBDR requires a VCT router design and its impact on router frequency is 30% on an MPSoC router and no impact on a CMP pipelined router design.

To sum up, a clear tradeoff lies between coverage of irregular 2-D mesh derived topologies and performance of applications. Future research will target an in-deep tuning of router architectures with uLBDR, specially reducing the paid overhead, along with a reconfiguration methodology of the configuration bits.

References

[1] Intel. (2010). *The Single-Chip Cloud Computer* [Online]. Available: http://techresearch.intel.com/articles/Tera-Scale/1421.htm

[2] Tilera. (2010). *Tilera Tile Multicore Processors* [Online]. Available: http://www.tilera.com/products/processors.php

[3] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy, "Introduction to the CELL multiprocessor," *IBM J. Res. Dev.*, vol. 49, nos. 4–5, pp. 589–604, 2005.

[4] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar, "An 80-tile 1.28TFLOPS network-on-chip in 65 nm CMOS," in *Proc. IEEE ISSCC Dig. Tech. Papers*, Feb. 2007, pp. 98–589.

[5] S. Rodrigo, S. Medardoni, J. Flich, D. Bertozzi, and J. Duato, "Efficient implementation of distributed routing algorithms for NoCs," *IET Comput. Digital Tech.*, vol. 3, no. 5, pp. 460–475, 2009.

[6] Faraday Technology. (2010). *UMC Free Library: 90 nm IPs* [Online]. Available: http://freelibrary.faraday-tech.com/ips/90library.html

[7] H. Sullivan and T. R. Bashkow, "A large scale, homogeneous, fully distributed parallel machine, I," in *Proc. 4th Annu. ISCA*, 1977, pp. 105–117.

[8] A. S. Tanenbaum, *Computer Networks*. Upper Saddle River, NJ: Prentice-Hall, 2003.

[9] A. Gara, M. A. Blumrich, D. Chen, G. L.-T. Chiu, P. Coteus, M. E. Giampapa, R. A. Haring, P. Heidelberger, D. Hoenicke, G. V. Kopcsay, T. A. Liebsch, M. Ohmacht, B. D. Steinmacher-Burow, T. Takken, and P. Vranas, "Overview of the Blue Gene/L system architecture," *IBM J. Res. Dev.*, vol. 49, no. 2, pp. 195–212, 2005.

[10] M. E. Gómez, N. A. Nordbotten, J. Flich, P. López, A. Robles, J. Duato, T. Skeie, and O. Lysne, "A routing methodology for achieving fault tolerance in direct networks," *IEEE Trans. Comput.*, vol. 55, no. 4, pp. 400–415, Apr. 2006.

[11] J. Van Leeuwen and R. B. Tan, "Interval routing," *Comput. J.*, vol. 30, no. 4, pp. 298–307, 1987.

[12] M. E. Gómez, P. López, and J. Duato, "A memory-effective routing strategy for regular interconnection networks," in *Proc. 19th IEEE IPDPS*, Apr. 2005, p. 41.2.

[13] S. Borkar, R. Cohn, G. Cox, S. Gleason, and T. Gross, "iWarp: An integrated solution of high-speed parallel computing," in *Proc. ACM/IEEE Conf. Supercomput.*, Nov. 1988, pp. 330–339.

[14] J. Flich, A. Mejía, P. López, and J. Duato, "Region-based routing: An efficient routing mechanism to tackle unreliable hardware in network on chips," in *Proc. 1st Int. Symp. NOCS*, 2007, pp. 183–194.

[15] M. Palesi, S. Kumar, and R. Holsmark, "A method for router table compression for application specific routing," in *Proc. 6th SAMOS Workshop Mesh Topology NoC Architectures*, 2006, pp. 373–384.

[16] S. Rodrigo, C. Hernández, J. Flich, F. Silla, J. Duato, S. Medardoni, D. Bertozzi, A. Mejía, and D. Dai, "Yield-oriented evaluation methodology of network-on-chip routing implementations," in *Proc. 11th Int. Conf. SoC*, 2009, pp. 100–105.

[17] M. Koibuchi, H. Matsutani, H. Amano, and T. M. Pinkston, "A lightweight fault-tolerant mechanism for network-on-chip," in *Proc. 2nd ACM/IEEE Int. Symp. NOCS*, Apr. 2008, pp. 13–22.

[18] W. Song, D. Edwards, J. L. Nuñez Yanez, and S. Dasgupta, "Adaptive stochastic routing in fault-tolerant on-chip networks," in *Proc. 3rd ACM/IEEE Int. Symp. NOCS*, May 2009, pp. 32–37.

[19] J. Nuñez Yanez, D. Edwards, and A. Coppola, "Adaptive routing strategies for fault-tolerant on-chip networks in dynamically reconfigurable systems," *IET Comput. Digital Tech.*, vol. 2, no. 3, pp. 184–198, 2008.

[20] A. Kohler and M. Radetzki, "Fault-tolerant architecture and deflection routing for degradable NoC switches," in *Proc. 3rd ACM/IEEE Int. Symp. NOCS*, May 2009, pp. 22–31.

[21] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "Routing table minimization for irregular mesh NoCs," in *Proc. DATE*, 2007, pp. 942–947.

[22] I. Loi, F. Angiolini, and L. Benini, "Synthesis of low-overhead configurable source routing tables for network interfaces," in *Proc. Conf. DATE*, 2009, pp. 262–267.

[23] T. Skeie, F. O. Sem-Jacobsen, S. Rodrigo, J. Flich, D. Bertozzi, and S. Medardoni, "Flexible DOR routing for virtualization of multicore chips," in *Proc. 11th Int. Conf. SoC*, 2009, pp. 73–76.

[24] A. Mejía, J. Flich, J. Duato, S. A. Reinemo, and T. Skeie, "Segment-based routing: An efficient fault-tolerant routing algorithm for meshes and tori," in *Proc. Int. Parallel Distributed Process. Symp.*, Apr. 2006, pp. 84–93.

[25] F. A. Samman, T. Hollstein, and M. Glesner, "Planar adaptive router microarchitecture for tree-based multicast network-on-chip," in *Proc. Int. Workshop NoCArc*, Nov. 2008, pp. 6–13.

[26] A. Pullini, F. Angiolini, S. Murali, D. Atienza, G. De Micheli, and L. Benini, "Bringing NoCs to 65 nm," *IEEE Micro*, vol. 27, no. 5, pp. 75–85, Nov. 2007.

[27] E. S. Shin, V. J. Mooney, III, and G. F. Riley, "Round-robin arbiter design and generation," in *Proc. 15th ISSS*, 2002, pp. 243–248.

[28] Nangate. (2010). *The Nangate Open Cell Library, 45 nm Freepdk* [Online]. Available: https://www.si2.org/openeda.si2.org/projects/nangatelib

[29] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (gems) toolset," *SIGARCH Comput. Architecture News*, vol. 33, no. 4, pp. 92–99, 2005.

[30] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *Proc. 22nd Annu. ISCA*, 1995, pp. 24–36.

Samuel Rodrigo received the M.S. and Ph.D. degrees in computer engineering from the Technical University of Valencia, Valencia, Spain, in 2008 and 2010, respectively.

He is currently a Post-Doctoral Fellow with the Simula Research Laboratory, Oslo, Norway. His current research interests include routing and fault-tolerance schemes in networks-on-chip.

José Flich received the M.S. and Ph.D. degrees in computer science from the Technical University of Valencia, Valencia, Spain, in 1994 and 2001, respectively.

He joined the Department of Computer Engineering, Universidad Politécnica de Valencia, in 1998, where he is currently an Associate Professor of computer architecture and technology with the Parallel Architectures Group. He has published over 100 papers in peer-reviewed conferences and journals. His current research interests include high-performance interconnection networks for multiprocessor systems, cluster of workstations, and networks-on-chip.

Dr. Flich has served as a program committee member in different conferences, including NOCS, DATE, ICPP, IPDPS, HiPC, CAC, ICPADS, and ISCC. He is an Associate Editor of the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS. He is currently the co-chair of the CAC and INA-OCMC workshops. He is the Coordinator of the NaNoC FP7 EU-Funded Project (http://www.nanoc-project.eu).

Antoni Roca received the M.S. degree and the Advanced Studies Diploma, both in telecommunications engineering from the Technical University of Valencia, Valencia, Spain, in 2006 and 2007, respectively.

Currently, he is pursuing the Ph.D. degree from the Parallel Architectures Group, Technical University of Valencia. His current research interests include network-on-chip architectures, especially router implementation.

Simone Medardoni received the Ph.D. degree in electrical engineering from the University of Ferrara, Ferrara, Italy, in 2009, with a dissertation on technology-aware network-on-chip architecture design.

He is currently a Post-Doctoral Researcher with the Integrated Systems Laboratory of Minatec, Grenoble, France. His current research interests include embedded systems, networks-on-chip, and emulation techniques for system-on-chips.

Davide Bertozzi received the Ph.D. degree in electrical engineering from the University of Bologna, Bologna, Italy, in 2003.

Since 2005, he has been an Assistant Professor with the Department of Engineering, University of Ferrara, Ferrara, Italy, where he leads research activities on multi-processor systems-on-chip and on networks-on-chip in particular. He has been a Visiting Researcher with international academic institutions (Stanford University, Stanford, CA) and large semiconductor companies (NEC America Laboratories, Princeton, NJ, NXP Semiconductors, Eindhoven, The Netherlands, STMicroelectronics, Agrate Brianza MB, Italy, Samsung Electronics, Seoul, South Korea).

Dr. Bertozzi was the Program Chair of the International Symposium on Networks-on-Chip in 2008, of the NoC Track at the Design Automation and Test in Europe Conference in 2010 and 2011, and a guest editor of the special issues on networks-on-chip of the *IET Computer and Digital Techniques Journal* in 2009 and of the *Hindawi VLSI Design Journal* in 2007. He is a member of the Editorial Board of the *IET CDT Journal*. He is a member of the Hipeac-2 NoE (Interconnect Cluster) and is actively involved in STREP projects funded by the EU (Galaxy Project, NaNoC Project).

Jesús Camacho is currently pursuing the Ph.D. degree from the Parallel Architectures Group, Technical University of Valencia, Valencia, Spain.

His current research interests include design of novel high performance and power-aware networks-on-chip topologies.

Federico Silla received the M.S. and Ph.D. degrees in computer engineering from the Technical University of Valencia, Valencia, Spain, in 1995 and 1999, respectively.

He is currently an Associate Professor with the Parallel Architectures Group, Department of Computer Engineering, Technical University of Valencia. His current research interests include high performance on-chip and off-chip networks, as well as distributed systems. He is a member of the Advanced Technology Group of the HyperTransport Consortium.

José Duato received the M.S. and Ph.D. degrees in electrical engineering from the Technical University of Valencia, Valencia, Spain, in 1981 and 1985, respectively.

Currently, he is a Professor with the Parallel Architectures Group, Department of Computer Engineering, Technical University of Valencia, and is a Researcher with the Simula Research Laboratory, Oslo, Norway. He also developed RECN, a scalable congestion management technique, and a very efficient routing algorithm for fat trees that has been incorporated into the Sun Microsystems 3456-Port InfiniBand Magnum Switch. Currently, he leads the Advanced Technology Group in the HyperTransport Consortium, whose main result until now has been the development and standardization of an extension to HyperTransport (High Node Count HyperTransport Specification 1.0). He is the first author of the book *Interconnection Networks: An Engineering Approach*. His current research interests include interconnection networks and multiprocessor architectures.

Dr. Duato served as a member of the editorial boards of the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON COMPUTERS, and IEEE COMPUTER ARCHITECTURE LETTERS. He has been the General Co-Chair for the 2001 International Conference on Parallel Processing, the Program Committee Chair for the Tenth International Symposium on High Performance Computer Architecture, and the Program Co-Chair for the 2005 International Conference on Parallel Processing. Also, he served as the Co-Chair, a member of the Steering Committee, Vice-Chair, or a member of the program committees in more than 60 conferences, including the most prestigious conferences in his area (HPCA, ISCA, IPPS/SPDP, IPDPS, ICPP, ICDCS, Europar, HiPC).