Addressing Manufacturing Challenges with Cost-Efficient Fault Tolerant Routing

S. Rodrigo[1], J. Flich[1], A. Roca[1], S. Medardoni[2], D. Bertozzi[2], J. Camacho[1], F. Silla[1] and J. Duato[1]
[1]Parallel Architectures Group
Technical University of Valencia
Valencia, Spain
[2]ENDIF, Department of Engineering
University of Ferrara
Ferrara, Italy

***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.**

In this paper, uLBDR (Universal Logic-Based Distributed Routing) is proposed as an efficient logic-based mechanism that adapts to any irregular topology derived from 2D meshes, being an alternative to the use of routing tables (either at routers or at end-nodes). 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 trade-off 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 trade-off between fault tolerance and performance.

***Index Terms*—Networks-on-chip; Fault-tolerance; routing**

I. Introduction

Main microprocessor manufacturers have shifted to Chip Multi-Processors (CMPs) for their latest 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 $x86$ compatible, under the Tera-scale Computing Research Program [1]. Previously, Intel also developed a chip prototype [2] that included 80 cores (known as TeraFlops Research chip).

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 [3] 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.

CMPs and high-end MPSoCs rely on an on-chip (NoC) network able to handle all the communication traffic between cores. The most straightforward topology for NoCs is the 2D mesh structure as it offers regularity and simplicity for routing. All the links have the same length thus, exhibiting the same latency. Also, local traffic is well supported since latency is low. One of the inconveniences of the 2D mesh, however, is the relative higher hop count for messages travelling distant nodes. Fortunately, this impact is minimized with the use of wormhole and virtual cut-through switching (where hop count is additive to latency).

On the other hand, in CMP systems 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 2D meshes.

Due to the previous reasons, we advocate for 2D meshes in CMPs and high-end MPSoCs. However, even if the design of a CMP chip with a 2D mesh network is correct, the final on-chip network may face new raising challenges, leading to a non-regular network structure. Several challenges are being identified in the following years to come: manufacturing defects, effective chip utilization, voltage/frequency islands, and effective power saving techniques.

As technology advances, **correct manufacturing** becomes challenging and defective components are more frequent in final chips. A clear example is the allocation of a defective tile that, if not addressed, will ruin the 2D mesh structure of the network, leading to an irregular network not handed by the routing algorithm and thus making the chip useless.

Another challenge is getting enough parallelism to efficiently use tens and hundreds of cores. This will render to a **low utilization** of the cores, or alternatively, to the need to partition the chip into multiple domains, each one running a different application. In this scenario, and to fit as many applications as possible, the partitions of the chip resources will become irregular. The way applications are mapped onto the chip belongs to the concept known as virtualization, where a real chip is divided in several smaller virtual chips.

Voltage/frequency islands are being identified as a need, where different regions of the chip will have different operating conditions. This will require the isolation of such regions so to avoid conflicting mismatches that may lead to bottlenecks, e.g. a message crossing different domains.

Finally, another major challenge is the need of **efficient power saving methods**. As the number of cores increases, probably most of the cores will remain unpowered (in sleep mode) most of the time, so enabling large savings in power consumption. The same should be applied to the on-chip network, which has been reported to consume 30% of the total chip power consumption. Powering off and on different routers will lead to temporal irregular patterns.

All these previous challenges require some effort at the on-chip network level, specifically supporting irregular network topologies. But rather than addressing completely irregular topologies (more suitable for low-end MPSoC systems) the effort must be made to address irregular topologies derived from an original 2D mesh with the following properties: (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 and column.

In this paper we provide a step further and provide a mechanism able to cover all the possible cases derived from a 2D mesh, that is, with full support for any fail-

ure/virtualization/domain/region configuration. The mechanism, referred to as uLBDR (universal Logic-Based Distributed Routing) is an evolution of LBDR [4].

The remaining of the 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 conclude the paper and provide future directions.

II. Related Work

The first straightforward way to deal with irregular topologies corresponds to source-based routing. The node that sends a message computes a path, from a routing table stored at the end-node, and stores it in the message header. One example is Street-Sign Routing [5]. It compresses the message header to minimize the impact on network bandwidth. The router id and the direction for the next turn is included in the message header. However, a table is still needed at every end node.

In distributed routing, however, the message header only contains the destination ID and each router computes the next link to forward the message. One of the most used techniques to deal with non-regular topologies is to implement forwarding tables at routers, so each one stores the output port that must be used based on the destination. The main advantage of table-based routing is that any topology and any routing algorithm can be implemented, including fault-tolerant routing algorithms. However, routing tables do not scale in terms of latency, power consumption, and area, thus being impractical for NoCs [6].

Using solutions from the off-chip network domain for NoCs must be a step reviewed thoroughly. Message dropping proposals like TCP/IP protocols [7] would affect severely network performance, so they are not suitable. Techniques used in large parallel systems that switch off healthy nodes (lamb nodes) like the Blue Gene/L system [8] to keep topology and routing algorithm untouched also impacts performance. Other mechanisms, focused on routing optimization [9] 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 issues, a feature that could be difficult to implement in NoCs.

Region-Based Routing (RBR) [10] (and a similar proposal [11]) has been proposed as an attempt to achieve fault-tolerance in on-chip networks while requiring few resources. RBR groups at each router different destinations (region) that can be reached by an output port. The main drawback [12] of such mechanism is that, even with 16 regions defined, it still does not achieve 100% coverage. Also, it significantly increases the router delay path [12].

Default-Backup Path (DBP) [13] deals with the issue of maintaining healthy processing elements when the attached router fails. It adds redundant wiring and buffers connecting output and input ports directly. It, however, does not address routing in irregular topologies.

In [14] authors propose an architecture based on deflection routing that attempts to detect fault errors adding CRC modules at input and output ports for crossbar faults and SEC codes for link faults with the support of routing matrices, one for each type of fault. The link fault matrix is $n \times n$ being n the dimension. It uses a variant of deflection routing called delta XY with weighted priority. Deflection (or reflection) can lead to potential starvation that is solved with message dropping, thus impacting performance. Also, routing matrices exhibit poor scalability as the number of destinations augments.

There are several proposals relying on the use of tables, thus suffering from the same scalability problems and routing costs. Adaptive stochastic routing (ASR) [15] relies on a self-learning method to handle failures by assigning confidence fields to output ports for different tasks running in the system. Thus, it requires a table on each router having n entries for n tasks. In [16] the objective is to minimize routing tables, either at end-nodes or at routers. Three techniques are proposed: Turn-table (TT), XY Deviation Table (XYDT) and Source Routing Deviation Points (SRDP). Mainly, all these techniques rely on a routing table to handle irregular cases in combination with routing strategies like XY (combined with YX), source routing or the *don't turn* technique (meaning, a message does not change direction when traversing the router unless is noted). In [17], authors propose a compendium of state-of-the-art LUTs implementations for routing purposes. These designs shift from being fully hardwired to being partially or fully configurable depending on the degree of flexibility.

As a summary, all the proposals tend to use routing tables (either at sources or at routers) and/or rely on an excessive number of resources (virtual channels) to solve the deadlock problem. Also, none of the solutions (except when using tables) provides full coverage (all the possible failure cases) for a 2D mesh.

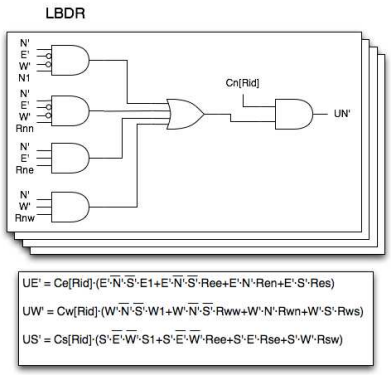
III. uLBDR Description

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 2D meshes we will refer to each router port by its direction, being N, E, W, and S for north, east, west, and south, respectively.

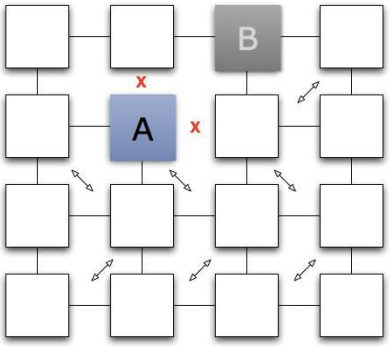
A. First mechanism: LBDR

The uLBDR evolves from the basic LBDR mechanism. LBDR requires 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 to define the routing and connectivity pattern. Those bits are used by a combinational logic to decide the output port to use for every message.

The LBDR mechanism relies on the use of minimal paths for every source-destination pair. Indeed, LBDR uses two comparators to decide the directions to use based on the relative position of the current router and the destination router. The directions (labelled as N', E', W', and S' in Figure 1(a)) are then checked against the routing/connectivity bits. Therefore, only minimal paths are allowed. This leads, as we will see in the evaluation, to a low percentage of topologies being supported. Indeed, in a 2D 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 center of the network is a clear example. Figure 1(b) 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.



(a) ULBDR core routing logic: LBDR



(b) Topology not supported by LBDR

Fig. 1. uLBDR core mechanism: LBDR.

B. Second Mechanism: $LBDR_{dr}$

We need to provide non-minimal support in an efficient way, that is, with as minimum logic as possible. Figure 2(a) shows the additional logic proposed. In particular, for every input port of the router a deroute option is provided. A set of two bits encode the deroute option that can be N, E, W, or S. 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 input port the message arrived from.

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). Figure 2(b) 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. It is worth mentioning that the deroute option needs to be computed in accordance to the routing algorithm, as it must not introduce cycles that could lead to deadlocks. In Figure 2(b) a deroute option at input port W at router B can not be set to S as it would let messages crossing a routing restriction.

The algorithm computes first the set of routing bits. This is done by taking into account the topology (including the failed/powered down routers and links) and the routing algorithm. The selected algorithm in this paper is SR [18] as it can be applied to any topology and does not require virtual channels[1]. LBDR bits are computed by taking into account the location of the routing restrictions. Thus, computation is straightforward and the complexity is low (linear with the number of routing restrictions).

Once LBDR bits are computed the deroute options are searched. To do this, the algorithm looks for a valid path for every source-destination pair (the algorithm is computed offline before any normal operation of the chip, thus computation complexity is not a major issue). As LBDR may allow multiple paths for a given source-destination, the algorithm deeply searches all the paths in a recursive way. Two end nodes are connected by LBDR if all the possible paths reach the destination. In the search of all paths, whenever it fails to provide a valid path, then, a misrouting action is needed. Figure 2(c) shows the case at router B for messages going from router A to router C. In this situation, the algorithm tries all the possible deroute options available, one per output port but avoiding U turns (so, west port is not considered). Options leading to crossing routing restrictions are also evicted. The algorithm starts with the first deroute option and keeps following the path, thus taking the deroute, checking if the path (and all their possible alternative paths) will reach the destination. In case of success, the deroute option is set. In case of failure (destination is not reached) another deroute option is tried. In case all deroute options fail the topology is not covered by $LBDR_{dr}$. Notice that several deroute options may be required for a single path. Figure 2(c) shows the route along the path using the deroute bits at router B.

This mechanism will enhance greatly the coverage of uLBDR. However, there are subtle cases that are still not covered by $LBDR_{dr}$. Figure 2(d) shows an example. The problem comes by the fact that for some destinations located at the same quadrant, at router B the routing engine should provide one port (N) for some destinations (router C) and another port (W) for other destinations (router A). As LBDR (or $LBDR_{dr}$) works in quadrants, there is no way to indicate the router which option should be given to the message.

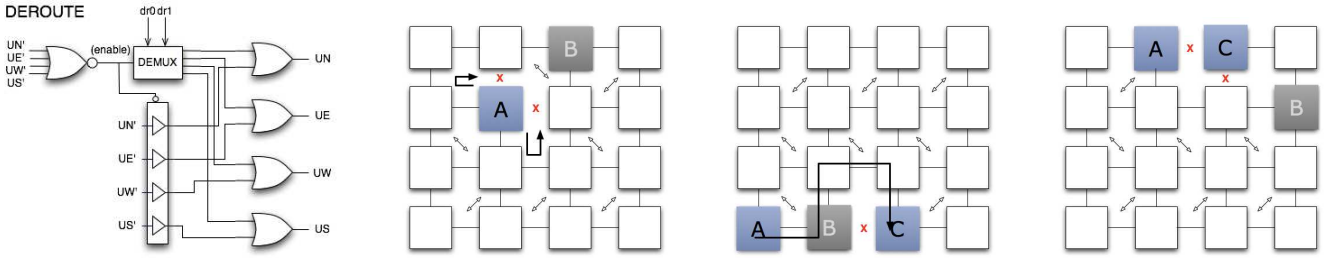
C. Third Mechanism: $LBDR_{fork+dr}$

To solve the previous problem, $LBDR_{dr}$ is enhanced with an additional feature. The message is simply forked through two output ports. This leads, however, to important changes in the router design. We describe them next.

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, therefore, 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 one read pointer is needed. In the second one, the arbiter may grant or deny access to one port regardless of the action performed for the other port. This leads to a complex input buffering, as message forwarding may be shifted for both output ports, thus each requiring a read pointer. We assume the first option because of its simplicity and the fact the fork operations will be used in few cases.

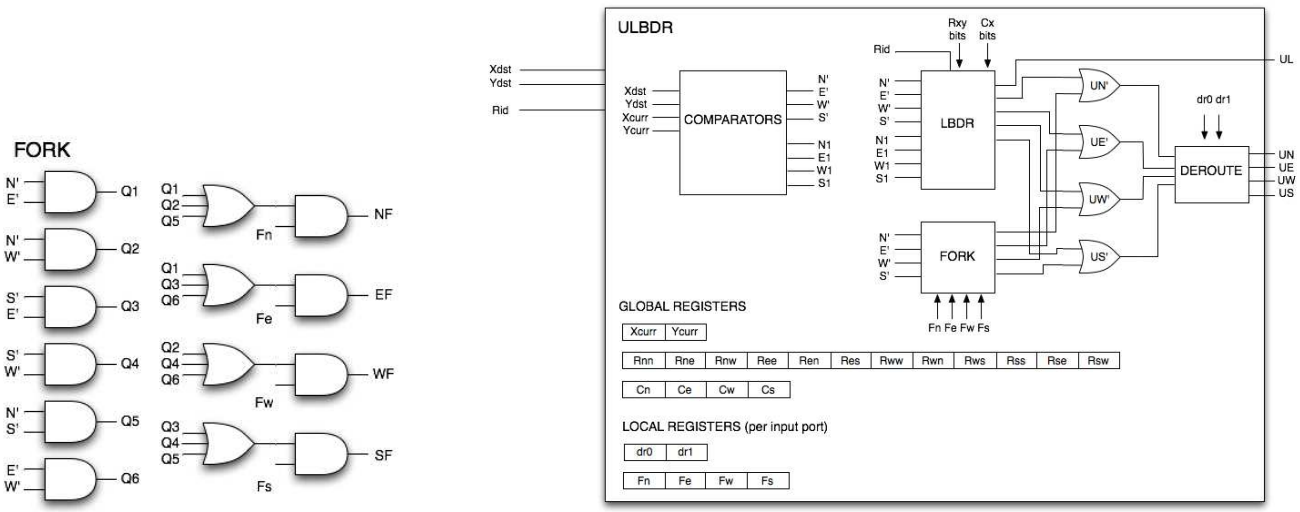
Deadlock may occur in wormhole switching, since two fork messages may compete for the same set of resources. Although the routing algorithm used is deadlock-free, performing fork actions (like collective communication) leads to deadlock.

[1]Alternatively, for a healthy chip the XY routing algorithm can be used.



(a) Deroute logic (b) Two different deroute options (c) Deroute solution (d) Topology not supported by $LBDR_{dr}$

Fig. 2. uLBDR: $LBDR_{dr}$ mechanism.



(a) Fork logic (b) Complete uLBDR mechanism

Fig. 3. uLBDR: $LBDR_{fork+dr}$ mechanism.

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 (from the problem's point of view) is the use of virtual cut-through (VCT) switching, thus ensuring a packet[2] 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 circuit switching [19] 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. The problem with this kind of solutions is that internal tables are required so to keep flit identifiers. We opt for the first solution, thus enforcing VCT switching. Although VCT is seen as demanding much buffer space at routers, a careful design of the router can minimize this impact. In this paper we show key design options that enable its use in NoCs (Section IV).

Asuming VCT switching, one of the packets being forked will reach the final destination. However, the other replica will not reach the destination. In this situation, the packet needs to be removed from the network. This will be easily achieved by silently destroying the packet at a router. In Section V we will show area and latency results for two routers designed for $LBDR_{fork+dr}$, thus showing upfront the real impact of such router changes. The $LBDR_{fork+dr}$ mechanism is shown in Figure 3(b). Additionally to the original mechanism more routing bits are added: R_{nn}, R_{ee}, R_{ww}, R_{ss}. Similarly to R_{xy} bits, these bits are used to indicate where vertical or horizontal routing restrictions (similar arrows in Figure 2(c)) exist at the next hop. Two additional comparators are used (comparators block in Figure 3(b)) to know if a packet is one hop away from its destination through any direction (signals N_1, E_1, W_1, and S_1). Also, new four additional fork bits (F_n, F_e, F_w and F_s) are used per input port. These bits are set to reflect the ports that must be used to fork a packet. Whenever a packet's destination is in the same quadrant defined by the fork bits, then the packet is forked. Fork bits are checked in parallel with the LBDR logic, just after computing the N', E', W', and S' signals. Keep in mind those bits are computed offline. Once LBDR and deroute bits are computed and do not provide a valid path to reach a destination, then fork bits are computed. The algorithm checks for every destination in the same quadrant (NE, ES, SW, WN, NS and WE) if the path (and all their possible alternative paths) will reach the destination with forks included. In case of failure (destination is not reached), the topology is not covered.

IV. Routers Description

In this section we provide a brief description of the routers used for the evaluation: a non-pipelined MPSoC router and

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

a pipelined CMP router. In both cases an initial wormhole router design has been evolved with minimum changes so to allow virtual cut-through switching. Changes required are: (1) in flow control, (2) in the arbiter logic (support fork operations), and (3) in the crossbar to remove stale copies of forked packets.

A. MPSoC Router Design

Typically, NoC building blocks for use in MPSoCs target lower operating speeds with respect to CMPs and are generally unpipelined [20]. The reference component that we consider to assess the feasibility of $uLBDR$ is an input buffered router implementing wormhole switching (Figure 4). Size of the input buffer is tunable and set to 4 slots here. In 1 clock 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 (one round-robin arbiter for each 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 signalling either a condition of buffer filled ("stall") or of buffer free ("go"). This latter signal is indicated as $flow\ control$ in Fig.4.

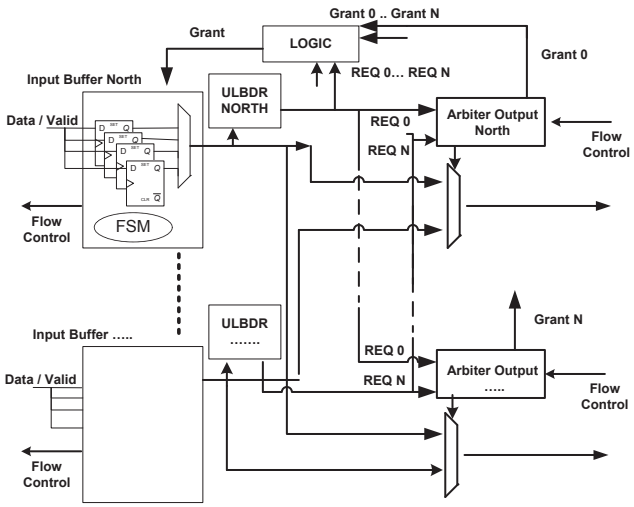


Fig. 4. MPSoC router schematic.

$LBDR$ 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 $LBDR_{fork+dr}$. The signals used and the architecture schematic are the same of Figure 4, just the meaning of flow control signals and the arbiter behaviour change. First of all, we had to evolve the basic stall/go flow control protocol 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 vs writes, variable number of write/read burst beats, etc.), 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 Figure 4 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 behaviour 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 LBDR$_{fork+dr}$, 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 Figure 4. 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. Flit size is set to 4 bytes. The input buffer size is set to four flits. The RT module has been implemented to support XY, $LBDR$, $LBDR_{dr}$, and $LBDR_{fork+dr}$ routing algorithms and the Stall/Go flow control. Finally, the SW module has been designed with a round-robin arbiter as in [21]. The router has been implemented using the 45nm technology open source Nangate [22].

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 packets 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 Figure 5. 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.

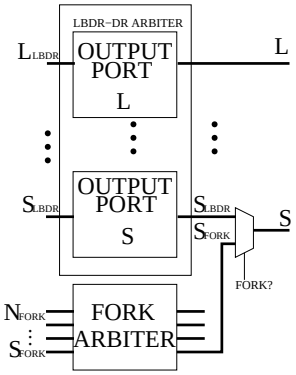


Fig. 5. New arbiter for the CMP router with fork requests.

V. Evaluation

In this Section we provide several evaluation results to assess the impact of the different LBDR versions. First, we provide a coverage analysis for each solution. Then, we analyze the overhead of the mechanisms in the router designs, including also the overhead incurred by the VCT router counterparts. Finally, we provide performance results by running real applications in a full system simulator environment. We test in particular if the packetization requirement by $LBDR_{fork+dr}$ affects final application's execution time. Also, performance results when using routing tables are obtained.

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, $LBDR_{fork+dr}$. 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 [12] 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 modelled on a real 65nm implementation NoC layout where all cores are identical, and their size is $1mm^2$. 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 ($LBDR_{1dr}$), $LBDR_{dr}$ as explained in Section III-B and $LBDR_{dr+fork}$ (or uLBDR), the full mechanism. In Figure 6(a) 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. Figure 6(b) shows the average percentage of deroutes and forks required per chip. As can be seen, an small set of deroutes are required. As irregularity and network size increases the use of deroutes also increases and the fork mechanism is even less utilized.

B. MPSoC Router Overhead

We synthesized the MPSoC router with a 65nm STMicroelectronics technology library and Synopsys Physical Compiler. Routers with all routing mechanisms (LBDR, LBDR$_{dr}$ and LBDR$_{fork+dr}$) were synthesized both for max. performance and for the same target speed (that of the slowest architecture, i.e., LBDR$_{fork+dr}$). 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: 1GHz for LBDR, 950 MHz for $LBDR_{dr}$, and 750 MHz for $LBDR_{fork+dr}$.

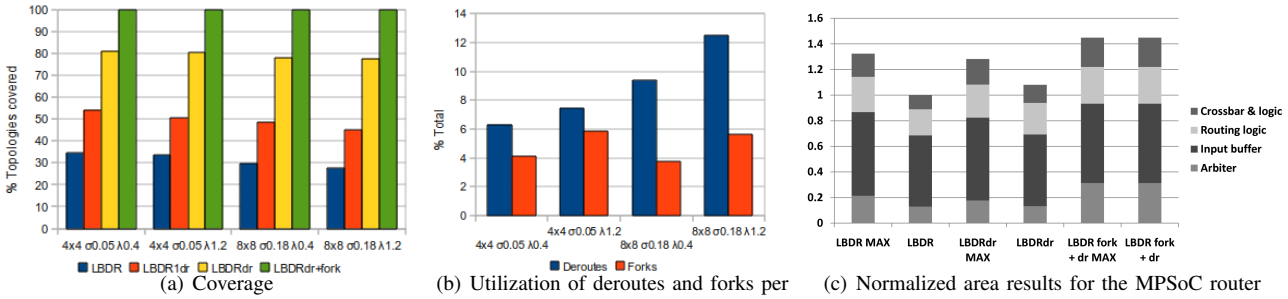
Post-synthesis area results for the routers are illustrated in Figure 6(c). By looking at the maximum performance figures, LBDR$_{fork+dr}$ is about 10% larger than LBDR, clearly due to the more complex port arbiters and to their need to handle true credit based flow control. To make this relatively 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 (the LBDR$_{fork+dr}$), 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 LBDR$_{fork+dr}$ 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 trade-off decision. When the target frequency is very low (a few hundred MHz, not showed in Figure 6(c)), 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.

C. CMP Router Overhead

Table I summarizes (upper part) frequency and area results of the CMP router for different switching techniques and routing mechanisms. The first thing to highlight is the

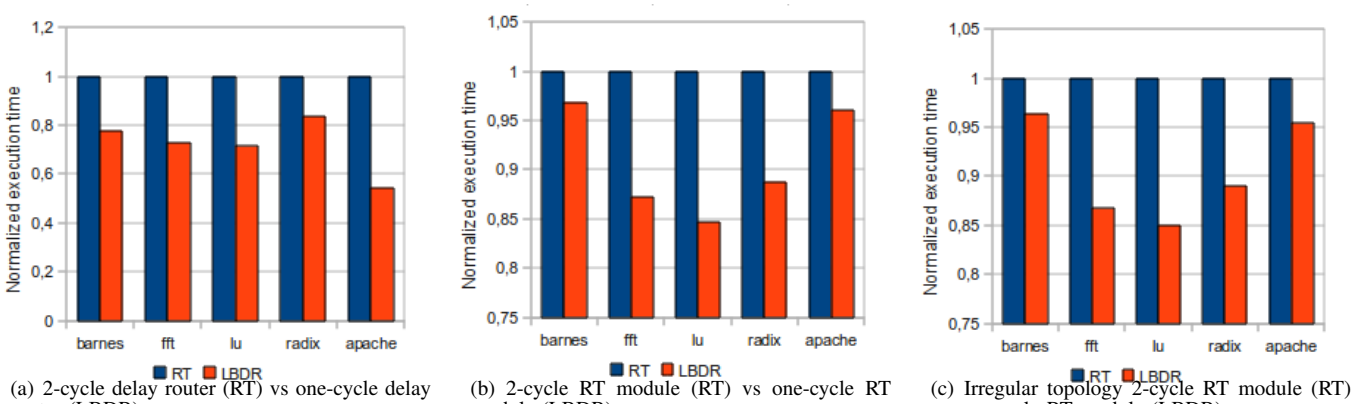


(a) Coverage (b) Utilization of deroutes and forks per router (c) Normalized area results for the MPSoC router

Fig. 6. Performance results. Coverage, utilization of deroutes and forks, and MPSoC router.

All the router					
area / Freq	WH-XY	WH-LBDR	WH-$LBDR_{dr}$	VCT-xy	VCT-$LBDR_{fork+dr}$
area ($\mu m2$)	19860	20335	20418	13644	14944
frequency (GHz)	1.33	1.33	1.33	1.53	1.47
Only the routing modules					
area / Freq	WH-XY	WH-LBDR	WH-$LBDR_{dr}$	VCT-xy	VCT-$LBDR_{fork+dr}$
area ($\mu m2$)	137.592	143.304	141.479	86.3	137.91
frequency (GHz)	3.45	2.78	2.56	4.55	2.56

TABLE I
AREA AND FREQUENCY FOR THE CMP ROUTER AND THE ROUTING MODULES, BOTH IN WH AND VCT VERSIONS, DIFFERENT ROUTING ALGORITHM IMPLEMENTATIONS.



(a) 2-cycle delay router (RT) vs one-cycle delay router (LBDR) (b) 2-cycle RT module (RT) vs one-cycle RT module (LBDR) (c) Irregular topology 2-cycle RT module (RT) vs one-cycle RT module (LBDR)

Fig. 7. Execution time of applications in a 4×8 CMP system. Normalized execution time of applications when routers with two cycle delays (RT) and one-cycle delay (LBDR) are used.

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 will be introduced (known as atomic buffer allocation).

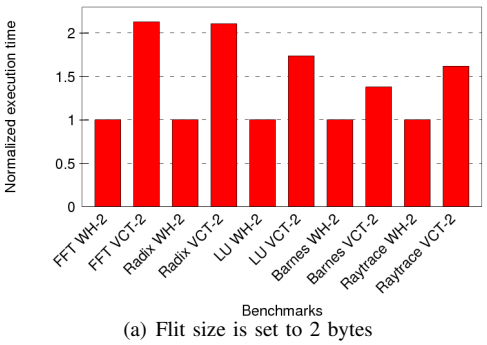
There is no difference in the operating frequency when using either XY, LBDR or $LBDR_{dr}$, and only a marginal increase in area (differences fall within the uncertainties of the synthesis optimization process). This is due to the RT stage not setting the maximum frequency of the router. $LBDR_{fork+dr}$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

Table I also shows (bottom part) 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 an small impact on area but a large one in frequency. Also, the complexity of $LBDR_{fork+dr}$ has a large impact on both area and frequency. However, remember that this module is not the one setting the router frequency.

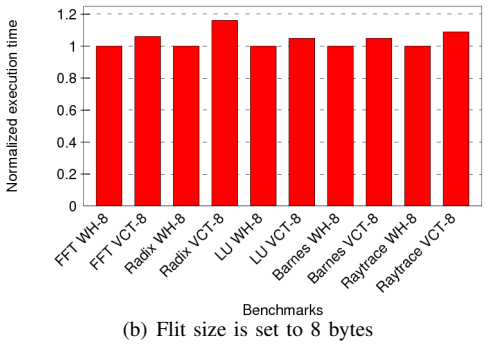
D. Performance Analysis

We have run several analysis for performance using the GEMS/SIMICS platform [23] upgraded with an event driven cycle-accurate network simulator. Several SPLASH-2 [24] applications (Barnes, FFT, LU, Radix, and Raytrace) and Apache application have been run in the platform. Two CMP chip configurations have been used, the first one with 16 cores spreaded in a 4×4 mesh, and the second one with 32 cores spread in a 4×8 mesh. In both cases a directory-based cache coherency protocol is used to keep coherency between private L1 caches and a shared (but distributed) L2 cache.

Figure 7(a) shows the normalized execution time when using a router with distributed routing tables (two cycles delay



(a) Flit size is set to 2 bytes



(b) Flit size is set to 8 bytes

Fig. 8. Execution time of applications in a 4×4 CMP system where packetization is performed.

router) and a router using $LBDR_{dr}$ engine (one cycle delay router). Wormhole switching is assumed and flit size is set to 4 bytes. As can be seen, a slow router (routing tables) affects greatly the execution time of applications, in some cases by a factor of 2. In all applications the slower router behaves worse. Therefore, the designer needs to avoid the large latency of routing tables.

Figure 7(b) shows the case when two routers are compared, one with the RT stage experiencing two cycles (only for header flits) and one with the RT stage having a single cycle. Notice that in this situation the overhead of routing tables is smaller.

Finally, Figure 7(c) shows performance achieved in an 4×8 topology where two links are missing, so deroutes and forks are necessary. In that situation, notice that applications could be run as the routing mechanism is able to support paths (coverage of the topology). The execution time of applications did not varied much. However, the interesting point is the success in running the application in a failed mesh network without using routing tables.

To conclude the evaluation, Figure 8 shows performance results when packetization is used in a VCT router. Also, results assuming WH switching are included for comparison purposes. Figure 8(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 Figure 8(b) where flit size is widened (8 bytes) the situation changes and now the impact 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. Conclusions

In this paper we have presented uLBDR, a logic-based routing layer for on-chip networks to support any irregular topology derived from a 2D mesh without using routing tables. The objective of the full mechanism, $LBDR_{dr+fork}$ 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 trade-off 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 $LBDR_{fork+dr}$ version 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 trade-off lies between coverage of irregular 2D-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.

Acknowledgements

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 the project *NaNoC* (project label 248972) which is funded by the European Commission within the Research Programme FP7.

References

[1] http://techresearch.intel.com/articles/Tera-Scale/1421.htm.
[2] http://techresearch.intel.com/articles/Tera-Scale/1449.htm.
[3] http://www.tilera.com/products/processors.php.
[4] J. Flich, S. Rodrigo, and J. Duato, "An Efficient Implementation of Distributed Routing Algorithms for NoCs", NOCS, 2008.
[5] S. Borkar et al "iWarp: An Integrated Solution to High-Speed Parallel Computing", Supercomputing, 1988.
[6] S. Rodrigo et al., "Efficient Implementation of Distributed Routing Algorithms for NoCs", IET Computers & Digital Techniques, 2008.
[7] A.S. Tanenbaum, "Computer Networks", Prentice Hall, 2003.
[8] A. Gara et al., "Overview of the Blue Gene/L system architecture", IBM Journal of Research and Development, Volume 49, Number 2/3, 2005.
[9] M.E. Gómez et al.,"A Routing Methodology for Achieving Fault Tolerance in Direct Networks", Transactions on Computers, Apr. 2006.
[10] J. Flich, A. Mejía, P. López and J. Duato, "Region-Based Routing: An Efficient Routing Mechanism to Tackle Unreliable Hardware in Network on Chip", NoCS, 2007.
[11] M. Palesi, S. Kumar, R. Holsmark, "A Method for Router Table Compression for Application Specific Routing in Mesh Topology NoC Architecture", SAMOS, 2006.
[12] S. Rodrigo et al., "Yield-oriented Evaluation Methodology of Network-on-Chip Routing Implementations", SoC, 2009.
[13] M. Koibuchi, H. Matsutani, H. Amano and T. Pinkston, "A Lightweight Fault-tolerant Mechanism for Network-on-chip", NOCS, 2008
[14] A. Kohler, M. Radetzki, "Fault-Tolerant Architecture and Deflection Routing for Degradable NoC Switches", NOCS, 2009
[15] W. Song, D. Edwards, J.L. Nunez-Yanez, S. Dasgupta, "Adaptive stochastic routing in fault-tolerant on-chip networks", NOCS, 2009
[16] E. Bolotin, I. Cidon, R. Ginosar and A. Kolodny, "Routing Table Minimization for Irregular Mesh NoCs", DATE, 2007
[17] I. Loi, F. Angiolini and L. Benini, "Synthesis of Low-Overhead Configurable Source Routing Tables for Network Interfaces", DATE, 2009
[18] A. Mejía et al., "Segment-based routing: an efficient fault-tolerant routing algorithm for meshes and tori", IPDPS, 2006.
[19] F.A. Samman et al., "Planar Adaptive Router Microarchitecture for Tree-Based Multicast Network-on-Chip", NoCArc, 2008
[20] A. Pullini et al., "Bringing NoCs to 65nm", IEEE Micro, pp.75-85, 2007.
[21] E.S. Shin et al., "Round-robin Arbiter Design and Generation", in ISSS, Oct. 2002, pp. 2-4.
[22] "The Nangate Open Cell Library", 45nm FreePDK, https://www.si2.org/openeda.si2.org/projects/nangatelib.
[23] M. Martin, et. al, "Multifcet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset"', in Computer Architecture News, 2005.
[24] S.C. Woo et al, "'The SPLASH-2 Programs: Characterization and Methodological Considerations"', 22nd ISCA Conference, 1995.