Efficient Routing in Heterogeneous SoC Designs with Small Implementation Overhead

José Cano, José Flich, Antoni Roca, José Duato, Marcello Coppola and Riccardo Locatelli

Abstract—In application-specific SoCs, the irregularity of the topology ends up in a complex and customized implementation of the routing algorithm, usually relying on routing tables implemented with memory structures at source end nodes. As system size increases, the routing tables also increase in size with non-negligible impact on power, area and latency overheads. In this paper we present a routing implementation for application-specific SoCs able to implement in an efficient manner (with no routing tables and using a small logic block in every switch) a deadlock-free routing algorithm in these irregular networks. The mechanism relies on a tool that maps the initial irregular topology of the SoC system into a logical regular structure where the mechanism can be applied. We provide details for both the mapping tool and the proposed routing mechanism. Evaluation results show the effectiveness of the mapping tool as well as the low area and timing requirements of the mechanism. With the mapping tool and the routing mechanism complex irregular SoC topologies can now be supported without the need of routing tables.

Index Terms—Systems-on-Chip, Networks-on-Chip, Routing, Evaluation.

1 Introduction

As technology advances, systems-on-chip (SoC) designs become more complex with the inclusion of many IP components. Tens (and in the near future several hundreds) of elements need to be connected within the same chip, thus requiring an efficient on-chip interconnect. Usually, the system is designed taking into account the future application that will be running on the system, thus, the design is customized and adapted to the application needs. Traffic patterns are known in advance, and the interconnect is customized. The net result of such design approach is a network within the chip [1], [2] with no regular shape and with varying switch complexities and link bandwidths. Figure 1 shows a possible example where IP blocks are connected by using an on-chip network with 28 switches. The target utility of this example (and similar ones) could be a high-end home multimedia entertainment subsystem or an application processor. As can be observed, the network topology is totally irregular and heterogeneous.

Two key pillars of an interconnect are the topology and the routing algorithm. The topology sets the physical connection pattern between end nodes and, as indicated previously, in application-specific SoC systems is usually irregular and customized to the application. However, there are other topologies with regular patterns like 2D meshes, tori and fat trees, mostly used in other environments like Chip MultiProcessor (CMP) systems.

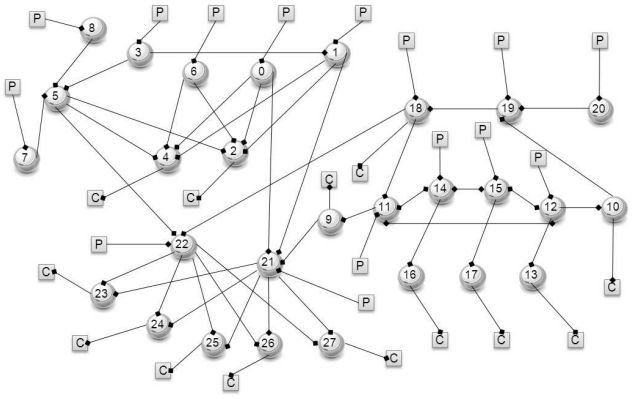

Fig. 1. Example of a complex irregular topology for an application-specific SoC system. P means producers and C means consumers.

The routing algorithm, on the other hand, sets the paths that messages need to take within the network. One important issue in the routing algorithm is the prevention of deadlock. A deadlock situation occurs when messages in the network block cyclically forever as they request resources already occupied by other messages. Since message dropping (and later retransmission) is not efficient in such systems, the routing algorithm needs to be designed carefully in order to prevent any potential deadlock situation.

Once the topology is set, then, the routing algorithm needs to be applied and messages need to be instructed about the paths to follow. In order to implement the routing algorithm two trends can be followed: source routing [3] and distributed routing [3]. In source routing, the entire path of the message is

- *J. Cano, J. Flich, A. Roca and J. Duato are with the Department of Computer Engineering (DISCA), Universitat Politècnica de València, Camí de Vera s/n, 46022 Valencia, Spain. E-mail: jocare@gap.upv.es; jflich@disca.upv.es; anrope2@gap.upv.es; jduato@disca.upv.es.*
- *M. Coppola and R. Locatelli are with STMicroelectronics, 12 Rue Jules Horowitz, 38000 Grenoble, France. E-mail: marcello.coppola@st.com; riccardo.locatelli@st.com.*

encoded in the message header, and switches simply read the header and take the corresponding output port to forward the message. In this case, the sender node has memory blocks (tables) to encode all the possible paths for every destination node. Also, as the system size increases, paths tend to increase and the packet header increases, thus sometimes requiring more network bandwidth. In distributed routing, the message header includes the destination identifier and switches are in charge of computing the appropriate output port for the message. In this situation, the length of the header is independent of the system size. However, switches need to implement the routing algorithm.

Today, few MPSoC systems are using regular topologies (like picoArray technology [4], Tilera products [5] and AsAP [6]). In these cases, a simple, yet powerful, routing algorithm (e.g. dimension-order routing (DOR) [3]) can be implemented with minimum cost (few gates) on every switch (distributed routing). Simply, the coordinates of the destination switch in the message are compared with the coordinates of the current switch.

Contrary to this, the majority of application-specific SoC systems in current products are using irregular topologies based on well-known on-chip technologies (examples are Spidergon STNoC [7], Arteris NoC [8], Sonics MicroNetwork [9] and AMBA [10]). Those irregular solutions are mainly based on source routing and address decoding, and normally need a complex implementation of the routing algorithm (with routing tables using memory structures). Indeed, the lack of regularity in the topology prevents simplifications in the routing algorithm design. As system size increases, the routing table increases in size with non-negligible impact on power, area and latency overheads (for a comparison between logic-based routing and tables, refer to [11]). In addition, in source-based routing as the system size increases the header overhead is larger and in some cases the limited amount of header information that can be included prevents reaching all the nodes.[1]

In this paper we address the implementation of the routing algorithm in application-specific SoC systems where the topology is set by the application, thus being totally irregular. The aim is to design a mechanism and an algorithm that enables the use of table-less distributed routing on every switch with a constant and reduced logic cost, regardless of system size.

The proposal builds from the LBDR (Logic-Based Distributed Routing) mechanism [11], a previous proposal suited for CMPs and high-end MPSoC systems where initial regular 2D mesh topologies are used but manufacturing defects end up inducing some irregularities in the topology. LBDR is able to implement in an efficient manner (with no need of routing tables) a routing algorithm in most of these topologies. In this paper we extend the LBDR approach to cover complex topologies derived from SoC designs, thus enabling the use of the LBDR approach in application-specific SoC systems. We also provide a tool able to map the initial irregular topology into a logical regular structure where the enhanced LBDR approach can be used. By doing this, the routing algorithm can be efficiently implemented in the SoC design with no need of routing tables and with no topology change.

The rest of the paper is organized as follows. Section 2 shows the related work with some routing solutions for irregular topologies. In Section 3 we first describe the concrete contribution of the paper in a preliminary subsection, in order to clarify and focus the description of the mechanism and the mapping tool. Then, we describe the LBDRx mechanism to cover practical topologies from SoC designs. Section 4 describes the mapping tool. Finally, in Section 5 we provide evaluation results and conclude the paper in Section 6.

2 Related work

There are considerable previous works addressing routing algorithms for irregular NoCs, which can be separated into table-based and logic-based.

In [12], a deadlock-free and highly adaptive routing solution for irregular 2D mesh-based NoCs is proposed. This solution is table-based and uses well-known principles from parallel computer architecture. However, it does not achieve 100% coverage for fully irregular topologies.

A technique based on applying a fixed routing function combined with minimal deviation tables, is described in [13]. The objective is to reduce the size of routing tables either at end-nodes or at routers. Using this methodology, three hardware efficient routing methods for irregular mesh topology NoCs are compared. For each method, path selection algorithms minimize the overall cost. However, deadlock freedom is not assured unless virtual channels are used.

In [14], authors propose a synthesis approach that, depending on the degree of routing flexibility, can significantly reduce the area cost of fully reprogrammable routing tables by adopting partially reprogrammable routing logic. This solution implements configurable routing, which allows bypassing faulty nodes or links without impacting the consistency of traffic flows.

An adaptive and stochastic routing algorithm is proposed in [15]. The algorithm supports function oriented routing and provides strong fault-tolerance to permanent errors. Each router learns the network status from acknowledgement flits and stores the outcomes in a routing table, thus, suffering the same scalability problems and routing costs related to routing tables.

1. In the evaluation section we provide an analysis of reachability of source-based routing.

FDOR [16] is a flexible routing algorithm based on dimension ordered routing, which supports a large variety of irregular mesh topologies and is based on the idea of dividing an irregular mesh topology into regular sub-meshes, a core mesh and one or more flask meshes. FDOR provides a cheap and efficient routing solution but it does not offer full coverage.

More recently, the application-specific cycle elimination and splitting (ACES) method [17] has been proposed as a scalable technique to provide deadlock-free routing in irregular NoCs. ACES uses virtual channels and has been implemented using routing tables, trying to find an optimal trade-off point between power and performance.

The uLBDR approach [18] is proposed as an efficient logic-based mechanism which offers 100% coverage to faulty 2D-meshes, being an alternative to the use of routing tables. To sum up, it achieves a good trade-off between coverage of irregular 2D mesh-derived topologies and performance of applications.

Finally, in [19], authors introduce a hybrid scheme multiphase routing algorithm for irregular 2D meshes which integrate oversized rectangle cells. The idea of this work is borrowed from fault tolerant networks, where the network topology can be rendered irregular due to faulty regions.

As a summary, none of the previous solutions allow distributed routing implementation in fully irregular NoCs topologies with no routing tables and minimum logic. In general, in application-specific designs, the topology is set without accounting for the routing algorithm. Once set, routing paths are searched and their implementation is assumed to be with routing tables, due to the excessive irregularity required by the application. The LBDRx approach, removes tables in such environment by minimizing and condensing all the routing information in a small and bounded set of bits that are finally hardwired.

3 LBDRx DESCRIPTION

The description of the proposed LBDRx mechanism will be presented as an evolution from the basic LBDR mechanism previously proposed (with low coverage for complex irregular topologies) to the most enhanced version (with full coverage to all the complex topologies analyzed).

3.1 Preliminary: Basic Idea

Prior to describing the mechanism and the mapping tool, we need, however, to briefly describe the underlying basic idea. In regular networks, e.g. a 2D mesh network, the regular connectivity pattern is useful when designing the routing algorithm. Indeed, with the Dimension-Order routing (DOR), the implementation is quite straightforward as messages are forwarded with minimal paths first in the X direction and then in the Y direction. Thus, there is no need for a routing table, only a set of gates is enough. This renders to an efficient implementation in terms of area, power, and delay.

If we consider small irregularities on 2D mesh networks, for instance due to manufacturing defects, then the inherent irregularity complicates the routing implementation. For instance, DOR is no longer valid as some paths are not possible now. However, other routing algorithms are still suitable for such topologies, for instance, topology-agnostic routing algorithms like up*/down* [20]. Their implementation is usually performed with routing tables. Efforts to provide efficient implementations of such algorithms in those irregular topologies have been performed in the recent years. One important method is LBDR, which condenses all the routing information required on every switch on a small set of bits, thus reducing significantly implementation costs. LBDR still relies on the fact that the topology is a 2D mesh network but with some missing links. Adding some bits enables LBDR to successfully deal with the irregularity induced by missing links. However, LBDR still relies on the fact that every switch has at most four links connecting neighboring switches (at North, East, West, or South directions).

LBDR uses, as DOR does, the coordinates of the destination switch in the message and the coordinates of the current switch, to compute the appropriate set of output ports. Thus, LBDR still benefits from the original 2D mesh layout.

In this paper, what we propose is the extension and applicability of the LBDR concept to truly application-specific and irregular networks (as an example see Figure 1). The approach we follow is, first, to map the irregular topology into a 2D grid (notice, however, we do not change the initial topology) providing coordinates to every switch in the grid. Once the topology is mapped, and based on the coordinates of the destination switch and the current switch, the derived LBDR logic will decide the output port that needs to be used to forward the packet towards its destination. In order to correctly map the topology into a 2D grid we have developed a mapping algorithm that will search the space of combinations and will deliver the most suitable ones, always guaranteeing deadlock freedom and connectivity.

Due to the mapping performed, and because of the high irregularity we will find, some switches will require a varying number of ports to connect to other switches, and in that situation some links will connect switches not placed closely in the 2D grid. This kind of connectivity has not been provided by the original LBDR mechanism, and thus, requires modifications. In this paper, we further extend LBDR for supporting this kind of mappings.

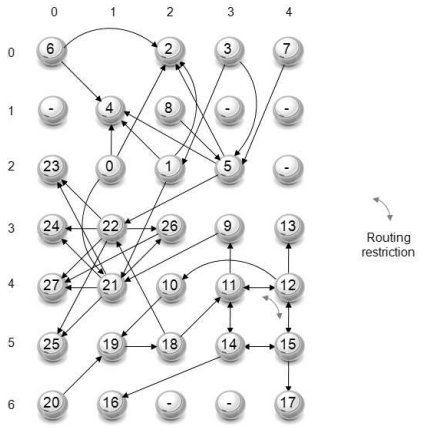

Fig. 2. Mapping example for the initial topology.

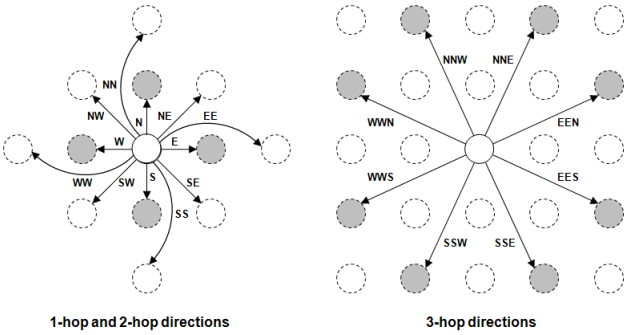

Fig. 3. Possible port directions in LBDRx.

3.2 LBDR Extension: LBDRx

We start the description with the mechanism required at every switch to deal with the irregular topologies. In order to be concise, we take as a reference the mapping of the initial topology (shown in Figure 1) that appears on Figure 2. This mapping is obtained with the mapping tool that will be described in the next section. The mapping is representative of all the connectivity patterns between switches that we need to address in this section.

As we can see in the figure, there are switches with varying connectivity patterns with other switches. For instance, switch 1, mapped at row 2 and column 2, is connected to switches 4, 2, and 21 with different link mapped lengths. [2] In particular, mapped length of links are 2 hops for links connecting to switches 2 and 4, and 3 hops for the link connecting to switch 21. In addition, links with the same number of mapped hops have different orientations/directions, thus, being different. This is the case for link connecting to switch 4 which is located one hop north and one hop west from switch 1, and link connecting to switch 2 that is two hops north.

As previously described, LBDR relies on switches with up to four ports, and each one connecting switches in one direction in the 2D mesh plane (N, E, W and S). Also, the maximum distance covered with a link is 1 hop in the 2D grid. Thus, links with higher mapping lengths are not supported, and thus, the mapped topology is not supported by LBDR.

In order to overcome this limitation, the new mechanism, LBDRx, allows switches with up to 20 ports for connecting to other switches (ports used to connect end nodes are excluded). Also, any of these ports can be configured as a 1-hop port, a 2-hop port, or a 3-hop port. A X-hop port connects two switches that are at mapping distance X. In order to uniformly refer to X-hop ports, we define additional directions. In particular, 20 different directions are supported for the ports, and each of the 20 possible ports of the switch can be configured to any of the 20 directions. The supported directions are: the initial 1-hop four directions (supported by LBDR): N, E, W, S; four 2-hop straight directions NN (two hops along the north direction), SS, EE, WW; four 2-hop diagonal directions NE (one hop north and one hop east), NW, SE, SW; and eight 3-hop directions: NNE, EEN, EES, SSE, SSW, WWS, WWN, NNW. Figure 3 shows all the possible port directions supported by LBDRx.

Notice that simplified versions of the mechanism can be conceived by restricting the type of ports that can be supported. For instance, the LDBR mechanism is embedded in the proposed mechanism when only 1-hop ports are allowed. Another implementation is allowing 1-hop and 2-hop ports only, thus obtaining a LBDR2 mechanism. Therefore, the LBDRx proposal can be seen as a method to further extend the connectivity of switches when mapped on a 2D grid. As we will see in the evaluation section, LBDR3 is enough to map all the tested topologies, thus not requiring a more complex implementation. Anyway, we will show also results for the LBDR2 approach. It is worth mentioning that although 20 ports are allowed on every switch, not all of them need to be implemented. Indeed, only a subset of ports will be implemented, e.g. switch 1 at Figure 2 will be implemented with only 3 output ports.

The logic required for LBDRx is shown in Figure 4. The mechanism relies on some configuration bits grouped in two sets: routing bits and connectivity bits. Routing bits indicate which routing options can be taken, whereas connectivity bits indicate whether a switch is connected with its neighbors. As an alternative view, the connectivity bits set the mapped topology and the routing bits set the routing algorithm. As we support 20-port switches, at maximum we will have 20 connectivity bits per switch. We represent the connectivity bit for a port X as C_x, where X can be a possible direction of any mapped port (N, E, W, S, NN, NE, ..., NNE, ...). Notice these bits will be

2. Notice that the link length is set in the original unmapped topology (Figure 1) and the showed length in the mapped topology (Figure 2) is the same, although in the grid is set by the number of hops in each direction. We will refer to physical length for original unmapped topologies, and to *mapped lengths* for link lengths in the mapping layout.

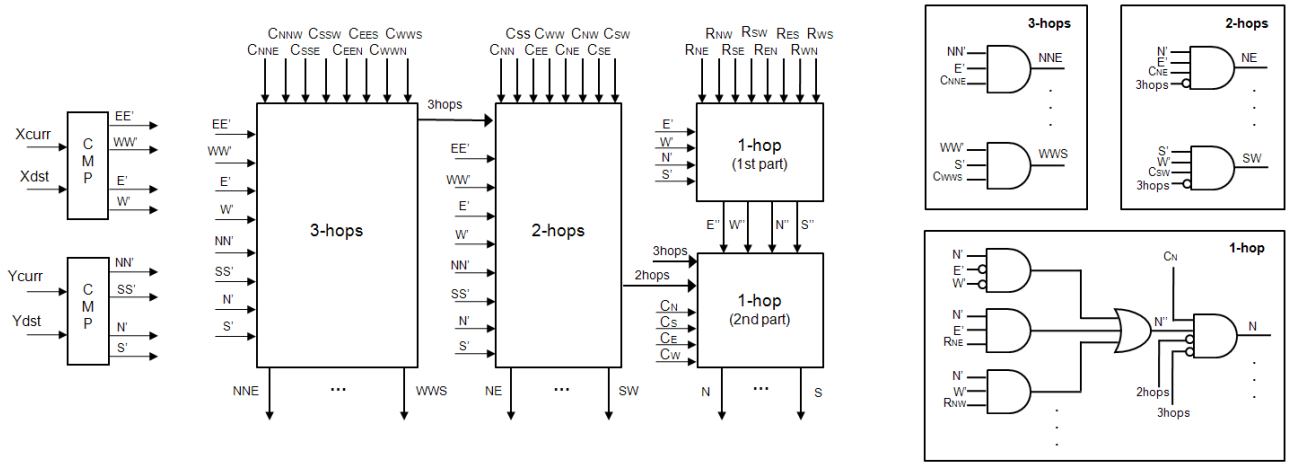

Fig. 4. Logic of LBDRx

hardwired depending on the final topology and the radix of each switch, thus not being implemented with flip-flop registers.

The routing bits R_{xy} (where x and y can be n, e, w, and s) indicate whether messages routed through the x output port may take at the next switch the y port. In other words, these bits indicate whether messages are allowed to change direction at the next switch. The value of these bits is computed in accordance to the applied routing algorithm and to prevent deadlock while still guaranteeing connectivity. In order to simplify the routing logic, however, not all the possible routing bits are implemented. Indeed, no new routing bits are used except those already defined in LBDR: R_{ne}, R_{nw}, R_{en}, R_{es}, R_{wn}, R_{ws}, R_{se}, R_{sw}. Notice that routing bits are used only between 1-hop links. By default, the LBDRx mechanism will assume messages can take 2-hop and 3-hop links without restriction along their path without risk of inducing deadlock. The mapping strategy described in Section 4 will guarantee in those cases the absence of deadlocks. Although allowing more routing bits would lead to greater flexibility, we noticed that they are not needed in order to reach our objective (shown in the evaluation section). This will also help to keep a low implementation cost of the mechanism. Indeed, the routing bits can be hardwired in the final implementation of the chip as well.

The routing logic of LBDRx (Figure 4) computes, in first place, the relative position of the message's destination (left part of the figure). For this, the coordinates of the current switch (X_{curr} and Y_{curr}) are compared with the coordinates of the message's destination (X_{dst} and Y_{dst}) located in the message header. At the output of this logic, one or more signals associated with the X and Y coordinates may be active simultaneously. For instance, if the packet's destination is in the NW quadrant then N' and W' signals are active at the same time. Signals NN, SS, EE and WW indicate whether the destination is at least 2 hops away in the given direction. Similarly, the signals of 1 hop are computed.

Notice that in some situations, signals in the same direction will be active at the same time, for instance signals NN' and N'. These cases are filtered in the second part of the logic. Higher priority will be given to larger hop ports. We refer to the signals produced by the comparators as intended *direction signals*. Note also that packets forwarded to the local port are excluded from the routing logic.

Once the direction signals are computed, the logic is divided into three parts in order to address the different type of output ports (1-hop, 2-hop, and 3-hop ports). Figure 4 shows the details, where the right part shows the details of each of the three blocks. Notice that 3-hop ports have the highest priority followed by 2-hop ports. This means that if a 3-hop output port is eligible for routing a message then, ports with lower mapping length will not be considered. To implement this priority scheme, two control signals ($2hop$ and $3hop$ signals) are used. Besides this, the logic to compute 2-hop and 3-hop ports is quite straightforward. Indeed, a port X is eligible if the port exists in the switch (C_x bit is set), and the message's destination is in the same direction of the output port (direction signals). As an example, output port NNE is eligible for routing if the message's destination is in the NNE direction (both direction signals NN' and E' are set).

The logic for 1-hop ports is, however, slightly more complex. It also deals with the routing bits. In this case, the logic is implemented with four inverters, four AND gates and one OR gate (right part of the figure). The logic filters out the routing options that could lead to deadlock situations (by using the routing bits). Figure 4 shows the logic for the North 1-hop output port. Notice that excepting for the priority signals, the logic for the 1-hop output ports is the same used in LBDR.

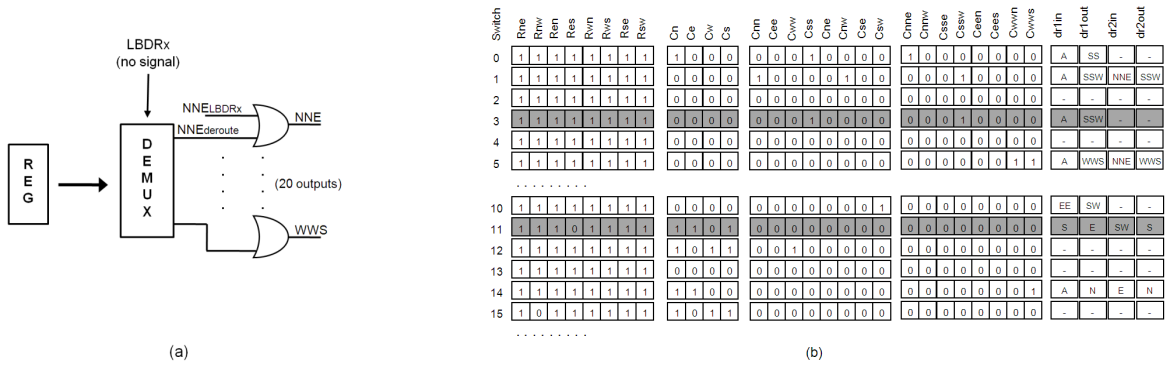

Fig. 5. (a) Logic for non-minimal path support (deroutes) in LBDRx. (b) LBDRx configuration bits (with deroutes) for the mapping shown in Figure 2. Routing bits are set to 1 when no restriction is applied; connectivity bits are set to 1 when the switch is connected to another. Deroute bits are represented with the input port (*in* column) and the deroute option for that input port (*out* column). "A" means the input port is the local one.

3.3 Support for Non-minimal Mapped Paths

The previous logic guarantees minimal path routing in the mapped topology. As each output port is used when the destination is located in the same direction of the output port, then every hop performed guarantees the message will get closer to its destination. Indeed, this fact renders with a very simple implementation of the routing algorithm (as seen in Figure 4). However, there are mapping cases that can not be solved with only minimal path support. As an example, in the mapped topology shown in Figure 2 a message going from switch 11 (mapped in row 4 and column 3) requires a mapped non-minimal path to reach switch 21, as it needs to be forwarded N and then WWS.[3] This fact simply renders the mapped topology as unsupported by LBDRx (Figure 4) as it is now.

One possible solution is to discard the mapped topology and obtain one that guarantees all the paths will follow minimal paths. Indeed, this is one of the targets of the mapping tool shown later in the paper. However, in some complex topologies, this kind of mappings will simply not exist. To solve this problem in a smooth way, and allowing much more flexibility to the mechanism, we introduce a small additional logic on every input port to allow such non-minimal path support. Going back to the non-minimal path example in Figure 2, if at switch 11 we force the message to go north, then the message will be able to reach its final destination and the mapping will be valid.

Figure 5 (a) shows the logic for the non-minimal path support we propose. The logic is derived from [18]. In particular, the logic forces messages to take a non-minimal port whenever the LBDRx logic fails in routing the message. For the example provided, at switch 11 none of the output ports will be eligible by LBDRx. Thus, in this case, the logic will take the configured non-minimal port (port N). The logic requires a 5-bit configuration register per input port (to select an output port out of maximum 20 ports). The logic uses a demultiplexer to decode the output port. Notice that deroutes are taken only if the LBDRx logic does not provide a valid output port.

As an overall example, in Figure 2 we can observe how a message being forwarded from switch 3 to switch 27 is using the configuration bits shown in Figure 5 (b). At switch 3, the message could take port SSW, or port SS. However, the 3-hop port (SSW) filters out the 2-hop port (SS). The message, then, moves to switch 1 and from that switch takes output port SSW again. Finally, at switch 21 the port W is selected. This example is straightforward, and as can be easily deduced the way the topology is mapped onto a 2D grid will influence on the applicability of the LBDRx mechanism.

Furthermore, in Figure 2 two different deroute options are required for two different input ports at router 11 (see Figure 5 (b)). If going W, and the message comes from input port S, then a deroute is set to E. On the other hand, if the message is coming from SW, and the intention is to go SSW, then a deroute is set to S. It is worth mentioning that deroute options need to be computed in accordance to the routing algorithm, as they must not introduce cycles that could lead to deadlocks.

As a final remark for the LBDRx routing mechanism, its success depends strongly on the mapping performed for the topology. Indeed, there are mappings that allow all the links to conform to the LBDRx link structure (1-hop, 2-hop, 3-hop links) and there are mappings that have larger links. Also, there are mappings that introduce deadlocks in the routing algorithms or even prevent the connectivity between particular source-destination flows. Therefore, the mapping tool, described in the next section, is a key element to guarantee applicability of LBDRx.

3. Notice, however, the path in the original unmapped topology is minimal and the same.

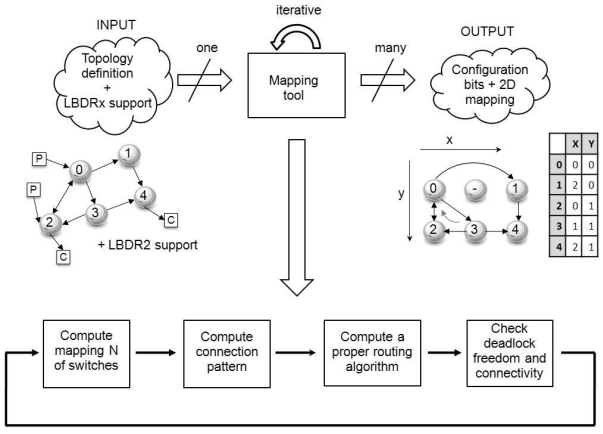

Fig. 6. Mapping tool.

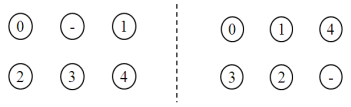

Fig. 7. Example of two initial mappings.

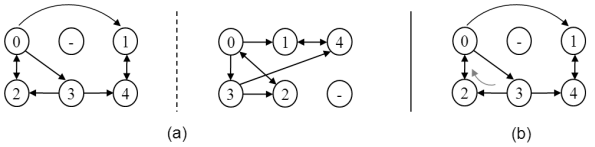

Fig. 8. Example of (a) connectivity pattern applied to two different mappings, and (b) mapped topology with the routing algorithm applied.

4 Mapping Tool

In this section we describe the mapping tool required by the LBDRx mechanism. The mapping tool can be adapted to different versions of LBDRx routing, e.g. with 3-hop links, 2-hop links, and with/without non-minimal path support. It takes as an input the physical topology and the type of LBDRx support and outputs several possible solutions, each of them able to be used with the target LBDRx version. Indeed, the tool provides for any possible solution the set of configuration bits together with the mapping coordinates of every switch into a 2D grid (Figure 6). It is worth highlighting that the mapping tool does not physically change the topology, indeed it only logically maps the topology onto a 2D grid. Figure 6 also shows the different stages of the mapping tool. For the sake of understanding, in the next subsections we describe the details of each stage along with an example.

4.1 Compute Mapping of Switches

The first stage provides an initial mapping of the switches into a 2D grid. Some basic assumptions are considered:

1) Only switches and links between switches are considered for the mapping, thus not considering end nodes.
2) The mapping grid (the diameter of the 2D mesh) will be minimized and made as square as possible (differences between diameters of each dimension will be minimized).
3) Every possible mapping onto the 2D mesh is explored, and the best solutions are extracted and further analyzed (in the following stages).

The last assumption may lead to a large number of mapping combinations, and most of them will result in mappings not supported by LBDRx. As an example, Figure 7 shows two possible mappings corresponding to the example topology provided in Figure 6. At first sight, we cannot deduce which ones can be supported. For this, we need to compute directions and link connectivity (connection pattern).

4.2 Compute the Connection Pattern

For each mapping in the previous step, the connection pattern needs to be performed. The connection pattern considers only links connecting switches (switch-to-switch links) and the direction of each link (unidirectional links are considered). Several restrictions are enforced in this step (considering LBDR2 support):

1) Any switch has at maximum 12 outgoing ports and 12 incoming ports, possibly having less number of ports, and not necessarily the same number of input and output ports.
2) In every switch one possible direction out of 12 can be taken, in the 2D mesh mapping, through a single output port. The directions are the ones supported by LBDR2 (1-hop and 2-hop links depicted in Figure 3).

Taking into account the previous restrictions, some mappings will become not valid, e.g. those with link lengths longer than the targeted LBDRx version or those that lead to unconnected networks. In any case, those mappings are excluded.

Figure 8(a) shows the connectivity pattern for the previous two mapping cases. As can be seen, the second mapping case is not compatible with LBDR2 since it contains a 3-hop link. Furthermore, we can observe how this mapping ends up in an unconnected network (once the routing algorithm is applied). The issue comes from the fact that switch 1 cannot be reached from switch 3 with the LBDR2 logic. From the point of view of switch 3, switch 1 is at the North-East quadrant but there is no North-East link nor any combination of 1-hop links (with North and East directions) leading to the destination, instead, the North-East-East exists. The first mapping case does not suffer from that problem, and thus, that case is compatible with LBDR2. Mappings leading to unconnected networks are filtered out at this stage. Notice however, that if we use LBDR3 or deroutes are allowed, both mappings are, then, supported.

4.3 Compute a Proper Routing Algorithm

Once we have obtained a correct mapping we need to check whether the mapped topology contains cycles or not. In that case, in order to avoid cycles, it will be necessary to apply a routing algorithm. In the first mapping in Figure 8(a), a cycle can be formed between switches 0, 3, and 2 in the counter clockwise direction. Applying a routing algorithm on top of the topology will remove such cycle.

In our case, the routing algorithm used is the Segment-based Routing (SR) [21], a technique that divides the network into segments and puts a routing restriction in each segment. A routing restriction is placed between two consecutive links and prevents any message from using both links sequentially. Drawing routing restrictions is a way of representing a routing algorithm since restrictions establish the allowed paths, those not crossing routing restrictions. In order to compute the routing restrictions, only 1-hop links in the mapped topology are assumed. As commented above, this assumption simplifies the LBDRx logic and still allows to reach our objective of avoiding cycles. Figure 8(b) shows the valid mapped topology in Figure 8(a) with the unidirectional routing restriction applied only at switch 2. Notice that no cycles exist without crossing the routing restriction. LBDR2 computes the routing bits from the routing restrictions defined by the routing algorithm.

4.4 Check Deadlock-Freedom and Connectivity

The last step of the mapping tool is to verify that the mapping is deadlock-free and guarantees the connectivity of the initial topology. The routing algorithm applied in the previous step ensured deadlock-freedom. However, when applying the routing algorithm and when not using deroutes, some pair of end nodes may be unconnected. Since LBDRx without deroutes relies exclusively on minimal paths (those always getting closer to its destination), a routing restriction may lead to a path being routed non-minimally, which needs the use of deroutes to be supported. At this stage, the tool iterates on all the communicating flows of the application (a flow is defined as the path from a producer to a consumer). For each flow, the tool searches a valid LBDRx path using the connectivity and routing bits set by the mapped topology. If for a flow, there is no connectivity, then the mapping is not valid and we will need either to search a new mapping or use deroutes.

Figure 9 represents the algorithm that checks deadlock-freedom property, which searches for cycles within the network. To do this, at the beginning all the input ports of switches are set with the *visited* flag reset. Then, for each possible input port cycles are searched by calling the function *fnCheckDeadlockFree*, which returns whether a cycle is found or not. If a cycle is found then the algorithm returns *False*. On the other hand, if no cycles are found for all the input ports of switches the algorithm returns *True*.

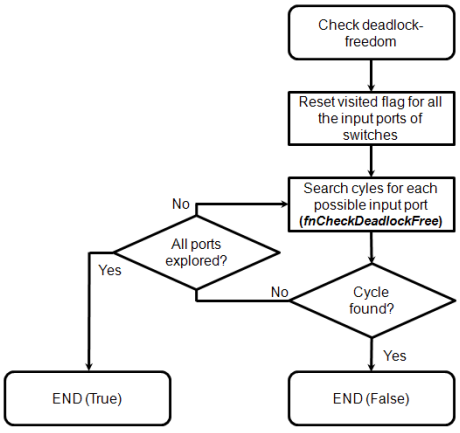

Fig. 9. Deadlock freedom flowchart.

The function *fnCheckDeadlockFree* searches all the possible paths allowed by the computed routing and connectivity bits. The allowed paths are encoded in a matrix that sets whether at a given switch and input port, the output port can be used. If so, the function recursively advances through that port. Then, all the possible output ports at the current switch are explored. If the output port can be used, then the next switch is computed, the associated input port at the next switch is computed, and the recursive function is called again. After the checking of the output ports, the input port is labeled as not visited (since the recursive function is backtracking).

Now we describe how deroutes are computed by the mapping tool. Initially, the set of routing and connectivity bits are computed. This is done by taking into account the mapped topology and the routing algorithm. Once LBDRx bits are computed, the deroute options are searched. To do this, an algorithm looks for a valid path for every producer-consumer pair (Figure 10 represents this algorithm). Note that the algorithm is computed offline before any normal operation of the chip, thus computation complexity is not a major issue. In Figure 10, we first check connectivity for each producer-consumer pair. For this, a function receives as a parameter the source switch, the final switch, and the input port at the source switch (the local port in this case). If for a particular flow there is no connectivity, then FALSE is returned and the mapping is not valid.

For each flow, all the possible paths granted by LBDRx are searched. The function for checking connectivity is recursive and finishes once the final switch is reached, or when no valid deroute option can be used. In this function, all the ports delivered by the LBDRx mechanism (including the possible deroutes already set) are retrieved. If valid output ports are provided, then the function is recursively called for every possible output port (notice LBDRx may grant

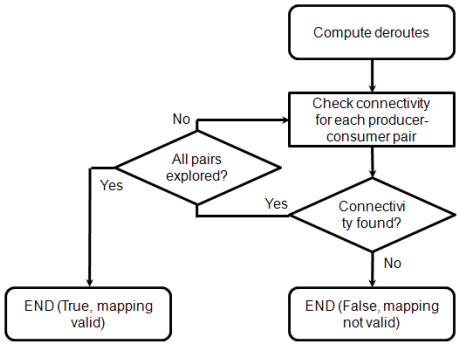

Fig. 10. Deroutes computation flowchart.

several output ports of the same mapped length). Notice that the destination should be reached through all the possible output ports provided by LBDRx. If, on the other hand, LBDRx fails to provide a valid output port, then all the possible deroute options are searched. For each possible deroute set, connectivity is checked. On success the deroute is left. On fail, the deroute is removed and another deroute is tested. If all the deroutes fail then the mapping can not provide connectivity and the function fails.

On success of a mapping topology, the final output is the mapping of each switch into the 2D grid and the configuration (connectivity, routing, and deroute) bits. Notice that many mapping solutions exists and the mapping tool succeeds if at least one mapping solution is obtained. Also, if no mapping solution exists for a grid size, the mapping tool extends the grid by one row and/or column thus having much larger flexibility. However, this will incur in larger register files at switches to store the relative position of the switch in the grid. As an example, Figure 2 shows a successful deadlock-free mapping for the initial topology depicted in Figure 1 where connectivity between switches is assured (due to its complexity, the version used in order to obtain that mapping was LBDR3 with deroutes).

5 Performance Evaluation

In this section, we provide a comprehensive evaluation of the LBDRx mechanism organized into three parts. In the first one, we analyze the mapping tool, showing the results of applying it to different sets of topologies with increasing complexity. Moreover, different mappings of the same topology are evaluated in terms of performance to check possible deviations between them (note however, the tool is focused in providing connectivity and routing implementation, instead of high performance). In the second part, our attention is focused on the switch implementation, describing its design and analyzing the implications of the LBDRx mechanism. An analysis of how the switch design evolves in latency and area for different switch radices is also provided. Finally, the third part analyzes the reachability of the source-based routing approach in comparison with the LBDRx mechanism.

5.1 Mapping Tool

5.1.1 Topologies Analysis

In order to thoroughly test the ability of the mapping tool, we have performed several groups of experiments using different sets of sample topologies with increasing complexity in each group. These sets of topologies have been generated randomly, deriving their complexity according to three input parameters: i) the number of switches, ii) the number of links, and iii) the number of Producers and Consumers. Table 1 shows the results.

	Grid size	Correct	Deroutes	Time
Type 1	3x3	>10.000	not needed	<1 min
Type 2	4x4	>10.000	not needed	3-5 min
Type 3	5x5	>100.000	not needed	10-15 min
Type 4	5x6	>100.000	not needed	30-45 min
Type 5	5x7	>100.000	10-15	1-2 hours
Type 6	6x6	>1.000.000	15-18	2-3 hours
Type 7	7x7	>1.000.000	18-23	3-5 hours
Figure 1	5x7	578.952	10-15	2 hours

TABLE 1
Topology mappings.

The main purpose was to compute the number of correct mappings generated for every analyzed topology and the time required to complete the procedure. In each case, the table shows: i) the minimum grid size needed to map the topology (3x3, 4x4, 5x5, ..., 7x7), ii) the number of correct mappings obtained and, iii) the average number of deroutes used (per mapping), if necessary. Note that correct mappings will be those which met the restrictions imposed by the LBDRx version applied in each case. In the last column, the computation time to complete the entire process is shown. This time depends mainly on the complexity of each topology, and for these examples ranges from some minutes to several hours. The last row represents the topology shown in Figure 1.

Note that with the proposed mechanism, using 20 port directions and deroutes, we were able to map all the tested topologies (with a maximum grid size up to 7x7), thus not requiring a more complex implementation. Analyzing larger grid sizes remains, if any, for future work. Notice that the mapping capability strongly depends on the topology complexity (referred previously) and the LBDRx used. As an example, if we consider a set of topologies with the same number of nodes and LBDRx, each of them may require different grid sizes to be mapped depending on its complexity. Furthermore, topologies containing few links will generate less number of valid mappings than topologies containing a great number of links (because the number of valid combinations of nodes and links is smaller).

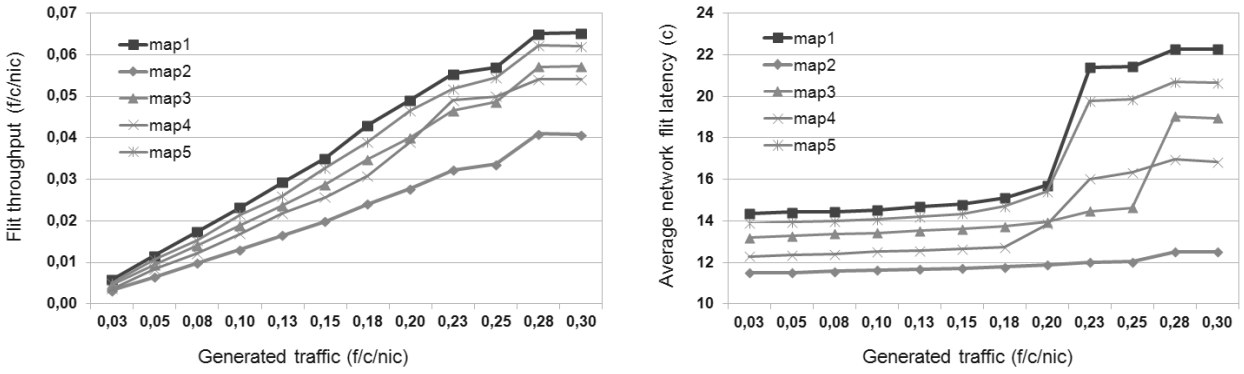

Fig. 11. Simulation of different mappings for the initial topology.

5.1.2 Mappings Analysis

The objective now is to analyze whether different mappings of the same topology can derive in different performance numbers, that is, to study if there can exist variability. The graphs shown in Figure 11 compare different mappings for the topology of Figure 1 (for the sake of clarity and understanding, only five different mappings are represented in each graph). The mappings have been evaluated into the gNoCSim simulator (an in-house cycle-accurate flit-level simulator). The left graph compares the traffic generated (in flits per cycle per nic) against the traffic actually received, while the right graph compares the traffic generated (again in flits per cycle per nic) against the average network flit latency.

As we observe, there exist differences for the different mappings in both graphs. It is important to note that although performance is limited by the irregularity of the network (topology) and not by the mapping, however, different mappings for the same irregular topology (different locations of the nodes in the 2D grid) can in fact use different LBDRx-based paths for the same Producer-Consumer pairs, thus producing different performance values. In addition, differences can increase with larger topologies because the number and length of the possible LBDRx paths between Producers and Consumers also increase.

Finally, note that having different mappings of the same topology can be useful for selecting different paths between a set of alternatives. That is, depending on the nodes location in the 2D grid and the version of LBDRx used, some routes could have preference over others. Therefore, selecting different mappings could be possible to give preference to some routes without using any additional logic or mechanism.

5.2 Switch Implementation

In this section we describe a switch design incorporating the LBDRx mechanism. We also analyze the area and latency of the switch under different port configurations and LBDRx versions. The goal is to evaluate the efficiency of the implementation and the comparison with table-based approaches.

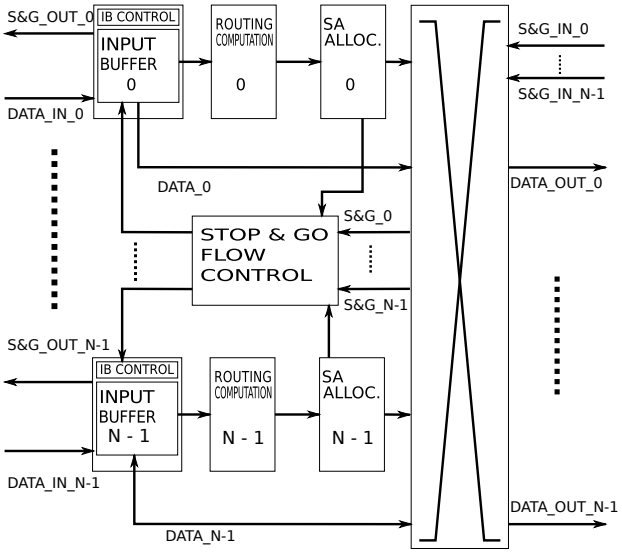

Fig. 12. Switch Design.

5.2.1 Switch Design

Figure 12 shows the main components of the used switch. The switch is a pipelined wormhole switch with four stages: input buffer (IB), routing computation (RT), switch allocator (SA), and switch traversal or crossbar (XB). Link width and flit size are set to four bytes. The input buffers can store four flits. A Stop&Go flow control protocol has been deployed in order to control the advance of flits between adjacent switches. Additionally, the routing (RT) stage has been implemented to support the LBDRx mechanism. In addition, there is an RT module for each input port and an SA module for each output port. The SA module determines when and which input port is connected to its requested output port. Finally, each SA module has been designed using a round-robin arbiter according to [22].

The switch has been implemented using the 45nm technology open source Nangate [23] with Synopsys DC. We have used M1-M3 metallization layers to perform the Place&Route with Cadence Encounter.

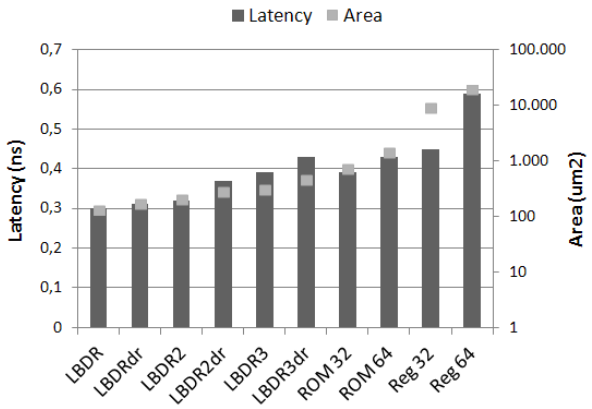

Fig. 13. Area-Delay results for the RT module with different routing mechanisms. 5-port switch design used.

5.2.2 *Router Overhead*

The goodness of the LBDRx algorithm must be performed in terms of performance but in terms of resources used. In this section we compute the area and delay of the different versions of LBDRx that have been implemented in the switch and synthesized with the 45nm Nangate opensource library. Also, the LBDRx mechanism is compared with a distributed routing approach using tables (which presents identical results as a source-based routing algorithm). In this case, each routing module has been designed with tables of 32, and 64 entries of size 24 bits[4]. Two kind of tables have been implemented. First, tables built with registers using the same design methodology than before. Secondly, an ideal ROM memory has been built by using Virtuoso Analog Design Environment by Cadence. Note that, a ROM is the simplest cell-based memory that does not admit reconfiguration, and hence, it is less flexible than LBDR. On the other hand, a ROM defines the minimum boundary in terms of area and delay when compared to other kind of cell-based memory designs.

Figure 13 shows the area and delay of the RT module of the switch. LBDR, LBDR2, and LBDR3 represent the mechanism with support for 1-hop, 2-hop, and 3-hop mapped links, respectively. Also, the plots with *dr* label represent the cases where also deroutes are used. A 5-port switch is assumed. In terms of delay (a power comparison is available in [24]) we can see LBDR3 with deroutes shows similar delay as ROM memories and register with 32 entries. However, ROMs have no flexibility at all and registers do not scale in area and delay. It can be seen that area is much larger for registers with 32 entries, doubling for registers with 64 entries. Thus, scalability is compromised.

Notice that the configuration bits of LBDRx (routing, connectivity, and deroute) can be hardwired once the mapping is fixed. This means an extra saving in area and even in latency. Figure 14 shows the area and delay for each switch stage in a 5-port switch and using the LBDR3dr solution. Results are shown for a single instance of each component. As can be seen, in both cases the values of the RT stage are smaller than the other stages. This means, the RT stage has room for the LBDR3dr solution without compromising the router frequency (set by the slowest stage). Also, we can see (by comparing with the previous figure) hardwiring the bits leads to a large saving in area (from around 700 μm^2 down to 80 μm^2).

4. 24 bits allows to code 8 hops in switches of 5 input ports.

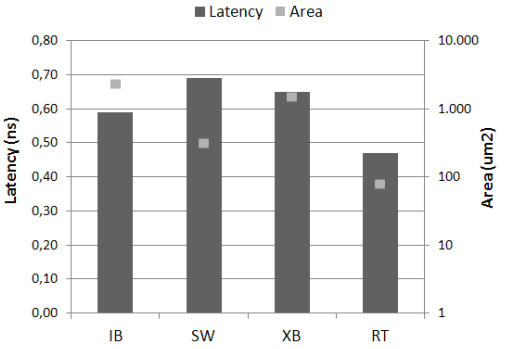

Fig. 14. Area-Delay results for the different switch stages. 5-port switch design and LBDR3dr used.

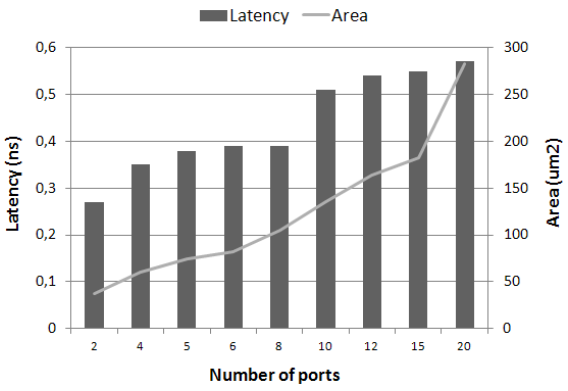

Fig. 15. Area-Delay results for the RT module with different number of I/O port switches. LBDR3dr used.

In the previous analysis, we assumed a 5-port switch design. However, in a SoC design different switch radices are used. There is a need, then, to explore the scalability of the mechanism with the number of switch ports. In order to reduce the latency impact of higher-radix switches, the slower switch stages (SA and XB) can be decoupled (as in [25], [26]). Basically, in a switch with $N_i \times N_o$ input/output ports, the XB and SA stages are designed as the sum of N_o independent modules of size $N_i \times 1$ instead of the conventional design where each task is centralized in a single module of size $N_i \times N_o$.

Assuming this optimization, Figure 15 shows the area and delay for the RT module in different switch configurations by varying the number of ports. As can seen, both area and delay grow with the increasing number of switch ports. This is due to the added

logic for each output port (the set of logic gates and the priority mechanisms between ports of different number of mapped hops). However, the increment of area is always much lower than the solution with routing tables. Indeed, for a 20-port switch, the RT module is implemented in less than 300 μm^2. Latency is also bounded to 8-port switches below 400 ps, and increases up to 550 ps for a 20-port switch.

5.3 Source Routing vs LBDRx

Finally, we provide a theoretical study that compares source-based routing with LBDRx. This analysis is made taking into account the number of routing bits used at the message header, which in some SoC systems can be bounded. The aim is to check which is the maximum number of hops that messages can make into the network by varying the number of routing bits allowed at the message header for such purposes. Based on the information provided by the ST Spidergon network [7], in all the examples analyzed we have set the maximum number of routing bits at the message header to 12.

In source-based routing, message headers are organized into groups of bits, and each group is used in a different switch. The number of groups represents the number of hops a message can make within the network. Therefore, the maximum number of hops a message can reach depends on: a) the number of total routing bits allowed in the message header and b) the number of bits needed at every visited switch. The latter depends on the number of output ports of the switch. These bits will be used to select one port in every switch. As an example, if the number of total routing bits is 6 and we have 8-ports switches, the message header is organized in 2 groups of 3 bits (we need 3 bits to address 8 ports).

In LBDRx, message headers are organized into 2 groups of bits. The first group represents the destination switch in the mapping used. The second group represents the selected consumer in the destination switch (each switch may be connected to other switches, to producers or to consumers). The number of bits needed to select the final port at the destination switch (connecting the destination consumer) is set by the switch having the largest number of consumers. As an example, if the number of total routing bits is 6 and we have up to 8-consumers per switch, the packet header will be organized with 3 bits to select the consumer and 3 bits to select the destination switch. If we have 3 bits to select the destination switch, the size of the 2D mesh is 2X4 (8 switches), and the maximum number of hops will be 5.

To analyze reachability of source-based routing (SR) and LBDRx when using a limited number of bits at the header, we have evaluated the following cases:

- *SR 8bp*: 8 bidirectional port switches (8in/8out)
- *SR 4bp*: 4 bidirectional port switches (4in/4out)

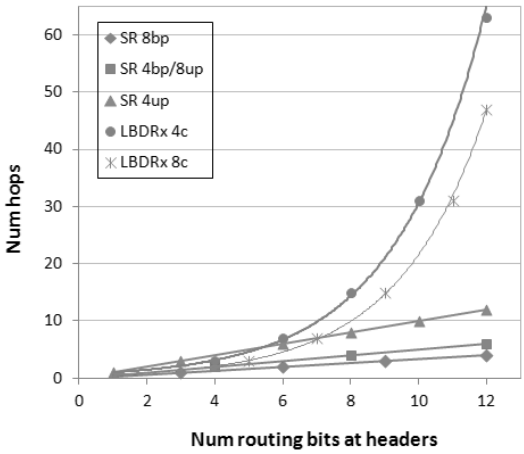

Fig. 16. Reachability comparison between source routing and LBDRx.

- *SR 8up*: 8 unidirectional port switches (4in/4out)
- *SR 4up*: 4 unidirectional port switches (2in/2out)
- *LBDRx 4c*: 4 consumers per switch at maximum
- *LBDRx 8c*: 8 consumers per switch at maximum

Figure 16 shows the number of hops for the previous cases. For each case, we observe how the maximum number of hops that can be reached in the network depends on the number of routing bits available. As an example, if we consider *SR 8bp* with 3 routing bits, the number of hops we can reach is only 1, because we need the 3 bits to select one of the 8 available ports on the switch. But if we consider that the maximum number of routing bits is 12, we can reach up to 4 hops, since it takes 3 bits for each new switch reached.

The area between *SR 8bp* and *SR 4bp* represents the range of hops we can reach considering a mix of 8- and 4-bidirectional port switches in the network. Depending on the total number of available routing bits, more or less hops can be achieved. The area between *SR 8up* and *SR 4up* represents the same but considering unidirectional ports. As we see, the maximum number of hops that can be achieved with 12 routing bits considering source-based routing is 12. In contrast, we see that with LBDRx (*LBDRx 4c* and *LBDRx 8c*), the number of hops is much larger. For example, with 12 routing bits we can reach 64 hops, thus allowing for a larger system.

The conclusion in the comparison is clear. With LBDRx, the number of hops a packet may take is much larger than when using source-based routing with a bounded number of header bits. Although a relative high number of hops can be reached with source-based routing with 4-unidirectional port switches, this corresponds to unrealistic switch designs (a switch has only two output ports). More practical switches (e.g. 8 bidirectional port switches) have a much lower reachability when using source-based routing, less than 10 hops in all the cases.

6 Conclusion

In this paper, we have presented two major contributions. In first place, LBDRx, a series of routing mechanisms (with support to non-minimal paths) for application-specific SoC systems where the topology is totally irregular. The main goal of LBDRx is to enable the use of table-less distributed routing on every switch with a constant and reduced logic cost, regardless of system size. LBDRx builds from the Logic-Based Distributed Routing (LBDR) mechanism, a previous proposal suited for CMPs and high-end MPSoC systems. In second place, we presented a mapping tool able to obtain different mappings of the same irregular topology onto a 2D mesh. The tool is key for the application of the LBDRx mechanism, so we have included a detailed description of its most important algorithms.

Finally, the evaluation results demonstrate the applicability of the mapping tool onto a wide set of topologies. In all cases, a valid mapping was achieved, thus routing tables were replaced by the LBDRx mechanism. Implementation costs also showed the benefits of such replacement. As future work we plan to further explore the mapping tool focusing on performance issues. As we have seen, different mappings can end up in different performance numbers. Thus, we plan to optimize the tool to provide the best mapping for the target application.

Acknowledgments

This work was supported by the Spanish MEC and MICINN, as well as European Comission FEDER funds, under Grant CSD2006-00046. It was also partly supported by the COMCAS project (CA501), a project labelled within the framework of CATRENE, the EUREKA cluster for Application and Technology Research in Europe on NanoElectronics.

References

[1] L. Benini and G. De Micheli, *Networks on Chips: Technology and Tools*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006.

[2] J. Flich and D. Bertozzi, *Designing Network On-Chip Architectures in the Nanoscale Era*. Boca Raton, FL, USA: CRC Press, Inc., 2010.

[3] J. Duato, S. Yalamanchili, and N. Lionel, *Interconnection Networks: An Engineering Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002.

[4] A. Duller, D. Towner, G. Panesar, A. Gray, and W. Robbins, "Picoarray technology: The tool's story," in *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*. IEEE Computer Society, 2005, pp. 106–111.

[5] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C. Miao, J. F. Brown III, and A. Agarwal, "On-chip interconnection architecture of the tile processor," *IEEE Micro*, vol. 27, no. 5, pp. 15–31, September/October 2007.

[6] Z. Yu, M. Meeuwsen, R. Apperson, O. Sattari, M. Lai, J. Webb, E. Work, T. Mohsenin, M. Singh, and B. M. Baas, "An asynchronous array of simple processors for dsp applications," in *IEEE International Solid-State Circuits Conference, (ISSCC '06)*, Feb. 2006, pp. 428–429.

[7] M. Coppola, M. D. Grammatikakis, R. Locatelli, G. Maruccia, and L. Pieralisi, *Design of Cost-Efficient Interconnect Processing Units: Spidergon STNoC*. Boca Raton, FL, USA: CRC Press, Inc., 2008.

[8] I. Arteris, "Arteris noc," http://www.arteris.com/, 2010, [Online; accessed 31-August-2010].

[9] D. Wingard, "Micronetwork-based integration for socs," in *In Proceedings of the 38th Design Automation Conference*, 2001, pp. 673–677.

[10] A. Ltd., "The advanced microcontroller bus architecture (amba)," http://www.arm.com/products/system-ip/amba/, 2010, [Online; accessed 01-September-2010].

[11] J. Flich, S. Rodrigo, J. Duato, S. Medardoni, and D. Bertozzi, "Efficient implementation of distributed routing algorithms for nocs," in *IET Computers and Digital Techniques*. IET, 2009, pp. 460–475.

[12] M. K. F. Schafer, T. Hollstein, H. Zimmer, and M. Glesner, "Deadlock-free routing and component placement for irregular mesh-based networks-on-chip," in *Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design (ICCAD '05)*. IEEE Computer Society, 2005, pp. 238–245.

[13] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "Routing table minimization for irregular mesh nocs," in *Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07*, april 2007, pp. 1 –6.

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

[15] W. Song, D. Edwards, J. L. Nunez-Yanez, and S. Dasgupta, "Adaptive stochastic routing in fault-tolerant on-chip networks," in *Proceedings of the 2009 3rd ACM/IEEE International Symposium on Networks-on-Chip*, ser. NOCS '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 32–37.

[16] T. Skeie, F. Sem-Jacobsen, S. Rodrigo, J. Flich, D. Bertozzi, and S. Medardoni, "Flexible dor routing for virtualization of multicore chips," in *Proceedings of the 11th international conference on System-on-chip*, ser. SOC'09, 2009, pp. 73–76.

[17] J. Cong, C. Liu, and G. Reinman, "ACES: application-specific cycle elimination and splitting for deadlock-free routing on irregular network-on-chip," in *Proceedings of the 47th Design Automation Conference*, ser. DAC '10, 2010, pp. 443–448.

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

[19] X. Duan and Y. Li, "A multiphase routing scheme in irregular mesh-based nocs," in *Proceedings of the 2011 Fourth International Symposium on Parallel Architectures, Algorithms and Programming*, ser. PAAP '11, 2011, pp. 277–280.

[20] D. Gelernter, "A dag-based algorithm for prevention of store-and-forward deadlock in packet networks," *IEEE Trans. Comput.*, vol. 30, pp. 709–715, October 1981.

[21] Flich, J., Mejia, A., López, P., and Duato, J., "Region-based routing: an efficient routing mechanism to tackle unreliable hardware in networks on chip," in *1st ACM/IEEE Int. Symp. Networks on Chip (ISNOC)*, 2007.

[22] E. Shin, I. Mooney, V.J., and G. Riley, "Round-robin arbiter design and generation," in *System Synthesis, 2002. 15th International Symposium on*, October 2002, pp. 243 – 248.

[23] "The nangate open cell library, 45nm freepdk," Available from http://www.si2.org/openeda.si2.org/projects/nangatelib.

[24] S. Rodrigo, J. Flich, J. Duato, and M. Hummel, "Efficient unicast and multicast support for cmps," in *Microarchitecture, 2008. MICRO-41. 2008 41st IEEE/ACM International Symposium on*, 2008, pp. 364 –375.

[25] A. Roca, J. Flich, F. Silla, and J. Duato, "A low-latency modular switch for CMP systems," *Microprocessors and Microsystems*, Aug. 2011. [Online]. Available: http://dx.doi.org/10.1016/j.micpro.2011.08.011

[26] F. G. Villamón, M. E. Gómez, S. Medardoni, and D. Bertozzi, "Improved utilization of noc channel bandwidth by switch replication for cost-effective multi-processor systems-on-chip," in *NOCS*, 2010, pp. 165–172.

José Cano received the MS and PhD degrees in computer science from the Universitat Politècnica de València, Spain, in 2004 and 2012, respectively. He was a member with the Networking Research Group (2005-2012) and also with the Parallel Architectures Group (2009-2012) in the Department of Computer Engineering at the Universitat Politècnica de València. He joined the ARCO Research Group in the Department of Computer Architecture at the Universitat Politècnica de Catalunya (Barcelona, Spain) in March 2012, where he was a postdoctoral researcher until December 2013. Currently, he is a research associate with the CArD group in the Institute for Computing Systems Architecture, School of Informatics, at The University of Edinburgh, United Kingdom. His research interests include computer architecture and computer systems, with special emphasis on processor microarchitecture, hw/sw co-designed systems, and compilers, plus multiprocessor systems-on-chip, networks-on-chip, large-scale heterogeneous multi-cores, parallel programming models, and ubiquitous computing.

Antoni Roca received the M.S. degree and the Advanced Studies Diploma, both in telecommunications engineering from the Universitat Politècnica de València, Valencia, Spain, in 2006 and 2007, respectively. Currently, he is pursuing the Ph.D. degree from the Parallel Architectures Group, Universitat Politècnica de València. His current research interests include network-on-chip architectures, especially router implementation.

José Duato received the M.S. and Ph.D. degrees in electrical engineering from the Universitat Politècnica de València, Valencia, Spain, in 1981 and 1985, respectively. Currently, he is a Professor with the Parallel Architectures Group, Department of Computer Engineering, Universitat Politècnica de València, 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 Hyper-Transport 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).

José Flich received the M.S. and Ph.D. degrees in computer science from the Universitat Politècnica de València, Valencia, Spain, in 1994 and 2001, respectively. He joined the Department of Computer Engineering, Universitat Politècnica de València, 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).

Marcello Coppola received the Laurea degree in computer science from the Pisa University, in 1992. Previously, he was in the architecture group at the INMOS Bristol (UK). Currently, he is working for STMicroelectronics within the a Research LAB. His current research interests are discrete event simulation, SoC modeling and architecture, programming languages, RTOS, concurrent programming and software/hardware engineering tools . He is responsible for R&D programs in real-time hardware, and software developement techniques.He is contributing to SystemC standardization. He has published several papers in the areas of system modeling. He has chaired international conferences on SoC design and helped to organize several others. He is a member of OSCI and is contributing to MEDEA+ roadmap.

Riccardo Locatelli received the Laurea degree (summa cum laude) in electronic engineering, and the Ph.D. degree in information engineering from the University of Pisa, Pisa, Italy, in 2000 and 2004, respectively. In 1999, he was a Research Intern with the Microelectronics Section of the European Space Agency, the Netherlands, and a Visiting Researcher with the Advanced Search Technology Grenoble Laboratory of STMicroelectronics, Grenoble, France, in 2003. At Pisa University, he worked on definition and prototyping of video architectures with emphasis on low-power techniques and system communication. He was a Digital Design Engineer with CPRTEAM, a microelectronic design house in Pisa, where he worked on advanced signal processing schemes for VDSL applications. Since 2004, he has been a Technical Leader and an Architecture and Design Team Leader with STMicroelectronics, Grenoble, France, and the Spidergon ST Network-on-Chip (SSTNoC) Interconnect Processing Unit (IPU), where he introduced novel concepts beyond networks-on-chip (NoC). He has published about 30 papers in international journals and conference proceedings and has filed nine international patents on NoC. He is the co-author of a book on Spidergon STNoC technology (Boca Raton, FL: CRC Press, 2008). Dr. Locatelli is a member of the technical program committee the NoC Symposium and the Design, Automation and Test in Europe, a Reviewer of the IEEETransactions on Computer-Aided Design journal and of several International Conferences.