Efficient Routing Implementation in Complex Systems-on-Chip Designs

José Cano
Parallel Architectures Group
Universitat Politècnica de València, Spain
jocare@gap.upv.es

José Flich
Parallel Architectures Group
Universitat Politècnica de València, Spain
jflich@disca.upv.es

José Duato
Parallel Architectures Group
Universitat Politècnica de València, Spain
jduato@disca.upv.es

Marcello Coppola
STMicroelectronics
Grenoble, France
marcello.coppola@st.com

Riccardo Locatelli
STMicroelectronics
Grenoble, France
riccardo.locatelli@st.com

ABSTRACT

In application-specific SoCs, the irregularity of the topology ends up in a complex implementation of the routing algorithm, usually relying on routing tables implemented with memory structures. As system size increases, the routing table increases 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 (without requiring routing tables and using a small logic block in every switch) a 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 on the mapping tool as well 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 use of routing tables.

Categories and Subject Descriptors

C.2.2 [**Processor architectures**]: Other Architecture Styles —*Heterogeneous (hybrid) systems*; C.2.2 [**Computer Communication Networks**]: Network Protocols—*Routing protocols*; D.2.8 [**Software Engineering**]: Metrics—*Performance measures*

General Terms

Design, Experimentation

Keywords

Systems-on-Chip; Networks-on-Chip; Routing

.

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.

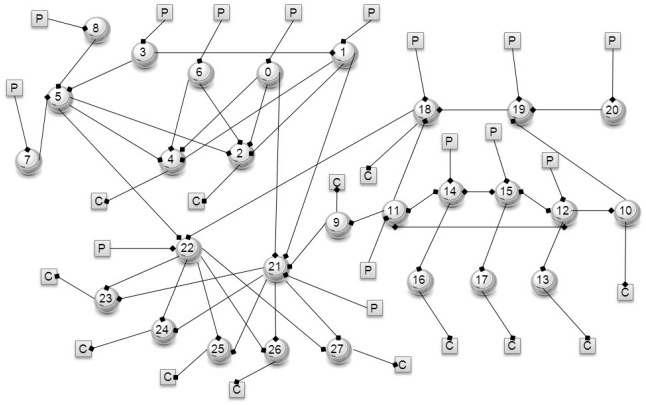

Figure 1: Example of a complex irregular topology for an application-specific SoC system. P means producers and C means consumers.

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. 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.

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 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. DOR routing [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 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 (without requiring 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. In Section 4 we describe the mapping tool. Finally, in Section 5 we provide evaluation results and conclude the paper in Section 6.

2. RELATED WORK

There has been considerable work on routing algorithms for irregular NoCs. In [12], authors use well-known principles from parallel computer architecture to develop a deadlock free highly adaptive solution for irregular mesh-based NoCs. Bolotin et al. [13] propose three hardware efficient routing methods that combine a fixed routing function (such as XY routing) and reduced size routing tables based on distributed and source routing techniques. For each method, they developed path selection algorithms that minimize the overall cost. Finally, in [14], authors propose a synthesis approach that, depending on the degree of routing flexibility, can reduce very significantly the area cost of the configurable source routing tables by adopting partially reprogrammable routing logic instead of fully reprogrammable tables. Configurable routing provides intelligent adaptation without impacting the consistency of traffic flows.

None of the solutions shown allow the implementation of distributed routing algorithms in 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. Our approach tries to remove tables in such environment by minimizing and condensing all the routing information in a small and bounded set of bits.

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 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* [15]. 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 collapses 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 to map the topology into a 2D grid (notice, however, we do not change the initial topology). Once the topology is mapped, we need to provide coordinates to every switch in the network. 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 its support in this kind of mappings.

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.

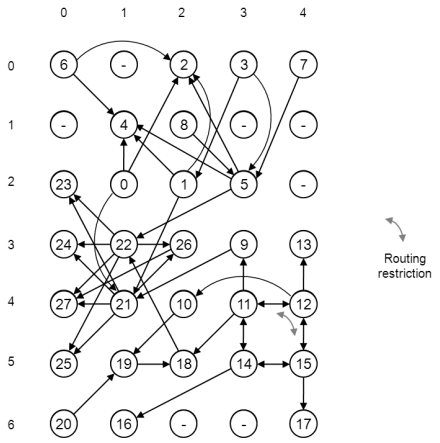

Figure 2: Mapping example for the initial topology.

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. [1] 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, the 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 for 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.

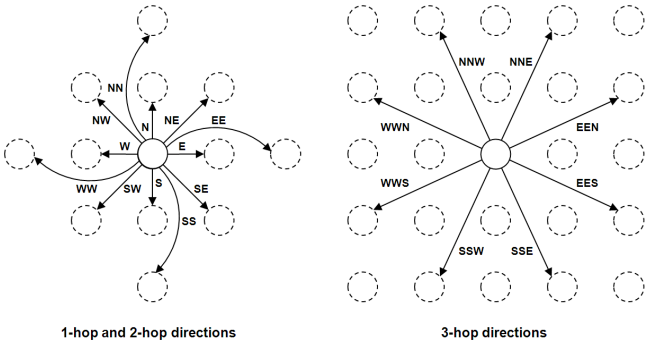

Figure 3: Possible directions in LBDRx.

Simplified versions of the mechanism can be conceived by restricting the type of ports that can be supported. For instance, a 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

[1] It is worth mentioning 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.

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. As an example, the mapped topology in Figure 2 requires some ports being 3-hops (link from switch 1 to switch 21), thus not being suitable for an 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.

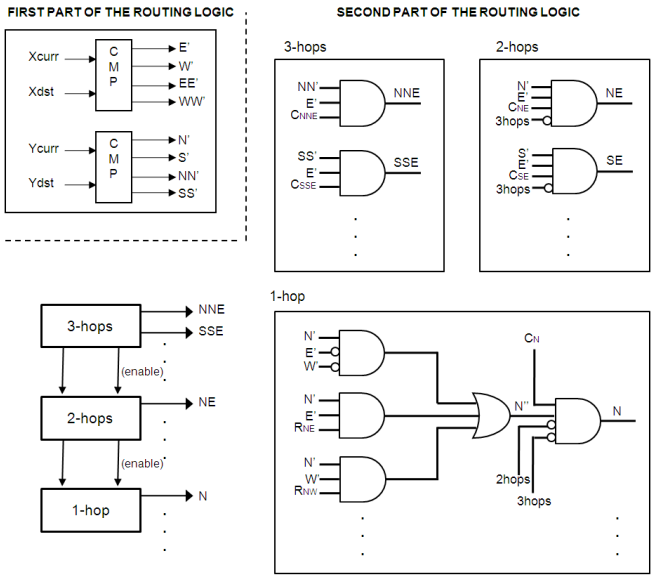

Figure 4: Logic of LBDRx.

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 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.

Routing logic of LBDRx is divided into two parts (see Figure 4). The first part of the logic computes the relative position of the message's destination. For this, two comparators are used and 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 two signals may be active (e.g. if the packet's destination is in the NW quadrant then N' and W' signals are active at the same time). Note also that packets forwarded to the local port are excluded from the routing logic.

Additionally, four extra signals, NN', EE', WW' and SS' are computed. These signals are set to one if the message's destination is at least two hops away in the corresponding direction (if NN' is active, then at least two hops must be performed in the N direction to get closer to its destination). These signals can be easily computed with additional comparators with the X_{curr} and Y_{curr} coordinates shifted in one position. Notice that in some situations different signals 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*.

Once the direction signals are computed, the second part of the logic comes into play. It consists of a logic at each input port in the switch. Figure 4 shows the details. The logic is divided in three parts in order to address the logic for the different type of output ports (1-hop, 2-hop, and 3-hop ports). 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 (2*hop* and 3*hop* 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 deals also with the routing bits. The logic, in this case (excluding the priority signals) is implemented with two inverters, four AND gates and one OR gate. 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. As an example, the incoming message is forwarded to the North output port (signal N" is set) if either one of the following three conditions is met:

- The message's destination is on the same column ($N' \times \overline{E}' \times \overline{W}'$).
- The message's destination is on the NE quadrant and the message is allowed to take the E port at the next switch through the N port ($N' \times E' \times R_{ne}$).
- The message's destination is on the NW quadrant and the message can take the W port at the next switch through the N port ($N' \times W' \times R_{nw}$).

Obviously, connectivity bits and priority signals are used in combination with the previous logic. The logic for the 1-hop output ports is the same used in LBDR. For a deeper description of LBDR, refer to [11].

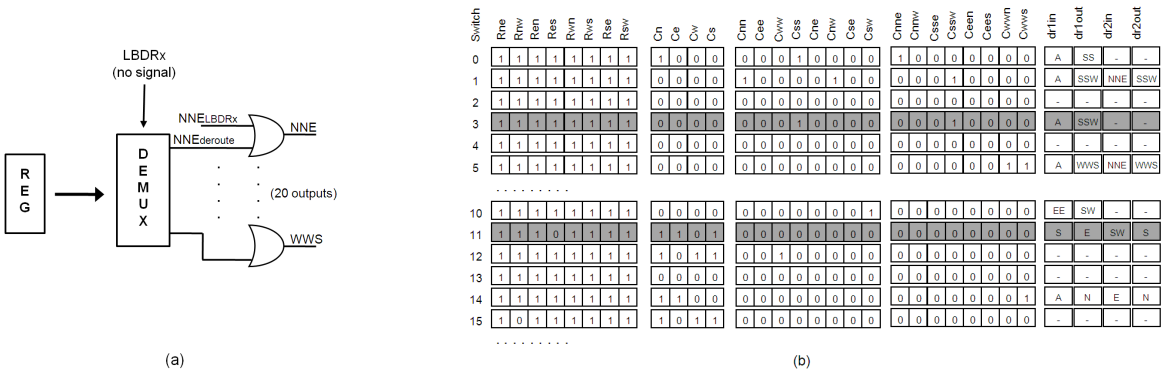

Figure 5: (a) Non-minimal path support for LBDRx. (b) LBDRx configuration bits (with deroutes) for the mapping shown in Figure 2 (Routing bits are 1 when no restiction is applied; Connectivity bits are 1 when the switch is connected to another. Deroute bits are "A" when the flow departs from itself).

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.[2] This fact simply renders the mapped topology as unsupported by LBDRx (Figure 4).

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. We evaluate this issue later. 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. Indeed, going back to the non-minimal path example in the figure, 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 support we propose. The logic is borrowed from [16]. 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. In this case, the selected port is N. The logic requires a configuration register per input port, in our case, of size 5 bits (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 the deroute option needs to be computed in accordance to the routing algorithm, as they must not introduce cycles that could lead to deadlocks. For example, if we would have a routing restriction South-East at router 11 in Figure 2, a deroute option at input port S at router 11 can not be set to E as it would let messages crossing the routing restriction. In this example, the only restriction is located at router 12.

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 the LBDRx mechanism.

4. MAPPING TOOL

In this section we describe the mapping tool required by the LBDRx mechanism. The mapping tool is adapted to different versions of LBDRx routing, e.g. with 3-hop links, 2-hop links, and with/without non-minimal path support. The mapping tool takes as an input the 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 mapping tool (see Figure 6) provides for any possible solution the set of configuration bits together with the mapping coordinates of every switch into a 2D grid. 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 7 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.

[2]Notice, however, the path in the original unmapped topology is minimal and the same.

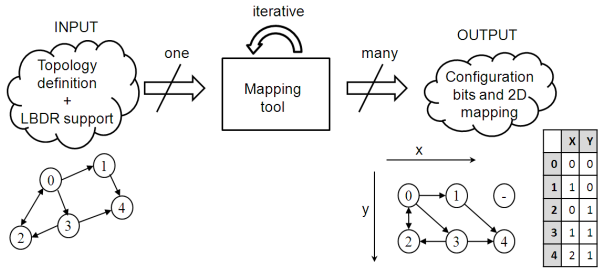

Figure 6: Mapping tool.

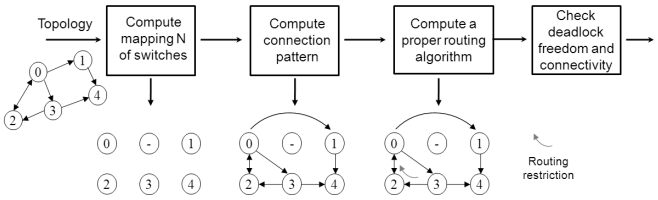

Figure 7: Stages in the mapping tool.

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 links and 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 of switches onto the 2D mesh is explored, and the best solutions are extracted and further analyzed (in the following stages).

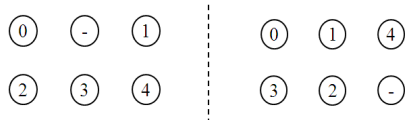

Figure 8: Example of two initial mappings.

The last assumption may lead to a large number of mapping combinations, most of them will result in mappings not supported by LBDRx. As an example, Figure 8 shows two possible mappings corresponding to the example topology provided in Figure 6. Considering LBDR2, at first sight we cannot deduce which one can be supported. For this, we need the next step: computing 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):

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 (2-hop and 1-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.

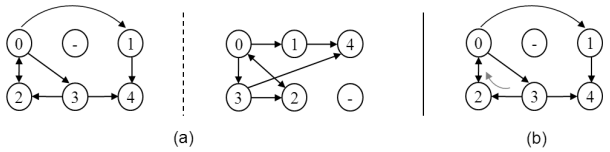

Figure 9: Example of (a) connectivity pattern applied to two different mappings, and (b) mapped topology with the routing algorithm applied.

Figure 9(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 9(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) [17], 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 9(b) shows the valid mapped topology in Figure 9(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 and Connectivity

The last step of the mapping tool is to verify 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. The reason is because LBDRx without deroutes relies exclusively on minimal paths, those always getting closer to its destination (the comparator modules in the first part of the logic enforce this property). Therefore, 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.

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).

Now we describe how deroutes are computed. Once LBDRx 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 LBDRx 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 LBDRx if all the possible paths reach the destination. In the search of all paths, whenever it fails to provide a valid path, then, a deroute action is needed.

At this stage and at the switch where LBDRx is not providing a valid output port, the tool tries all the possible deroute options available, one per output port but avoiding U turns. 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 mapping is discarded. Notice that several deroute options may be required for a single path.

5. EVALUATION

In this section, we provide a comprehensive evaluation of LBDRx. Firstly, we show the results of applying the mapping tool to different sets of topologies with increasing complexity. In second place, all the LBDRx versions are compared between them and with an example using routing tables from an implementation cost and efficiency viewpoint, with a glance at scalability properties (a power comparison is available in [18]). Finally, two different mappings of the same topology are evaluated in terms of performance to check possible deviations between them.

5.1 Mapping Tool Analysis

Table 1 shows the results of some initial experiments performed in order to test the mapping tool. To this end, we used several sets of sample topologies (with increasing complexity in each type) corresponding to high-end home multimedia entertainment (sub)systems. The mapping tool was run in AMD Opteron (2,8 Ghz dual core, 8Gb RAM) computers.

	Grid size	Correct	Deroutes	Time
Type 1	4x4	>10.000	not needed	3-5 min
Type 2	5X5	>100.000	not needed	10-15 min
Type 3	5X6	>100.000	not needed	30-45 min
Type 4	5X7	>100.000	10-15	1-2 hours
Figure 1	5X7	578.952	13-15	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 the minimum grid size needed to map the topology (4x4, 5x5, ..., 5x7), the number of correct mappings obtained and 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.

5.2 LBDRx vs Routing Tables

LBDRx versions were designed and synthesized with 45nm Nangate opensource library. Also, comparisons with routing tables, by using Memaker, are provided. In Figure 10, we observe that the differences between all the LBDRx versions are slight in terms of both area and delay. When compared with a RAM memory of 256 entries, the LBDRx version are much compact (notice the logarithmic scale). For delay, although similar, the LBDRx versions have lower access latency.

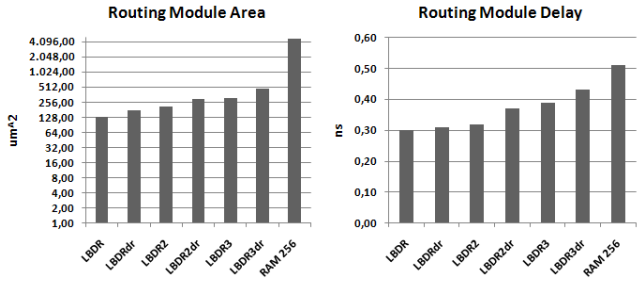

Figure 10: Area-Delay comparison tests.

5.3 Performance Analysis

Finally, the following graphs compare two different mappings of the same topology being simulated into gNoCSim simulator (a cycle-accurate flit-level simulator). We can observe that differences are small. The left graph compares the traffic generated against the traffic actually received. These values are almost equal until the network is saturated and received packets remain constant. The right graph compares the traffic generated against the average network flit latency.

Although we get small differences regarding performance, however, having different mappings can be useful for select-

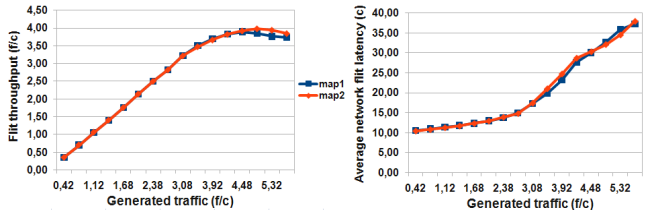

Figure 11: Simulation of different mappings.

ing a different path between a set of alternatives. That is, depending on the nodes location in the 2D mesh and the version of LBDRx used, some routes could have preference over others. In other words, selecting different mappings could be possible to give preference to some routes without using any additional logic or mechanism. This is left, however, to future work.

6. CONCLUSIONS

In this paper we have presented LBDRx, a series of routing mechanisms (with support to non-minimal paths) for application-specific SoC systems where the topology is totally irregular. Moreover, we presented a mapping tool able to obtain different mappings of the same topology onto a 2D mesh. The main goal of LBDRx is to enable the use of tableless 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. A tool to effectively map irregular topologies onto a 2D grid has been proposed. The tool is key for the application of the LBDRx mechanism.

The provided 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 LBDRx mechanism and the mapping tool, focusing on performance issues. Different mappings will end up in different performance numbers. Thus, we plan to optimize the tool to provide the best mapping for the target application.

Acknowledgment

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. Finally, the authors would like to thank Antoni Roca for his assistance in the Area-Delay comparison tests.

7. 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] Bolotin, E., Cidon, I., Ginosar, R., and Kolodny, A., "Efficient routing in irregular topology nocs," Israel Institute of Technology, Technion Department of Electrical Engineering, Tech. Rep. CCIT 554, 2005.

[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] 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.

[16] 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.

[17] 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.

[18] 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*, nov. 2008, pp. 364 –375.