



Universidad Politécnica de Valencia
Departamento de Informática de Sistemas y Computadores

# Cost Effective Routing Implementations for On-chip Networks

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy
(Computer Science)

*Author*
Samuel Rodrigo Mocholí

*Advisor*
José Flich Cardo
Valencia, 2010

# Acknowledgements

First of all, I would like to start with some thoughts. This has been a long journey (or short, depending who I ask, but time is always relative) but it is only the first step of another journey. And this is have been possible thanks to Pepe, my advisor, which gave me the opportunity around 3 years ago to start developing a new concept idea. His counseling, support and top of it, patience and guidance, have been very useful all the path along. I can not forget the colleagues who have also supported this research, in many ways. First to all my lab colleagues and professors from GAP (specially Fede, Crispin, Blas and Toni), and the italian guys, Davide, Simone and Daniele from Ferrara and Maurizio from Catania.

Of course, thanks to my family (specially to my mother Suni who has also achieved her objective not so long ago and my father Jose for his incredible support and caring), for the support all these years and the opportunity to follow a path which it was a dream years before, and hope my brother Diego achieves also his path. I love you. I have to thank you also to my closest friends for every moment of fun, dialogue and good memories (Albert, Carla, Manolo, Eduardo, Salva, Juan Ángel, Vicente, Ignacio, Su'ad, Pablo, Raquel, Lara, Néstor, Carlos, Javier, Jesús, Ainara, Xavi, Alberto, Jordi, Jon, Koldo, José María and so many more). And a final thank you to you, Mona, watching over me as the best friend I had in many years.

# Contents

# List of Figures

# List of Tables

# Abstract

Current many-core architectures like Chip Multiprocessors (CMPs) and Multiprocessor System-on-Chips (MPSoCs) rely on the effectiveness of the on-chip network (NoC) for inter-core communication. An effective NoC has to be scalable while meeting tight power, area, and latency constraints. 2D mesh topologies are usually preferred for general-purpose NoC designs as they fit the chip layout. However, designers must address new emerging challenges. The increased probability of manufacturing defects, the need for an optimized use of resources to enhance application-level parallelism or the need for efficient power-aware techniques may break the regularity in those topologies. In addition, collective communication support is a desired feature to effectively address communication needs from cache coherence protocols. Under these conditions, efficient routing of messages becomes a challenge.

The objective of this dissertation is to lay the foundations of a new logic-based distributed routing architecture that is able to adapt to any irregular topology derived from a 2D mesh structure, thus providing full coverage for any topology pattern induced by any of the challenges mentioned above. And this take is done, first by starting from the grounds of a concept idea, then looking through an evolution of several mechanisms and finally, arriving to a final implementation that encompasses several modules accomplishing the objective mentioned before. In fact, this last implementation is named $eLBDR$ (effective Logic-Based Distributed Routing), but the study will span from the first mechanism, $LBDR$, to the next mechanisms that have been emerged progressively, describing them in detail accompanied with evaluations and results to show a cost/applicability trade-off analysis.

Referring to the full architecture, $eLBDR$ presents area, latency and power

consumption requirements that are comparable to the most efficient solutions in routing mechanisms like Dimension-Order-Routing (DOR), reflected on real router implementations designed with first attempts of bringing ideas that are still underutilised in the NoC domain, like virtual cut-through switching. NoC scenarios modelled after link variability analysis show a 100% coverage achievement of the full mechanism in all scenarios that were configured. So, it is fair to assume that *eLBDR* is prepared to face the new challenges present in the NoC research field. *eLBDR* can be used as an effective fault-tolerant mechanism in CMP and MPSoC systems with defective components at the NoC level, aggressive power-down techniques switching off entire irregularly shaped NoC regions can be designed as the remaining network topology is still supported by *eLBDR*, virtualization of the chip (mapping applications to disjoint paths) can be also achieved with *eLBDR* by defining disjoint network regions, and finally, *eLBDR* allows the use of broadcast communication primitives to support effective cache coherency protocols. Broadcast is allowed inside a region in *eLBDR* and previous alternatives, thus implementing multicast support at the chip level.

In short, the objective of the concept idea is to offer an alternative to the use of routing tables (either at routers or at end-nodes). Although the use of routing tables at routers is extremely flexible, it does not scale in terms of latency, area, and power consumption. As described in the next chapters, all mechanisms require a small set of configuration bits, thus being more compact than large routing tables implemented in memories. More important, from the first mechanism to the most recent developed, the requirements of any of them, do not grow with system size, thus providing scalability.

# Resumen

Arquitecturas de múltiples núcleos como multiprocesadores (CMP) y soluciones multiprocesador para sistemas dentro del chip (MPSoCs) actuales se basan en la eficacia de las redes dentro del chip (NoC) para la comunicación entre los diversos núcleos. Un diseño eficiente de red dentro del chip debe ser escalable y al mismo tiempo obtener valores ajustados de área, latencia y consumo de energía. Para diseños de red dentro del chip de propósito general se suele usar topologías de malla 2D ya que se ajustan a la distribución del chip. Sin embargo, la aparición de nuevos retos debe ser abordada por los diseñadores. Una mayor probabilidad de defectos de fabricación, la necesidad de un uso optimizado de los recursos para aumentar el paralelismo a nivel de aplicación o la necesidad de técnicas eficaces de ahorro de energía, puede ocasionar patrones de irregularidad en las topologías. Además, el soporte para comunicación colectiva es una característica buscada para abordar con eficacia las necesidades de comunicación de los protocolos de coherencia de caché. En estas condiciones, un encaminamiento eficiente de los mensajes se convierte en un reto a superar.

El objetivo de esta tesis es establecer las bases de una nueva arquitectura para encaminamiento distribuido basado en lógica que es capaz de adaptarse a cualquier topología irregular derivada de una estructura de malla 2D, proporcionando así una cobertura total para cualquier caso resultado de soportar los retos mencionados anteriormente. Para conseguirlo, en primer lugar, se parte desde una base, para luego analizar una evolución de varios mecanismos, y finalmente llegar a una implementación, que abarca varios módulos para alcanzar el objetivo mencionado anteriormente. De hecho, esta última implementación tiene por nombre $eLBDR$ (effective Logic-Based Distributed

Routing). Este trabajo cubre desde el primer mecanismo, $LBDR$, hasta el resto de mecanismos que han surgido progresivamente, describindolos en detalle, junto con las pertinentes evaluaciones y resultados para mostrar los anlisis de costes y aplicabilidad.

En el caso de la arquitectura completa, $eLBDR$, se obtienen unos requisitos de área, latencia y consumo de energía que son comparables a soluciones de encaminamiento tan eficientes como Dimension-Order-Routing (DOR), quedando reflejado en implementaciones reales de routers diseñadas con conceptos que siguen estando infrautilizados en del dominio de redes dentro del chip, como encaminamiento virtual cut-through. Pruebas hechas sobre instancias de redes dentro del chip modeladas a partir de un análisis de variabilidad en enlaces muestran un logro en el 100% de cobertura de este mecanismos en todas las configuraciones. Por lo tanto, es razonable suponer que $eLBDR$ est preparado para enfrentarse a los nuevos desafíos presentes en el campo de la investigación de redes dentro del chip. $eLBDR$ es un mecanismo eficaz capaz de soportar tolerancia a fallos en soluciones multiprocesador (CMP y MPSoC) que tienen componentes defectuosos a nivel de red, preparado para implementar tcnicas agresivas de apagado selectivo de regiones irregulares dentro de la red ya que la topología todavía es completamente encaminable, con proyección para soportar virtualización del chip (desde mapeo de aplicaciones a caminos disjuntos) con definicin de regiones disjuntas dentro de la red, y, por último, promueve las primitivas de comunicación colectiva para apoyar protocolos efectivos de coherencia de cach. $eLBDR$ (y alternativas anteriores) permite la comunicacin broadcast dentro de una región, lo que se puede traducir en un soporte de comunicacin multicast a nivel de chip.

En resumen, el objetivo de esta idea conceptual es ofrecer una alternativa a la utilización de tablas de encaminamiento (ya sea en los routers o en los propios nodos). Aunque el uso de tablas de encaminamiento en routers es extremadamente flexible, no escala en términos de latencia, área y el consumo de energía. Como se describe en los capítulos siguientes, todos los mecanismos sólo requieren un pequeño conjunto de bits para su configuración, de forma que conseguimos ms compactación que usando las tablas de encaminamiento implementadas en memorias. Además, desde el primer mecanismo hasta el último, se cumple que los requisitos de cualquiera de ellos, no crecen con el

tamaño del sistema, proporcionando una buena escalabilidad.

# Resum

Arquitectures de múltiples nuclis com a multiprocessadors (CMP) i solucions multiprocessador per a sistemes dins del xip (MPSoCs) actuals es basen en l'eficàcia de les xarxes dins del xip (NoC) per a la comunicació entre els diversos nuclis. Un disseny eficient de xarxa dins del xip ha de ser escalable i al mateix temps obtindre valors ajustats de àrea, latència i consum d'energía. Per a dissenys de xarxa dins del xip de propòsit general es sol usar topologies de malla 2D ja que s'ajusten a la distribució del xip.

No obstant aix, l'aparició de nous reptes ha de ser abordada pels dissenyadors. Una major probabilitat de defectes de fabricació, la necessitat d'un ús optimitzat dels recursos per a augmentar el parallelisme a nivell d'aplicació o la necessitat de tècniques eficaces d'estalvi d'energia, pot ocasionar patrons d'irregularitat en les topologies. A més, el suport per a comunicació collectiva és una característica buscada per a abordar amb eficàcia les necessitats de comunicació dels protocols de coherència de cau. En estes condicions, un acarrerament eficient dels missatges es convertix en un repte a superar.

L'objectiu d'esta tesi és establir les bases d'una nova arquitectura per a acarrerament distribut basat en lògica que és capa d'adaptar-se a qualsevol topologia irregular derivada d'una estructura de malla 2D, proporcionant així una cobertura total per a qualsevol cas resultat de suportar els reptes mencionats anteriorment. Per a aconseguir-ho, en primer lloc, es partix des d'una base, per a després analitzar una evolució de diversos mecanismes, i finalment arribar a una implementació, que comprén uns quants mòduls per a aconseguir l'objectiu mencionat anteriorment. De fet, esta última implementació té per nom *eLBDR* (effective Logic-Based Distributed Routing). Este treball cobrix des del primer mecanisme, *LBDR*, fins a la resta de mecanismes que

han sorgit progressivament, descrivint-los en detall, junt amb les pertinents avaluacions i resultats per a mostrar les anàlisis de costos i aplicabilitat.

En el cas de l'arquitectura completa, *eLBDR*, s'obtenen uns requisits d'àrea, latència i consum d'energia que són comparables a solucions d'acarrerament tan eficients com Dimension-Order-Routing (DOR), quedant reflectit en implementacions reals de routers dissenyades amb conceptes que continuen estant infrautilitzats en del domini de xarxes dins del xip, com a acarrerament virtual cut-through. Proves fetes sobre instàncies de xarxes dins del xip modelades a partir d'una anàlisi de variabilitat en enllaos mostren un èxit en el 100% de cobertura d'estos mecanismes en totes les configuracions. Per tant, és raonable suposar que *eLBDR* està preparat per a enfrontar-se als nous desafiaments presents en el camp de la investigació de xarxes dins del xip. *eLBDR* és un mecanisme efica capa de suportar tolerància a fallades en solucions multiprocessador (CMP i MPSoC) que tenen components defectuosos a nivell de xarxa, preparat per a implementar tècniques agressives d'apagat selectiu de regions irregulars dins de la xarxa ja que la topologia encara és completament suportada a nivell d'acarrerament, amb projecció per a suportar virtualització del xip (desde mapeig d'aplicacions a camins disjuntos) amb definició de regions disjuntes dins de la xarxa, i, finalment, promou les primitives de comunicació collectiva per a recolzar protocols efectius de coherència de cau. *eLBDR* (i alternatives anteriors) permet la comunicació broadcast dins d'una regió, la qual cosa es pot traduir en un suport de comunicació multicast a nivell de xip.

En resum, l'objectiu d'esta idea conceptual és oferir una alternativa a la utilització de taules d'acarrerament (ja siga en els routers o en els propis nodes). Encara que l'ús de taules d'acarrerament en routers és extremadament flexible, no escala en termes de latència, àrea i el consum d'energia. Com es descriu en els capítols segents, tots els mecanismes només requerixen un xicotet conjunt de bits per a la seua configuració, de manera que aconseguim més compactació que usant les taules d'acarrerament implementades en memòries. A més, des del primer mecanisme fins a l'últim, es complix que els requisits de qualsevol d'ells, no creixen amb la grandària del sistema, proporcionant una bona escalabilidat.

# Chapter 1

# Introduction

*"You are about to take the first steps of the longest journey of your life."*

The Longest Journey.

In this chapter, we first introduce the reasons that have motivated this dissertation (Section 1.1). Then, we briefly define the specific objectives aimed by the dissertation (Section 1.2). Finally, we outline the structure of the remaining chapters in this document (Section 1.3).

## 1.1 Motivation

Nowadays, Chip Multiprocessor (CMP) is accepted as the design paradigm by the largest microprocessor manufacturers. CMP chips are made of multiple cores in the same die and, as technology advances, more cores are expected to be included. Typically, a core includes a processor and some cache structures. As an example, Intel has recently developed a research chip with 48 cores, each being $x86$ compatible, under the Tera-scale Computing Research Program [7]. Previously, under the same research program, Intel also manufactured a chip prototype [8] that included 80 cores (known as the TeraFlops Research chip) but with very simple cores (2 FPUs). It is well accepted in the community that processor performance will be increased in the following years by including more cores in the same die.

In addition to CMPs, multicore solutions for System-on-Chip (MPSoCs) are appearing in the embedded system market. Tilera [9] offers several high-end SoCs where multicore computing platforms provide support for a wide range of computing applications, including advanced networking, high-end digital multimedia, wireless infrastructure, and cloud computing. The most recent product by Tilera offers 100 cores in the same chip.

CMPs and high-end MPSoCs rely on an on-chip network to handle all the communication traffic between cores. Initial chip designs with few cores included buses and rings as the communication subsystem. Such solutions are well suited for chips with a small number of cores, like the Cell processor [23]. However, as the number of cores scales up, the bus structure suffers from lack of enough bandwidth. Thus, designers evolved to more scalable solutions. The most recurring solution is the 2D mesh topology (and alternatively, the 2D torus topology) as it fits the chip layout. In a 2D mesh, each core is attached to a network router and routers are laid out in a 2D structure where each router is connected to four routers at most, each one in each direction and dimension.

Good properties of the 2D mesh topology include its regularity and simplicity for routing. All the links have the same length, thus exhibiting the same latency (with the same number and type of repeaters). Also, applications with local traffic patterns obtain a good performance as message latency is low (few routers and links to cross). However, one of the inconveniences of the 2D mesh structure is its relatively high hop count for messages travelling to distant nodes. Fortunately, this impact is reduced when using wormhole and virtual cut-through switching strategies (where hop count is additive to latency, not multiplicative).

A routing strategy in a network manages the way messages are routed from a sender to a receiver. There are many routing algorithms and strategies. One of the most used is Dimensional-Order-Routing (DOR; also called $XY$ when applied to 2D mesh topologies) [64] because of its simplicity, but the main drawback of DOR is that it does not support any irregularity in the network.

At the same time, tile-based design is gaining momentum for CMP and high-end MPSoC systems. In a Tiled CMP (Figure 1.1) the tile is designed in isolation and, once designed, the chip is finally built by replicating tiles. By

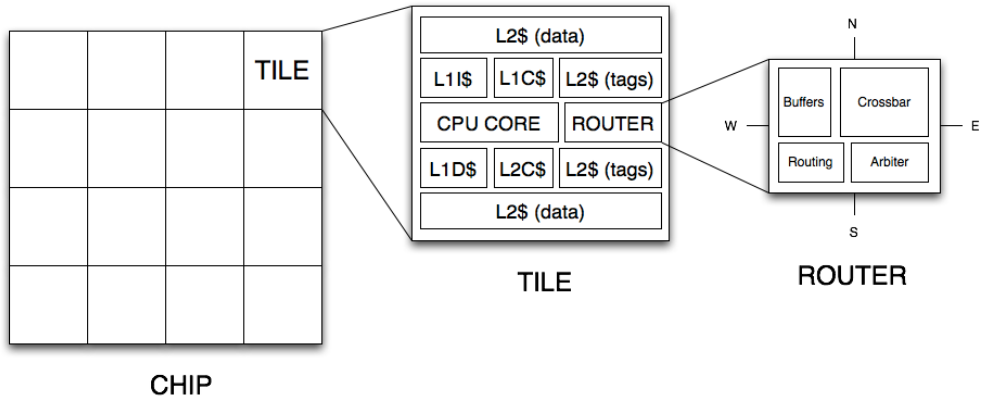



Figure 1.1: Tiled CMP.

doing so, the effort to design, test, and build the chip is drastically reduced. Tiled CMPs also advocate for regular network structures like 2D meshes. Typically, each tile incorporates a processor core, a private L1 cache memory (data and instruction cache), and a bank of the L2 shared cache, distributed over the entire chip (each tile adds a slice of the cache). In addition, a directory structure, and optionally, a memory controller may be allocated in each tile. The directory reduces the overhead introduced by the cache coherency protocol between L1 and L2 caches. The memory controller is in charge of accessing external memory. A router is also included to enable communication among tiles.

Due to the reasons mentioned above, we advocate for homogeneous 2D meshes in Tiled CMPs and high-end MPSoCs. However, even if the design of a CMP chip with a 2D mesh network is correct, the on-chip network may face new challenges leading to non-regular heterogeneous topologies. Several challenges have been identified that may severely impact the homogeneity of the network in the years to come: manufacturing defects, effective chip utilization, voltage/frequency islands, and aggressive power saving techniques. In the following paragraphs we briefly introduce those challenges.

As technology advances, correct manufacturing becomes challenging and defective components will become frequent. As a consequence, a defective tile in the chip, if not addressed, will ruin the 2D mesh structure of the network, leading to an irregular network that cannot be handled by the routing algo-

rithm, thus rendering the chip useless. An example can be seen in Figure 1.2, where the east link between routers 10 and 11 has failed. Unless the routing mechanism is prepared to reroute a message around the defective link, it would be impossible for both routers to communicate.

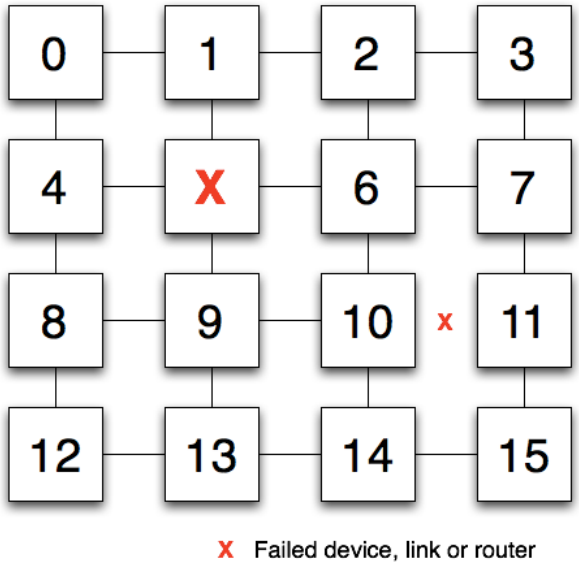



Figure 1.2: Some network components have failed.

Another challenge is the problem of not extracting enough parallelism from applications so to efficiently use tens and hundreds of cores in future chips. As a solution, the chip can be partitioned into multiple domains, each one running a different application (or serving a different customer). In this scenario, and in order to fit as many applications as possible, the partition of the chip resources may lead to irregularly shaped domains. The way applications are usually mapped onto the chip is known as virtualization, where a real chip is partitioned into several smaller virtual chips. In Figure 1.3 an example is shown. Three applications, $A0$, $A1$ and $A2$ are already mapped on the chip using a different number of nodes. If a new application named $A3$ needs only three nodes to perform its task it could then be mapped on the nodes attached to routers 8, 9 and 12, grouping them into an irregular region. If such irregular mapping is not allowed, then $A3$ will need to wait the completion of $A0$, $A1$ or $A2$. The objective is to minimize any fragmentation by allowing irregular

patterns.

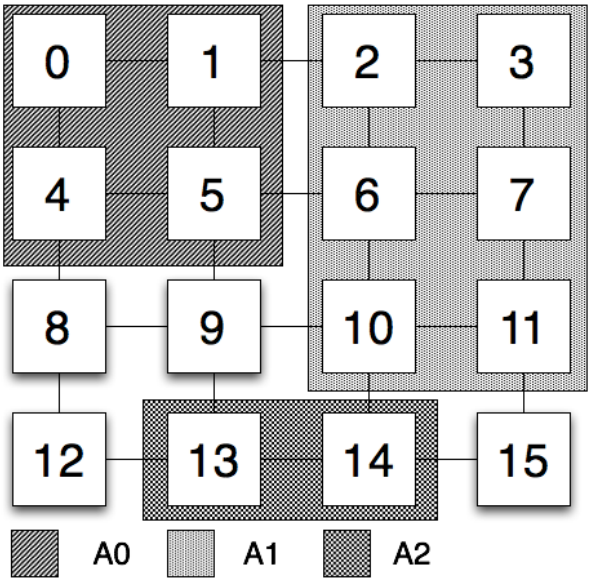



Figure 1.3: Allocation of applications in groups of nodes.

To reduce (if possible, minimize) power consumption on the chip, current trends on NoC design shift from completely synchronous systems to GALS (Globally Asynchronous, Locally Synchronous) communication models [66] as clock lines can use up to 50% of the total power budget [39]. This leads to the occurrence of voltage/frequency islands that introduce a new challenge regarding on-chip communication. In such scenario the isolation of such regions may be required so as to avoid conflicting mismatches in voltage and frequency that may lead to bottlenecks, e.g. a message crossing different domains. Indeed, due to the high integration scale, chip manufacturing is becoming more affected by variability inducing these islands. Let us see an example in Figure 1.4, where the chip has been divided into three isolated regions due to different voltage thresholds. Isolation, even with overlapped regions, is a feature that will be requested on future chip designs for multiple purposes.

As a major challenge for chip design, there is a clear need to introduce efficient aggressive power saving methods. As the number of cores increases, probably many of the cores will remain unpowered (in sleep mode) most of the time, thus achieving large savings in power consumption. The same strategy

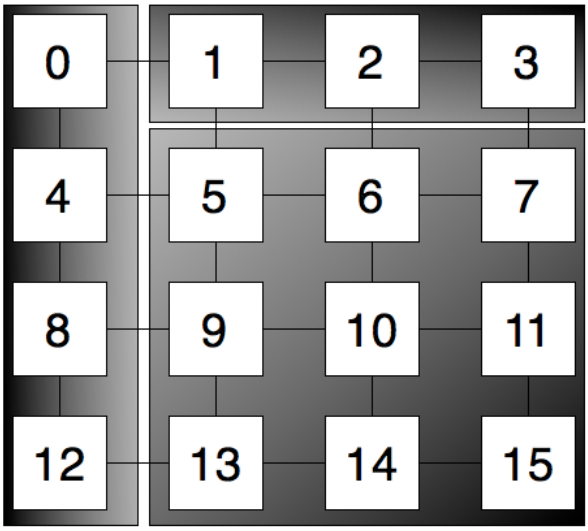



Figure 1.4: Voltage/frequency islands.

should be applied to the on-chip network, which has been reported [69] to consume around 30% of the total chip power consumption. Powering routers off and on will lead to temporary irregularities in the topology. An example is present in Figure 1.5. Tiles 1, 6 and 10 are idle, and as a result of a power-aware technique implemented, they are also powered down until requested by another computation task. This is translated into an irregular topology with the rest of nodes that are still active and need an effective routing layer capable of handling the communication between them.

Finally, in addition to the previous issues, support for collective communication in CMPs is also needed. Collective communication primitives are required either for implementing barrier synchronization or to support coherency traffic: a broadcast/multicast mechanism accelerates higher level entities like cache coherence protocols (either directory-based [28] or token-based [32]) or new techniques (virtual hierarchies [34]). Collective communication in CMPs may also be needed for an effective management of the NoC (control/management messages). Although there are solutions for collective communication in NoCs (see Chapter 2) either they do not support irregular topologies or their implementation is costly. An example of this is shown in Figure 1.6 where some links have failed and node 4 wants to perform a broad-

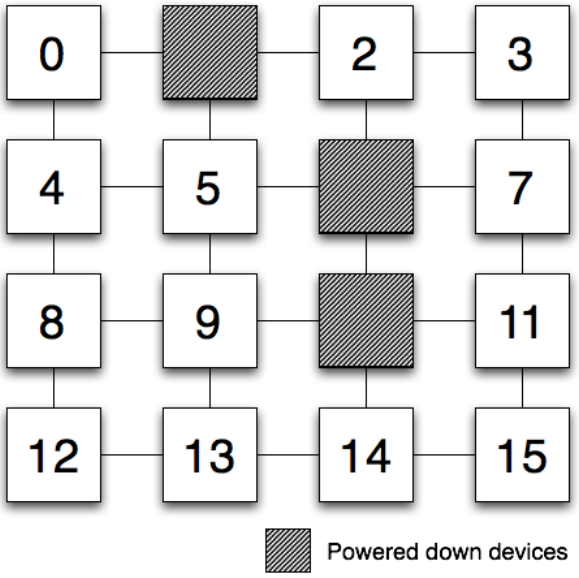



Figure 1.5: Some cores are powered down.

cast operation (i.e. send the message to all nodes visiting them only once), so it spreads across the network following a tree-based path supported by the routing layer. This challenge does not only comprehend delivering support for collective communication in presence of failed devices, but it also requires traffic isolation support as broadcast/multicast operations should be contained to the region/domain they belong to (if virtualization concepts are applied), thus not flooding other regions in the chip.

It is important to note that addressing all the previous challenges requires some effort at the on-chip network level and that will drive the way how future routing algorithms, implementations and techniques will be implemented. In particular, the most important thing to address is providing efficient support for irregular network topologies. Indeed, all the challenges mentioned above can be correctly addressed by deploying a routing mechanism able to deal with any of the derived topologies.

Current solutions that advocate for irregular topologies are based on routing tables. The basic mechanism consists either on implementing look-up tables at every end node (when using source-based routing), or forwarding tables at every router (when using distributed-based routing). For every destination,

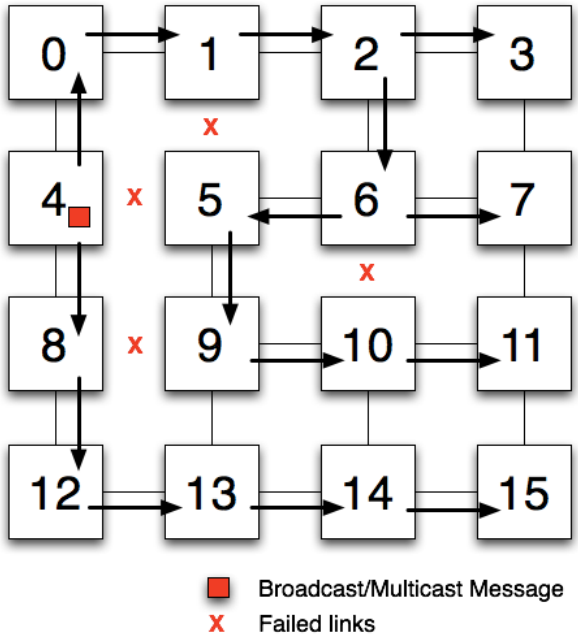



Figure 1.6: Multicast operation example with temporary failed links.

there is a path or output port associated, respectively to each kind of routing, that the message follows, or chooses, to arrive to its destination. Let us see an example in Figure 1.7. In source-based routing, if a message had node 10 as a source and node 0 as destination, the path to follow, $N - N - W - W$ (north, north, west, west), would be coded on the message header (this will be explained later). In distributed routing, there is no path coded at the message header, just the destination, so on every router, the routing table is accessed to see the output port the message has to take on its way to its destination. In the example, at router 10, for destination 0, the output port chosen is $N$ (north).

The main advantage of table-based routing is that any topology and any routing algorithm can be used, including fault-tolerant routing algorithms. However, as routing tables are implemented with memories, they do not scale in terms of latency, power consumption, and area, thus being impractical for large NoCs [50]. Indeed, a routing table with as many entries as the number of nodes and input ports is needed in the worst case, with the possible addition

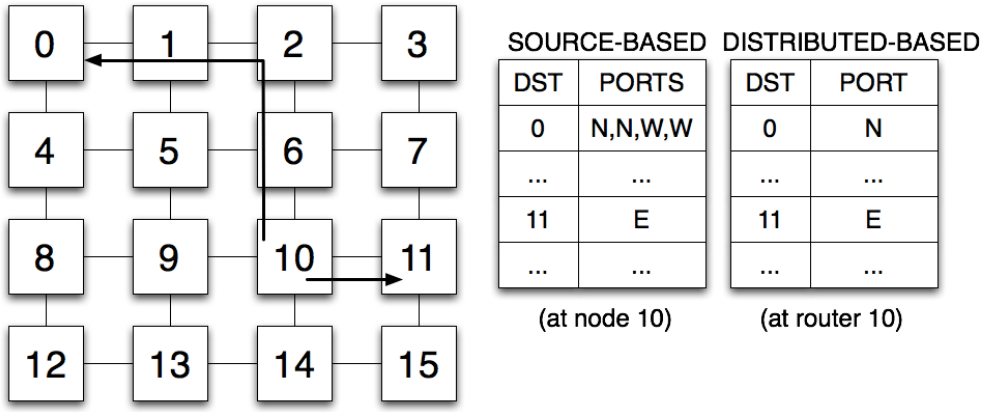


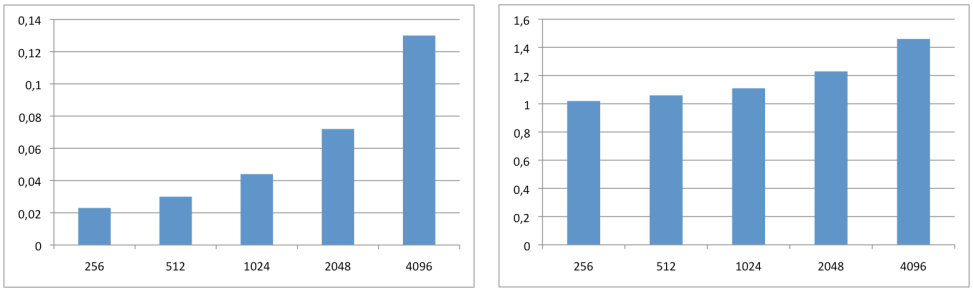

Figure 1.7: Routing tables example.

that every entry needs to store different output ports returned by the routing algorithm. Hence the cost of this implementation is $N \times d \times d$, where $N$ is the number of nodes and $d$ is the number of ports. If we assume minimal routing (only ports that get the message closer to its destination are provided) the cardinality is reduced to two at maximum, so the cost is translated to $N \times 2 \times d$. There are algorithms that do not require the use of the input port, this, in such cases memory requirements can be $N \times 2$. Anyway, in the entire network, memory requirements grow quadratically ($N$ node with $N$ entries each).

An example of poor scalability of tables can be seen in Figure 1.8 which shows the synthesis of memory macros with 90nm technology obtained with Memaker[1] [6]. As can be seen in Figure 1.8(a), there is an exponential increase of area with respect to the number of entries (table size). This is worsened by the fact these results are obtained by modelling the memory block for one tile, so the impact is multiplied for the total area related of the chip. Latency results are less significative, but the trend is clear. Indeed, execution time of applications may be affected. Also, the increase on router delay could change the critical path of the router leading to lower performance.

Another study [47] performs a comparison between a DOR implementation and an implementation with routing tables on a $8 \times 8$ NoC mesh. The

[1]Memaker (from Faraday Technology Corporation) produces memory macro models on UMC Logic LL-RVT (LowK) Process technology.




(a) Area ($mm^2$) (b) Latency (ns)

Figure 1.8: Area and latency for memory macros/blocks as a function of the number of entries for a 90nm technology node.

models were designed and synthesized on a 90nm industrial technology library. Even with a small number of nodes there is a significative overhead between both implementations. For example, the routing tables model shows a power consumption over $20x$ when compared to the DOR implementation. DOR is based on the concept of minimal-path routing and its concept is very simple: first route the message in one dimension ($X$) and then, in the other dimension ($Y$), when related to a 2-dimensional mesh. DOR is an efficient solution (logic-based implementation) in terms of area, power and delay overheads, but as opposed to a routing tables implementation it lacks the associated flexibility and it cannot support any of the challenges mentioned before.

So, it is imperative for current and future designs to face these challenges, benefiting from the flexibility of routing tables, while achieving important savings in three critical key aspects that influence every design at the nano-scale domain: area, latency (critical paths) and power-awareness, as shown in DOR implementations. Indeed, effective and efficient designs that accomplish to get overall savings in any of the aspects mentioned before will be a significant contribution to the NoC field. In this dissertation we take on such a challenge. But, rather than addressing completely irregular topologies (more suitable for MPSoC systems) we focus on irregular topologies derived from an initial 2D mesh structure where the following properties still remain: (1) a router is connected to at most four routers, each one in a different direction and dimension, and (2) a hop along a valid direction and dimension will not cross

more than one row or column. If we assume that the previous two properties are guaranteed by the final topology, then the solutions to provide efficient routing in those topologies get simplified. In this dissertation we present a conceptual architecture able to cover all the possible cases derived from a 2D mesh with full support for any failure, virtualization, domain or region configuration that keeps the network physically connected. Both unicast and broadcast operations are supported.

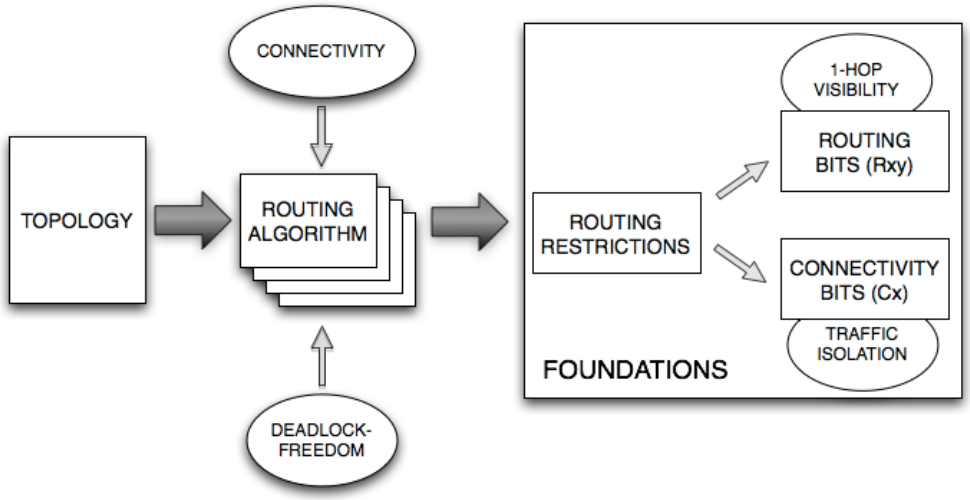



Figure 1.9: Foundations for the new routing mechanisms.

Every mechanism proposed and described in this dissertation is based on an alternative way of representing both the topology of the system and the routing algorithm. This method, referred to as *the foundations*, will allow a compact representation using only two sets of configuration bits: routing and connectivity bits. Those bits are computed according to the routing algorithm instance present in a given topology. Referring to Figure 1.9, from the current topology of the network, a set of routing algorithm instances are computed. These instances are defined conceptually by certain routing restrictions, that from the router viewpoint are translated into the two sets of configurations bits, which provide different associated properties.

Through all the dissertation we present a set of solutions that start from a basic mechanism, *LBDR* (Logic-Based Distributed Routing). The basic

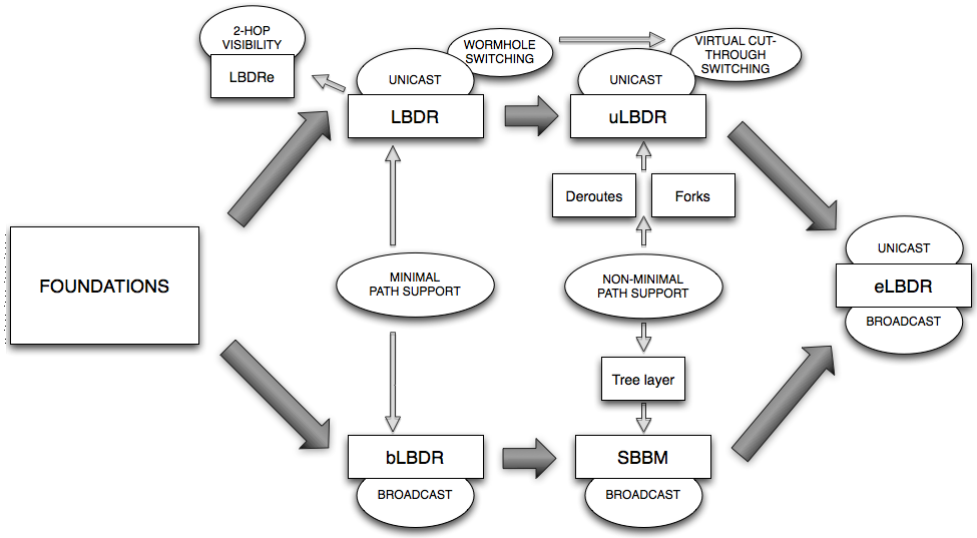



Figure 1.10: Evolution of the research.

mechanism will be enriched by new additions. On every step taken to arrive to the full conceptual architecture both unicast and collective communication will be addressed. This will be described thoroughly in the next chapters, but let us see a general overview in Figure 1.10. Starting from *the foundations*, with the basics established, first steps were made on *LBDR*, which is based on unicast routing and provides support for minimal-path routing. Broadcast support is provided, from *the foundations*, in the *bLBDR* mechanism. As with *LBDR*, only minimal-path support is provided. Although showing better flexibility than a DOR implementation, better savings than routing tables, and also with the addition of the broadcast counterpart, *bLBDR*, lacks the support on non-minimal paths, needed in some configurations, so the next steps were to add non-minimal path support to both mechanisms (*LBDR* and *bLBDR*). The evolution of *bLBDR*, *SBBM* was straightforward as compared to the unicast counterpart, *uLBDR*, as it required a different perspective of how the configuration bits were processed with a minimal addition. For *uLBDR*, two extensions are added for non-minimal path support, *deroutes* and *forks*, both covering different cases. Finally, *uLBDR* and *SBBM* are merged into *eLBDR*. With the full mechanism, *eLBDR* (effective

Logic-Based Distributed-Routing) enables the implementation of communication primitives in any irregular network derived from a 2D mesh without the need for routing tables, thus providing effective fault-tolerance, allowing the use of aggressive power saving techniques and enabling the virtualization concept at the network level. The technique works also in 2D meshes with no failures/irregularities, thus enabling its use also in fault-free chips.

## 1.2 Objectives

This section presents the objectives of this dissertation. The key objective we seek is to describe a simple logic-based distributed routing solution with low hardware overhead that provides support for both unicast and collective communication in a multicore chip where irregularities in its regular 2D mesh network may be present. In order to do this, we pursue the following specific goals:

- Provide an insight of the most recent related work in fault-tolerance and routing in on-chip networks.
- Describe a new functional methodology that allows us to encode efficiently routing algorithms and related issues into a compact representation of configuration bits. This is the foundations for our solutions.
- Propose a routing implementation, based on the previous methodology, that is able to support unicast communication while achieving a substantial fault-tolerance support.
- Propose a routing implementation, based on the previous methodology, that is able to support collective (broadcast, multicast) communication while achieving a substantial fault-tolerance support.
- Provide a thorough evaluation on every proposed implementation on aspects like performance, hardware overhead and coverage and provide comparisons to previous and related routing implementations.

The completion of these goals is reflected on the final mechanism, $eLBDR$, which achieves 100% coverage for all the evaluated cases, while showing similar costs to DOR-based routing implementations, and exhibiting flexibility

comparable to routing table implementations. It is important to remark that none of the proposals in this thesis is a routing algorithm by itself. This thesis proposes routing implementations, instead. Indeed, as will be explained in Chapter 3, any routing implementation can be used for any routing algorithm and its generated instances as long as it keeps these key characteristics: deadlock-freedom and connectivity between each pair of end nodes (these concepts are explained in the next chapter).

## 1.3 Dissertation Outline

This dissertation begins with this introductory chapter (Chapter 1). After, we continue with Chapter 2 that describes the basics of on-chip interconnection networks and an analysis of the current related work that contributes to the matter of this dissertation. Chapter 3 presents the foundations of the conceptual architecture. Chapters 4 and 5 present the different routing implementations shown in Figure 1.10. The routing implementations are described from the grounds to the full mechanism, each one with its associated evaluation and comments. In these chapters we also present some real implementations of the architecture and the trade-off related to the applicability of the design. Finally, the dissertation ends with Chapter 6, which summarizes the conclusions and displays the contributions related to the research field.

# Chapter 2

# Technical Background and Related Work

*"Live for a century, learn for a century."*

Russian proverb.

In this chapter, the goal is to describe the basics and terminology of on-chip interconnection networks. For the sake of understanding we will cover the main aspects, but it is not the intention of this chapter to provide an in-depth view on the subject, since the on-chip network field is as complex as the general interconnection network field, and there exist several aspects that are beyond the scope of this dissertation. We refer the reader to the established textbooks on this topic and related ones for further background and introductory material [11, 13].

First, in Section 2.1 a brief description of the design parameters that involve networks-on-chip are presented. Then, in Section 2.2, we dive into a more extensive description of the aspects that surround this kind of networks and the basics of routing types, strategies and implementations and how they are related. Finally, this chapter, in Section 2.3, shows the related work and existent contributions that serve as a reference for this dissertation.

## 2.1 On-chip Interconnection Networks

In the field of interconnection networks, there is a growing interest and amount of research in the on-chip domain. The integrated circuit technology has evolved to accommodate a multiprocessing device capable of high-performance computation. As a result of the high integration scale in the deep sub-micron domain and the increasing number of connecting elements, on-chip interconnection has become a need and will influence the performance of the final system. So, any gain in the efficiency of the on-chip interconnection layer will be highly beneficial.

### 2.1.1 Design Factors

As aforementioned, NoCs play a major role in the design of the modern high-performance computers, nevertheless, they are not simple; there are many factors that affect the choice of an appropriate interconnection layer at design time. The main factors are:

- *Performance.* As commented, performance is a key point in interconnection networks, not only from the point of view of raw throughput, also from the point of view of latency. Latency is a critical design issue in several systems such as real-time systems. Moreover, in on-chip networks, messages must reach destinations in terms of nanoseconds.

- *Scalability.* Scalability is the first design rule that an interconnect designer should keep in mind. Scalability in interconnection networks implies that the bandwidth of the network increases proportionally to the number of elements of the system. Latency should also be kept to reasonable limits when increasing the system size. Otherwise, the interconnection network would become a bottleneck, limiting the efficiency of the whole system. Scalability also implies that network cost and resources are proportional to the network size.

- *Reliability.* An interconnection network should be able to deliver information in a reliable way. Interconnection networks should be designed for continuous operations in the presence of a limited number of faults.

More important, as technology scales, manufacturing defects will increase, thus demanding an efficient treatment.

- *Simplicity.* Not only for the sake of cost, but making simpler designs leads to the implementation of architectures that work with higher working frequencies, thus, increasing the system performance, and occupying less area. In fact, the silicon area usage is a critical aspect in on-chip networks. Indeed, reducing the area, translates into the opportunity for making room for more devices inside the chip.

- *Power consumption.* One of the most important aspects in networks-on-chip, not as important in other network environments, is the reduction or minimization of power consumption. Indeed, effective power-aware techniques are needed to bring better management of the total power consumed by the processing cores.

All these previous factors must be specifically considered when designing an on-chip network. In this thesis all the contributions take these factors, directly or indirectly, as a reference. In the next section we present the basics for interconnection networks.

## 2.2 Interconnection Network Basics

The network architecture design is the result of several design choices like network *topology*, *switching and flow control* techniques or *routing strategies*. The network topology defines the physical interconnection between nodes and other elements. The switching and flow control techniques define how and when the information is transmitted through the network resources. Finally, the routing strategies manage the different path choices of communication between the nodes.

There are some common elements that conform a network architecture. The first elements are the nodes. Nodes are the elements that communicate through the network and perform basically two main tasks: computation and storage. Nodes connect to other nodes through a network interface that could be associated to routing devices, depending on the topology of the network. A

routing device connects multiple devices. Link are the elements that connect all the devices (network interfaces and routing devices) present in the network architecture. See an example in Figure 2.1.

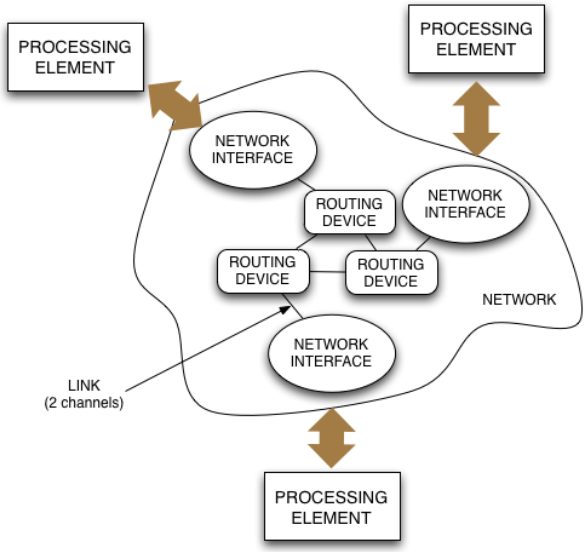



Figure 2.1: A general overview of a network architecture.

### 2.2.1 Network Topology

Different network categories can be devised based on how all the elements of a system are connected to the network (see examples in Figure 2.2):

- *Shared-medium networks:* In this kind of network there is a transmission medium that is shared by all the nodes, and only one node is able to begin communication at a time, the rest of nodes read (and monitor) from the shared medium. Every device has the circuitry to handle addressing of other nodes and the data management. In this kind of networks, the routing device is the shared medium, called also bus. Buses have limited bandwidth, so they suffer from scalability problems, as the number of connected nodes increases.

- *Direct networks:* Each node has a routing device attached, called router, which is the component that establishes the connection to other nodes

through point-to-point links. Each node is directly connected usually to a small subset of nodes in the network. The concept of network interface is weak in this type of networks as the end node and the routing device are tightly connected. Nodes are connected according to a certain interconnection pattern.

- *Indirect networks:* Instead of connecting directly the nodes through point-to-point links, the communication between a pair of nodes can be performed by intermediate stand-alone routing devices called switches. Every node has a network interface that connects to a switch (through a point-to-point link) and switches are connected between them (also through point-to-point links).

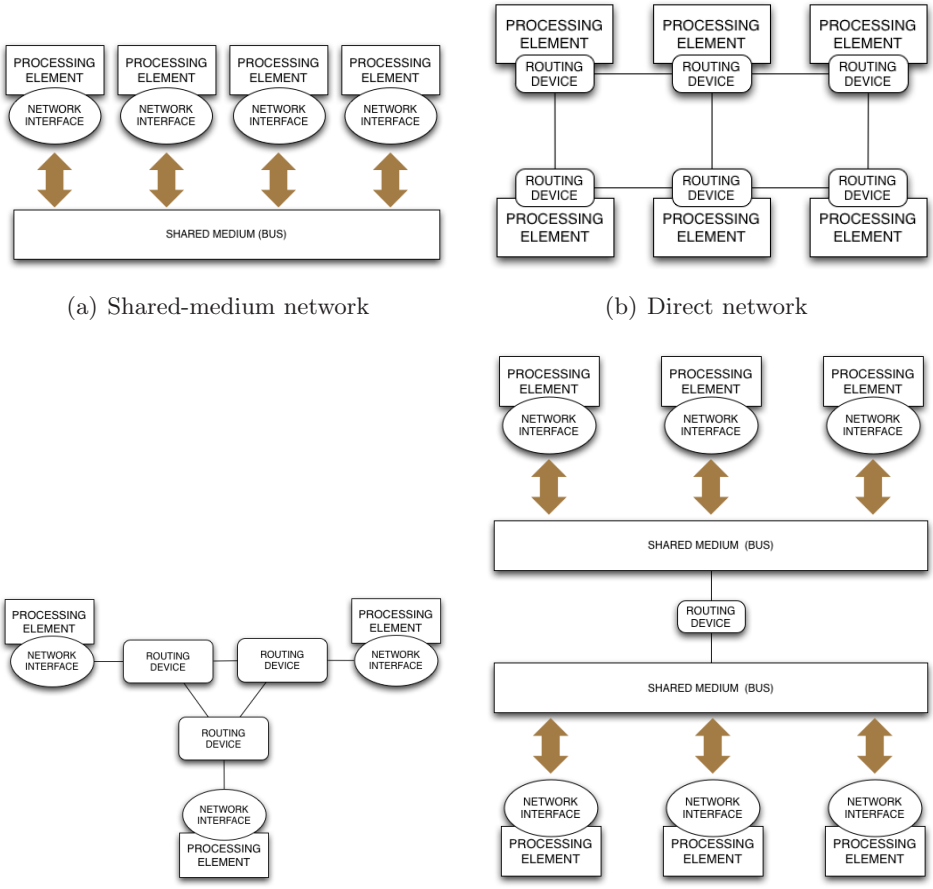
- *Hybrid networks:* This kind of networks is a mixture of the previous approaches. In general, they combine mechanisms from shared-medium-networks and direct or indirect networks.

Although there are very subtle differences between direct and indirect networks, the functionality is similar in many aspects. An indirect network in which every switch is connected to a single node is equivalent to a direct network. Also, terms router and switch, although having different meanings, are used with no distinction by the community, so both terms for the routing devices are interchangeable. In the rest of the dissertation, unless noted, the term router is assumed.

There are also some common aspects to all these types of networks. Although links are usually formed by two communication channels, one in each direction, one of the basic aspects of a network is how communication channels are arranged. Network performance significantly differs if links are bidirectional or unidirectional. This choice impacts directly on the routing techniques and algorithms and associated issues, like deadlock avoidance. We assume the use of bidirectional channels on every link, though.

Each type of network can also be categorized with different properties:

- *Router degree:* This property refers to the number of channels that connect a router to its neighbours.




(a) Shared-medium network

(b) Direct network

(c) Indirect network

(d) Hybrid network

Figure 2.2: Network architectures.

- *Diameter:* It is the maximum distance between a pair of end nodes in the network.
- *Regularity:* A network is defined as regular when all the routers have the same degree.
- *Bisection Bandwidth:* Bisection of the network encompasses the minimum set of links that split the network in two equal halves. Bisection bandwidth is the resulting bandwidth at the bisection.
- *Homogeneity:* A network is homogeneous if every node is equal in all

aspects to the rest of nodes.

There are three common basic topologies used in interconnection networks. The first one is the *crossbar*. A crossbar (see Figure 2.3) allows the connection from any node to any other node simultaneously at the same time other connections are established (as long as the requested input and output are free). Crossbar networks, typically, are used for high-performance computing multiprocessor solutions and in the design for routers in direct networks. The drawback with crossbar topologies is that they do not scale as system grows due to the quadratic requirement of connections.

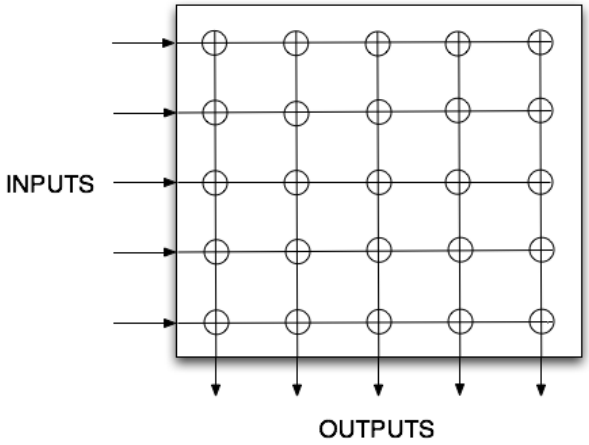
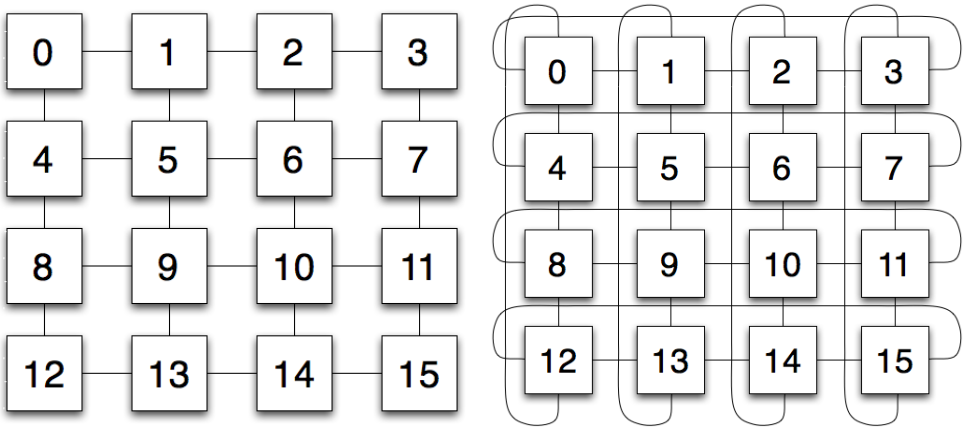



Figure 2.3: A crossbar network.

Strictly orthogonal topologies are the second common type. In this kind of networks we can find the *n-dimensional meshes and tori* (see Figure 2.4). A n-dimensional mesh or torus has $k$ nodes placed along each dimension. A mesh differs from a torus because it does not have the wraparound channels that connect the nodes in the borders of the topology. Note that the torus topology duplicates the bisection bandwidth of the mesh topology and reduces the diameter. These topologies are the most common example of direct networks.

*Multistage interconnection networks* (MINs) are topologies driven by the concept of indirect networks as seen in Figure 2.5. Between input and output devices there are several switch stages. The arrangement of stages and the connection patterns determine the routing in these networks. MINs have




(a) Mesh (b) Torus

Figure 2.4: A $4 \times 4$ 2-dimensional mesh and torus.

been heavily used to interconnect parallel computers with large number of processors in commercial and high-performance solutions. However, for on-chip networks mapping of such topology pattern in the 2-dimensional surface of the chip is a big challenge.

### Networks-on-chip Topology

Earlier on-chip communication architectures fall on the share-medium network paradigm, that included buses as the communication subsystem. Examples of this kind of architecture is the Cell processor [23], that is well suited for a small number of processing elements (or nodes). But the trend nowadays in the industry of high-performance computing is to include a reasonably large number of processing cores inside the chip, and shared-medium network designs have poor scalability and bandwidth impacting heavily on the network performance.

Network-on-chips emerged, thus, as a response to effective on-chip communication. On-chip networks are based on a paradigm that is a mixture of the concept of direct and indirect networks. Current multicore architectures are composed as a wall made of elemental brick nodes that work together to achieve the high-performance computing goal, i.e, the chip is formed of several processing devices. These devices are called usually *tiles*. A tile, fun-

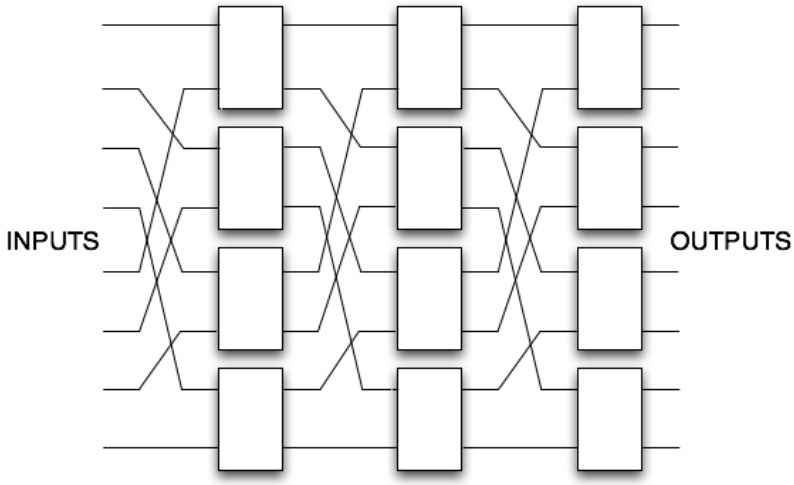



Figure 2.5: A multistage network topology.

damentally, apart from the processing elements, has also a router attached that handles the communication between tiles. See a simplified schematic of a tile in Figure 2.6.

As the chip can be seen as a collection of tiles, there is a major taxonomy where chips can be differentiated between homogeneous (inducing regular topologies) and heterogeneous designs (more suited with irregular topologies). Every tile is connected to a subset of other tiles through an on-chip network. An example of homogeneous configurations are the tiled chip multiprocessors (CMPs) where all the tiles are equal, i.e, tiles are replicated along the chip (see Figure 2.7(a)). Instead, high-end multiprocessor systems-on-chip (MPSoCs) are an example of heterogeneous designs where tiles are different in many aspects: size, functionality, performance, throughput, etc (refer to Figure 2.7(b)).

A popular choice in NoC designs is the use of orthogonal topologies as most of the direct network architectures are implemented with this property in mind. Orthogonal topologies, which are associated with regular patterns, allocate the nodes in a $n$-dimensional space, with $k$ nodes along each dimension. Every router has at least one link crossing one dimension. Every router is labelled with an identifier depending on the coordinates, and all links that communicate to other routers are bidirectional (formed by two channels, one

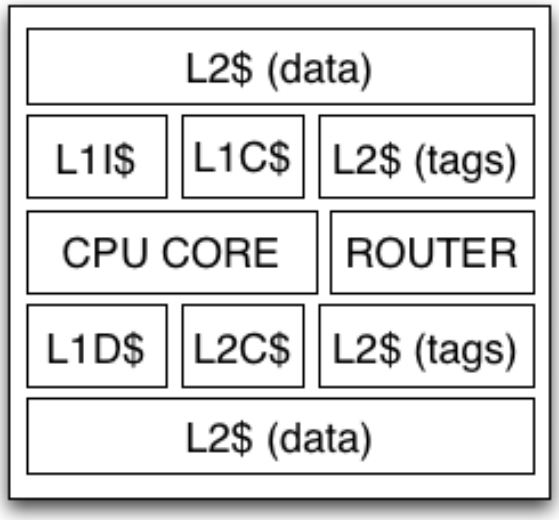
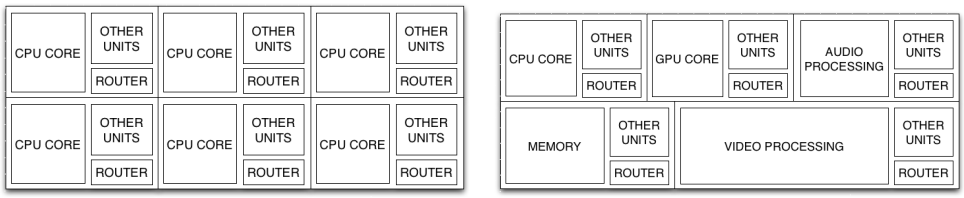



Figure 2.6: The processing element in a tile-based CMP.




(a) Tiled chip multiprocessor (b) High-end multiprocessor system-on-chip

Figure 2.7: Different tile-based designs.

in each direction). As the distance between a pair of routers is the sum of the offsets in all dimensions, the routing strategy is usually implemented as a function of selecting the links that decrement the absolute value of the coordinate offsets between a source node and a destination node, a very simple mechanism. The most popular design in NoCs is the n-dimensional mesh, used in most of the commercial and non-commercial (prototypes) NoC designs. The most suitable topology is the 2-dimensional mesh (Figure 2.8). This kind of topology is vastly used (or at least assumed) because it fits the chip layout.

As every router is located within the network by its coordinates on a n-dimensional space, a router in a 2-dimensional graph will be numbered by a

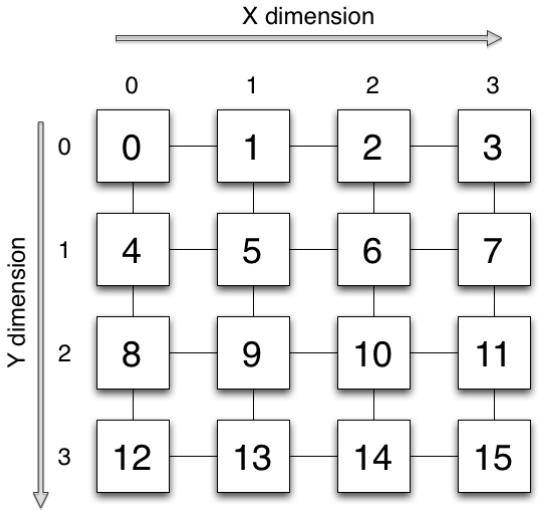



Figure 2.8: A $4 \times 4$ 2-dimensional mesh.

group of two coordinates, $(x, y)$, one for each dimension. Crossing a link means decrementing or adding an unitary value to the offset of the dimension between the two nodes that share the associated link. See an example in Figure 2.9. Moving from node 1, with coordinates $(1, 0)$, in $Y+$ direction results in node 5, coordinates $(1, 1)$. Typically, nodes are numbered by a single id, computed as a function of the coordinates and the number of nodes per dimension. In the case of the example for the 2-dimensional mesh, the value follows this equation: $ID_{Node} = X_{coordinate} + k \times Y_{coordinate}$, being $k$ the number of nodes per dimension. So, in the example in Figure 2.9, node $(3, 1)$ has an id of 7.

### 2.2.2 The Routing Device

As aforementioned, each tile is composed of several elements. The router is in charge of the communication between the associated node and the rest of the nodes through the network layer. Typically, a router[1] architecture is structured by the following general parts (Figure 2.10):

- *Buffers:* Buffers are a key component of the router and its design and

[1]Note that, as previously commented, the community makes no distinction on the terms of router and switch as they have similar meanings. We prefer the term router as it involves making routing decisions, not just switching.

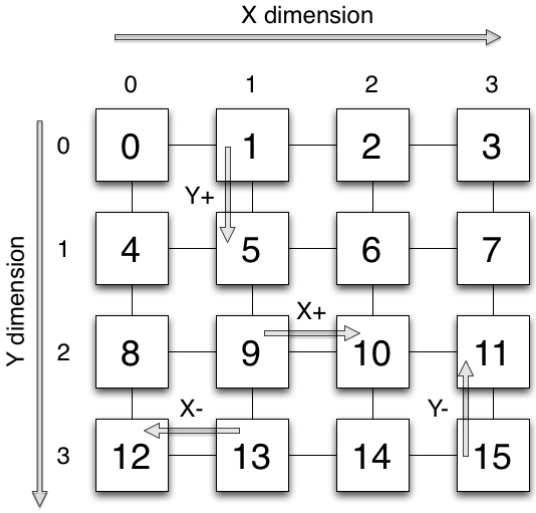



Figure 2.9: Different link crossings in a 2-dimensional mesh.

position inside the router affect other aspects of the router design. The task of a buffer is to store temporarily units of information (typically called messages and/or packets). Buffers are associated to the channels that are connected to the router. Channels, also called ports, are divided into input ports, streams that receive the messages and are subject to the routing decisions, and output ports, streams the send the messages to other routers or nodes. Note that, to save area and power, buffers at the output ports are usually not implemented.

- *Crossbar:* The crossbar is the switching element and is non-blocking. Crossbars allow the connection between all inputs of the router to all the outputs. Crossbars are classified by their radix, i.e. the maximum numbers of connections they can make. As has been already identified, crossbars do not scale, thus routers with many ports do not scale neither.

- *Routing unit:* This unit is the responsible for decoding the unit of information provided by the incoming message, and based on the routing function and destination of the message, computes the most suitable output ports for transmitting the message.

- *Arbiter unit:* This unit feeds from the routing unit and configures the

crossbar accordingly to the requests between input and output ports, taking into account *switching* and *flow control* issues (both will be explained later).

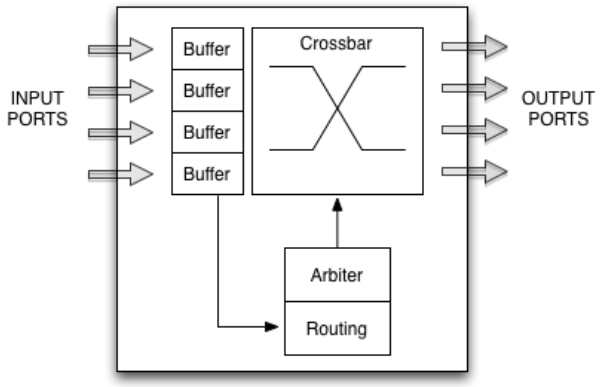



Figure 2.10: Router in a tile.

### 2.2.3 Data Units

In an interconnection network, the general routing unit of information between nodes is the *message* (see Figure 2.11). A message is a collection of bits that the sender wishes to transmit to a destination (or a set of destination nodes), i.e. it contains the data that must be transmitted. This information unit, however, due to resource restrictions affected by design choices, may need to be divided into smaller units, called *packets*, through a packetization process (usually performed at the network interface). A packetization of a message implies some reassembly and order handling at the destination. A packet (or the message) is comprised of a header, which contains the information for routing and control, to be used by the routing devices, a body which contains the data, and optionally a tail, for flow control or arbitration purposes. Often, packet and message terms are interchangeable by the community, when both are equal in size. The term packet is usually employed even when the message has not been packetized. In this thesis we use the term message when wormhole switching is assumed (see next section) and the term packet when virtual cut-through switching is used (see also next section).

A packet is divided further into *flits* (flow control digits), which are the smallest unit of information flow controlled. As the width of the link can be lower than the size of a flit, the flit is further divided at the physical level, into *phits* (physical digits). It is left to the designer and the parameters involved, the size of every unit. However, in on-chip networks, due to the vast amount of bandwidth available, the phit size usually equals the flit size.

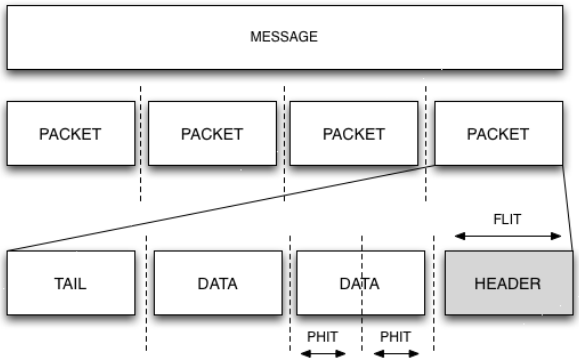



Figure 2.11: Data units.

### 2.2.4 Switching

Switching techniques are the responsible for the allocation of network resources to messages/packets inside the routers. Their basic function is to perform the setting of the connections between the buffers of the input and the output ports. The choice imposes several design constraints in the router that impact the performance, fabrication cost and power consumption of the elements in the network. Next we describe the main switching techniques used in on-chip networks.

#### Circuit Switching

In circuit switching (Figure 2.12), the network establishes a reserved path between source and destination nodes prior to the transmission of the message. This is performed by injecting in the network a flit header, which contains the destination of the transmission. This header acts as some kind of routing probe that progresses towards the destination node reserving the channels

that it gets. When the probe reaches its destination, a complete path between destination and source node has been set up due to the acknowledgement that is sent back to the source node. As the path has been reserved for this flow, messages cross the network avoiding buffer needs and collisions with other flows. The circuit is torn down when the transmission finishes. An example of a circuit switching-based on-chip network is described in [71].

Circuit switching can be very advantageous when messages are very frequent and long. Nevertheless, this switching technique has several important drawbacks. If circuit set up time is long compared to transmission time of the data, it will strongly penalize the performance of the network since links will be poorly used. Additionally, as channels are reserved for a given flow, no other flows can use them even if the connection is idle, thus channels may become even more under utilized.

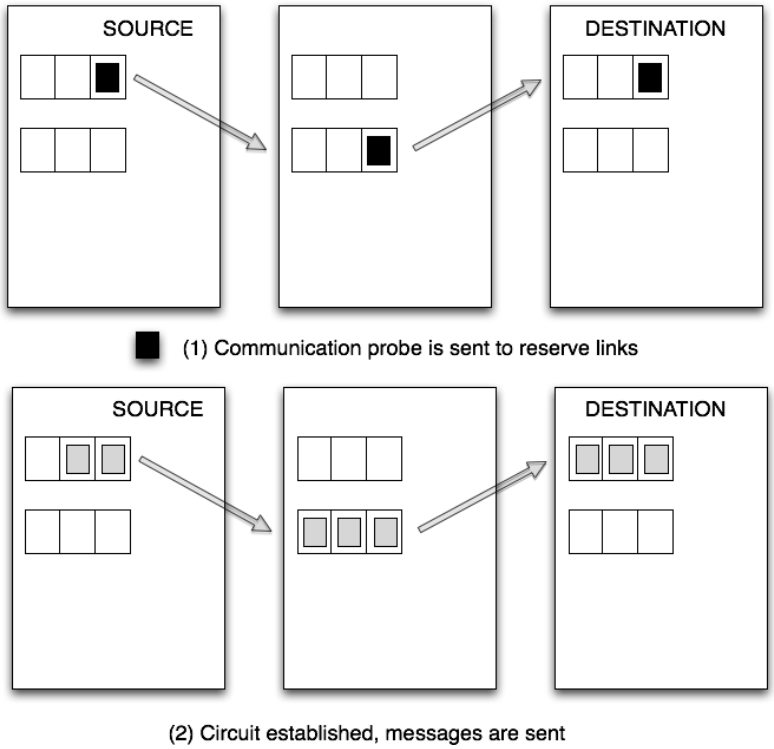



Figure 2.12: Circuit switching.

### Store and Forward

Instead of reserving all the path for a certain flow, there are some techniques that operate at packet granularity. These techniques are referred as packet switching. The most basic technique related to packet switching is *store and forward* (SAF). When a packet arrives to a router, the router waits to store the whole packet in its input port buffer before the packet is forwarded. So, input port buffers must be large enough to store a packet (see Figure 2.13).

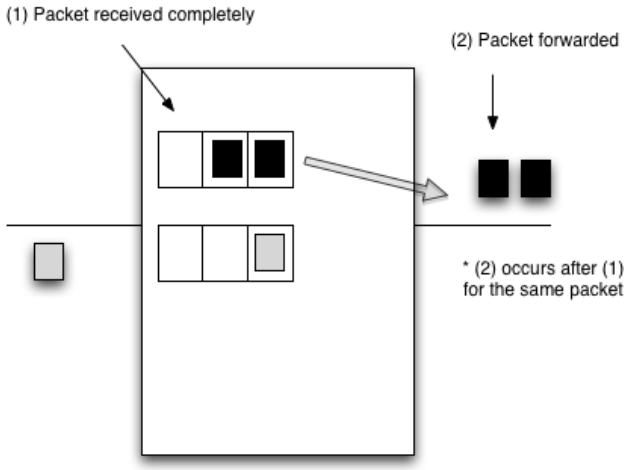



Figure 2.13: Store and forward switching.

As can be deduced SAF has longer buffer requirements than circuit switching. In addition, latency of packets is multiplicative with hop count along the path (as the forward operation waits for the completion of the store operation).

### Virtual Cut-Through Switching

SAF switching is based on completely receiving a packet before any routing decision is made. But, this is not a very practical decision, since packet header contains all the required information to perform the routing, and it is physically located at the beginning of the packet (typically in the first flit). So, the routing process can be started once the packet header arrives to the input buffer, without waiting for the rest of the packet. Thus, the packet can be forwarded provided the selected output link chosen by the routing strategy

is free. This is what is done in *virtual cut-through* (VCT) switching (Figure 2.14).

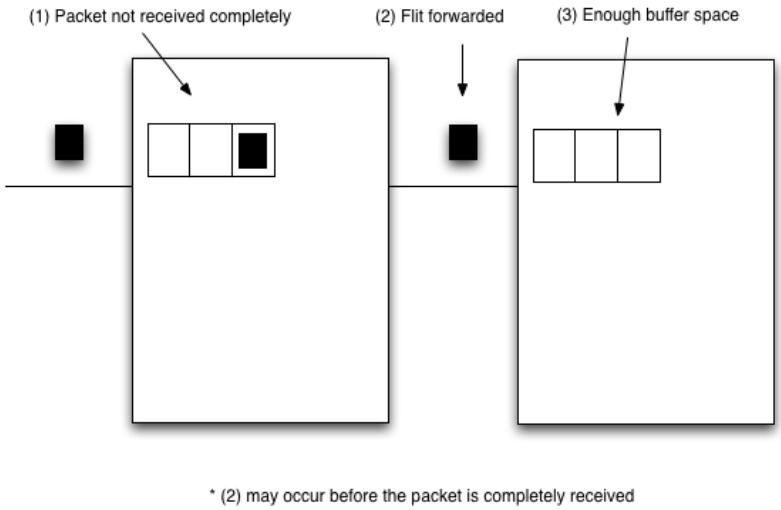



Figure 2.14: Virtual cut-through switching.

In this case, as packets can advance through the routers of the network once the packet header has arrived to each buffer (and has been decoded), the base latency for this switching technique is mostly additive to the distance between the nodes (hop count). Despite this, buffer requirements are the same for VCT and SAF. VCT requires there is enough free buffer space to store the entire packet. In fact, VCT behaves like SAF when the output link is busy. The router needs to completely allocate the entire packet. This is the switching technique commonly used in off-chip high-performance interconnects [11, 13] due that buffer size is not as critical as in NoCs.

### Wormhole Switching

VCT switching is an improvement over SAF, but in some network architectures, the choice of a buffer size to hold an entire packet could be critical. The requirement to completely store a packet in the buffer of a router may prevent to design a small, compact, and fast router [13]. In wormhole switching (WH) buffers at the ports of a router only have to provide enough space to store

only few flits, depending on the round-trip time delay (RTT) [2], instead of the whole message. In WH switching (Figure 2.15), the packet is forwarded immediately before the rest of the packet is entirely received, but as opposed to VCT, there is no need to have enough space for the rest of the message in case the message blocks. In that case, the entire message remains stored through the buffers of several routers. The major advantage of WH switching is the low storage requirements at routers. However, the most important drawback is that WH switching could lead to high contention levels at the network, because a message may block several resources when is traversing the network, causing low utilization of links and buffers.

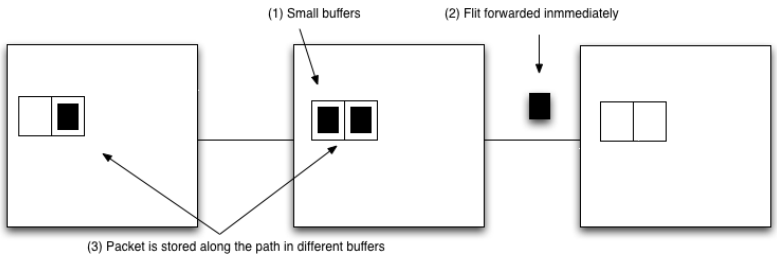



Figure 2.15: Wormhole switching.

### Virtual Channels

To overcome the problem of contention induced by wormhole switching, *virtual channels* [10] were proposed. Buffers basically are operated as FIFO (First-in, First-out) queues. Therefore, if a message reserves the channel but due to the saturation of the network it remains blocked at the current router, no other message can use the physical channel even if its requested output port is available. This problem is known as *head-of-line blocking*.

When using virtual channels the buffer at the input port is divided into different virtual buffers and the channel is shared by all the virtual buffers (see Figure 2.16). Of course this virtual multiplexing requires some local arbitration and must be taken into account by flow control and switching techniques. Virtual channels can be used to improve message latency and network

[2] Round-trip time delay can be defined as the elapsed time between a unit of information is sent and the acknowledgement of that transmission is received

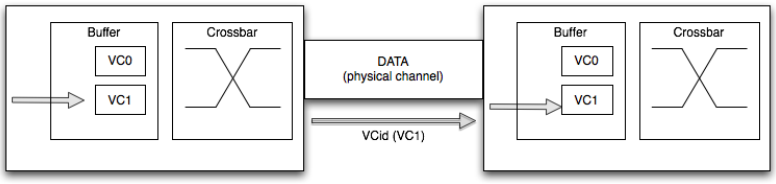



Figure 2.16: Virtual channels.

throughput. Their major drawback is that the available link bandwidth is distributed over all the virtual channels sharing a physical link, resulting in lower speeds. Again, in the on-chip network domain, the designer must evaluate the trade-off and the impact overhead on the network. Virtual channels are not restricted to wormhole switching, the concept can be extrapolated to other design choices, depending on the need of their functionality (examples are deadlock-free routing algorithms and quality-of-service protocols)

### 2.2.5 Flow Control

Transmission of a flit between the input and output ports in a router is a task performed by the switching technique. *Flow control*, however, is in charge of administering the advance of information between routers. Buffers are a temporary resource where to store flits, but they are finite. Flow control techniques are in charge of determining when the flits can be forwarded evaluating the capacity of the buffers and the link bandwidth.

There are three flow control mechanisms that are commonly used: *ack/nack*, *stop & go* and *credit-based*. The ack/nack flow control mechanism is based on data acknowledgements. When a flit arrives to a buffer, if the buffer has space available, then the flit is accepted and an acknowledgement signal (ack) is sent back. Instead, if there is no space available, the flit is dropped and a negative acknowledgement is sent. The flit must be retained at its origin until it receives a positive acknowledgement.

Stop & go emerged as an alternative to reduce the signalling (control traffic) between the sender and the receiver. Stop & go flow control is based on every buffer having two thresholds corresponding to certain sizes computed from the round-trip time. When the space occupied in the buffer reaches the

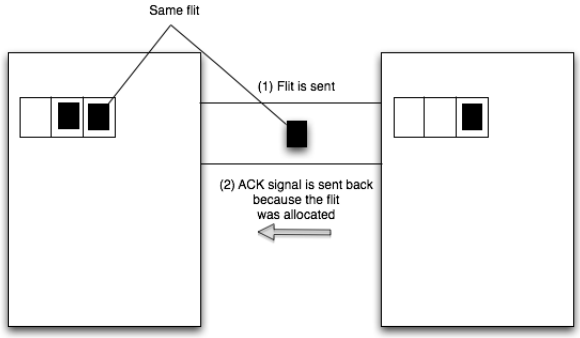



Figure 2.17: Ack/nack flow control.

stop threshold, a signal is sent back to the sender precisely to stop the transmission, taking into the account that still remains enough buffer space for the flits that are still being transmitted by the sender. When the buffer occupancy diminishes under or equal to the second threshold, go, then another signal is sent to reactivate the flow of flits.

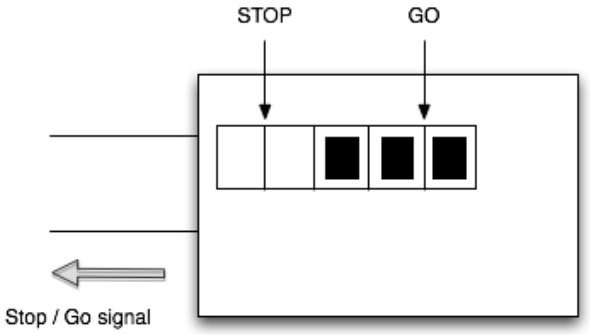



Figure 2.18: Stop & go flow control.

With *credit-based* flow control, each sender, at its end of the link, maintains a count of credits, which is equal to the number of flits that can still be stored at the buffer on the receiver side. Whenever a flit is forwarded to the receiver buffer, as it occupies a slot, then the counter is decremented. If the counter reaches zero, it means that there is no available buffer space at the other end, and no flit can be forwarded. On the other hand, whenever a flit is forwarded and frees the associated buffer space, a credit is sent back to

increment the counter. The drawback of this flow control mechanism is the significant amount of credit signalling sent backwards, which could impact on network performance.

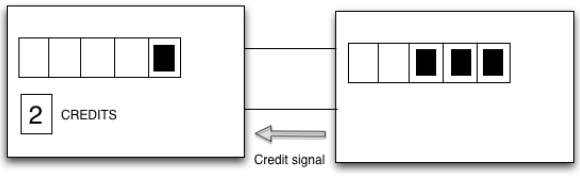



Figure 2.19: Credit-based flow control.

### 2.2.6 Arbitration

A router is composed of multiple input and output ports with their associated buffers and channels. Multiple inputs, according to routing decisions, may request the same output port. In this scenario, an arbitration operation is required to decide which one of the requests is allowed to connect to the output port. The arbitration mechanism must ensure to assign the output to only one of the inputs that have requested it, and the others must wait until they are allowed. As the arbitration operation introduces a latency to determine the assignment of the different output ports, it is critical for a network-on-chip environment that these operations are performed fast enough to keep low latencies.

The main goal of an arbitration mechanism is to provide fairness between all the ports while achieving maximal matchings between requests and resources. Although there are many proposals for arbitration algorithms and implementations, we can distinguish two general arbitration techniques that are differentiated on how to assign priorities between the requesters.

The first one is *fixed priority*. An arbiter with fixed priorities assigns the requests in an established order to the different input ports. This order is determined by the priority assigned to each input port. In this mechanism, the arbitration is simple, but introduces unfairness and potentially, starvation. If one of the input buffers with higher priority keeps requesting the associated output, the inputs with lower priority get blocked, even, inducing the chance that the inputs with low priority never get the request satisfied.

The second one is called *round-robin*. An arbiter that implements round-robin arbitration cycles priorities between all the input ports by assigning the lowest priority to the input port which request was last served. This arbitration technique introduces better fairness between the requesters, but is more complex to implement. Usually, represents a trade-off between implementation cost and performance.

### 2.2.7 Routing

In this section we tackle the basics of routing strategies. As we have described before, topology defines the physical organization of the network composed by the nodes. In fact, a given topology defines the available paths between all the nodes. The routing algorithm is the responsible of deciding which path has the message to follow to be effectively routed from its source to its destination. The choice of the routing algorithm becomes of key importance in the network performance. Indeed, in the on-chip network domain not all solutions from the off-chip network domain are suitable due to environmental restrictions. The designer must find a trade-off between efficiency, flexibility and cost of the routing implementation (the core of this dissertation).

We must remark, for the sake of the description and interpretation of this dissertation, the concept of a routing algorithm and a routing implementation. Indeed, few authors disassociate both issues and, when talking about NoCs the implementation becomes a big issue. Taking as an example the DOR routing algorithm, we can distinguish between the routing algorithm itself (the rules to apply to every incoming message) and the way such rules are implemented. Figure 2.20 shows the algorithm written in pseudo-code and Figure 2.21 shows a possible implementation with some logic gates. This distinction will be key to properly identify and implement algorithms in NoCs addressing future challenges.

#### Problems and Issues

In the on-chip network domain, and more generally in any interconnection network scenario, the desired behaviour is that every generated message from a source node arrives to its destination. However, even in the presence of phys-

```
Inputs: Coordinates of current router (Xcurrent, Ycurrent) and destination router (Xdest, Ydest)
Output: Selected output port (Port)

Procedure:
Xoffset := Xdest - Xcurrent;
Yoffset := Ydest - Ycurrent;

if Xoffset=0 and Yoffset=0 then
  Port:=local          //Arrived to destination
endif

if Xoffset < 0 then
  Port:= X-
endif

if Xoffset > 0 then
  Port:= X+
endif

if Xoffset=0 and Yoffset<0 then
  Port:= Y-
endif

if Xoffset=0 and Yoffset>0 then
  Port:= Y+
endif

end procedure
```

Figure 2.20: DOR algorithm pseudocode.

ical available paths, there are several situations that prevent message delivery. Next we describe the different issues that may prevent message delivery.

The first issue is *deadlock* (see Figure 2.22). A deadlock occurs when a message cannot advance toward its destination because the buffer requested by the message is full, being blocked by another message that is also waiting. A cyclic set of such events could make the messages to be blocked permanently because each message involved in the situation requests a resource that is hold by another one. As no one message will advance before getting its requested buffer granted we get a deadlock situation. There are two common ways to deal with deadlock events. The first one is to guarantee in the routing strategy the avoidance of deadlock situations, which basically is done by the exploration of cycles in the Channel Dependency Graph (CDG). The CDG is built by graphically representing channels as vertices and channel dependencies

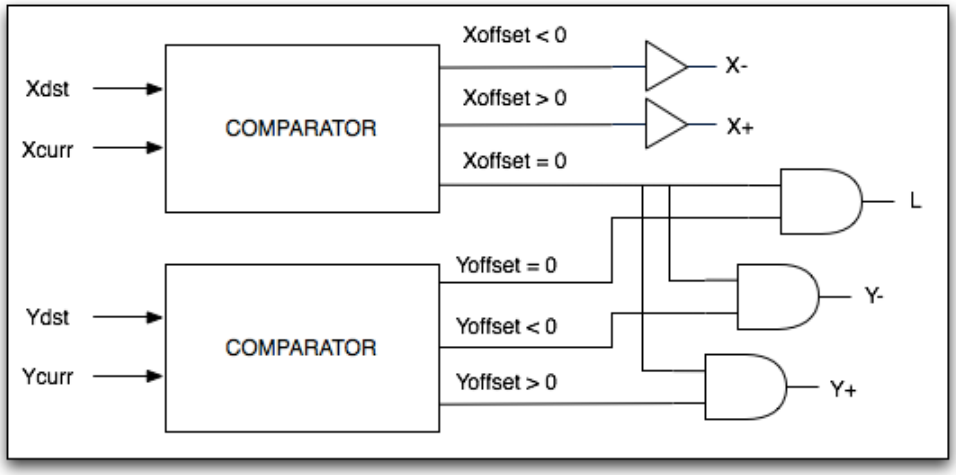



Figure 2.21: DOR implementation.

as edges. A dependency exists between two channels if a message uses both channels. The resulting CDG for DOR routing is acyclic as shown in Figure 2.23. If a cycle exists in the CDG, then a deadlock situation may arise in the network. The second one consists of detection of deadlock events that are present in the network and providing a recovery methodology to dismiss the deadlock situation.

The next issue to deal is *livelock*. Livelock scenarios are similar to deadlock, but they happen when a message is misrouted and never reaches its destination as the links required to do so are always reserved by other messages. So, there is no situation of permanently blocked messages, but more adjusted to a dynamic blocking (take the example of two people that want to cross a narrow corridor, so both of them try to be polite by moving aside to the same direction). These kind of situations arise when allowing non-minimal path routing. Fortunately, this problem can easily be solved by bounding the number of times a message can be misrouted.

Finally, the last problem a routing algorithm must face is *starvation*. This issue is triggered when a message is permanently stopped holding a resource and cannot advance because the network traffic is so intense that the resources requested are always granted to other messages with higher priorities. This scenario is the result of an incorrect arbitration. Starvation is easily avoided

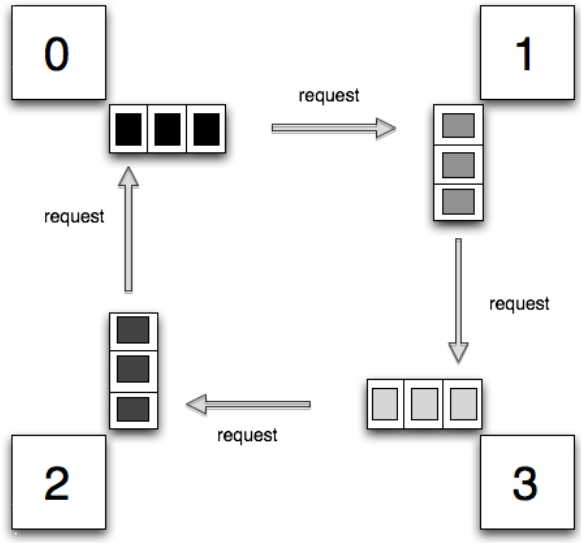



Figure 2.22: Deadlock event.

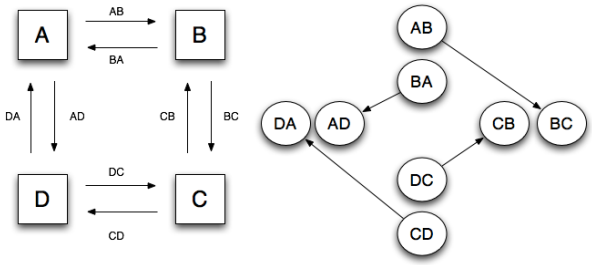



Figure 2.23: CDG for DOR routing in a $2 \times 2$ mesh network.

by a proper design of arbiter and priority mechanisms.

All these issues occur because the number of resources (buffers) is finite, and specially in the on-chip network domain, is reduced. To face these problems, there are two ways of implementing the routing schemes and algorithms. The most suitable for networks-on-chip is to prevent the formation of such scenarios (acyclic CDGs, no misrouting allowed, fair arbiters). The second one consists on recovery techniques to solve these kind of situations (cyclic CDGs). Recovery techniques also need extra circuitry to detect the presence of these issues. In this thesis we focus on routing algorithms that guarantee acyclic CDGs.

### Implementation Types

Although any implementation is specific to the nuts and bolts of the technology, there are two main trends to implement the routing strategy.

The first one is *logic-based* routing. This kind of routing is the result to translate a logical or arithmetical function of a routing algorithm into the equivalent in circuitry inside the router. So, when the message header is decoded at the input buffers, the output port is computed based on the hardware that represents the routing function. Logic-based routing is a good design choice in terms of delay, area, and power consumption. The main drawback is the lack of flexibility as these implementations could become non-functional if the topology scenario changes due to manufacturing defects, just to name a reason.

As we anticipated in Section 1.1, on the other hand we have *table-based* routing. Routing tables are basically composed of row-like structures that match destinations with table entries. So, given the destination for a certain message, there is some circuitry associated that decodes this information, and accesses the routing table to find the routing decision associated to that destination. The most conventional way to implement these tables is to use memory structures. The advantage of table-based routing is flexibility, as the information of routing decisions stored on routing tables could be the answer of more complex routing algorithms, that are not only based on logical or arithmetical assumptions. On the other hand, routing tables implementation suffers from scalability, area, power consumption, and latency problems. For example, there is a penalty time (that increases with table size) associated to accessing memory structures.

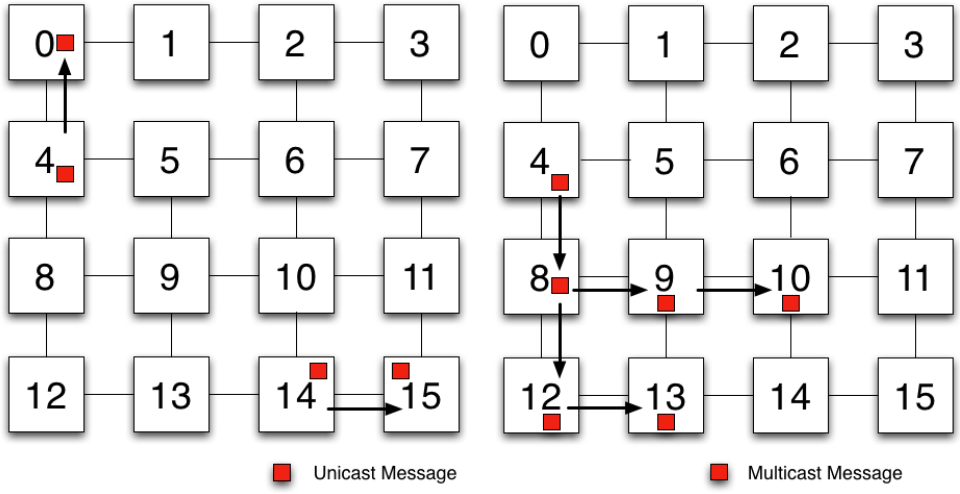
### Schemes Classification

There are different taxonomies to classify routing algorithms. In the following paragraphs we provide a discussion for the most well-known types of routing algorithms.

The nodes of an interconnection network send and receive messages through the routing devices present in the network. Given there is connectivity between all the end nodes on the network-on-chip, a node (or several nodes) may re-

quire to send the same information to several nodes, instead of only one. From the perspective of the sender, and given the amount of nodes that are meant to receive the data, there is a first distinction. If the message is sent to only one destination, we are talking about *unicast* or one-to-one communication $(1:1)$. An example is represented in Figure 2.24(a).

On the other hand, if the message must be sent to several destinations (that could include all the nodes on the network) then we are talking about some types of *collective communication*, again from the perspective of the source node. If the message must be sent from a source node to the rest of the nodes in a chip then the routing operation is named *broadcast communication*, or one-to-all $(1:all)$. On the other hand, multicast communication or one-to-many, occurs when the sender distributes the message to a limited group of destination nodes $(1:many)$. Broadcast communication can be seen as an specific case of multicast communication. Broadcast communication is easier to implement because the incoming message is just replicated to the rest of router ports but the drawback is flooding the network with unneeded messages. There is an example of multicast communication in Figure 2.24(b).




(a) Concurrent unicast operations (b) Multicast operation

Figure 2.24: Different routing types.

There are three basic methods to implement broadcast or multicast rout-

ing. The first one is the *unicast-based approach* or multiple one-to-one communication. This technique implements a collective communication operation by sending, in a sequential manner, a unicast message to every destination. While this solution requires minimum routing infrastructure, it tends to flood the network with many messages, resulting in higher latency communications. Power consumption is also high as there are many redundant messages in the network. The second one is the *path-based approach*. This solution relies on the injection of a single message with as many headers as destinations. The message uses a long path visiting all the destinations sequentially. Its downside is the message header overhead as well as the long path used, which impacts network latency. Also computation of paths is not trivial (so as to avoid deadlock) usually using a Hamiltonian cycle. The last one is the *tree-based approach*. Tree-based multicast or broadcast solutions rely on the use of a spanning tree mapped on the network (typically on a 2D mesh network) or region, providing collective communication to a set of destinations with the minimum amount of time. Routers create replicas when new branches are formed along the tree. This solution minimizes the number of messages sent through the network (the sender only injects one message per tree) with the associated reduction in power consumption and network latency. However, this approach usually requires a costly implementation, and is the one where avoiding deadlock is more complex.

Another classification in routing strategies is based on the location where routing decisions are taken, i.e, which devices are responsible for computing the path for the message. Routing can be implemented in two straightforward ways: source-based or distributed-based. In the first case, source-based routing, the responsible agent for computing the path is the end node. The entire path, then, is stored on the message header. In this routing scheme, network routers just read the stored path in the message header and forward the message through the indicated output port, which makes routing very quick and router designs very simple. Nevertheless, storing the whole path in the message header results in poor scalability, since header size grows with network size and it consumes network bandwidth as it is transmitted through the network, impacting on network performance.

In distributed-based routing, however, each router is in charge of comput-

ing the next step that the message will take while travelling across the network. The message header only contains a reference to the destination node (usually an identifier or destination coordinate). Distributed routing allows for more flexibility, as it enables the fact that, at each router, different output ports can be available to reach a given destination. Note that with distributed routing, routers require additional logic in order to take the routing decisions, thus increasing the complexity.

Another aspect in the taxonomy of routing schemes is related on the number of paths that are available for the message to be routed. Here we can differentiate two different ways. The first approach is deterministic routing. It means that from a given source node and an specific destination node, a message can only be routed by a single predetermined path. With deterministic routing the traffic of the network may make an unbalanced use of the network links depending on the traffic pattern, thus limiting the performance of the network. On the contrary, an adaptive routing algorithm, as it name implies, adapts the routing of messages as a response of the current network situation (imposed by the traffic and saturation present). But, adaptive mechanisms require complex designs starting from a selection function, that manages the routing choices with the network information that is available. Of course, between a fully deterministic and a fully adaptive mechanism there is a range of hybrid schemes. A quasi-deterministic scheme could offer several complementary predetermined paths instead of a single one, or an adaptive strategy could be restricted to a set of links in each router becoming partially adaptive. The final decision relies, as many aspects before, on a trade-off. For NoC environments, still there is not recent related work on adaptive mechanisms, but as technology evolves, it would be a great asset for better efficiency of the network. There is also a type of routing algorithms, called oblivious routing algorithms [56, 67], that generate paths not taking into account traffic knowledge at all (contrary to deterministic routing that may take few assumptions), and could be interesting for NoCs due to their low overhead once implemented.

There is another major distinction that affects the functionality of a routing scheme. A routing algorithm can be classified as having minimal or non-minimal path support. A minimal path routing strategy will only function on environments where it is assured that between each pair of nodes there

is always a path that is composed of minimal hops to reach a destination node from a source node. This kind of routing is related to regular topology scenarios. As it is shown in Figure 2.25 the path between nodes 13 and 8 is a minimal path because on every hop the message becomes closer to its destination.

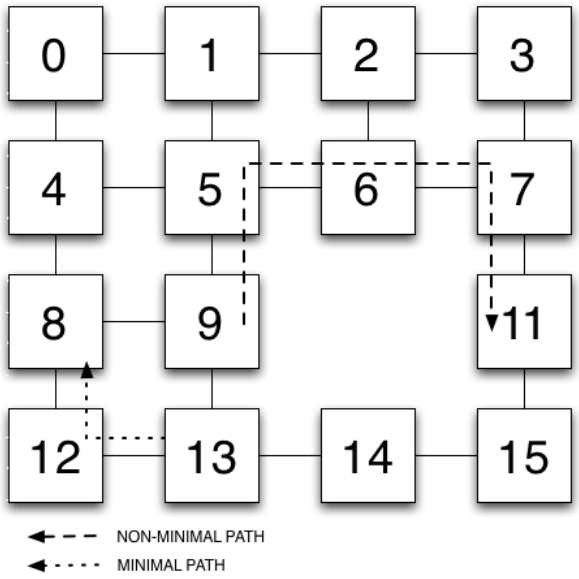



Figure 2.25: Minimal and non-minimal paths.

Instead, a routing algorithm with non-minimal path support is the one that delivers routing decisions that are able to route a message in situations where, at some step, the message is derouted around over other channels. Referring again to Figure 2.25 the path between nodes 9 and 11 would normally cross node 10, but as node 10 is unavailable due, for example, to a failure, the message is derouted through a path surrounding the failed node. In fact, non-minimal paths introduce misrouting and the routing algorithm capable of handling these situations must also take care of deadlock and livelock issues. Of course, in the NoC domain, minimal paths are preferred for shorter routes as they reduce the average message latency, but non-minimal path support is useful in the event of fault-tolerant or congestion-management scenarios.

## 2.3 Related Work

In this Section the most recent contributions to routing in NoCs are described, both for unicast and collective communication. Starting from unicast approaches and ending with techniques that deal with collective communication, the key is to identify their features and how they address the incoming challenges applied to NoCs. Designers must take into account that they have to build effective scalable mechanisms while looking for area, latency and power consumption saves. Good designs require some effort at the on-chip network level. Here we must remember to decouple between routing implementations and routing algorithms. The first ones are mechanisms and techniques applying the second ones, and some implementations may support one or several routing algorithms.

### Routing Algorithms

Dimension-Ordered Routing (DOR) [64] has been widely used for meshes. This routing algorithm forwards every message through one dimension at a time, following an established order of dimensions. Sometimes it is alternatively called $XY$, when applied to meshes. $XY$ is based on a simple idea to avoid deadlock issues (i.e. avoid cycles in the CDG). Messages are first routed in the $X$ dimension to reduce to zero the $\Delta x$ offset between the coordinates of current router and destination router. Then, the next step is to route the message through $Y$ dimension while decreasing $\Delta y$ offset until they reach the destination node. No message is allowed, on the contrary, to first be routed through $Y$ dimension and later through $X$ dimension. Its implementation is very cost-effective, but it is not able to route messages in presence of any irregularity of the current topology used. Different alternative combinations have been proposed, like XY-YX routing [57], where separate virtual channels are used for messages routed in XY or in YX. Note that the combination of two algorithms could end up in a deadlock situation (thus virtual channels guarantee traffic isolation and thus deadlock avoidance). Other derived solutions from DOR are pseudo-adaptive XY [12] and surrounding XY [2].

But let us assume the situation in Figure 2.26. Router at node 15 has failed, so if there is a message from node 14 to node 11, it will never get to its

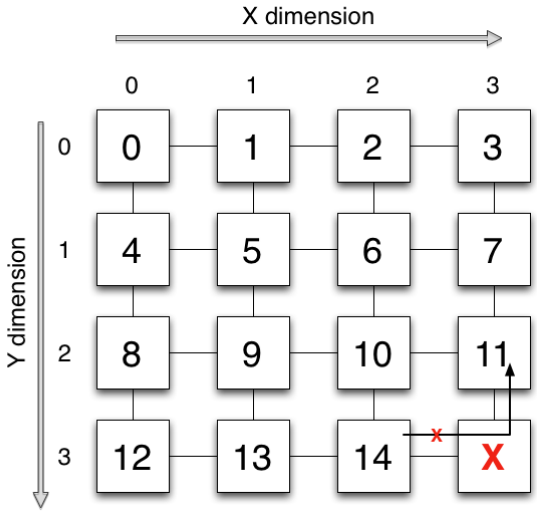



Figure 2.26: $XY$ mechanism is not able to route in the presence of failures.

destination. Message can not traverse through router at 10 as it is not allowed to cross $Y$ dimension before crossing entirely $X$ dimension. It lacks flexibility.

To support irregular topologies (e.g. due to failures), one popular routing algorithm is up*/down* (UD) [55]. It performs a breadth-first search (BFS) from a root node, assigning one direction to each unidirectional channel as *up* (towards the root node) or *down*. The net result is a tree made of up and down links. Computed paths are composed first of a sequence of up links, followed by a sequence of down links, and no message is allowed to traverse an up link after a down link has been used. Further refinements and derivations of this routing algorithm comprehend DFS (depth-first spanning tree) [53], the Flexible Routing scheme (FX) [53] and the left-up-first routing algorithm (LTURN) [25]. All these algorithms need to be implemented in look-up tables (either at the end node if source-based routing is implemented or at the router if distributed-based routing is implemented). Indeed, most of these algorithms have been proposed for off-chip networks, where typically the use of large look-up tables on every router is not a major issue.

One of the main limitations of the UD routing algorithm is the concentration of the traffic (50% when assuming uniform distribution of message destinations) around the root node. Another limitation is the use of non-minimal

paths that tends to increase average message latency. To overcome (or at least attenuate) both issues, there exists a collection of routing algorithms and solutions: In-Transit Buffers (ITB) [16], Layered Shortest Path (LASH) [61], Multiple UD (MUD) [30], Transition Oriented Routing (TOR) [54], and Descending Layers (DL) [26]. Most of these algorithms use virtual channels to allocate different paths that, on the contrary, would create cycles in the CDG, thus inducing deadlock situations. As virtual channels is a precious and expensive resource in NoCs, their use to guarantee deadlock-free routing should be minimized if not used at all. Indeed, these algorithms were also initially proposed for off-chip networks.

One significant routing algorithm, able to deal with any topology, is the Segment-based Routing algorithm (SR) [35]. It renders in a up*/down* tree, however, the way is computed allows for much large flexibility, ending up in many routing instances. SR uses a divide-and-conquer approach, partitioning a topology into subnets, and subnets into disjoint segments, and placing bidirectional turn restrictions (turn restrictions, or alternatively, routing restrictions, are introduced later in the chapter) locally within each segment. SR benefits from a larger degree of freedom compared to previous routing strategies. Indeed, SR has been proposed to tackle the irregularities found in initial chip designs where a 2D mesh has been broken in one or two locations, thus ending in an almost regular 2D mesh structure.

Other routing algorithms, like adaptive-trail [46], minimal adaptive [60],fully adaptive [11] and smart-routing (SMART) [5] achieve performance improvements, but are specific to some (off-chip) network technologies and require extra functionality at the switches or have high computational cost.

All the routing algorithms can be classified as deterministic or adaptive. Most of the routing algorithms, however, are partly adaptive (e.g., UD, DFS, MUD, SR) in the sense that some alternative options are available by the routing algorithm and none of the options lead to a deadlock situation. However, the use of alternative paths may introduce out of order issues, and this could conflict with the application or the cache coherency protocol run on top of the network. Anyway, adaptive routing techniques like turn-model routing and planar adaptive routing [11, 13] offer better throughput and fault tolerance by providing a set of alternative paths, depending on the network congestion and

the presence of faults, and are subject of careful research for the NoC domain.

In the research of new routing algorithms that follow simplicity but effectiveness for NoCs, we must also point to deflective routing [37]. This technique routes messages through one of the profitable output channels (those getting the message closer to its destination). If such channels are busy, then, misrouting is applied. These kind of algorithms can be implemented in buffer-less NoCs.

As opposed to the off-chip domain, power, thermal and reliability issues are important design restrictions in a NoC architecture. In [58], authors propose ThermalHerd, a distributed, collaborative run-time thermal management scheme for on-chip networks to tackle thermal emergencies ensuring thermal safety with little performance impact. Every router is equipped with a set of registers (or counters) to estimate the traffic workload and the temperature sensor to adjust the traffic, but they only support minimal path routing. In [31], authors describe the problem of transient failures of on-chip networks and the impact on applications that require a guaranteed message arrival probability and response time. Their approach combines temporal and spacial redundancy taking into account energy consumption. The drawback of this model is that it is deterministically selected at design time the links to be used by each message and the number of copies to be sent on each link.

As previously commented, one important issue in routing algorithms for on-chip networks is the implementation cost of the algorithm. In the next sections we describe some of the recent proposals that deal also with the implementation cost of the algorithm. We focus our attention first on unicast-based routing algorithms and then on multicast/broadcast solutions. We describe the solutions that are conceived to support fault-tolerance in the network, and the assumed network topology is the 2D mesh. Indeed, different causes may break the initial homogeneous and regular structure of the 2D mesh. In such scenario, efficient routing of messages under irregularly shaped topologies becomes a challenge under the assumption of non expensive routing solutions. To reinforce the need of extending research in the field of fault-tolerance for NoCs, there are some studies [43, 44] that explore the possibilities of how NoC routing algorithms could be developed to provide routing around faults while maintaining the network operational. Specifically, in [43], authors make a

comparison between two methodologies of fault-tolerant routing algorithms, flood-based algorithms and random walk algorithms, both based in probabilistic mechanisms with the support of $n \times n$ probability matrices, being $n$ the number of nodes. In [44], authors propose Immunet, a table-based routing mechanism that tolerates failures for interconnection networks by doing a hardware reconfiguration of all the network resources that have not been affected by the failures. To effectively manage this objective, each router incorporates two system size tables for routing messages through the different virtual networks (to avoid deadlock issues) and one table (as large as the network degree) that records the current shape of the safe topology.

### 2.3.1 Unicast-based Implementations

There are design solutions from the off-chip network domain that could be applied to the NoC field. All these mechanisms do not fit properly in NoCs unless they are thoroughly redesigned. As an example, proposals for TCP/IP protocols [65] are not suitable for NoCs as they rely on message dropping, and would severely affect network performance. There are also techniques used in large parallel systems like the Blue Gene/L system [19] where entire sets of healthy nodes (lamb nodes) are switched off to keep topology and routing algorithm unchanged. Other mechanisms, focused on routing optimization [22], require the use of virtual channels (up to five in some cases) but they do not achieve 100% coverage[3] practically. Also, these mechanisms rely on adaptive routing, and the network must deal with out-of-order delivery issues, a feature that could be difficult to implement in NoCs. Other examples are Interval Routing [68] and extensions [21], which group sets of destinations requesting the same output port, and are an initial attempt to compress the routing table in routers. However, these techniques are not easily applicable to irregular networks.

Street-Sign Routing [4], a source-based routing implementation, compresses the message header so to minimize the impact on network bandwidth. Street-Sign Routing includes only the router id of the next turn and the direction of the turn in the message header. Although message header is reduced it

[3]We define the coverage term as the percentage of failure cases that are supported.

still consumes bandwidth. In addition, a table including the paths for every destination is required at every end node.

Regarding distributed routing solutions, first we focus on Region-Based Routing (RBR) [17] (and a similar proposal [42]). It tries to achieve fault-tolerance in regular and irregular on-chip networks while requiring few resources. At each router RBR groups into a region different destinations that can be reached through a given output port. The main drawback of such mechanism is that, even with 16 regions defined, it still does not achieve 100% coverage [49]. Also, when the mechanism is implemented on a router, it induces a long critical path [49].

Default-Backup Path (DBP) [27] tries to keep healthy processing elements (PEs) when the attached router fails. It consists of adding redundant wiring and buffers that connect output and input ports directly. However, it does not address routing in irregular topologies and requires redundant hardware.

Adaptive stochastic routing (ASR) [63] is a recently proposed routing algorithm (an improvement from the COSR algorithm [38]) that relies on a self-learning method to handle failures by assigning confidence fields to output ports for different tasks (or applications) running in the system. Thus, it requires a routing table at each router, having $n$ entries for $n$ tasks and suffering from the same scalability and cost problems related to routing tables.

In [24], authors propose an architecture based on deflection routing that attempts to detect fault errors by adding CRC modules at input and output ports for crossbar faults, and SEC codes for link faults with the support of routing matrices, one for each type of fault. The link fault matrix is $n \times n$, $n$ being the router radix (i.e the number of inputs/outputs), and each column is also divided into the bit parity. Routing matrices are also $n \times n$, $n$ being the router radix, that represent the routing decisions on a message level, using a variant of deflection routing called delta $XY$ with weighted priority. Deflection (or reflection) can lead to potential starvation solved with message dropping, thus potentially impacting performance.

Another recent proposal, whose objective is to minimize the size of routing tables, either at end nodes or at routers, is described in [3]. In this article, three techniques are proposed: Turn-table (TT), XY Deviation Table (XYDT) and Source Routing Deviation Points (SRDP). All of these techniques consist of a

routing table created by different routing algorithms to handle irregular cases in combination with routing strategies like $XY$ (combined with $YX$), source routing or the *don't turn* technique (meaning that a message must not change direction when traversing the router unless indicated). Deadlock-freedom, however, is not assured in all these strategies (e.g. when changing from $XY$ to $YX$) unless virtual channels are used, and if the technique supports all the possible failure cases (coverage term). Anyway, in the worst case, the supporting routing table will need $n$ entries for $n$ destinations.

In [29], a compendium of state-of-the-art look-up tables (LUTs) implementations for routing purposes is proposed. These designs shift from being fully hardwired to being partially or fully configurable, depending on the degree of flexibility.

Novel implementations based on Dimension-Ordered Routing, like FDOR [62], arise to provide coverage on irregular topologies. This routing methodology is based on the idea of dividing the dimensional mesh irregular topology into regular submeshes, a core mesh and one or more flank meshes. Depending on the division, at the core mesh, messages are routed with $XY$ routing and on the flank meshes with $YX$, or vice versa. One bit per router is needed to configure $XY$ or $YX$ routing. FDOR provides a cheap and efficient routing solution to offer coverage on a set of irregular topologies that abide by certain conditions, but it does not offer full coverage (there is no non-minimal path support).

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

### 2.3.2 Collective Communication Implementations

One of the most recent proposals for tree-based collective communication is Virtual Circuit Tree Multicasting (VCTM) [14]. VCTM builds a dynamic tree mapped on top of a mesh providing support for cache coherence protocols like

Virtual Tree Coherence (VTC) [15]. For its implementation, VCTM uses two sets of tables. First, it uses a content-addressable memory at every end node, storing references for the current active trees. Second, it uses a VCT (Virtual Circuit Tree) table at every router. The number of entries of the VCT table is the product of the number of end nodes times the number of trees used (e.g. 1024 entries for a 16-node system with a maximum of 64 trees per core). This results in significant (and non-scalable) area, delay and power overheads. In addition, VCTM requires virtual channels to keep the ordering of the tree for different regions. Most important, VCTM is designed around *DOR* routing, thus not supporting irregular topologies (2D meshes with potential faulty links or routers).

XHiNoC [51], offers a very detailed multicast router architecture with a different approach, not only focused on coherence protocols. XHiNoC provides a parallel pipelined multicast wormhole switching technique. While XHiNoC is flexible and extensible, it requires (on every router) look-up tables (LUTs) and additional logic allocated at each port. In particular, it uses one table for routing (as many entries as destinations) and a table for message identity management (IDM) to break cyclic dependencies between multicast branches. As an additional feature, XHiNoC uses a hardware logic called LCFS (Link Controller and Flow Supervisor) for controlling the links in the crossbar router to prevent congestion states.

MRR (Multicast Rotary Router) [1], is a router with multicast support based on two concepts. First, a topology-agnostic table-based solution, and second, two internal ring buffer structures at each router, cyclically routing messages in opposite directions. All destinations of the collective operation are encoded in the message header (using one bit per node). Instead of a centralized table, every output port has its own table with the reachable destinations from that output port as a bit-encoded ($N$ destinations, $N$ bits) solution. As said before, memory requirements increase quadratically with the number of end nodes. Also, MRR, has a large number of buffering stages, the ones in input/output ports and the intermediate ones in the rings, increasing power consumption.

RPM (Recursive Partitioning Multicast) [70] is a recent table-less solution for multicast. It uses a minimal logic to partition the entire network (from the

perspective of the source end node), into eight partitions or quadrants. All the destinations of the collective operation are encoded in the message header. RPM generates message replicas at certain output ports based on the assigned partitions and on priorities. The proposal, however, is confined to a limited set of minimal routing paths and does not face the challenges mentioned above that may end up in irregular topologies. In addition, it requires a number of virtual channels on every input port to solve the deadlock problem with multicast traffic. The number of virtual channels depends on the blocking possibilities due to wormhole switching, thus increasing with network and message size.

Following the summary for unicast-related work, multicast/broadcast proposals either rely on the use of tables for routing purposes (and for building multicast trees) and/or rely on simple routing algorithms not adapted to faulty networks.

# Chapter 3

# The Foundations

> *"An idealist believes the short run doesn't count. A cynic believes the long run doesn't matter. A realist believes that what is done or left undone in the short run determines the long run."*
>
> Sydney J. Harris.

The most straightforward solution for routing implementations is the use of routing tables due to its flexibility. Virtually any routing algorithm instance applied to a given topology can be represented as a set of routing entries in a table. On small systems the hardware cost and power consumption related to the memories used to build routing tables is affordable, but as more and more cores are integrated on the chip, causing the system size to grow, the solution becomes expensive due to its poor scalability.

In this chapter, we take a step forward, and propose a new methodology for networks-on-chip routing implementations. The objective of this proposal is to offer a compact, simple and flexible methodology to aid the development of new routing implementations in the on-chip network domain. The basics of this methodology is to offer a simple way to represent the topology and routing algorithm. The aim is to simplify the way we understand a routing algorithm, focusing on local actions performed at each router. Note that we focus only on deterministic and/or partly adaptive routing algorithms, where the CDG is acyclic. Adopting the techniques proposed in this thesis to fully adaptive routing algorithms would be, however, straightforward. In the

following chapters we will describe the method to represent a routing algorithm in order to achieve efficient and scalable routing implementations.

For the sake of explanation, all the network routing implementations shown in this dissertation are applicable to 2-dimensional meshes. Extensions of this methodology to other n-dimensional topologies are beyond the scope of this work. A remark must also be made concerning the interoperability of the tile-based node. We assume only one end node per tile, so the router serving as the interface to the network dedicates one local link. Expanding to tiles with two or more end nodes can be done with the addition of more local links between the router and the network interface.

## 3.1 Methodology

In order to enable an efficient implementation of a routing algorithm, a compact representation of this algorithm is desirable, thus potentially requiring a small silicon area and achieving low latency while providing flexibility. The methodology we propose for such representation is based on two assumptions:

1. The 2-dimensional topology is modelled as a graph where each node contains one router. For the sake of simplification, each router is connected to at most four other routers, serving the communication between tiles. Therefore, the nomenclature for each direction in the network is changed from $X+$, $X-$, $Y+$ and $Y-$ to *east* ($E$), *west* ($W$), *south* ($S$) and *north* ($N$), respectively. Additionally, the bidirectional link connecting the processing device to the network router is called *local* ($L$).

2. For each topology, a set of deterministic (or partially adaptive) routing algorithms is applicable to that topology. Each routing algorithm that can be applied on a 2-dimensional mesh topology would be suitable, but in order to offer efficient routing (see Section 2.2.7) the routing algorithm must comply with some restrictions:

   - *Deadlock-freedom:* The routing algorithm must ensure that the messages routed avoid deadlock scenarios.
   - *Connectivity:* It is mandatory that the routing algorithm is able to offer at least one path between each pair of end nodes.

Once the proper routing algorithm is selected, it is represented on the topology as a set of *routing restrictions*. Routing restrictions are a compact description of any routing algorithm that follows the previous properties aforementioned. The most straightforward solution to ensure deadlock-freedom is to avoid cycles, by preventing some paths. As shown in Figure 3.1, if a message arrives to router 3, that message is forbidden to take the west channel reaching router 2 afterwards.

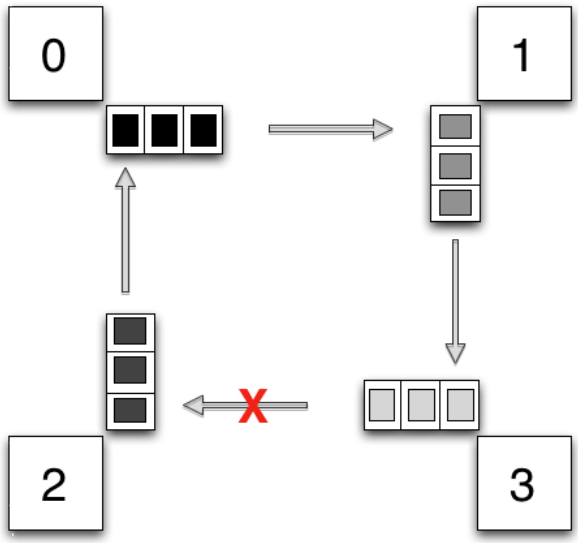



Figure 3.1: Avoiding deadlock by breaking the cycle between routers.

A routing restriction is defined between two consecutive channels ($c_1$, $c_2$), thus, crossing a router. The routing restriction indicates that no message can cross both channels in the order the routing restriction is defined ($c_1$ then $c_2$) consecutively. However, a message can cross both channels if there are other channels in between ($c_1$, then other channels, and then $c_2$). Figure 3.2 shows the routing restrictions on a $4 \times 4$ 2-dimensional mesh for two well-known routing algorithms. Figure 3.2(a) shows an example of the Segment-based Routing algorithm ($SR$) [35] represented by its set of routing restrictions. $SR$ is a topology-agnostic routing algorithm since it can be implemented on any topology. With $SR$, many different instances of the routing algorithm can be obtained by simply placing routing restrictions in different locations (but

ensuring deadlock freedom and keeping connectivity among all the routers). Figure 3.2(a) shows also the example of a valid path and an invalid path crossing a routing restriction.

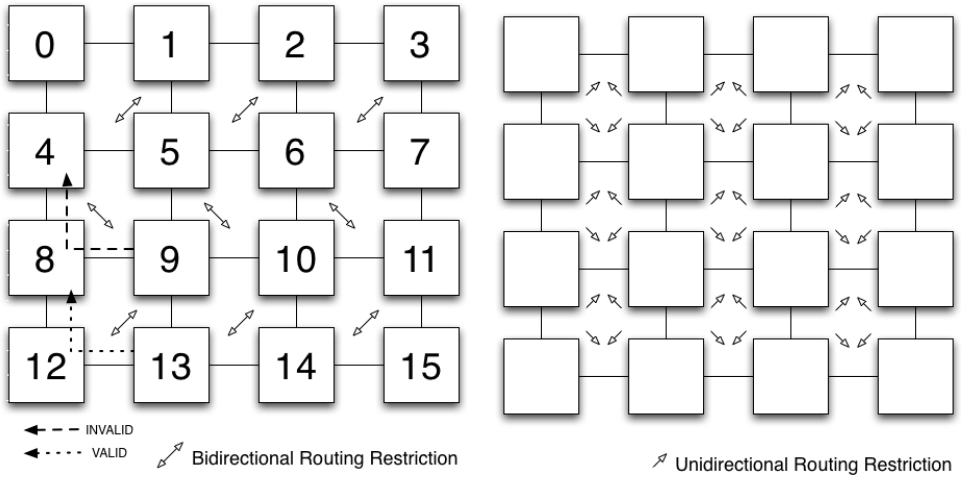
Also, traditional routing algorithms like Dimension-Order-Routing ($DOR$) [64] and the odd-even routing algorithm can be represented by their set of routing restrictions. Figure 3.2(b) shows the routing restrictions for the $DOR$ routing algorithm. Note that, in the $SR$ case, bidirectional routing restrictions are defined. Therefore, two links connected to the same router can not be used in neither direction. In the figure a bidirectional routing restriction is drawn by a bidirectional arrow. For the DOR algorithm, however, only unidirectional routing restrictions are used (represented by unidirectional arrows). The DOR algorithm forbids messages from taking Y-X transitions. Therefore, routing restrictions are only defined from N and S links to E and W links. The opposite transition is allowed by DOR, thus no routing restrictions exist from E and W links to N and S links.

It is important to note that different routing algorithms can be represented with routing restrictions even for irregular topologies. In Figure 3.3, two $SR$ routing instances and the up*/down* routing algorithm are represented for the same irregular topology. As can be seen, different algorithms lead to different locations of the routing restrictions, thus being different algorithms. For some source-destination pairs, non-minimal paths are needed, thus, non-minimal routing algorithms are also compatible with the routing restrictions representation.

Fully routing algorithms (those that allow cycles in the CDG), can be represented by routing restrictions, however, such restrictions only apply to the escape path of such algorithms. Anyway, we focus in this thesis in acyclic routing algorithms.

## 3.2 Configuration bits

Representing a routing algorithm by a set of routing restrictions will allow us a compact and efficient implementation of the algorithm. To allow the usability of this representation it is desirable to translate the graphical representation into real hardware. To do so, we introduce the router configuration bits. We

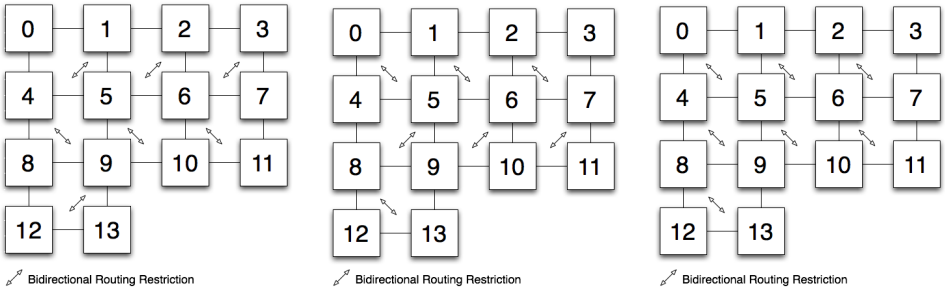



(a) *SR* routing algorithm (b) *DOR* routing algorithm

Figure 3.2: Routing algorithms represented as a set of routing restrictions.

pursue an implementation that is distributed among all routers. Also, we need to track both the routing algorithm (through the location of routing restrictions) and the topology. Therefore, each router will include the configuration bits of neighbour routing restrictions and connectivity patterns.

Each router in the network will handle globally two sets of configuration bits. Each set has a different purpose as their function is to serve as a compact and flexible translation of the current topology and the routing algorithm applied. The first set of these configuration bits is called the *routing bits*




(a) *SR* (b) *SR* (c) up*/down*

Figure 3.3: Different routing instances for the same irregular topology.

($R_{xy}$). Routing bits represent the changes of direction that can be made at the neighbour routers. It means that if a $R_{xy}$ bit is set, the message can be first routed on $x$ direction and at the next router it can be routed through $y$ direction, picked from combinations of the four basic directions: north, east, west and south. There are certain combinations that are not allowed, i.e. to avoid *U turns* as to prevent that a message returns to a router that has previously routed the message. Therefore, as we assume 2D meshes as the initial topology we define twelve routing bits. They are listed and briefly described next:

- $R_{ne}$, $R_{nw}$, and $R_{nn}$. These bits indicate whether messages can take the north port and at the next router the message can be forwarded through the east port, the west port, or the north port, respectively.

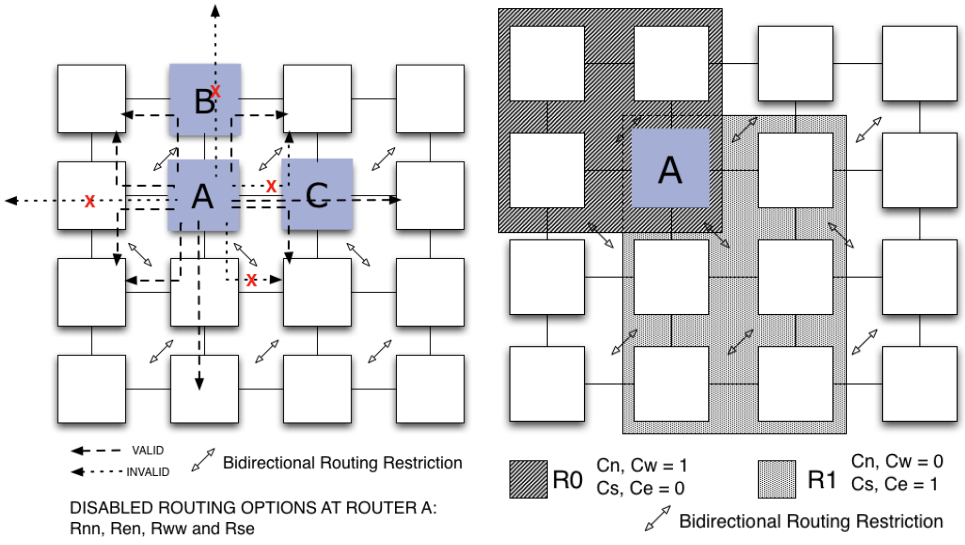
- $R_{en}$, $R_{es}$, and $R_{ee}$. These bits indicate whether messages can take the east port and at the next router the message can be forwarded through the north port, the south port, or the east port, respectively.

- $R_{wn}$, $R_{ws}$, and $R_{ww}$. These bits indicate whether messages can take the west port and at the next router the message can be forwarded through the north port, the south port, or the west port, respectively.

- $R_{se}$, $R_{sw}$, and $R_{ss}$. These bits indicate whether messages can take the south port and at the next router the message can be forwarded through the east port, the west port, or the south port, respectively.

Note that routing bits are the opposite of routing restrictions. Indeed, if a routing restriction exists the associated routing bit is reset. Also, note that routing bits are computed in a straightforward manner, as they mimic the routing restrictions at the neighbour routers. In Figure 3.4(a) there is an example of the allowed and forbidden paths a message could take according to router $A$, and based exclusively on the routing bits. Specifically, bit $Rne$ at router $A$ is set and indicates a message is allowed to go first to the north and afterwards, at the next router, to the east. On the contrary, the routing decision of going first to the east and then to the north is not allowed, due to the restriction present in router $C$, at the east of $A$. Note that four of those

routing bits ($R_{nn}$, $R_{ee}$, $R_{ww}$ and $R_{ss}$) indicate whether a message can advance along the same direction in $X$ or $Y$ dimensions or, on the contrary, either should take a non-minimal path, or has reached the mesh border. Following the description of the example, in the case of router $A$, bit $R_{nn}$ is reset due to the fact that there is no path to the north after traversing router $B$. Note that routing bits associated to local ports (links connecting end nodes to routers) are not needed, since routing restrictions do not exist that affect such ports. Therefore, no additional routing bits are required and, most important, they do not increase as the system size increases.




(a) Routing bits (b) Connectivity bits

Figure 3.4: Configuration bits at a specific router. $SR$ routing algorithm used.

The second set of configuration bits is related to the topology. They are labelled as *connectivity bits*. A $C_x$ bit defines the connectivity at the $x$ output port. For example, if the $C_n$ bit is set it means that there is a neighbour router connected through the $N$ port. $C_x$ bits act as a filter for connectivity and topology definition.

In order to provide a complete example, Figure 3.5 shows the bits computed for a $4 \times 4$ 2D mesh when using the $DOR$ routing algorithm (routing restrictions can be seen in Figure 3.2(b)).

As can be seen in the figure, bits $R_{ne}$, $R_{nw}$, $R_{se}$ and $R_{sw}$ are all set to

| Router | Cn | Ce | Cw | Cs | Rnn | Rne | Rnw | Ree | Ren | Res | Rww | Rwn | Rws | Rss | Rse | Rsw |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 2 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| 4 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 5 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 6 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 7 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 8 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 10 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 11 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 12 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 13 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 14 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 15 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |

Figure 3.5: Routing and connectivity bits for a $4 \times 4$ 2D mesh with *DOR*. Routers are numbered row-wise. (See Figure 3.2(b))

zero, representing the Y-X routing restrictions DOR imposes. Bits $R_{nn}$, $R_{ee}$, $R_{ww}$ and $R_{ss}$ are all set to one (allowed by DOR) except those cases where the message would go out of the network (at the boundaries). $R_{en}$, $R_{es}$, $R_{wn}$ and $R_{ws}$ are set to one (X-Y transitions allowed) except for the cases the message would go out of the network. Finally, connectivity bits are set appropriately and only those cases at the network boundaries are reset. It is worth mentioning now that routing bits that cross network boundaries can be either set or reset depending on the use/intention of such bits. This will be discussed later.

### 3.2.1 Multiple Regions

Although a single bit needs to be used per output port to provide information on topology connectivity, $C_x$ bits may be extended to 8-bit registers, thus providing flexibility when defining regions (which may enable effective virtualization and aggressive power consumption mechanisms). This means, for example, that the $N$ port of a router has its connectivity defined from $C_n[0]$ to $C_n[7]$ for regions 0 to 7. Note that while defining regions or domains, by using connectivity bits, those regions may even overlap. We can find an example in Figure 3.4(b). Router $A$ is shared by two different regions. For region 0 the $C_s$ bit is reset and for region 1 the bit is set. With this configuration, messages (labelled with the appropriate region identifier) can be managed appropriately.

Also, failed links and boundary routers can be configured accordingly.

Figure 3.6 shows a possible static partition of a chip with $8 \times 8$ tiles when using 8 connectivity layers. We define a layer as the regions that can be defined with the same identifier from the connectivity vector. Note that more than eight regions can be defined since regions with the same layer identifier do not need to overlap. Indeed, in the figure we can see how many regions can be defined, some of them irregular and some of them regular ones. Also, failed components (tiles) can be excluded with a proper configuration of the connectivity bits. All the regions in the figure required only 3 identifiers/layers. Thus, with eight identifiers/layers many regions can be defined, different ones for different purposes. The feature of connectivity layers will be exploited by different mechanisms we will define in the following chapters.


Figure 3.6: $8 \times 8$ mesh partitioned into different overlapped regions.

### 3.2.2 Bits Computation

The way these bits (routing and connectivity bits) are computed is critical for the success of the mechanisms. Indeed, not all the combinations of bits guarantee connectivity and/or deadlock freedom. The first step is computing the routing bits ($R_{xy}$) and connectivity bits ($C_x$), which are computed

offline. This is done by analysing the topology (including the failed/powered down routers and links) and applying the routing algorithm. The choice of the routing algorithm is critical since it must guarantee deadlock freedom and connectivity. As by default, Segment-based Routing algorithm ($SR$) [35] is chosen as it is topology-agnostic and does not require virtual channels. Nevertheless, any other topology-agnostic routing algorithm may be used. However, $SR$ provides many instances of valid routing algorithms, thus being flexible. Alternatively, for a healthy chip (no failures) the $XY$ routing algorithm can be used by computing the corresponding bits.

Once the algorithm is selected, the routing algorithm is represented by the routing restrictions it enforces. Routing bits ($R_{xy}$) are then computed by taking into account the location of the routing restrictions. Note that when the the location of routing restrictions is known, the computation of $R_{xy}$ bits is straightforward and thus, its computation complexity is low (linear with the number of routing restrictions). Connectivity bits ($C_x$) are computed based on the existence of links in the router and based on the static partitioning necessities in the chip.

Figure 3.7 shows the algorithm in pseudo-code for the computation of the routing bits and the connectivity bits. As can be seen, first the connectivity bits are computed for each region defined with the appropriate link definition. Function *checkLink* indicates whether a link belongs to a region or not.

In the case of routing bits, from the viewpoint of a router that has a restriction, the routing bit is set accordingly to its immediate neighbour based on the type of restriction. The pseudo-code extends this computation to unidirectional routing restrictions, as any bidirectional restriction can be defined as a pair of unidirectional restrictions. Obviously, the code can be simpler if the designer assumes always bidirectional restrictions. Function *getNeighbour* delivers the identifier of the router attached to a given link to a router. Function *has_restriction* indicates if a routing restriction exists between two links.

Note that routing bits different from $R_{nn}$, $R_{ee}$, $R_{ww}$ and $R_{ss}$, that are not strictly associated to a routing restriction and at least one of the directions is non existent (e.g. outside the network), can be computed either as set or reset depending on the use the routing implementation will make. In Figure

```
Inputs: Topology definition with location of routing restrictions and existent links
Outputs: Configuration bits

Procedure:

for each router
  for each region
    router.Cn[region] := checkLink(North, region); //function checks link at the direction provided
    router.Ce[region] := checkLink(East, region); //at the router for each region
    router.Cw[region] := checkLink(West, region);
    router.Cs[region] := checkLink(South, region);
  end for
end for

for each router //checking restrictions, defined as unidirectional
  getNeighbour(router, North).Rse := router.has_restriction(North, East);
  getNeighbour(router, East).Rwn := router.has_restriction(East, North);
  getNeighbour(router, North).Rsw := router.has_restriction(North, West);
  getNeighbour(router, West).Ren := router.has_restriction(West, North);
  getNeighbour(router, South).Rne := router.has_restriction(South, East);
  getNeighbour(router, East).Rws := router.has_restriction(East, South);
  getNeighbour(router, South).Rnw := router.has_restriction(South, West);
  getNeighbour(router, West).Res := router.has_restriction(West, South);
  getNeighbour(router, South).Rnn := router.has_restriction(South, North);
  getNeighbour(router, North).Rss := router.has_restriction(North, South);
  getNeighbour(router, East).Rww := router.has_restriction(East, West);
  getNeighbour(router, West).Ree := router.has_restriction(West, East);
end for

end procedure
```

Figure 3.7: Pseudo-code for the computation bit algorithm.

3.4(a), at router $B$, routing bit $R_{en}$ can be set or reset, as a message can go first to the east of router $B$, but afterwards, there is no link to the north, so in any case, the connectivity bit could filter the final choice. We will expand this idea in the next chapter. Indeed, the algorithm to compute routing and connectivity bits will be extended in the following chapters to provide a better use and to expand functionality.

## 3.3 Conclusions

Routing and connectivity bits are a simple yet powerful mechanism to route messages in combination with the routing implementations described later. Indeed, the main objective of these bits is to provide support for topologies de-

rived from an initial 2D mesh structure, in some cases leading to non-minimal paths, and to operate with the same degree of flexibility that routing tables would offer. This allows for an efficient method to deal with the challenges of fault-tolerance, power consumption issues, and virtualization-enabled systems. With this methodology, good scalability of the system will be provided. In the next chapters we will take advantage of such bits in order to define efficient implementations of both unicast and broadcast/multicast communication.

# Chapter 4

# Unicast Communication

*"Do or do not... there is no try."*

Yoda, Jedi Master.

In this chapter, we propose a compact, simple and flexible routing mechanism for unicast communication that removes the need for routing tables at every router, thus enabling the distributed implementation of any routing algorithm on regular and irregular topologies derived from 2-dimensional meshes. Three different mechanisms are proposed, starting with the simplest one with acceptable coverage results and ending with the most complete one providing full coverage. All the mechanisms rely (partly or completely) on the foundations described in the previous chapter. The complete mechanism offers full coverage and is prepared to face the new challenges present in routing implementations in the on-chip network domain. We define the term coverage as the percentage of irregular topologies derived from a 2D mesh topology that are supported by the routing algorithm and its implementation.

The chapter is organized as follows. From Section 4.1 to Section 4.4 we describe the evolution of the mechanism from its grounds, its very basic implementation and evolving until full coverage of irregular topologies is provided. Then, in Section 4.5 deadlock freedom and connectivity is discussed with a formal demonstration. In Section 4.6 two real router implementations are presented with the complete unicast routing mechanism, each one conceived both for MPSoCs and CMPs. In Section 4.7, evaluations and results are presented,

including the hardware overhead for the routing implementations. Finally, in Section 4.8, some conclusions are provided.

## 4.1 LBDR: Logic-based Distributed Routing

DOR can be represented with the foundations presented in the previous chapter. Figure 4.1 shows the routing restrictions for DOR in a $4 \times 4$ 2-dimensional mesh. With this representation, the properties of the routing algorithm still remain. As an example, there is a routing restriction at router 10 that forbids messages coming from the south being routed to the east. Indeed, the routing bit $R_{ne}$ at router 14 is reset. The choice of the routing algorithm that will serve to define the valid routing paths, represented as a set of routing restrictions, is key to the success of a routing implementation. We must ensure that the chosen routing algorithm is able to offer full connectivity between each pair of end nodes, including topologies with irregularities and maintaining the deadlock-freedom condition. As discussed in Section 2.3, for the sake of the routing implementation, Segment-Based Routing [35] algorithm is used for the rest of the dissertation.

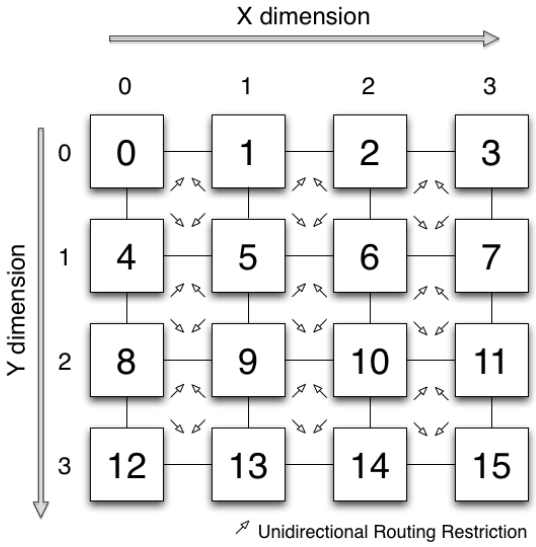



Figure 4.1: Routing restriction representation of DOR algorithm on a $4 \times 4$ 2D mesh.

*Logic-based Distributed Routing* (LBDR) [18] is the simplest routing implementation based on the routing and connectivity bits methodology presented in the previous chapter. This first basic mechanism is designed only for routing algorithms where minimal path support is guaranteed in the applied topology. See an example in Figure 4.2, where there are some minimal paths displayed for different pair of sources and destinations. In addition, Figure 4.3 shows several topologies where LBDR can be applied.

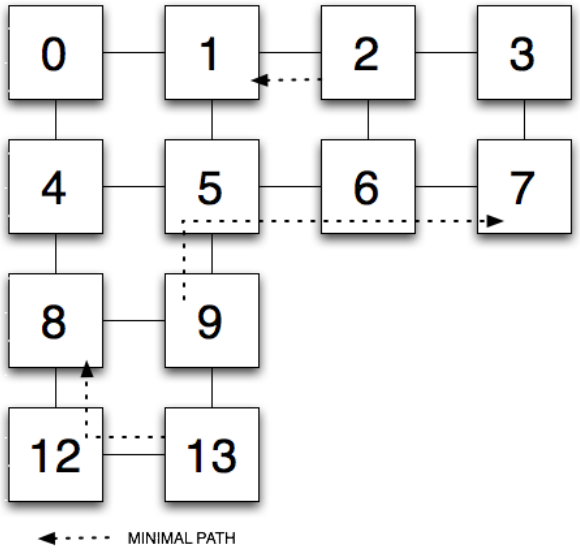
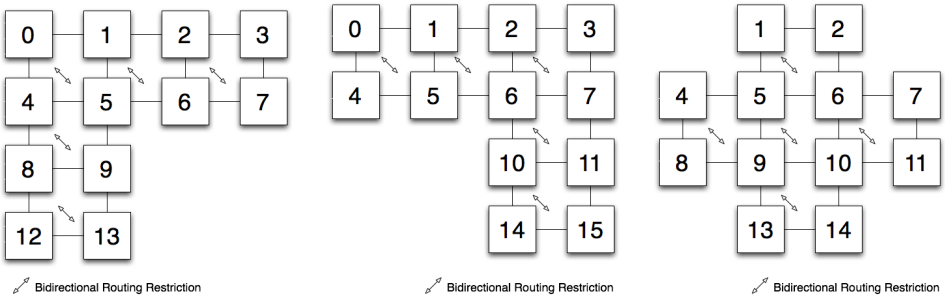



Figure 4.2: Example of minimal paths.




(a) $p$ layout (b) $q$ layout (c) + layout

Figure 4.3: Topologies supported by LBDR.

As minimal path is guaranteed by definition, LBDR requires only eight routing bits per router, as $R_{xx}$ bits are always set to one. In particular: for output port $N$, $R_{ne}$ and $R_{nw}$ bits; for output port $E$, $R_{en}$ and $R_{es}$ bits; for output port $W$, $R_{wn}$ and $R_{ws}$ bits; and for output port $S$, $R_{se}$ and $R_{sw}$ bits. For the connectivity bits the mechanism relies only in one layer, thus no vector bits are used for the connectivity. Instead, only four bits per router ($C_n$, $C_e$, $C_w$, and $C_s$) are used. Notice, however, that connectivity layers can be added in an straightforward manner (at a slight cost increase).

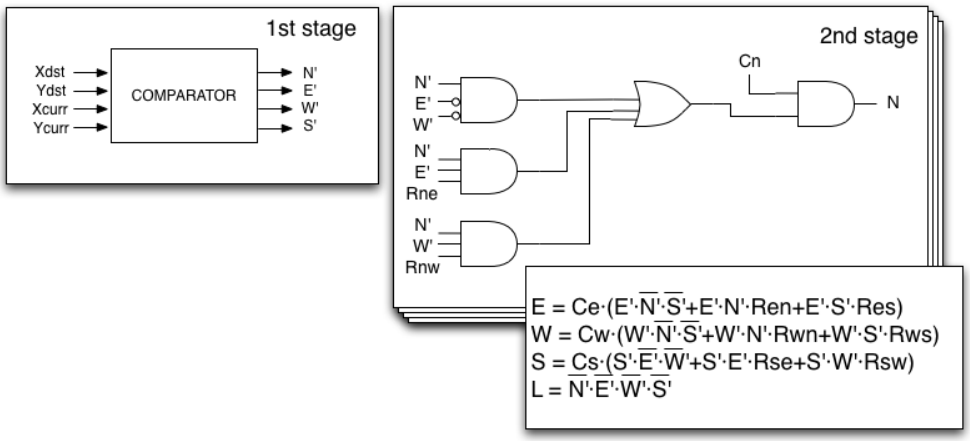



Figure 4.4: *LBDR* implementation, detail on the north output port case.

*LBDR* is based on the relative position in the mesh of the router the destination node is attached to and the current router. A general schematic of *LBDR* implementation is shown in Figure 4.4. First, in order to compare both positions, the COMPARATOR module is used at the first stage, which generates four control signals. These signals, $N'$, $E'$, $W'$ and $S'$, indicate the relative position of the final router from the viewpoint of the current router. For example, in Figure 4.5, if the current router is 5 and our destination is router 2, signals $N'$ and $E'$ would be activated, because it is located at the north-east quadrant. With these control signals, and using the routing ($R_{xy}$) and connectivity ($C_x$) bits, the *LBDR* mechanism computes a set of routing decisions in the second stage.

The second stage requires four logic units, one for each output port. Each one can be implemented with only two inverters, four AND gates and one OR

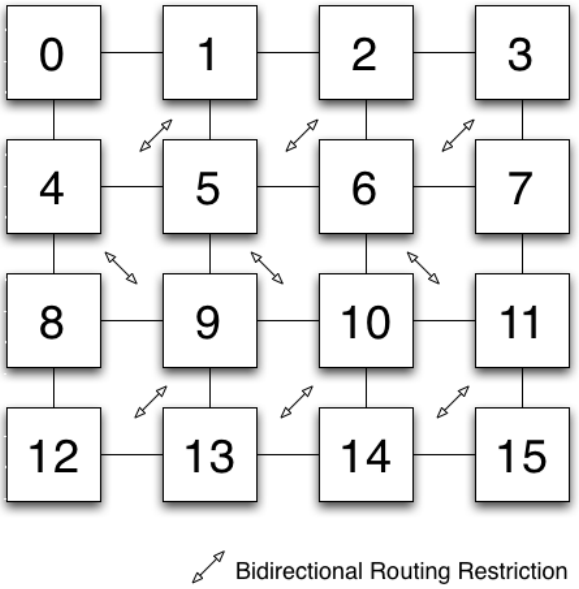



Figure 4.5: $4 \times 4$ 2D mesh with routing restrictions applied from the $SR$ algorithm.

gate. As all of them are similar we describe here only the logic associated with the $N$ output port.

The $N$ output port is considered for routing the incoming packet when either one of the following three conditions is met. If none of the conditions is met, then the $N$ port can not be considered for routing the packet (additionally, the connectivity bit $C_n$ is inspected in order to filter the $N$ port):

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

As stated, this logic provides support for minimal paths in the network, and generates a signal per output port. The $L$ signal is set when the message has reached the final router destination, which equals when $N'$, $E'$, $W'$ and $S'$ signals are reset.

| Router | Cn | Ce | Cw | Cs |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 2 | 0 | 1 | 1 | 1 |
| 3 | 0 | 0 | 1 | 1 |
| 4 | 1 | 1 | 0 | 1 |
| 5 | 1 | 1 | 1 | 1 |
| 6 | 1 | 1 | 1 | 1 |
| 7 | 1 | 0 | 1 | 1 |
| 8 | 1 | 1 | 0 | 1 |
| 9 | 1 | 1 | 1 | 1 |
| 10 | 1 | 1 | 1 | 1 |
| 11 | 1 | 0 | 1 | 1 |
| 12 | 1 | 1 | 0 | 0 |
| 13 | 1 | 1 | 1 | 0 |
| 14 | 1 | 1 | 1 | 0 |
| 15 | 1 | 0 | 1 | 0 |

| Rne | Rnw |
|---|---|
| 0 | 0 |
| 0 | 0 |
| 0 | 0 |
| 0 | 0 |
| 1 | 0 |
| 1 | 1 |
| 1 | 1 |
| 0 | 1 |
| 1 | 0 |
| 1 | 1 |
| 1 | 1 |
| 0 | 1 |
| 1 | 0 |
| 1 | 1 |
| 1 | 1 |
| 0 | 1 |

| Ren | Res |
|---|---|
| 0 | 1 |
| 0 | 1 |
| 0 | 1 |
| 0 | 0 |
| 0 | 1 |
| 0 | 1 |
| 0 | 1 |
| 0 | 0 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 0 | 0 |
| 0 | 0 |
| 0 | 0 |
| 0 | 0 |
| 0 | 0 |

| Rwn | Rws |
|---|---|
| 0 | 0 |
| 0 | 1 |
| 0 | 1 |
| 0 | 1 |
| 0 | 0 |
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 0 | 0 |
| 0 | 1 |
| 0 | 1 |
| 0 | 1 |
| 0 | 0 |
| 1 | 0 |
| 1 | 0 |
| 1 | 0 |

| Rse | Rsw |
|---|---|
| 1 | 0 |
| 1 | 0 |
| 1 | 0 |
| 0 | 0 |
| 0 | 0 |
| 0 | 1 |
| 0 | 1 |
| 0 | 1 |
| 1 | 0 |
| 1 | 0 |
| 1 | 0 |
| 0 | 0 |
| 0 | 0 |
| 0 | 0 |
| 0 | 0 |
| 0 | 0 |

Figure 4.6: Routing and connectivity bits computed for $SR$ algorithm on a $4 \times 4$ 2D mesh.

### 4.1.1 Detailed Example

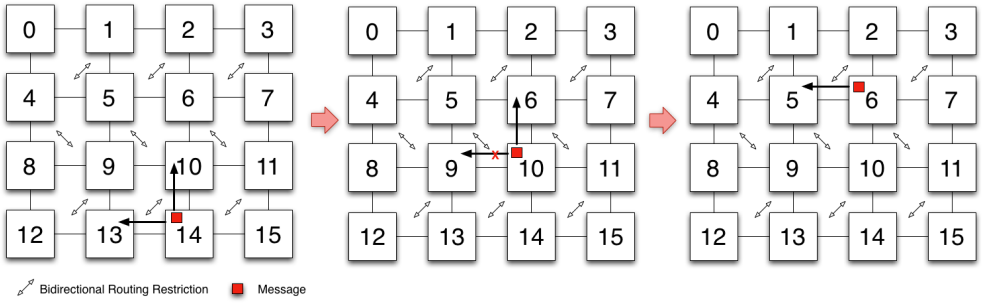



Figure 4.7: Example of routing decisions in $LBDR$.

Figure 4.7 shows an example of routing in $LBDR$. Router 14 wants to send a message to destination router 5. At router 14, signals $N'$ and $W'$ are activated at the first stage of $LBDR$ as destination is on the $NW$ quadrant. At the second stage, $N$ signal is considered for routing as $N'$ and $W'$ signals are active, $R_{nw}$ is set (see Figure 4.6), and there is connectivity to the north ($C_n$ is also set at that router). Note that also $W$ output port is valid to route the message, as $R_{wn}$ is set, and there is connectivity to the west. However, let us assume that the arbiter chooses the $N$ output port finally, therefore the

message reaches router 10. At the next hop, at router 10, a routing operation is started again. $N'$ and $W'$ are active, again. After the second stage, $LBDR$ provides $N$ direction as a valid choice, but unlike in the previous case, $W$ is discarded, because of the $R_{wn}$ bit at router 10, which represents the routing restriction at router 9. Message is sent north to router 6. At this router, the process is repeated again, as the message is still not at its destination. In this case, only $W'$ signal is active, and after the routing process, $W$ is the only routing option available. The message is forwarded to router 5, where it will be delivered to the local port connected to the node, as the message has arrived to its destination ($L$ signal is activated).

### 4.1.2 Arbitration Issues

Note that several signals ($N$, $E$, $W$, or $S$) could be activated at the same time for a message, as there can be more than one valid output choice at the same time. This occurs when partly adaptive routing algorithms are implemented on top of $LBDR$. For instance, with XY routing only one output port is provided regardless of the positions of the current router and the destination router. For other algorithms, however, different output ports can be considered. This is the case in Figure 4.5, for a message at router 9 that has its destination at router 3. The message can use ports $N$ and $E$ at router 9 (the same at router 10). If we need to enforce deterministic routing, then both output ports can later be filtered by the arbiter that selects the final port according to its priority strategy, whether adaptiveness is allowed or the routing algorithm is deterministic. After the output port choice is considered, the arbiter configures the crossbar to route the message to the corresponding output port.

However, in some NoC designs the arbiter may be simple and does not allow multiple routing options from the same input port. This is the case of routers designed with a two-phase arbiter where arbiters are implemented only at the output boundaries of the router (commonly know as allocators). To tackle with this issue, we provide two alternative basic modifications to the logic in order to provide only one routing option per input port. In the first one, referred to as *fixed priorities*, each input port will filter some routing options in order to provide only one. This will be done locally at every input port, thus no need for communication between different $LBDR$ implementations at

each input port of the router. The second one, referred to as *smart*, will make more elaborated decisions when filtering the routing options.

### Fixed Priorities

$LBDR$ logic provides the following sets of two routing options: $NE$, $ES$, $SW$, and $WN$. This is because all the provided paths are minimal within the topology and the messages are forwarded to one of the possible quadrants ($NE$, $ES$, $SW$, and $WN$ quadrants). The idea behind *fixed priorities* is to filter such routing options but providing equal probabilities to every output port. This is achieved by providing higher priority to each output port in a different quadrant. Figure 4.8 shows the extended $LBDR$ logic providing priorities to the $N$ port for the $NE$ quadrant, $E$ port for the $ES$ quadrant, $S$ port for the $SW$ quadrant, and $W$ port for the $WN$ quadrant.

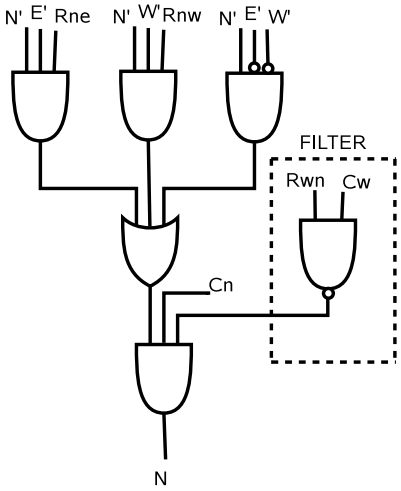



Figure 4.8: LBDR with fixed priorities.

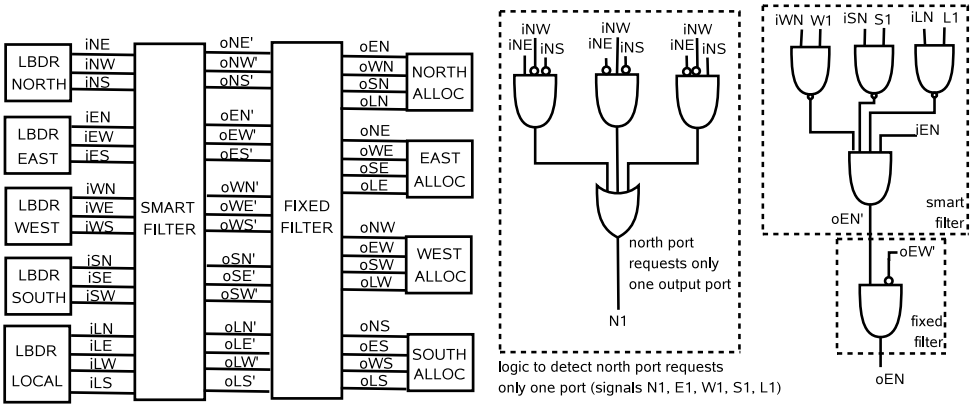
In particular, for the $N$ port, the new logic filters the port if the packet can be forwarded also through the $W$ port (in that case the $N$ port is not eligible for routing purposes). Notice that the $N$ port is not filtered if the packet is going only through $N$ direction or through the $NE$ quadrant (the $N$ port has priority in this quadrant). Similar deductions can be obtained for the remaining set of logic equations.

The main benefit of *fixed priorities* is the fact that LBDR logic is compact and still isolated at every input port. At the end *LBDR* provides only one routing option per input port.

### Smart Priorities

The original LBDR logic providing more than one routing option can be easily coupled with a three-phase arbiter. At the first phase each input port provides requests to the output ports (more than one is accepted). At the second phase each output port independently selects just one routing option and notifies it back to the input port. At the third phase the input port selects one of the accepted routing options. The main drawback of the three-phase arbiter approach consists of the increased arbitration latency and thus increased packet latency. This is one of the reasons to prefer two-phase arbiters in NoCs.

However, in some cases there is still margin to implement some common logic sharing all the requests from input ports in order to make smarter arbitration decisions. It is a design that fits with the lightweight implementation of routers with two-phase arbiters.




(a) Filter and control signals layout (b) Details of a control signal

Figure 4.9: LBDR with smart priorities.

Figure 4.9 shows a possible implementation of a *smart priority* mechanism coupled with *LBDR*. In this case, there is a common logic (Figure 4.9(a)) that reads all the routing options provided by every input port at each possible

arbitration cycle. In particular, three control signals are activated from each input port, plus four control signals from the local port. Each control signal, labelled $iXY$, indicates if input port $X$ is requesting output port $Y$. Possible ports are $NEWS$ and $L$ (local port). The filter logic provides four control signals to every output port of the router, thus providing sixteen control signals. The output signal labelled $oXY$ means input port $X$ finally requests output port $Y$. Notice that the filter logic will assert only one control signal for every output port (remember that a two-phase arbiter is assumed).

The filter logic is shown in Figure 4.9(b). The goal of the filter logic is to remove routing options whenever the requesting input port is providing two routing options and at least there is another input port (or local port) requesting only that output port. In other words, priority is given to input ports requesting only one output port by filtering routing options whenever possible. The figure shows a possible logic implementation for such mechanism. The first part of the logic computes five internal control signals ($N1$, $E1$, $W1$, $S1$, and $L1$). $X1$ means $X$ port is requesting only one output port.

The second part of the logic filters routing options. Focusing on routing option $iEN$ (input port $E$ requests output port $N$) we can see that this routing option is filtered (output signal $oEN'$ is reset) by the smart filter if there is at least one input port requesting only that output port (control signal $iWN$ is set and input port $W$ is only requesting one output port, $iSN$ is set and input port $S$ is only requesting one output port, or $iLN$ is set and input port $L$ is only requesting one output port). The remaining routing options are filtered in the same way and in parallel. Also, notice that routing options are filtered again using a fixed priority module in order to provide one routing option per input port. In the provided example the final $oEN$ routing option is filtered if the input port is also requesting output port $W$ (for the quadrant $NW$ west direction has priority).

### 4.1.3 More Cores inside a Node

Although for this work, it is assumed that a tile-based node contains only one core attached to the router, other designs could include two or more cores per router. Under certain operating conditions, concentrated topologies become attractive. Basically, the idea consists of reducing the number of topology

dimensions or (as in our case) of routers in each dimension of a $k$-ary 2-mesh and to increase the number of cores attached to each router, instead of assigning only one core per node. This way, bisection bandwidth is traded for low latency, area and power in this kind of topologies. On the other hand, assigning more cores to a router can increase the complexity of it, specially in the arbiter and routing modules, resulting in an impact of area, power and latency, so the designer must evaluate the trade-off of these kind of topologies.

LBDR can also be extended to support multiple cores per router in concentrated mesh topologies. For this purpose, a different labelling scheme had to be devised, since $LBDR$ requires the router coordinates within the 2D mesh. Now, multiple cores might be associated with the same router coordinates. The basic idea is that (see Figure 4.10) one local core inherits the same coordinates of the router, while the other ones have an incremental $x$ coordinate. From a network viewpoint, $x$ coordinates of the routers appear to increase at a coarse granularity, where the granularity is determined by the number of cores attached to each router. The figure illustrates the case with 4 cores per router.

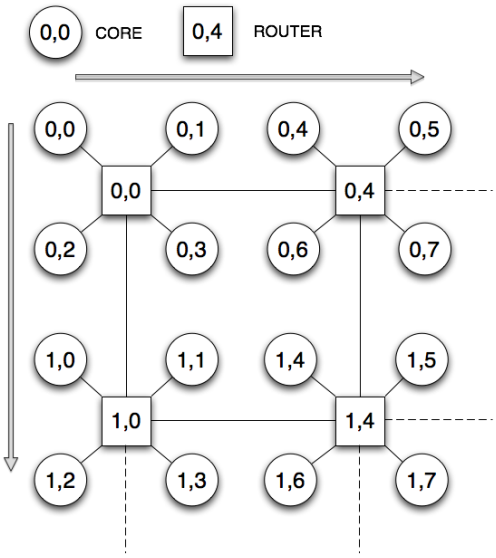



Figure 4.10: LBDR oriented node labelling in concentrated $k$-ary $n$-mesh topologies.

### 4.1.4 Configuration Bits Computation

The routing and connectivity bits are computed for $LBDR$ with the algorithm described in the previous chapter (foundations). That is, connectivity bits are set for those channels that connect to routers and routing bits are set for those cases where no routing restrictions exist. However, there is a special case for the $LBDR$ mechanism to cover with a slightly change of the routing bits significance. See the example in Figure 4.11. In this figure, although there are some failed routers that make the topology irregular, each pair of end nodes can still be connected by a minimal path. In this specific case (or similar where a submesh is removed from the original mesh), $LBDR$ to route a message from router 14 to router 7 needs a slight change of the routing bits at the routers in the boundary. In particular, with no change of routing bits, the $R_{ne}$ bit at router 14 would be reset as there is no connectivity to the east of router 10 (we can think there is a routing restriction at router 10 between $S$ and $E$ ports). However, such situation will impede a message at router 14 to be properly routed: the $N$ port will be dismissed as $R_{ne}$ bit is reset and the $E$ port will be dismissed as $C_e$ bit is reset.

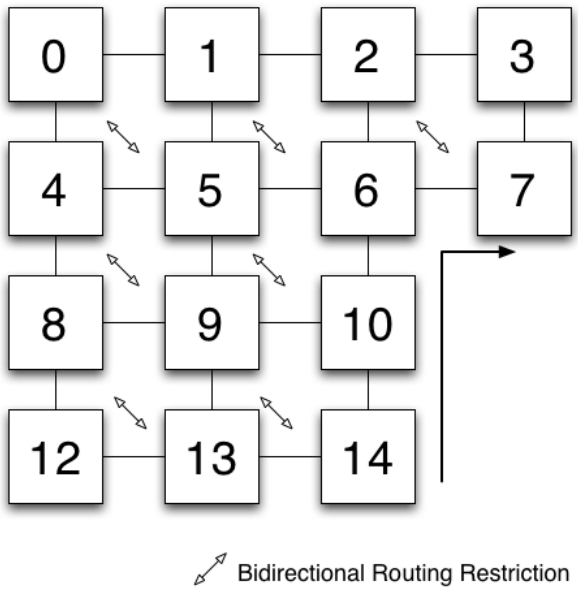



Figure 4.11: A $p$ irregular topology that can be supported by $LBDR$.

This is an important interpretation of the routing bits. Indeed, routing bits are expected to reflect current routing restrictions in the topology, and therefore, if the routing bit is associated with a channel that does not exist in the topology (the case of the east port at router 10) then it is expected the associated routing bit will be set to zero. However, in that case, a message going at a router set in the NE quadrant would filter the N port as the $R_{ne}$ bit is reset. Thus, the message can not be routed although there is a valid minimal path. To overcome this issue, all the routing bits associated with missing links in the network will be set to one. Note that indeed, there is no routing restriction if one of the links does not exist, thus routing bits really mimic routing restrictions. Take into account also, that in the next improvements of the mechanisms those bits can be set to zero (as new functionality will override such cases).

Figure 4.12 shows all the routing and connectivity bits for the topology and routing algorithm shown in Figure 4.11. As can be noticed routing bit $R_{ne}$ at router 14 is set to one.

| Router | Cn | Ce | Cw | Cs | Rne | Rnw | Ren | Res | Rwn | Rws | Rse | Rsw |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 2 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 4 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 5 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 6 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 7 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 8 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 9 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 10 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 11 | - | - | - | - | - | - | - | - | - | - | - | - |
| 12 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 13 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 14 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 15 | - | - | - | - | - | - | - | - | - | - | - | - |

Figure 4.12: Routing and connectivity bits of $p$ irregular topology shown in Figure 4.11.

### 4.1.5 LBDRe

At the first performance evaluations of the $LBDR$ mechanism under the Noxim (a network-on-chip simulator) platform [40], some degradations of performance were detected when using some routing algorithms. Specifically, the tests were made confronting up*/down* ($UD$), Segment-Based Routing ($SR_h$) and $XY$ under different routing implementations, $LBDR$ and routing tables (and subsequently, $LBDRe$). A particular result of the test is shown in Figure 4.13. As it can be seen, tests under $SR$ algorithm perform worse than ones with $UD$ routing algorithm. However, the interesting point is that $SR$ implemented with $LBDR$ works worse than when implemented with tables.

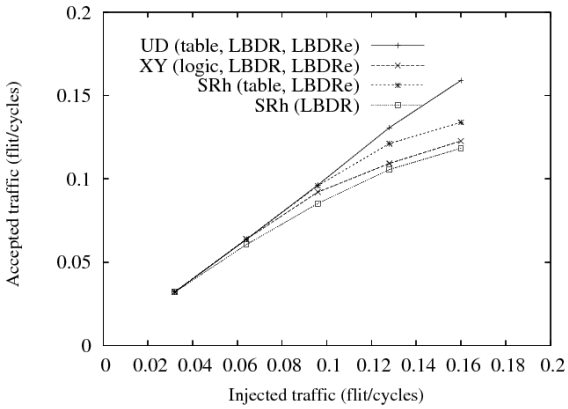



Figure 4.13: Results of performance tests with $LBDR$.

The reason behind this degradation can be explained with the example provided in Figure 4.14. A message from router 1 is sent to destination router 8. At the first router, $S$ direction is discarded as a valid routing option by the $LBDR$ mechanism, preventing that a possible message crosses the routing restriction at router 5. So the path followed by the message is $1--0--4--8$, but $1--5--9--8$ would also be a valid path as the message does not cross any routing restriction. However, in order to guarantee deadlock-freedom and being conservative, i.e. no cycles are formed when routing a message, the possible adaptiveness a routing algorithm could offer is reduced, so reducing overall performance. Indeed, at router 1 the LBDR mechanism does not see that two rows below messages could take $W$ turns, thus sends the message

through the $W$ port at router 1. This is a conservative decision.

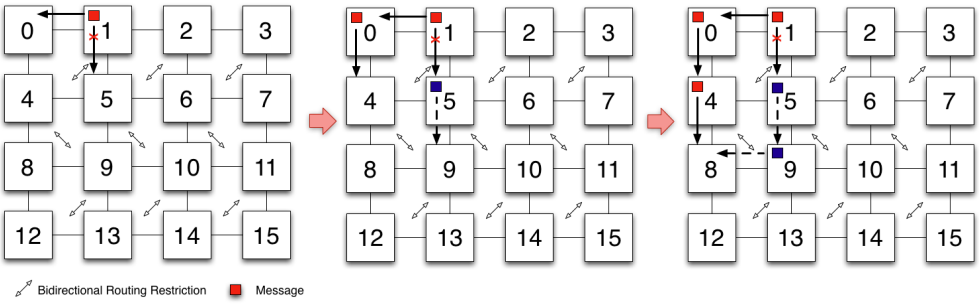



Figure 4.14: Alternate routing decisions that will not lead to deadlock events.

Figure 4.15 describes the extended LBDR method to overcome this situation. We refer to is as *LBDRe* (*LBDR* extended). The set of routing and connectivity bits from *LBDR* are maintained in this extension. Four new bits per router output port are, however, added.

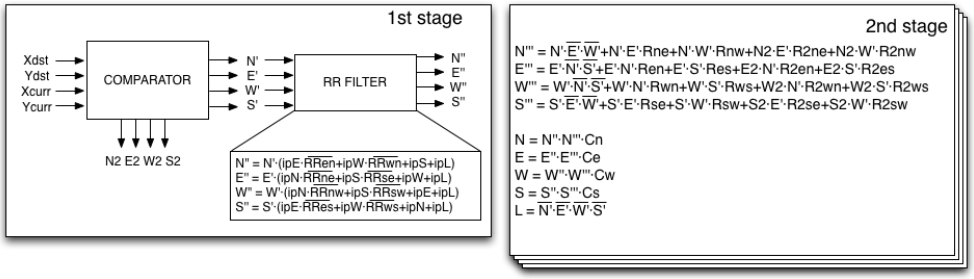



Figure 4.15: *LBDRe* implementation.

The bits labelled $R2_{xy}$ indicate whether the $y$ direction can be taken two hops away from the current router through the $x$ direction. For example, $R2_{ne}$ indicates whether a message is allowed to change direction to $E$ at the router located two hops in the $N$ direction. Notice that this set of bits have similar meaning with the ones used in *LBDR*. In some sense, these bits provide visibility to the current router of the routing possibilities two hops away. For instance, in Figure 4.16, which has the computed set of bits related to Figure 4.14 for *LBDRe*, the $R2_{sw}$ bit at router 1 is set. This means, that two hops away from this router in $S$ direction, it is allowed to route a message to $W$

| Router | Cn | Ce | Cw | Cs |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 2 | 0 | 1 | 1 | 1 |
| 3 | 0 | 0 | 1 | 1 |
| 4 | 1 | 1 | 0 | 1 |
| 5 | 1 | 1 | 1 | 1 |
| 6 | 1 | 1 | 1 | 1 |
| 7 | 1 | 0 | 1 | 1 |
| 8 | 1 | 1 | 0 | 1 |
| 9 | 1 | 1 | 1 | 1 |
| 10 | 1 | 1 | 1 | 1 |
| 11 | 1 | 0 | 1 | 1 |
| 12 | 1 | 1 | 0 | 0 |
| 13 | 1 | 1 | 1 | 0 |
| 14 | 1 | 1 | 1 | 0 |
| 15 | 1 | 0 | 1 | 0 |

| Rne | Rnw | R2ne | R2nw |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 |

| Ren | Res | R2en | R2es |
|---|---|---|---|
| 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |

| Rwn | Rws | R2wn | R2ws |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |

| Rse | Rsw | R2se | R2sw |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |

Figure 4.16: Routing and connectivity bits computed for $LBDRe$.

direction. There is, too, a set of bits labelled $RR_{xy}$, which indicate whether there is a routing restriction between $x$ and $y$ links at the current router. These bits are needed in order to avoid the formation of cycles, which is described in the example below.

To sum up, $LBDRe$ requires 24 routing bits grouped by 6 bits per output port. Additionally, the router needs five internal signals $ipN$, $ipE$, $ipW$, $ipS$ and $ipL$ to indicate the incoming port of the message being routed.

The first part of the routing logic is slightly augmented compared to $LBDR$. In particular, based on the $X$ and $Y$ coordinates of the current router and the destination router, the logic computes the relative directions $N'$, $E'$, $W'$, and $S'$. Additionally, four extra signals, $N2$, $E2$, $W2$ and $S2$, are computed. These signals are set if the destination of the message is at least two hops away in the corresponding direction (if $N2$ is set, then at least two hops must be taken in the $N$ direction to get closer to its destination). Note that these signals can be easily computed with additional comparators with the $X_{curr}$ and $Y_{curr}$ coordinates shifted in one position.

The first part of the logic is also in charge of inhibiting the possible output ports that would lead crossing a routing restriction. For this, the $RR$ (routing restriction) filter logic is used. This logic requires two inverters, three AND gates and one OR gate per output port. The resulting signals are labelled as $N''$, $E''$, $W''$, $S''$. They feed the final part of the logic.

The second part evaluates the routing options at the one-hop and two-hops neighbours. For this, the previous logic functions for $LBDR$ have been extended. For instance, for the output port $N$, the port will be selected if any one of the following conditions are met:

- The destination is on the same column ($N' \times \overline{E'} \times \overline{W'}$).
- The destination is on the $NE$ quadrant and the message can take the $E$ port at the next router through the $N$ output port ($N' \times E' \times R_{ne}$).
- The destination is on the $NW$ quadrant and the message can take the $W$ port at the next router through the $N$ output port ($N' \times W' \times R_{nw}$).
- The destination is on the $NE$ quadrant and is at least two hops away through the $N$ port, and the message can take the $E$ port at the two-hops neighbour router through the $N$ port ($N2 \times E' \times R2_{ne}$).
- The destination is on the $NW$ quadrant and is at least two hops away through the $N$ port, and the message can take the $W$ port at the two-hops neighbour router through the $N$ port ($N2 \times W' \times R2_{nw}$).

Finally, the connectivity bit $C_n$ and the routing-restriction filter ($N''$) are used to filter the output port. For the remaining ports, similar deductions are considered.

Note that the $LBDRe$ mechanism provides both paths explained in the example in Figure 4.14. At router 1, the $S$ output port can now be taken because the $R2_{sw}$ bit is set, so the internal $S2$ signal will be activated. Note also that router 5 has its $RR_{nw}$ bit active, thus avoiding taking the $W$ output port at the current router, which would lead to an invalid path.

It is worth mentioning that this implementation ($LBDRe$) may provide diminishing returns while requiring a larger area requirement for its implementation (more configuration bits and filter logic). Indeed, performance deviations between $LBDR$ and $LBDRe$ were analysed and only in algorithms like $SR$ differences were appreciated, although always lower than 3% in throughput increase [50]. With some routing algorithms like $UD$ (see Figure 4.13), the improvement is non-existent as the paths implemented with $LBDR$ are the same as the ones implemented with $LBDRe$. Note that routing restrictions are aligned (thus, one hop visibility equals two hop visibility). Since we advocated for simplicity at previous chapters, we considered the complexity added in $LBDRe$ being not worth for such marginal performance benefits. Moreover, extending the visibility of the $LBDR$ mechanism, as in $LBDRe$ with

two hops, to three hops or more, just increased the complexity with no gain whatsoever. As this path is not taken, we revert to the basic *LBDR* mechanism and in the following sections we enhance the mechanism so to provide non-minimal path support.

## 4.2 Deroutes

With *LBDR*, a topology will be supported if minimal paths are guaranteed to exist for each pair of communicating devices. But, let us assume a case where some links close to a router at the mesh present manufacturing defects that renders them inoperative. In this case, a message coming from the router trying to reach the adjacent routers would require non-minimal paths.

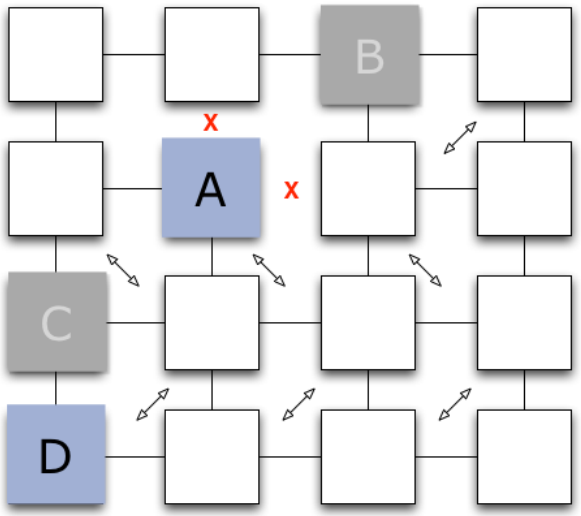



Figure 4.17: Some links are inoperative forcing non-minimal paths.

Figure 4.17 shows such a case, which would not be supported by the *LBDR* mechanism presented before. The reason is that the path from $A$ to $B$ cannot be supported. At router $A$, the possible directions to reach $B$ are $N$ and $E$, however, both links are missing, and therefore there is no possible way to reach $B$ through minimal paths. This motivates for a extension to the *LBDR* mechanism: the deroute logic. The deroute logic, when properly configured, acts as a filter to the output choices provided by *LBDR* by introducing the

possibility of using non-minimal paths. Figure 4.18 shows an example. At router $A$ messages coming from the south port and going north need a west deroute, whereas messages coming from the west port and going east need a south deroute.

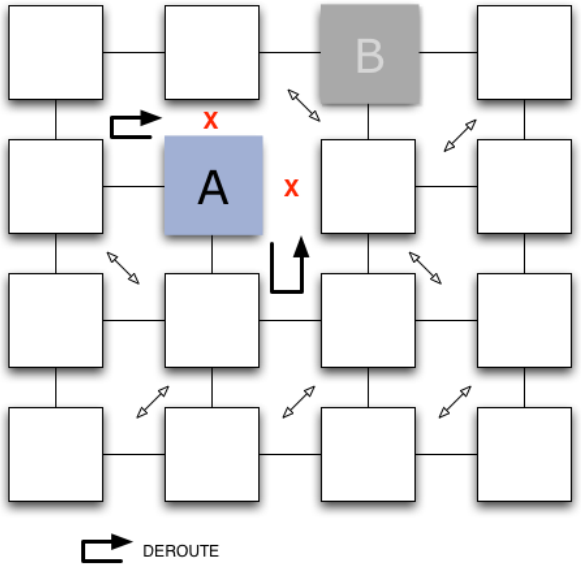



Figure 4.18: Deroutes at router $A$.

We need to provide non-minimal support in an efficient way, that is, with as minimum logic as possible. Figure 4.19 shows the *deroute* logic. In particular, at every input port of the router a deroute option ($dr_0$, $dr_1$ bits) is provided. This set of two bits encodes the deroute option ($N$, $E$, $W$, or $S$). Whenever the $LBDR$ mechanism is unable to provide a valid output port for a message (NOR gate with four inputs) and the message is not at its destination, the deroute option is selected. The logic and the $dr_x$ bits are replicated for every input port. Therefore, the deroute option for a message is the one configured at the input port it is. Alternatively, a single deroute option could be used for the entire router. However, flexibility is reduced (will be evaluated later).

### 4.2.1 Deroute Bits Computation

It is worth mentioning that the deroute option needs to be computed in accordance to the routing algorithm. In fact, the deroute option must not introduce

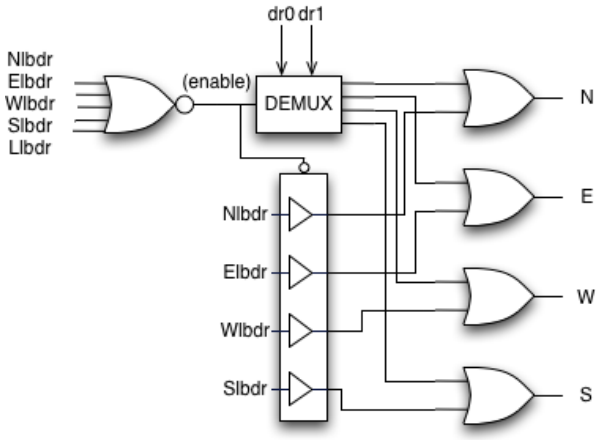



Figure 4.19: Deroute logic.

potential cycles that could lead to deadlocks. In Figure 4.18 the deroute option at input port $W$ at router $B$ can not be set to $S$ since this would lead to messages crossing a routing restriction.

Once the routing and connectivity bits are computed (as described in the previous chapter), the deroute options are searched. To do this, an offline algorithm checks the existence of valid paths for every source-destination pair. As $LBDR$ may allow multiple paths for a given source-destination, the algorithm deeply searches all the paths in a recursive way. A source-destination pair is connected if all the allowed minimal paths reach the destination. If the algorithm fails to support one of these paths, then, a misrouting action is needed. Figure 4.20 shows a case, where at router $B$, for messages going from router $A$ to router $C$, a deroute is needed. In this situation, the algorithm tries all the possible deroute options at router $B$, one per output port but avoiding U turns (so, west port is not considered). Options leading to crossing routing restrictions are also avoided. 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 and the deroute bit is configured. In case of failure (destination is not reached), then another deroute option is tried. Note that several deroute options may be required for a single path. A pseudo-code description of the algorithm is shown in Figure 4.21. The core of

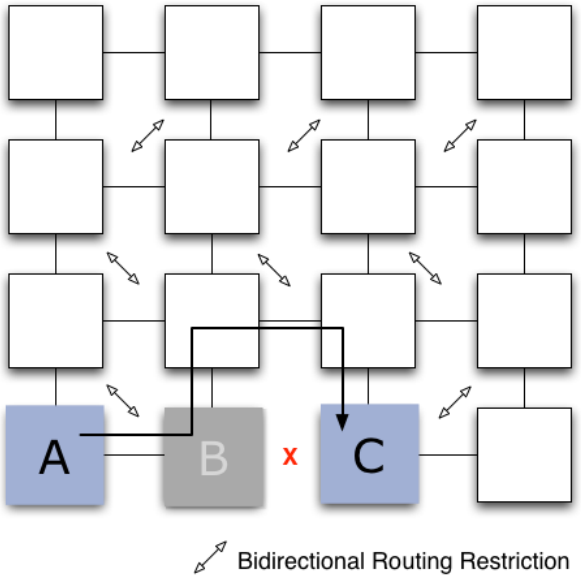



Figure 4.20: Derouting a message at router $B$.

the algorithm is the recursive function *testPath* that tests if there is a reachable path between a router and any other router in the network. In the case there is no direction provided by $LBDR$ (with function *doLBDRrouting*) then the deroutes options are tried, checking valid deroute options with the function *testDeroute*, that checks the direction taken is not a U turn and at the next hop there is no routing restriction that is crossed.

## 4.3 Forks

The *deroute* logic will enhance greatly the percentage of irregular topologies supported. However, there are subtle cases that are still not covered. Figure 4.22 shows an example. The problem in this case comes by the fact that for some destinations located at the same quadrant, at router $B$ the routing engine should provide one port ($N$) for some destinations (destination $C$) and another port ($E$) for other destinations (destination $A$). As the $LBDR$ mechanism works in quadrants for routing, there is no way to indicate the router which port should be granted for a particular message.

To solve the previous problem, the mechanism is enhanced with an addi-

```
Inputs: Routing and connectivity bits of all routers
Outputs: Deroute bits set for each router (if needed)

Procedure:

for each router
  for each other_router
    testPath(router, other_router);
  end for
end for
end procedure

function testPath(current, destination)
variables ok, neighbour, outport

ok := FALSE
outport := doLBDRrouting(current, destination);
if outport != NO outport //LBDR has found a valid outport
then
  if outport == LOCAL PORT //Destination
  then
  ok := TRUE;
  else
   neighbour := getNeighbour(current, outport);
   ok := testPath(neighbour, destination);
  endif
else
if outport == NO outport //LBDR can not provide a valid outport (presence of non-minimal routing)
then
  outport_array := testDeroute(current); //this function returns an array of valid outports for derouting,
                                         //i.e., checking with routing bits and destination quadrant signals
  for each outport in outport_array
    neighbour := getNeighbour(current, outport);
    ok := testPath(neighbour, destination);
    if ok := TRUE;
    then
      setDeroute(current, outport); //we set the deroute bits to the outport that allows misrouting
    endif
  end for
endif

return ok
end function
```

Figure 4.21: Pseudo-code for deroute computation algorithm.

tional and final feature. At router $B$, the message is simply forked through $N$ and $W$ output ports. The *fork* logic is shown in Figure 4.23. As shown, it relies on four additional configuration bits (fork bits) per router: $F_n$, $F_e$, $F_w$, and $F_s$. These bits are set to reflect the output ports that must be used to fork a message. Whenever a message comes and its destination is in the same quadrant defined by the fork bits, then the message needs to be forked.

As can be seen from the logic, the fork operations are performed per quad-

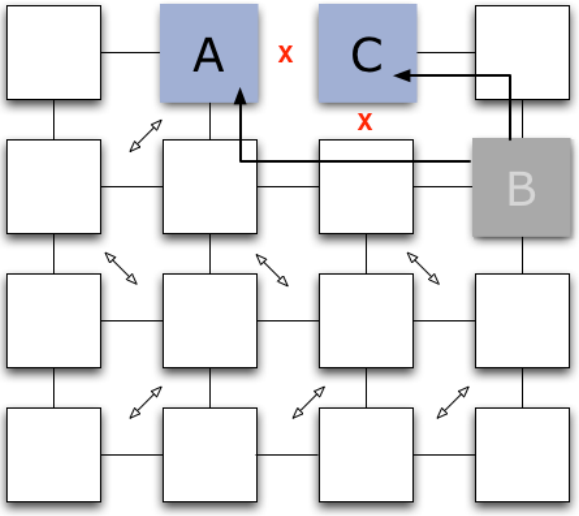



Figure 4.22: Case not handled by the deroute logic.

rants. That is, a message will be forked maximum through two output ports belonging to the same quadrant. Whenever a fork operation is detected (one of the four AND gates is set to one), then the $FORK$ signal is set and forwarded to the arbiter. The output of the fork logic is also made of four control signals ($NF$, $EF$, $WF$, and $SF$ signals) sent also to the arbiter (we will see later that the arbiter needs to be changed to support fork operations).

### 4.3.1 Fork Bits Computation

$F_n$, $F_e$, $F_w$, and $F_s$ bits are set appropriately depending on the topology and the applied routing algorithm. If due to the irregularity in the network considered, at least one pair of end nodes are in the same quadrant but require specific and different output ports, then, the fork operations are considered. To do so, the bit computation algorithm tests any possible fork operation at routers where no deroute succeeded. In Figure 4.24 we can see the addition of fork bits computation to the algorithm in pseudo-code. In the case that, even with deroutes, there is no valid path, then, forks options are searched and tested, with the difference that if any of the output ports provides a valid path, both are set at the router.

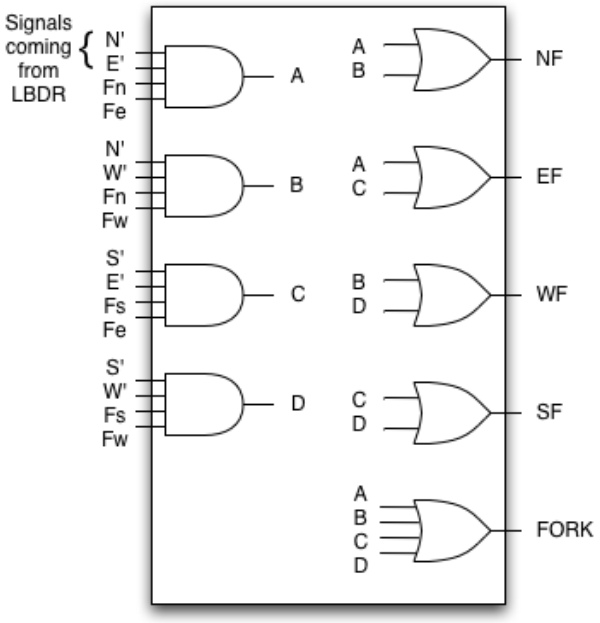



Figure 4.23: Fork logic.

### 4.3.2 Router Implications

The fork operation leads, however, to important changes in the router design. First, the arbiter must allow one message to compete for more than one output port at the same time. Two design alternatives are possible. In the first one, the arbiter may consider a request from a message to two output ports as an indivisible request, therefore, granting or denying access to both outputs at the same time. This leads to a simpler design of the buffering at the input port, since only one read pointer is needed (as in the normal case of a message requesting a single output port). In the second one, however, the arbiter may grant or deny access to one output port regardless of the action performed for the other output port. This leads to a more complex input buffering, since forwarding of the message is shifted for both output ports, thus each requiring a read pointer. We assume the first option because of its simplicity and the fact that fork operations will be required in some rare cases.

The second change at the router (to allow fork operations) is related with deadlock. Indeed, deadlock may occur in wormhole switching as two forked

```
Inputs: Routing and connectivity bits of all routers
Outputs: Deroute and fork bits set for each router (if needed)

Procedure:
for each router
  for each other_router
    testPath(router, other_router);
  end for
end for
end procedure

function testPath(current, destination)
variables ok, neighbour, outport

ok := FALSE
outport := doLBDRrouting(current, destination);
if outport != NO outport //LBDR has found a valid outport
then
  if outport == LOCAL PORT //Destination
  then
  ok := TRUE;
  else
   neighbour := getNeighbour(current, outport);
   ok := testPath(neighbour, destination);
  endif
else
if outport == NO outport //LBDR can not provide a valid outport (presence of non-minimal routing)
then
  outport_array := testDeroute(current); //this function returns an array of valid outports for derouting,
                                         //i.e., checking with routing bits and destination quadrant signals
  for each outport in outport_array
    neighbour := getNeighbour(current, outport);
    ok := testPath(neighbour, destination);
    if ok := TRUE;
    then
      setDeroute(current, outport); //we set the deroute bits to the outport that allows misrouting
    endif
  end for
endif

if ok :=  FALSE; //Deroute and LBDR operations have not succeeded, try again with fork operations
then
   outport_array := testFork(current); //this function gives an array of valid outport with forks, again checking
                                       //with routing bits and destination quadrant signals (N', E', W' or S')
   for each outport in outport_array
    neighbour := getNeighbour(current, outport);
    ok := testPath(neighbour, destination);
  end for
  if ok := TRUE;
  then
     setFork(current, outport_array); //we set fork bits to all outports provided by fork operations
  endif
endif

return ok
end function
```

Figure 4.24: Pseudo-code for combined deroute and forks computation algorithm.

messages may compete for the same set of resources. Although the routing algorithm used is deadlock-free, performing fork actions (like collective communication) may lead to deadlock. Imagine that a message $m_1$ gets access to output port $p_1$ at router $r_1$ and requests access to output port $p_2$ at router $r_2$. However, message $m_2$ gets access to output port $p_2$ at router $r_2$ and requests access to output port $p_1$ at router $r_1$. If messages are long enough they will block the current resources being used while requesting the new ones. Indeed, none of the messages will advance since the input buffers will fill and the output ports will never be released. Figure 4.25 shows an example where two multicast messages collide in the network. Although both of them follow the dimension-order routing algorithm, different branches of the tree induce new dependencies. Indeed, there are new dependencies between channels used and requested by different branches of the trees. If one of the branch blocks, then the other branch will also block (although resources are available).

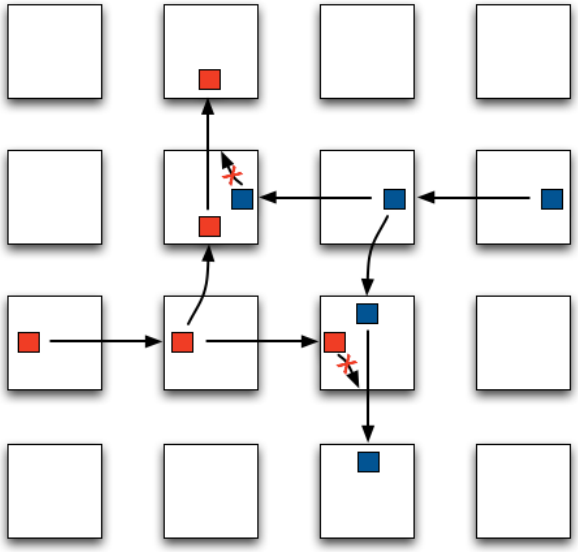


Figure 4.25: Two multicast messages induce a deadlock situation.

There are two solutions to this problem. The easiest way is the use of virtual cut-through (VCT) switching, thus ensuring a packet, coming from the division of a message, will fit always in a buffer. Thus, output ports in the previous example will be released (the packet has been forwarded entirely)

and the requests for the output ports will be granted. Other options rely on performing flit-level circuit switching [52], although wormhole switching is still used among routers. Basically, flits are labelled with identifiers, thus flits from different packets can be mixed in the same buffer. The problem with this kind of solutions is that internal tables are required to keep flit identifiers. We opted for the first solution, thus enforcing VCT switching. Although VCT is seen as demanding much buffer space at routers, a careful design of the router can minimize this effect. Note that fork operations resemble tree-based broadcast operations as a packet is replicated through different output ports. In Section 4.7 area and latency results from two real router implementations are presented, thus showing upfront the real impact of such router changes.

Finally, packets being forked will reach the final destination. However, one of the forked replicas will not reach the destination. In this situation, the packet needs to be removed from the network. This will be easily achieved by silently destroying the packet at a router.

## 4.4 uLBDR

In order to offer a complete unicast solution, $LBDR$ and the extensions, *deroutes* and *forks*, have been gathered in one single mechanism, called $uLBDR$ (Universal Logic-Based Distributed Routing) [48]. $LBDR$ is integrated as the core module of this mechanism, and the foundations have been extended to include the $R_{xx}$ routing bits, to reflect routing restrictions that prevent horizontal or vertical traversal of a router ($EW$, $WE$, $NS$ and $SN$ transitions). Also, the extension of several connectivity layers to support overlapped regions or domains has been added.

The addition of the $R_{xx}$ bits requires a minimal change of the $LBDR$ module as pictured in Figure 4.26. Now, on the first stage, at the $COMPARATOR$ module, eight control signals are generated. Apart from the regular ones ($N'$, $E'$, $W'$ and $S'$), signals $N1$, $E1$, $W1$ and $S1$ are computed. These signals indicate whether the final router is only one hop away in each direction. In a previous example, in Figure 4.17, at router $D$, $N1$ signal would be set if our destination is router $C$.

As shown in Figure 4.26, the $N$ port will be selected ($UN'$ signal activated)

if any of the following four cases is met. First (first AND gate), the destination is on the same column (to the north) and one hop away. Second (second AND gate), the destination is on the same column (to the north) but more than one hop away and the message is allowed to cross the next router from south to north (bit $R_{nn}$ is set). Third (third AND gate), the destination is on the $NE$ quadrant from the router viewpoint and the message can take the south-east turn at the next router (bit $R_{ne}$ is set). And finally (fourth AND gate), the destination is located on the $NW$ quadrant from the router viewpoint and the message is allowed to take the south-west turn at the next router (bit $R_{nw}$ is set). Therefore, the addition in LBDR is the use of $R_{xx}$ bits and the filtering of the case that the message is one hop away in any direction (in that case the port is chosen).

Note also that the connectivity bit filters the final decision, and such filtering is made by taking into account the region identifier the message includes in its header ($R_{id}$ field). The $UL$ signal is set when the message has reached the final router destination, which translates when $N'$, $E'$, $W'$ and $S'$ signals are reset.

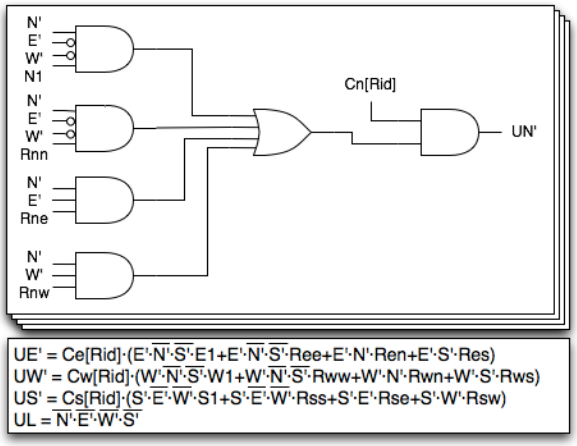



Figure 4.26: $LBDR$ mechanism with support for $R_{xx}$ routing bits, detail for $N$ direction.

An overview of the schematic of the complete mechanism can be seen in Figure 4.27. The inputs of the mechanism are the coordinates decoded from the header, as in the original $LBDR$, and the region identifier, primarily for

selecting the correct connectivity bit layer. The first signals are computed on the *comparator* module, and then are forwarded to the $LBDR$ and *fork* modules. Fork bits are checked in parallel with $LBDR$, just after computing the $N'$, $E'$, $W'$, and $S'$ signals. The outputs of the *fork* logic are combined with the outputs of the $LBDR$ mechanism to produce a possible set of valid signals. After crossing the deroute module, they are exposed to the arbiter. Note that fork operations have priority over deroute operations, as the *deroute* logic is only enabled if none of the signals combined coming from $LBDR$ or *fork* are set. If not enabled, the *deroute* module just acts as regular wiring.

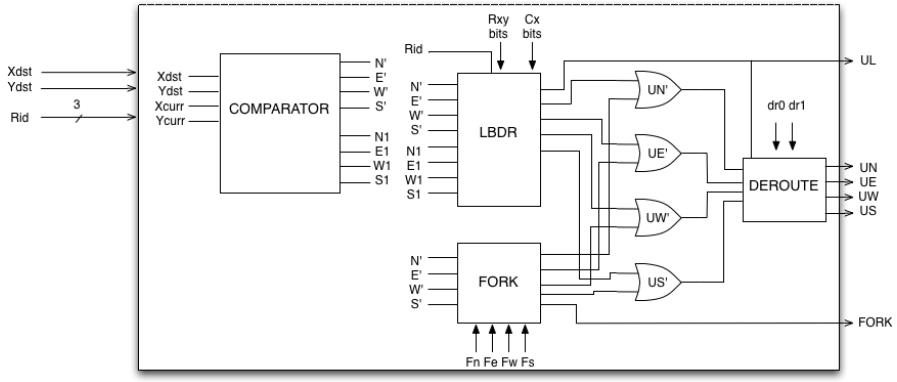



Figure 4.27: $uLBDR$.

### 4.4.1 Configuration Bits Computation

The $uLBDR$ routing implementation, as we will show in the evaluation section, is capable of full coverage (100%) of any topology case, regular or irregular, derived from a 2-dimensional mesh, as long as there is connectivity between each pair of end nodes, so non-minimal path support is guaranteed. Note that in the event that it is not possible to reach all the destinations, then the topology is not covered, but the assumption is related to the routing algorithm instance used. Indeed, with a particular instance of the routing algorithm, $uLBDR$ is able to offer full coverage. However, with another routing instance used, a particular topology may or may not be covered. Different routing instances of $SR$, for example, end up placing routing restrictions in different places and therefore, some instances could end up unusable. Figure 4.28

shows the same topology shown at Figure 4.20 but with a different instance of the $SR$ algorithm. In this case, deroute options can not be placed at router $B$ since would lead to messages crossing a routing restriction. Note that the previous instance of $SR$ (Figure 4.20) allows a proper deroute, thus guaranteeing connectivity. The algorithm for finding configuration bits relies on $SR$ routing. Indeed, it iterates different placement of routing restrictions (computed by $SR$) until one of them allows connectivity through the foundations of $uLBDR$, as seen in Figure 4.29. Indeed, from the pool of different instances computed by function *computeRestrictions*, every instance is checked, if all the paths from any pair of routers is viable then the instance is marked as valid and the topology is considered routable with that instance. Therefore, the success in covering a given topology is due to the $uLBDR$ mechanism and the routing instance used at the same time.

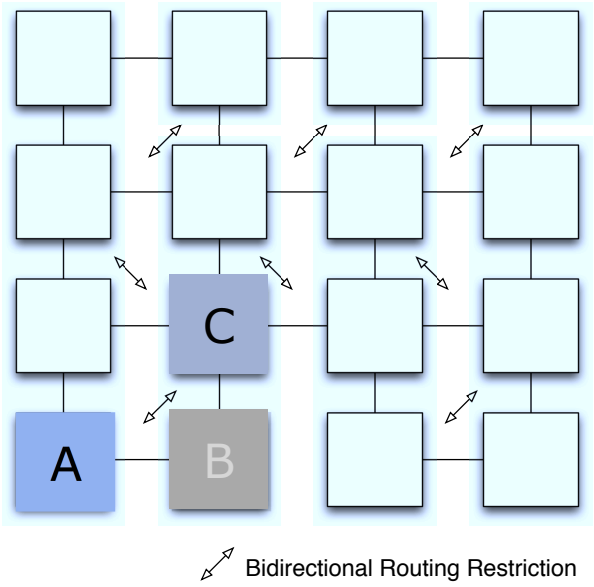



Figure 4.28: Not possible to place a deroute at router $B$.

## 4.5 Demonstration

In this section we discuss the deadlock-freedom and connectivity properties of the different $LBDR$ mechanisms. Before going into details we need, however,

```
Inputs: Topology definition with location of existent links
Outputs: Instance

Procedure:

variables topok, instanceok[max_instances], routerok[max_routers][max_routers]

topok := FALSE;

instance_array := computeRestrictions(SR); //computing several instances of different locations of routing restrictions
for each instance
 instanceok[instance] := FALSE;
 for each router
  computeConnectivityBits(router);
 end for
 for each router
  computeRoutingBits(router);
 end for

 for each router
  for each other_router
     routerok[router][other_router] := FALSE;
     routerok[router][other_router] := testPath(router, other_router); //function checks path with computation of deroutes and forks (if needed)
  end for
 end for
 instanceok[instance] := checkroutersok(routerok[][]) // if all the paths are valid, the instance is valid
end for
topok := checkinstancesok(instanceok[]) // if one of the instances of valid, topology is routable

return instance, topok
end procedure
```

Figure 4.29: Checking if a topology has a instance of the routing algorithm for valid routing.

to differentiate between the routing algorithm and the routing implementation, and to bear in mind that the applied routing algorithm is by itself deadlock-free and provides connectivity over all the end nodes. Indeed, there is no meaning in using the $LBDR$ mechanisms to implement a deadlock-prone routing algorithm or that the routing algorithm leaves some end nodes unconnected (under the assumption that at least all end nodes communicate with other end nodes, so routing paths must be provided, as there are practical cases in which the hypothesis of full connectivity can be relaxed [41]) .

Based on the previous observation, then we can deduce that we need to focus in guaranteeing the $LBDR$ mechanisms are able to keep both properties (deadlock-freedom and connectivity).

The way the routing algorithm is represented is by the use of routing restrictions. The algorithms applied have an acyclic CDG and the routing restrictions are located where channel dependencies do not exist. Thus, a set of routing restriction is the complementary view of a CDG. Indeed, the CDG is acyclic as no packets are allowed to cross routing restrictions.

The routing restrictions are coded in the LBDR implementations with the use of routing bits. Indeed, there is a one-to-one relation between routing

restrictions and routing bits.

From the previous comments we can deduce also that the network is deadlock free if no message crosses a routing restriction. In other words, deadlock freedom is guaranteed by the fact that the routing restrictions imposed by the routing algorithm (and coded in the routing bits) are preserved by the mechanisms.

As can be deduced from the LBDR logic, no message crosses a routing restriction. Indeed, an output port is not taken for a message if the message may potentially cross a routing restriction at the next router. The $R_{xy}$ bits are taken into account for routing as they represent the translation of the routing restrictions from the point of view of each router.

Let us imagine a deadlock occurs in a network using LBDR and a deadlock-free routing algorithm has been applied (with the proper set of routing and connectivity bits). If such case occurs, then a routing restriction necessarily has to be crossed by a message blocked in the deadlock cycle. In that situation, either the routing bits have been wrongly defined (not representing the routing algorithm routing restrictions).

Let us extend this with an example in Figure 4.30. Router 5 receives a message from router 9 that was originated at router 10. Router 9 has a routing restriction that forbids the use of two consecutive channels: $N$ and $E$, in both directions. If the bit computation algorithm has configured correctly router 10, bit $R_{wn}$ should be reset at that router. From router 10 (take Figure 4.30 as a reference), at the first stage of $LBDR$, $N'$ and $W'$ signals are active. If the message would take $W$ direction, at least one of the logic AND gates in the computation of this output port must have its output to 1. $W$ output port is valid, if:

- $W'$ signal is active, $N'$ and $S'$ signals are not active and $W1$ signal is active or routing bit $R_{ww}$ is set. Therefore, this alternative is not applicable because $N'$ signal is active.
- $W'$ and $N'$ signals are active and routing bit $R_{wn}$ is active. Not applicable because routing bit $R_{wn}$ must be reset.
- $W'$ and $S'$ signals are active and routing bit $R_{ws}$ is active. Not applicable because $S'$ is not active.

So, if a message arrives at router 5 from router 9, there is no way that was previously being routed from router 10 through its west link. In other words, the *LBDR* logic guarantees no message will cross a routing restriction, thus keeping the deadlock-freedom property of the applied routing algorithm.

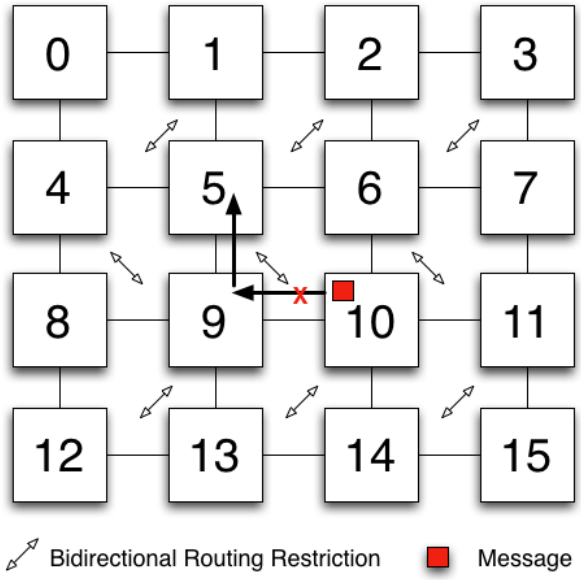



Figure 4.30: Path not suitable.

When using the extensions in the form of *deroutes* and *forks* in *uLBDR*, the computation algorithm that obtains such bits must run correctly as is critical for the success of the mechanism. Once the routing and connectivity bits are computed, the deroute options are searched. To do this, the algorithm checks the existence of valid paths for every source-destination pair. As *uLBDR* may allow multiple paths for a given source-destination, the algorithm deeply searches all the paths in a recursive way. A source-destination pair is connected if all the allowed minimal paths reach the destination. If the algorithm fails to support one of these paths, then, a misrouting action is needed. In case of success, the deroute option is set and the deroute bit is configured. A similar task is done to compute fork operations. If the algorithm fails in placing deroutes, the fork operations are considered. To do so, the algorithm tests any possible fork operation at routers where no deroute

succeeded. Preventing deadlock scenarios when applying deroutes or fork operations is done by an explicit restriction on the computation algorithm. Any non-minimal path that uses deroute or fork options is checked before validating the deroute and fork bits. If it crosses any routing restriction, then the path is discarded and thus, the deroute or fork operation is not set. See an example of pseudocode of this bit computation algorithm in Figure 4.24.

Fork operations, as commented before are not completely deadlock-free if the network relies on wormhole switching. Any operation close to the premises of collective communication is subject to deadlock scenarios with this kind of switching as explained in [13]. As long as this kind of scenario is avoided for routers with $uLBDR$, routing will be deadlock-free.

Connectivity is also guaranteed by $uLBDR$. However, in this case, demonstrating connectivity is hard as many special cases appear when dealing with derouted or forked messages in irregular topologies. Basically, the underlying routing algorithm ($SR$) ensures connectivity in the network. When computing the configuration bits, the algorithm searches for valid paths from every source node to every destination node. As commented above, on fail, the algorithm tries to include deroute/fork operations to keep connectivity. Thus, the same algorithm that computes the configuration bits checks connectivity. Based on this, whenever the algorithm succeeds in computing the bits then we can guarantee that connectivity in the network exists with $LBDR$-based mechanisms. If the algorithm fails, then the topology is not supported (for the tested routing instance).

## 4.6 Real Implementations

To asses the impact of the different mechanisms proposed in this thesis, two router designs have been deployed. In this section we provide a brief description of the routers used for the evaluation: a non-pipelined MPSoC router and a pipelined CMP router. In both cases an initial wormhole router design has been evolved with minimum changes so to allow virtual cut-through switching (required for fork operations). Changes required are: (1) in flow control, (2) in the arbiter logic (support fork operations), and (3) in the crossbar to remove stale copies of forked packets.

### 4.6.1 MPSoC Router Design

Typically, NoC building blocks for use in MPSoCs target lower operating speeds with respect to CMPs and are generally unpipelined [45]. The reference component that we consider to assess the feasibility of $uLBDR$ is an input buffered router implementing wormhole switching (Figure 4.31). Size of the input buffer is tunable and set to 4 flits. In 1 clock cycle, a flit covers the distance between two consecutive input buffers of connected routers through the inter-router link. The switch traversal inside the router is controlled by a modular arbiter (one round-robin arbiter for each output port). A lightweight stall/go flow control policy is implemented. It requires two control wires: one going forward and flagging data availability ("valid") and one going backward and signalling either a condition of buffer filled ("stall") or of buffer free ("go"). This latter signal is indicated as $flow\ control$ in Figure 4.31.

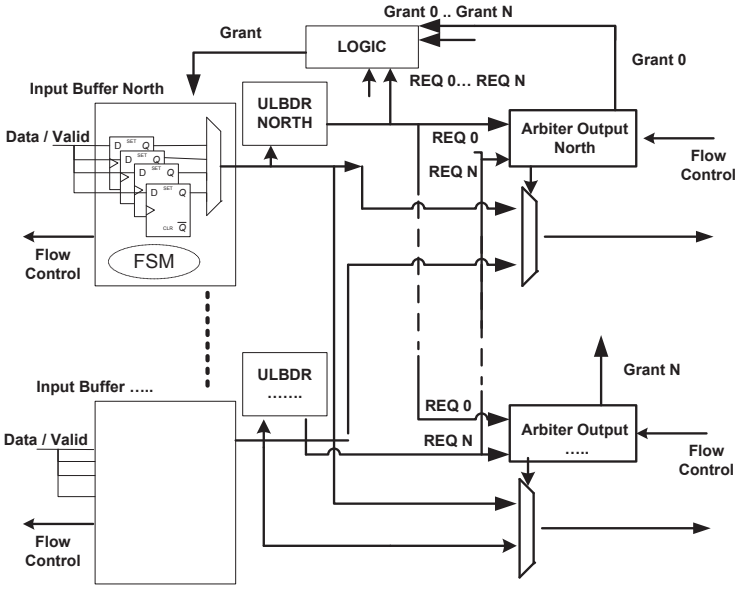



Figure 4.31: MPSoC router schematic.

$LBDR$ and deroutes ($LBDR_{dr}$) mechanisms are implemented in a similar way, from an architecture viewpoint. The head flit contains destination coordinates which are read, after storage in the input buffer, by the routing

logic. The output signals elaborated by the $LBDR/LBDR_{dr}$ module represent *match* signals sent to the arbiters. A *match* signal indicates that the packet from a given input port requires a specific output port. It is interesting to note that the $LBDR/LBDR_{dr}$ routing logic enables to preserve the modular design style of the router architecture (one routing module per input port).

This router was evolved to VCT switching to support $uLBDR$. The signals used and the architecture schematic are the same of Figure 4.31, just the meaning of flow control signals and the arbiter behaviour change. First of all, we had to evolve the basic stall/go flow control protocol to credit-based flow control. In fact, stall/go would have been acceptable only in case all packets were of the same length. If packets exhibit variable length (e.g., reads versus writes, variable number of write/read burst beats, etc.), then the router arbiter needs to know the number of available slots in the downstream buffer before granting a new packet head. Therefore, we now use the *flow control* signals in Figure 4.31 as *credits*. An input buffer asserts a credit high when it has a grant from the arbiter AND it has valid flits to send.

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

In $uLBDR$, packets can be forked through two output ports. When this happens, the $LBDR$ logic asserts two match signals heading to two different port arbiters. When BOTH of them assert their *grant* signals, a unique *grant* is sent to the requesting input buffer, as illustrated in Figure 4.31. One of the packets will reach destination. The other one will reach a router where the $LBDR$ logic will not provide a valid match signal. In that situation, the *grant* signal is set by default to asserted, thus the packet will be forwarded to the crossbar which is not configured for the input port, thus the packet will be filtered. The input buffer is not aware that no arbitration has been

performed for the forked packet, and the *grant* signal is kept asserted, thus will also correctly generate a *credit* to the upstream router, since buffer slots are cleared. This is the way the misrouted forked packet is silently discarded.

### 4.6.2 CMP Router Design

The CMP router is a pipelined input-buffered wormhole router with five stages: input buffer (IB), routing (RT), switch allocator (SW), crossbar (XB) and link traversal (LT). We used a simple router with no virtual channels and five input/output ports. The input buffer size is set to four flits. The RT module has been implemented to support $XY$, $LBDR$, $LBDR_{dr}$, and $uLBDR$ routing implementations and the Stall/Go flow control. Finally, the SW module has been designed with a round-robin arbiter as in [59]. The router has been implemented using the 45nm technology open source Nangate [36].

In order to adapt the basic CMP router to VCT we have performed the following changes. First, buffers at routers have been set to maximum packet size, in our case to four flits. In addition, packetization is performed at the interface nodes when required (notice that this is also needed for the MPSoC router, probably with a different packet size). Message sizes in CMPs (using a coherence protocol) are known beforehand. Usually a short message contains a memory address and a coherence command and a long message also includes the cache line.

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

SW is the most critical stage in our design. Thus, the arbiter modifications applied in the MPSoC arbiter are not affordable for the CMP router. To solve this we have implemented the arbiter shown in Figure 4.32. This arbiter is the same used in the WH design but it adds a new module performed in

parallel. This module arbitrates between fork requests. The grant signals of this module enable (or disable) the non-fork grants. Higher priority is given to fork requests. By doing this, minimum impact on the SW latency is expected. Stale packets (generated by fork operations) are silently discarded in the same way as in the MPSoC router.

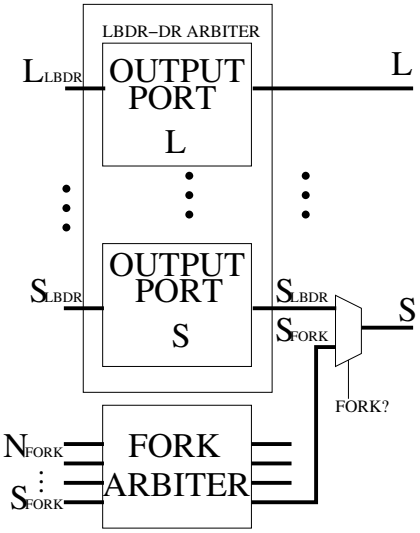



Figure 4.32: New arbiter for the CMP router with fork requests.

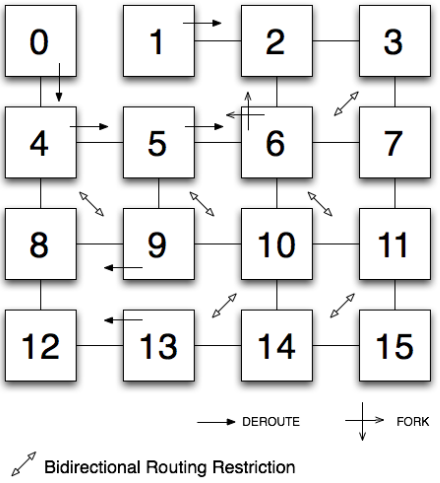
## 4.7 Evaluation

In this section we provide several evaluation results to assess the impact of the different *LBDR* versions. First, we provide a coverage analysis for each solution. Then, we analyse the overhead of the mechanisms in the router designs, including also the overhead incurred by the VCT router counterparts. Finally, we provide performance results by running real applications in a full system simulator environment. Also, performance results when using routing tables are obtained (for comparison purposes).

### 4.7.1 Coverage Analysis

In this section we evaluate the coverage provided by different versions of the mechanism, from the original *LBDR* to the full *uLBDR* mechanism (with

deroutes and forks). Coverage is measured as the percentage of topologies supported from a pool of topologies. A topology is considered supported (covered) if every node in the network reaches all possible destinations.




| ROUTER | INPUT | DEROUTE |
|---|---|---|
| 0 | L | S |
| 1 | L | E |
| 4 | L | E |
| 4 | N | E |
| 4 | S | E |
| 5 | L | E |
| 5 | W | E |
| 5 | S | E |
| 9 | L | W |
| 9 | N | W |
| 13 | L | W |

| ROUTER | INPUT | FORK |
|---|---|---|
| 6 | L | N,W |
| 6 | S | N,W |
| 6 | E | N,W |

Figure 4.33: Example of topology in the coverage analysis with deroutes and forks computed.

A set of topologies derived from the link variability analysis provided in [49] has been used. In particular, different NoC operating frequency thresholds were set and links not reaching those thresholds (due to variability effects) were labelled as faulty. Chips were modelled on a real 65nm implementation NoC layout where all cores are identical, and their size is $1mm^2$. Two different configurations were used, $4 \times 4$ and $8 \times 8$ NoCs with different values of spatial correlation ($\lambda$ 0.4 and $\lambda$ 1.2) and variance ($\sigma$ 0.05 and $\sigma$ 0.18). In total, 1423 topologies have been evaluated. Figure 4.33 shows an example of such topologies.

The evaluation comprehends four different scenarios, $LBDR$, $LBDR$ with 1 global deroute ($LBDR_{1dr}$), $LBDR_{dr}$ as explained in Section 4.2 and $uLBDR$ ($LBDR_{dr+fork}$), the full mechanism. In Figure 4.34 results show how the addition of the two enhancements, deroutes and forks, affects significantly the coverage. Although having one global deroute per router helps to increase coverage up to 50%, further benefits are obtained for deroutes per each input port (5 per router), as coverage further increases to 80%. Finally, the fork

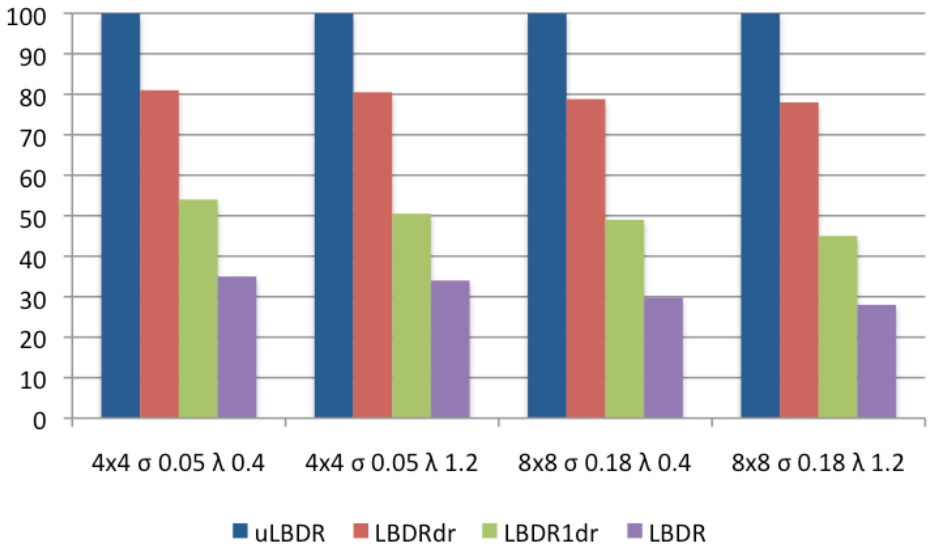



Figure 4.34: Coverage of several routing implementations.

mechanism is the one that guarantees full coverage (all the topologies were successfully supported). Figure 4.35 shows the average percentage of deroutes and forks required per chip. As can be seen, an small set of deroutes is required. As irregularity and network size increases the use of deroutes also increases and the fork mechanism is even less utilized.

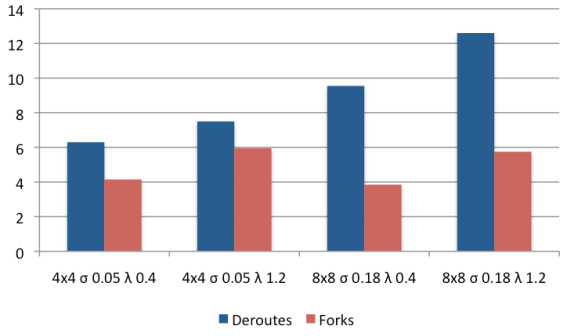



Figure 4.35: Average percentage of deroutes and forks per chip.

For a more exhaustive coverage of *uLBDR* routing implementation, a sec-

ond evaluation was made. Starting from $4 \times 4$ and $8 \times 8$ mesh configurations, several set of irregular instances were produced with 1, 2 and 3 random link failures across the meshes. Every set produced is composed of 2000 (random combinations) irregular instances.

Table 4.1: Random link failure coverage evaluation.

| | 1 failed link | 2 failed links | 3 failed links |
|---|---|---|---|
| $4 \times 4$ | 2000 | 2000 | 2000 |
| $8 \times 8$ | 2000 | 2000 | 2000 |

As seen in in Table 4.1, *uLBDR* is able to offer full coverage (100%) any irregular pattern in the scenarios provided.

From the results provided, it is straightforward that the *uLBDR* mechanism is the one that a designer probably should aim for. However, notice that overheads of the different mechanisms are not plotted in the figure, thus not showing all the picture. The designer must be aware of implementation overheads and of performance impacts. Thus, a tradeoff exists between coverage, overhead, and performance that needs to be considered. In the next section we provide the overhead of such mechanisms in the two router designs used for the exploration of *uLBDR*.

### 4.7.2 MPSoC Router Overhead

The MPSoC router was synthesized with a 65nm STMicroelectronics technology library and Synopsys Physical Compiler. Router versions for each routing mechanism (*LBDR*, $LBDR_{dr}$ and *uLBDR*) were synthesized both for maximum performance and for the same target speed (that of the slowest architecture, i.e., uLBDR). All routers implement the same amount of buffering (4 slots). The choice of a specific routing mechanism affects the maximum achievable speed by each router: 1GHz for LBDR, 950 MHz for $LBDR_{dr}$, and 750 MHz for uLBDR.

Post-synthesis area results for the routers are illustrated in Figure 4.36. By looking at the maximum performance figures, *uLBDR* is about 10% larger than *LBDR*, clearly due to the more complex port arbiters and to their need to handle true credit-based flow control. To make this relatively more complex circuit faster, the synthesis tool tried to speed up the crossbar at the cost

of further increased area. We also observe that $LBDR$ and $LBDR_{dr}$ feature approximately the same maximum area, except for hardly controllable specific optimizations that the synthesis tool applies to the two netlists. The take-away message here is that the logic complexity of these two routing mechanisms is pretty much equivalent.

When the three routers under test were re-synthesized to meet the performance of the slowest one, $uLBDR$, then of course the relaxation of the delay constraint for $LBDR$ and $LBDR_{dr}$ allowed the synthesis tool to infer a more area efficient gate level netlist for them. As a consequence, the area efficiency gap with uLBDR became as large as 44,7%, the absolute worst-case.

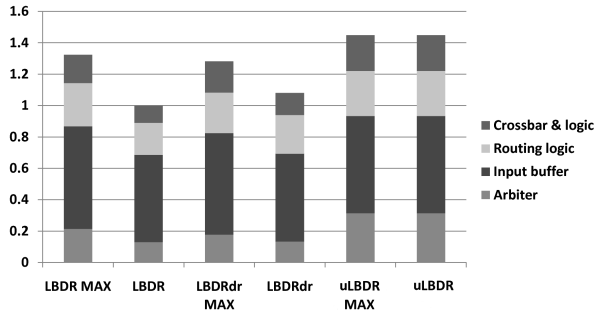



Figure 4.36: MPSoC router area, normalized results.

The conclusion is that whenever the three routing schemes are employed at their maximum performance, the area gap is not significant (around 10%) while tremendously gaining in fault coverage. When the target speed is affordable for each of them and close to that of the slowest scheme, then the choice between the routers becomes a true area-coverage trade-off decision. When the target frequency is very low (a few hundred MHz, not showed in Figure 4.36), then the gate level netlists of the three schemes can be almost equally optimized, resulting in almost the same area while keeping the coverage differences.

Figure 4.37 shows the latency breakdown of the three mechanisms, $LBDR$, $LBDR_{dr}$, and $uLBDR$. As can be seen, $uLBDR$ introduces 25% more delay compared to the basic system in the critical path of the router, while the gap of $LBDR_{dr}$ is around 3%. But the routing stage is not setting the critical path as shown in the figure. The latency for the routing stage for $LBDR_{dr}$

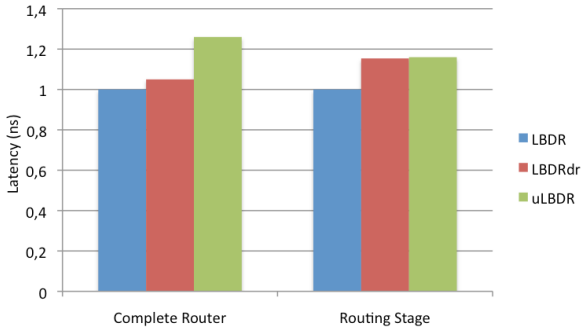



Figure 4.37: MPSoC router latency, normalized results.

and *uLBDR* is almost the same. The latency gap introduced in uLBDR for the total router is due to the changes to support VCT switching and this is reflected in other router elements like the input buffers.

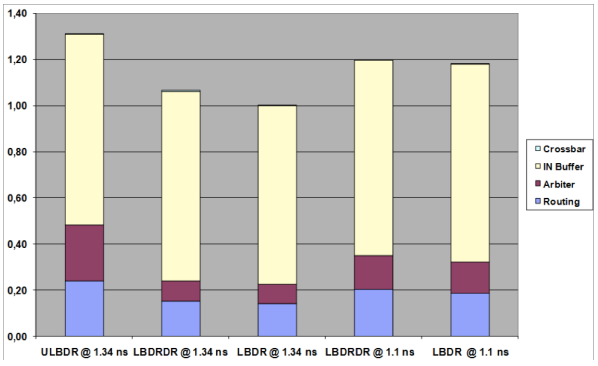



Figure 4.38: MPSoC router power analysis, idle power normalized results.

Power analysis has also been performed for the different routing implementations. Figure 4.38 reports normalized total idle power of the routers with their respective breakdowns. Routers with *LBDR* and $LBDR_{dr}$ have been implemented and characterized twice: in the first variant they are synthesized at their maximum speed (1.1 ns clock period), while in the other variant they are synthesized at the maximum speed achievable by uLBDR, thus resulting into a fair power comparison.

Idle power is dominated in all cases by the buffer contribution, since clock gating has not been applied. Power of the routing block inside all routers

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

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

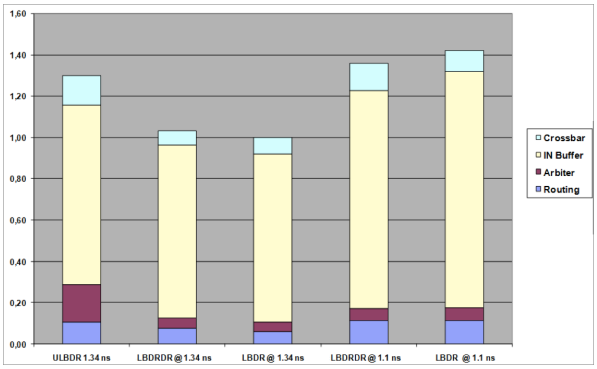



Figure 4.39: MPSoC router power analysis, dynamic power normalized results.

Figure 4.39 reports total router power in active mode, as derived through the average power computing capability of Synopsys PrimeTime PX (50% switching activity). First of all, the crossbar contribution to total power now becomes evident as an effect of the switching activity. In spite of this, the input buffer is still by far the largest contributor. Interestingly, the gap between $LBDR$ and $LBDR_{dr}$ powers at maximum performance and relaxed performance is now larger, due to the increased contribution of the combinational logic. The gap is such that by comparing only power of the routers synthesized at their respective maximum performance, $uLBDR$ turns out to be the least consuming scheme in dynamic power, but not achieving the same maximum performance of $LBDR$ and $LBDR_{dr}$. Again, when aiming at the same target

speed, uLBDR proves almost 30% more power hungry than the other schemes.

### 4.7.3 CMP Router Overhead

Figure 4.40 summarizes frequency and area results of the CMP router for different switching techniques and routing mechanisms. The first thing to highlight is the improvement of both area and frequency of the VCT router. The reason for this improvement is due to the use of buffers with the same size of packets. This has simplified the IB stage because in VCT the flit header of every packet is mapped always into the same buffer slot, thus simplifying read logic. Also, the logic to keep track the number of mapped flits in the buffer has also been simplified. Due to the per-packet flow control only a control signal is required. Although such simplifications can also be made in WH, bubbles would be introduced (known as atomic buffer allocation).

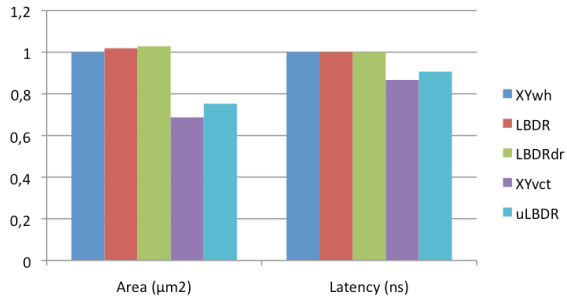



Figure 4.40: CMP complete router, normalized results.

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

Figure 4.41 shows the area overhead and frequencies of the different routing modules, thus not considering the entire router. Now, there are significative differences between the $XY$ module for WH and for VCT. These differences are due to the different flow control mechanism used in both versions. Also,

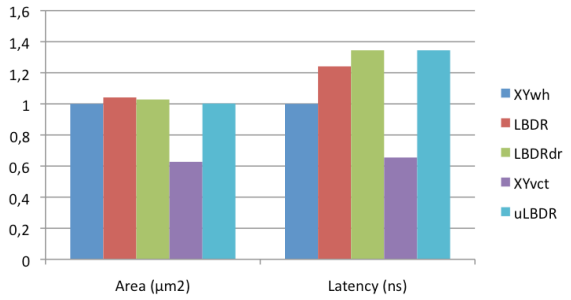



Figure 4.41: CMP RT stage, normalized results.

the different input buffer design affects the routing module. On the other hand, LBDR and $LBDR_{dr}$ mechanisms have an small impact on area but a large one in frequency. Also, the complexity of $uLBDR$ has a large impact on both area and frequency. However, remember that this module is not the one setting the router frequency.

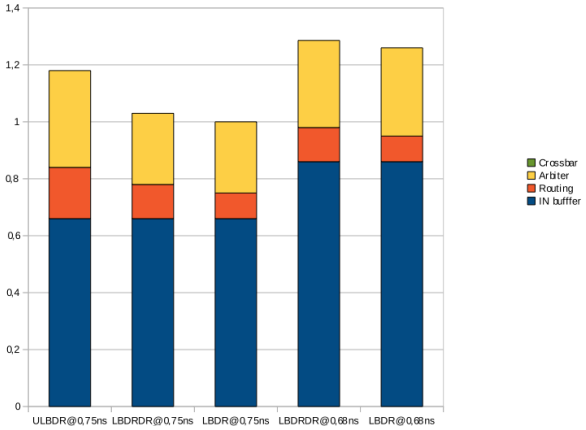



Figure 4.42: CMP router power analysis, idle power normalized results.

Power analysis for the CMP router architectures has also been performed: Wormhole routers with $LBDR$ and $LBDR_{dr}$ routing mechanisms and the VCT router with uLBDR routing mechanism. Figure 4.42 reports normalized total idle power of the routers. As same as the MPSoC counterparts, routers with $LBDR$ and $LBDR_{dr}$ have been implemented classified in two variants:

in the first one they are synthesized at their common maximum speed (in this case, 0.68 ns clock period), while in the second one they are synthesized at the maximum speed achievable by *uLBDR* (for CMP version, 0.75 ns clock period), for a fair power comparison in the results.

Idle power is dominated in all cases by the buffer contribution, since clock gating has not been applied. Power of the routing block inside all routers is relevant, but as it can be seen in Figure 4.42 the routing module presents the lowest idle power consumption. As mentioned before, the main element that causes the power consumption in the routing module is the registers that store the values of LBDR configuration bits (routing, connectivity, deroute and fork bits). Identically to the MPSoC router, $LBDR_{dr}$ and $LBDR$ have an almost equivalent power consumption, due to the very similar router structure with just minor relative modifications, which is consistently lower than that for *uLBDR*. Note that, when using *uLBDR* the routing module and the arbiter increase their power consumption. This increment is almost 20%.

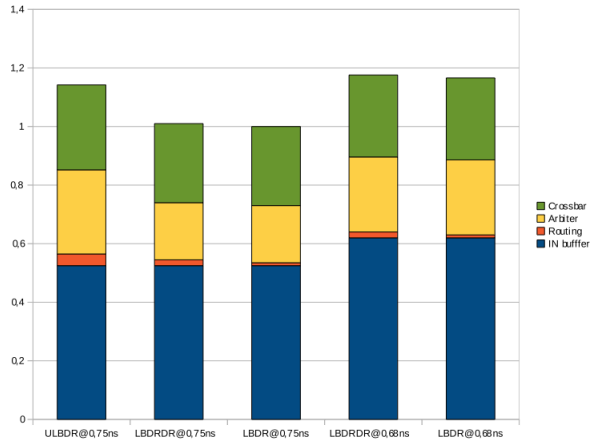



Figure 4.43: CMP router power analysis, dynamic power normalized results.

Figure 4.43 reports total router power in active mode, performed on the same scenario that was set for the MPSoC variants. Note that, router power presents similar results than idle power. However, the gap between *uLBDR* and $LBDR_{dr}$ with respect to $LBDR$ is reduced.

From this evaluation, now the designer will take a much broader view of the real implications of the different mechanisms. Indeed, not necessarily the

designer will opt for the most efficient mechanism that provides full coverage, as the area and frequency implications may lead to unacceptable lower performance. A possible good tradeoff is the choice of the LBDR mechanism with deroute bits which does not require VCT switching, thus requiring much lower area and frequency overheads, while still achieving high coverage results (80%). In the following section we further evaluate the different mechanisms but using the network as a whole, thus to identify the overall network performance effects of the different mechanisms.

### 4.7.4 Area and Latency Overhead versus Memory-macro Implementations

When synthetizing, usually, industrial tools and libraries perform a synthesis of a routing tables model in the form of a set of registers (or flip-flops). As routing tables are implemented on current designs with memory modules, the next step on the analysis was to find a suitable memory model. Memaker [6], a tool from Faraday Technology Corporation, is a compiler that produces memory macro models on UMC Logic LL-RVT (LowK) Process technology. At first, the objective was to compare both routing modules inside the router model in a 65nm technology solution, but Memaker produces synchronous memory macros and the model needed an asynchronous memory macro solution. Instead of changing all the router model, we decided to compare only the modules. The *LBDR* module was changed to be synchronous adding a previous step to the input. A new set of flip-flops (or registers) that feed the input of the module with a clock signal. To compare the models in a fair evaluation, the models were synthetized in a 90nm technology scenario.

For the evaluation, this scenario was assumed: routers are placed in a 2D mesh, with an IP (core, memory) attached to the router. Routers have a radix of 8 ports. The total of destinations it is the result of the mesh size, i.e for a $4 \times 4$ mesh there are 16 destinations. Every input port has a memory module attached with word size equivalent to the amount of destinations. Every word is composed of 3 bits, matching the radix of the router, so an output port is selected (from 0 to 7) given a destination.

The minimum size in words that Memaker, at 90nm technology, can produce is 256 words. But Memaker, also can produce for a size less than 256

words, a single-port register file macro. The results that the tool gives us for the register macro on area and delay are equally analyzed in Figure 4.44 if results in Figures 4.45 and 4.46 are extrapolated for a size in words less than 256. The minimum showed here is for 16 words, i.e, for a network with 16 destinations.

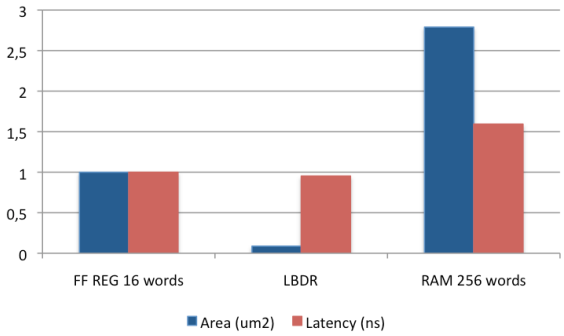



Figure 4.44: Area and data access time analysis, normalized results.

As is shown in Figure 4.44, *LBDR* implementation is about 91% smaller than the single-port register file macro, with a similar latency. This gap is even greater when increasing the number of destinations as seen in Figures 4.45 and 4.46. Remember, that any *LBDR*-based routing implementation only scales with the router radix (i.e. the number of input and output ports), not with system size.

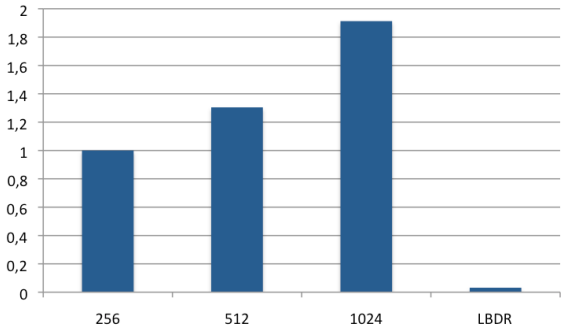



Figure 4.45: Area for different system sizes, normalized results.

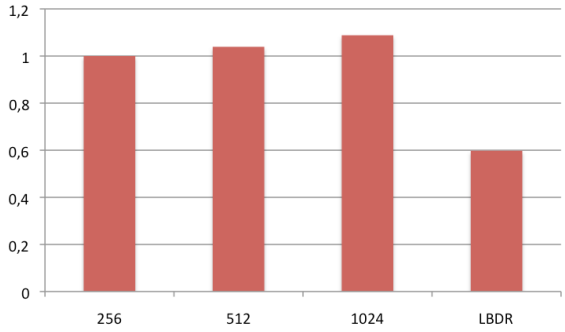



Figure 4.46: Latency for different system sizes, normalized results.

### 4.7.5 Performance Analysis

This section starts with the performance evaluation of several mechanisms: uLBDR, distributed routing tables assuming a two-cycle time delay router (resembling the MPSoC router with the addition of tables, meaning that the cycle time used by uLBDR router here is doubled), and distributed routing tables assuming a two-cycle time routing stage (resembling the CMP router with a larger RT stage with the addition of tables in which only the routing stage cycle time is doubled compared to the one in the uLBDR routing stage).

We have developed, for such purpose, and in-house cycle accurate flit-level network simulator. The simulator models the network cycle-by-cycle and is based on events, where each event represents a different action within the network (a flit arrived, a message header is routed, ...). Each workload simulates a maximum of 80k of transient and 160k of permanent messages, respectively. Flit size is set to 3 bytes. There are two messages sizes, short messages composed of 5 flits and long messages with 27 flits. The percentage of shot messages is set to 70% of the total of messages produced. Buffer size is set to 5 flits, and as virtual cut-through is mandatory for packet switching protocol, packetization is performed (with packet size equal to buffer size). There are two metrics: *throughput* is defined as the average rate of successful flit delivery (flits/cycle); and *latency* (cycles) is defined as the average end-to-end time for a flit to arrive to its destination.

The tests were made on a $4 \times 4$ mesh irregular topology to force the use

of deroutes and forks (thus, using full potential of uLBDR) under three traffic load types: uniform, bit-reversal and bit-complement. Uniform traffic workload returns for a given source a random destination. Bit-reversal traffic workload returns for a given source, the identifier of the source but reversing the bits (if source is id 100, destination is id 001). In bit-complement traffic workload, it is performed a bitwise operation from the source identifier to give a destination (if source is id 100, destination is id 011).

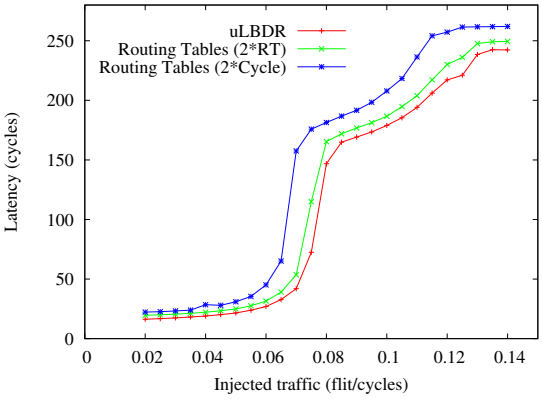



Figure 4.47: Latency of different mechanisms under uniform traffic.

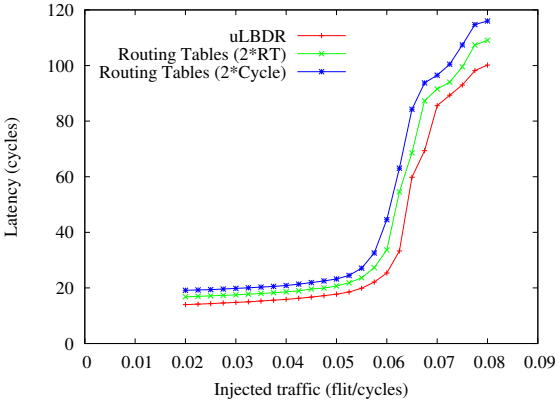



Figure 4.48: Latency of different mechanisms under bit-reversal traffic.

Figures 4.47, 4.48, and 4.49 show the average latency of messages for the different traffic types. For zero-load latency we can see how, obviously, solu-

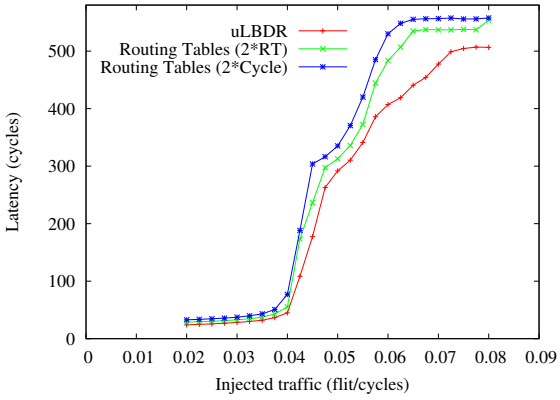



Figure 4.49: Latency of different mechanisms under bit-complement traffic.

tions relying with tables end up with higher numbers. Indeed, latency offset is constant for all the low and mid-traffic range. A solution with routers with 2-cycles delay is the worst one, whereas a pipelined router with 2 cycles in the RT stage approaches *uLBDR* performance. It has to be noted, however, that results are provided in cycles instead of being provided in time. Probably a table-based solution will end up in a router with long critical paths, thus, operating at a much lower frequency. Thus, differences will be higher. For the throughput we can observe also an increase when using *uLBDR* (Figures 4.50, 4.51, and 4.52).

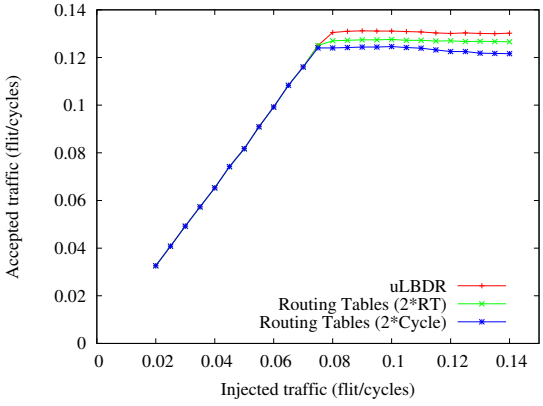



Figure 4.50: Throughput of different mechanisms under uniform traffic.

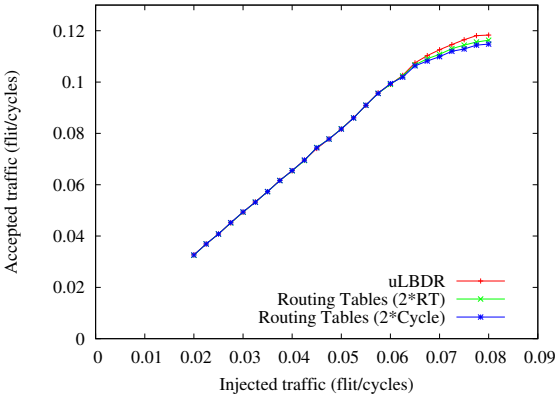



Figure 4.51: Throughput of different mechanisms under bit-reversal traffic.

We have run also several analysis for performance using the GEMS/SIMICS platform [33] upgraded with the event driven cycle-accurate network simulator mentioned before. Several SPLASH-2 [72] applications (Barnes, FFT, LU, Radix) and Apache application have been run in the platform.

- Barnes: This application simulates the interaction of a system of bodies (galaxies or particles, for example) in three dimensions over a number of time-steps, using the Barnes-Hut hierarchical N-body method.
- FFT: The FFT kernel is a complex 1-D version of the radix-$\sqrt{n}$ Six-step FFT algorithm, which is optimized to minimize interprocessor communication.
- LU: This application factors a dense matrix into the product of a lower triangular and an upper triangular matrix.
- Radix: This application performs an iterative algorithm for integer radix sort.
- Apache: This application simulates a traffic workload on the web server Apache.

Chip environment was configured as a 16-tile CMP system in a $4 \times 4$ mesh NoC, each tile including a core processor, a private 2 MB L1 cache,

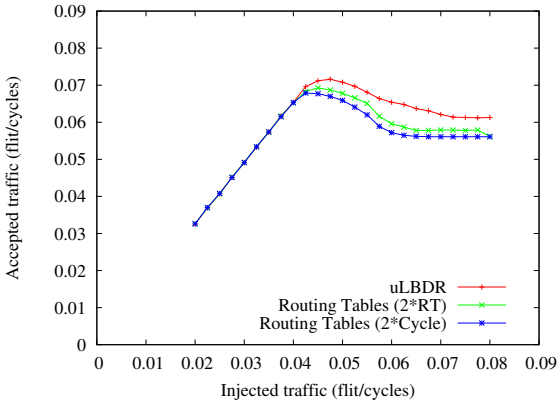



Figure 4.52: Throughput of different mechanisms under bit-complement traffic.

a 4 MB L2 cache bank, a memory controller and the router. Flit size has been set to 4 bytes and virtual cut-through switching is assumed. For this chip configuration, two cache coherency protocols have been used to keep coherency between private L1 caches and a shared (but distributed) L2 cache: directory-based and token-based.

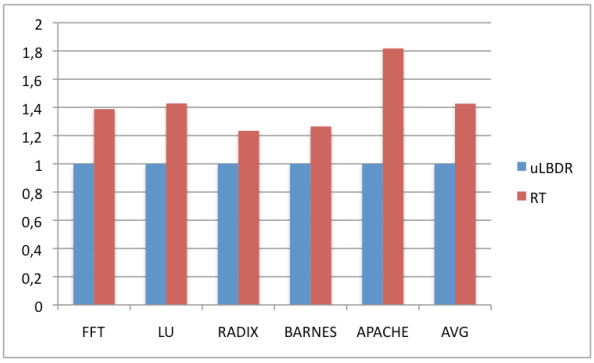



Figure 4.53: 2-cycle delay router (RT) vs one-cycle delay router (uLBDR), normalized execution time of applications, directory-based protocol.

In Figures 4.53, 4.54 and 4.55 the results for performance evaluation in a directory-based protocol are shown. Figure 4.53 shows the normalized execution time when using a router with distributed routing tables (two cycles

delay router) and a router using *uLBDR* mechanism (one cycle delay router). Virtual-cut-through switching is assumed and flit size is set to 3 bytes (performing packetization and padding when needed). As can be seen, a slow router (routing tables) affects greatly the execution time of applications, in some cases, like Apache, almost doubling it. In all applications the slower router behaves worse. Therefore, the designer needs to avoid the large latency of routing tables.

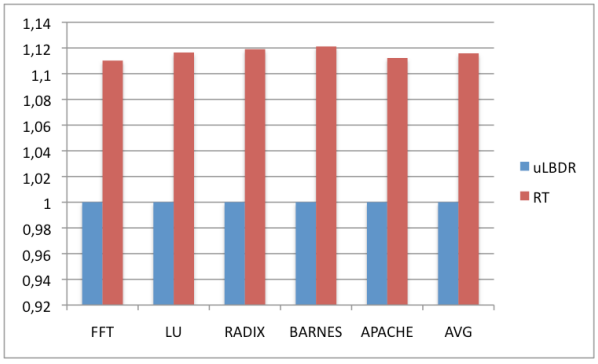



Figure 4.54: 2-cycle RT module (RT) vs one-cycle RT module (uLBDR), normalized execution time of applications, directory-based protocol.

Figure 4.54 shows the case when two routers are compared, one with the RT stage experiencing two cycles (only for header flits) and one with the RT stage having a single cycle. Note that in this situation the average overhead of routing tables is smaller, around 11%.

Figure 4.55 shows the performance achieved in the $4 \times 4$ topology, but with some links missing, so deroutes and forks are necessary. In this irregular scenario, applications could be run as the routing mechanism is able to support all the paths (coverage of the topology). *uLBDR* mechanism still gets better results, in average, a gap of 9% when compared with the router design with a 2-cycle table implementation in the RT stage. The interesting point, here, is the success in running the application in a failed mesh network without using routing tables.

Figures 4.56, 4.57 and 4.58 show the same previous evaluation, but this time using the token protocol for cache coherency issues. This time differences are not so significant, specially for the case of a pipelined router with 2-cycle

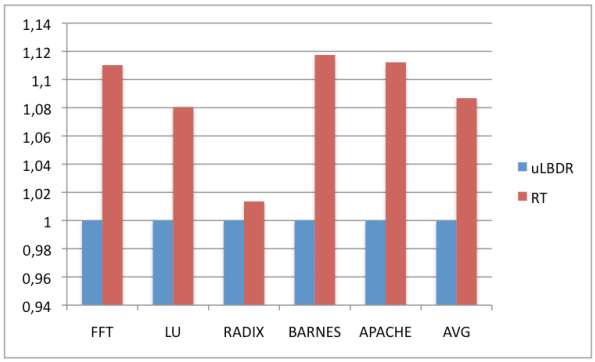



Figure 4.55: Irregular topology 2-cycle RT module (RT) vs one-cycle RT module (uLBDR), normalized execution time of applications, directory-based protocol .

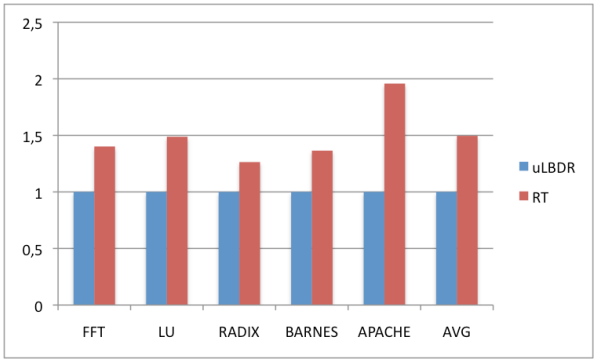



Figure 4.56: 2-cycle delay router (RT) vs one-cycle delay router (uLBDR), normalized execution time of applications, token-based protocol.

table implementation. The issue in this protocol is that most of the traffic is broadcast and the current *uLBDR* mechanism is focused on unicast communication. Later, when adding support for collective communication at the network level we will see much larger differences.

However, an important thing to consider here is the fact that *uLBDR* is not focused in achieving performance improvements over table-based approaches. The mechanisms proposed must be seen as cost-effective alternatives to the routing tables. Having similar (or slightly better) performance numbers must be seen as an additional feature.

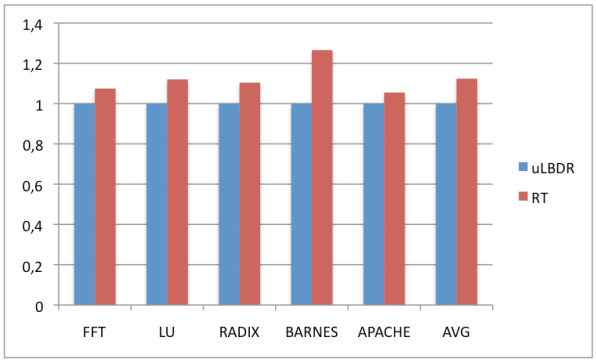



Figure 4.57: 2-cycle RT module (RT) vs one-cycle RT module (uLBDR), normalized execution time of applications, token-based protocol.

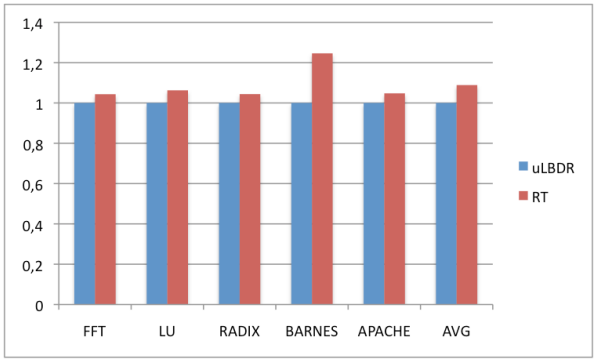



Figure 4.58: Irregular topology 2-cycle RT module (RT) vs one-cycle RT module (uLBDR), normalized execution time of applications, token-based protocol.

## 4.8 Conclusions

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

version requires a VCT router design and its impact on router frequency is 30% on an MPSoC router and no impact on a CMP pipelined router design.

# Chapter 5

# Broadcast/Multicast Communication and the eLBDR Architecture

*"It is your mind that creates this world."*

Gautama Buddha.

As the complexity of multicore architectures grows, the need for collective communication support is a desirable feature to handle efficiently scattered operations across the chip. Collective communication primitives are required either for implementing barrier synchronization or to support coherency traffic. A broadcast/multicast mechanism opens the door for cache coherence protocols or virtualization techniques. Collective communication in CMPs may also be needed for an effective management of the NoC (control/management messages). Although there are solutions for collective communication in NoCs (see Section 2.3) either they do not support irregular topologies or their implementation is costly (table-based).

In this chapter we tackle broadcast/multicast support for many-core architectures like Chip Multiprocessors (CMPs) and Multiprocessor System-on-Chips (MPSoCs). The mechanisms described in this chapter are based on the foundations presented in Chapter 3, thus, relying exclusively on the routing

and connectivity bits. The proposed techniques remove the need for routing tables for collective communication support and offer different levels of coverage of regular and irregular topologies derived from 2-dimensional meshes.

In this chapter we first describe the two mechanisms to provide broadcast/multicast communication support: *bLBDR* and *SBBM*. Later we demonstrate deadlock-freedom and connectivity of such mechanisms. Next, a detailed performance evaluation of the mechanisms is shown, starting with a coverage analysis, and ending with both router analysis and performance impact.

Later, we present a conceptual architecture, eLBDR, that aims to gather both unicast and broadcast/multicast communication. A description of this architecture and the associated evaluation is shown at the end of the chapter.

## 5.1 bLBDR: Broadcast Logic-based Distributed Routing

*bLBDR* (*Broadcast Logic-based Distributed Routing*) is the first mechanism we introduce. It is based on the same foundations of *LBDR*, thus, relying exclusively on the routing and connectivity bits. bLBDR relies on minimal path routing, therefore is designed to cooperate with the basic LBDR mechanism (without non-minimal path support).

In order to provide multicast support, bLBDR uses the concept of connectivity layers, thus using a vector of connectivity bits per output port at each router. In particular, we assume eight connectivity layers. Those layers will be considered to be programmed before normal chip operation. Layers/regions may be overlapped, however. Also, one of the layers will define a global region including all the routers, thus enabling the broadcast communication for the entire chip. *bLBDR* offers a tree-based broadcast operation inside a region defined within the network. This can be viewed as a multicast operation at the chip level. The following broadcast properties are enforced by *bLBDR*:

- Traffic derived from a broadcast action is bounded to the region where it was initiated, even if parts of the region overlap with other regions. Therefore, broadcast operations will not cross the region boundary they

belong to.

- A tree-based broadcast operation can be initiated from any node within the region.
- Traffic originated from a broadcast action will take always minimal paths and routers (and end nodes) will receive only one packet per broadcast action. Thus, traffic is minimized.
- *bLBDR* can be applied to any routing/topology combination where the basic *LBDR* mechanism is used, thus with no deroute or fork additions. Also, any region pattern compatible with *LBDR* can be used.
- The broadcast/multicast mechanism does not require any new information at routers and end nodes. Only a small logic is required to create dynamically the broadcast tree at each router.

For the sake of explanation, broadcast in *bLBDR* is described in two parts. First, when the broadcast is initiated, and second, when a router receives a broadcast action.

### 5.1.1 A Broadcast is Initiated

When an end node initiates a broadcast a message is created. A control signal ($B_h$) is used to differentiate broadcast messages from unicast ones. Additionally, four control signals ($B_n$, $B_e$, $B_w$, and $B_s$) are used. These signals indicate the router the directions the broadcast message must take at the given router. These signals travel together with the broadcast message through control lines. As the first router has to inject the broadcast message through all possible directions (regardless of the connectivity), the end node sets all the signals to one. Figure 5.1 shows the additional control lines required on every network channel.

It is left to the choice of the designer to include these signals as part of the message header or defined as control lines over the channel between two routers. As long as the signals are decoded and provided to the broadcast mechanism correctly, there is no real difference. For the sake of description, we assume they are served as control lines.

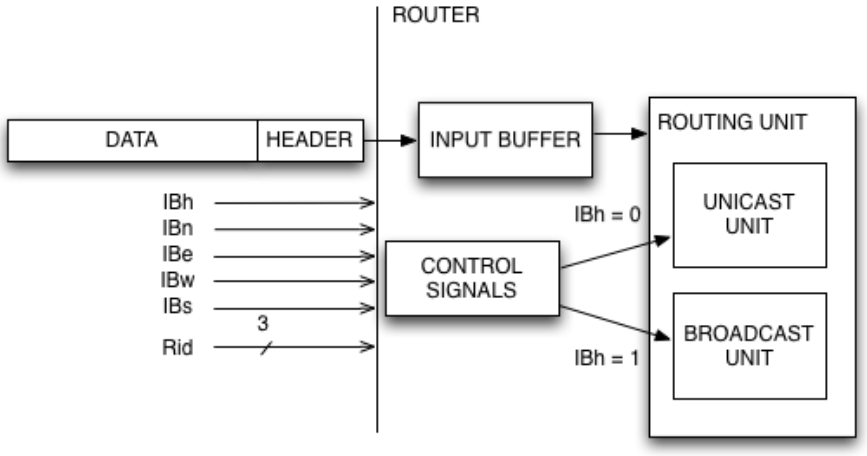



Figure 5.1: Simple schematic of the control lines.

### 5.1.2 A Router Receives a Broadcast Packet

A router may receive a broadcast packet either from its local port (from the end node) or through the incoming ports. Upon reception of a broadcast packet, the router injects through its output ports up to 4 packets ($NB$, $EB$, $WB$, and $SB$), each one being sent through a different output port ($N$, $E$, $W$, $S$) and each one being a broadcast packet with its corresponding set of signals ($B_h$, $B_n$, $B_e$, $B_w$, and $B_s$). These signals are, however, computed by the router in a different manner. For the sake of presentation signals for packet generated for the $N$ port are labelled $NB_h$, $NB_n$, $NB_e$, $NB_w$, and $NB_s$; signals for the packet for the $E$ port are labelled $EB_h$, $EB_n$, $EB_e$, $EB_w$, and $EB_s$; the same for signals for packets sent through $W$ and $S$ ports. Additionally, broadcast signals for the incoming broadcast packet are labelled $IB_h$, $IB_n$, $IB_e$, $IB_w$, and $IB_s$.

The input port stores both the packet and control signals. In particular, 8 control signals are received and stored: three signals encoding the region, and five signals with the broadcast info. After decoding the packet at the buffers and transferred the header data to the routing unit, if $IB_h$ signal is set, the broadcast logic is used.

*bLBDR* uses the connectivity bits and routing bits to compute the set of admissible output ports. The mechanism computes all the output ports

that must be used to broadcast the packet. To do this, the routing module computes all the broadcast signals (20 signals, five signals per output port). Figure 5.2 shows the logic required for computing each signal for each packet. The logic is replicated at each input port.

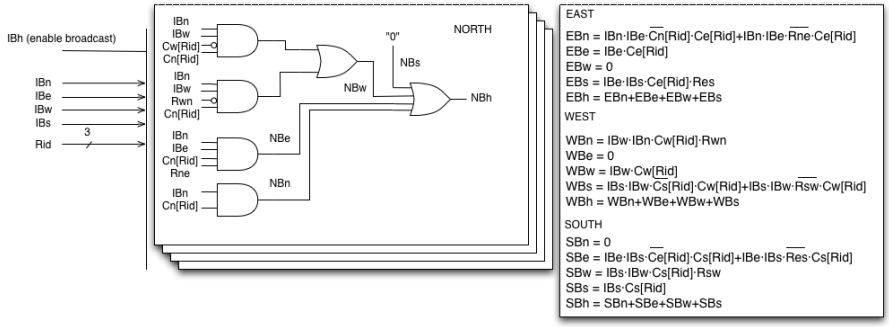



Figure 5.2: *bLBDR* logic.

Let us focus on the computed signals for the $NB$ packet (the control signals used for the broadcast packet that will be sent through the north port; see Figure 5.2):

- The $NB_n$ signal is set if the incoming broadcast packet ($IB$ packet) has its $IB_n$ signal set, that is, broadcast must go north and there is a connected router (belonging to the same region) through the north port ($C_n$ bit is set for region $R_{id}$).

- The $NB_e$ signal is set if the incoming packet must be broadcasted through the $NE$ sector (signals $IB_n$ and $IB_e$ are set), there is a connected router through the north port ($C_n$ bit is set) and there is no routing restriction at the next router that forbids turning towards $E$ (bit $R_{ne}$ is set). Notice that if the $NB_e$ signal is set, at the same router the $EB_n$ signal should not be set (in order not to duplicate broadcast packets). Notice that logic for $EB_n$ is only set if there is no connectivity through the north port (bit $C_n$ is reset) or routing towards $E$ is not allowed through the north port (bit $R_{ne}$ is reset). Therefore, both signals $NB_e$ and $EB_n$ will not be set at the same time for the same broadcast packet.

- The $NB_w$ signal is set if the incoming packet must be broadcasted through the $NW$ sector (signals $IB_n$ and $IB_w$ are set), there is a connected router through the north port ($C_n$ bit is set), and either there is no router through the $W$ port (bit $C_w$ is reset) or there is a routing restriction that forbids turning north at the router reached through the $W$ output port (bit $R_{wn}$ is reset). Notice that in this case, signals $NB_w$ and $WB_n$ must not be set at the same time.

- The $NB_s$ signal is reset since the packet will be sent through the $N$ port.

- The $NB_h$ signal is computed by ORing all the previous signals. If $NB_h$ is reset, then no broadcast packet is issued through the $N$ port.

Figure 5.2 shows the equations for signals intended for other ports. As can be seen, similar equations are obtained for the remaining output ports of the router.

Let us describe how the bLBDR mechanism faces different situations and irregularities in the topology. Figure 5.3 shows different situations we might face when broadcasting through the $NE$ sector at a given router (marked with an $X$). In case $A$, the router forwards two broadcast packets through the $N$ and $E$ ports. In the $NB$ packet the $NB_n$ and $NB_e$ signals are set, and in the $EB$ packet just the $EB_e$ signal is set. Notice that, in this situation, the $E$ output port is used only to broadcast through the row of routers, whereas at the next router through the $N$ port the new $NE$ sector is broadcasted. In this case, notice that $EB_n$ has been reset as there is a routing restriction through the $E$ port (bit $R_{en}$ is reset at router $X$). Contrary to this, in case $B$ the $NB_e$ signal is reset since there is a routing restriction through the $N$ port (bit $R_{ne}$ is reset). In this situation, the $N$ port is used only to broadcast the column of routers and the resulting new $NE$ sector is broadcasted through the $E$ output port.

An interesting case is $C$. In this situation notice that the router has no routing restrictions through $N$ and $E$ ports. However, in order to avoid duplicates, the $N$ output port is given priority. In this case, $N$ port is used to broadcast the resulting new $NE$ sector and $E$ port is only used to broadcast through the row of routers. $bLBDR$ provides priority to all the output ports depending on the sector that is being broadcasted. $N$ port has higher priority

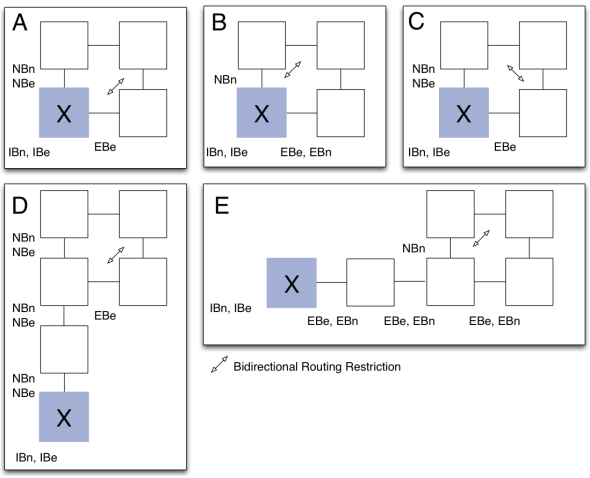



Figure 5.3: Different cases for the $NE$ sector.

than $E$ port (for the $NE$ sector), $E$ port has higher priority than $S$ port (for the $SE$ sector), $S$ port has higher priority than $W$ port (for the $SW$ sector), and $W$ port has higher priority than $N$ port (for the $NW$ sector).

In cases $D$ and $E$, boundaries of the region are handled by $bLBDR$. In particular, in case $D$, the $NB_n$ and $NB_e$ signals are set as there is no routing restriction for the $N$ port (bit $R_{ne}$ is set). Notice that at the next router the same happens and thus, the broadcast packet with its $B_n$ and $B_e$ signals set makes forward progress, until it reaches the router with three neighbouring routers. At that router, case $A$ is applied. In case $E$, the $EB_e$ and $EB_n$ signals are set. In particular $EB_n$ signal is set because there is no connectivity through the $N$ port (bit $C_n$ is reset). The broadcast packet will, thus, make forward progress until it reaches the router with three neighbours. At that router case $B$ is applied. The same deductions can be obtained to the remaining broadcast signals that are computed and the remaining sectors ($SE$, $SW$, and $NW$).

### 5.1.3 Detailed Example

Figure 5.4 shows an example of routing in $bLBDR$. Router 14 starts a broadcast message. At first, the message is replicated through all existent links to the neighbouring routers, but with different signals. For example, the replica that exits to the north to router 10 will have $NB_n$ and $NB_e$ set to one and

the rest to zero, meaning that, router 10, at thenext cycle, will take care of the $NE$ quadrant with its replicas. On the other hand, the message sent to the west to router 13 has $WB_n$ and $WB_w$ set to one (the rest are reset), so router 13 will replicate the message to the $WN$ quadrant at the next cycle. Router 15 will not replicate the message to the north as there is a routing restriction in this router and router 10 will cover router 11 at the next step. Indeed, the message sent to router 15 has only the $EB_e$ signal set to one, in the case there was a router at the east of router 15, but this is filtered by the corresponding connectivity bit. The figure shows the signals computed at routers 10, 6, 5 and 1. In five steps (cycles) the broadcast message has arrived to all destinations. Note that none of the replicas of the broadcast message crosses a routing restriction.

We must conclude this section with one remark. As $bLBDR$ was an evolution based on the $LBDR$ mechanism, it only supports minimal paths between each pair of end nodes. As $LBDR$ evolved to a more complete mechanism like $uLBDR$ to support full coverage, it is mandatory to find a mechanism, with the same assumptions, to offer communication primitives in the presence of irregularities that enforce non-minimal paths. The next mechanism, developed from the base of $bLBDR$, achieves that.

## 5.2 SBBM: Signal Bit Based Multicast

In order to tackle non-minimal path support, we define the SBBM (*Signal Bit Based Multicast*) mechanism. It offers a tree-based broadcast operation at the network chip level or inside a region, in both cases defined by the connectivity bits and supports non-minimal paths. Deroutes and fork bits are not used by the mechanism, as it relies exclusively on the concept of connectivity. However, is compatible with the uLBDR unicast mechanism.

As with bLBDR, the broadcast operation in SBBM can be initiated by any of the nodes contained within the region, and the broadcast packet reaches only the end nodes defined within the region. Only one packet is received by every destination of the collective communication, thus avoiding any possible duplicate. SBBM requires a minimum computation logic at each router. In particular, the following properties are attributable to SBBM:

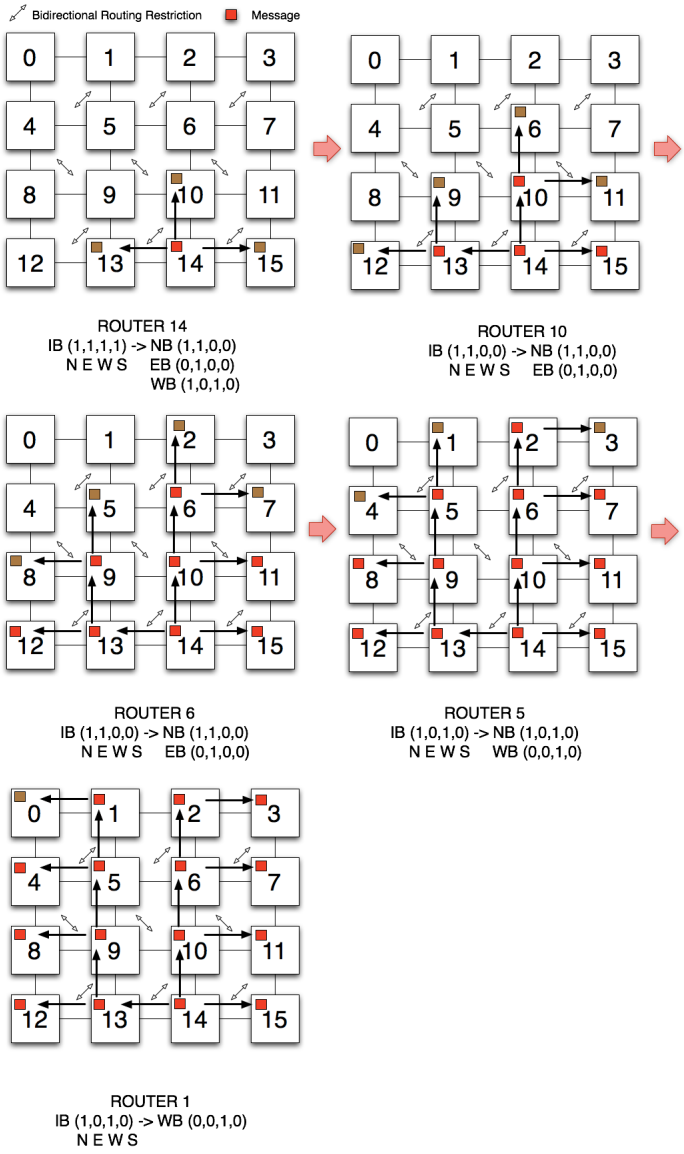



Figure 5.4: Example of bLBDR.

- Any irregular topology or region/domain derived from a 2D mesh NoC is supported. Irregular topologies with gaps or blocks inside the mesh are, thus, supported. This opens the possibility to use collective communication in heterogeneous MPSoC systems (with IP cores of different sizes). In addition, regular topologies are also supported by SBBM.

- The definition of overlapped regions in bLBDR is also supported by SBBM, but contrary to bLBDR now regions with completely irregular shapes can be defined and supported.

- SBBM works with any topology-agnostic[1] routing algorithm that enforces full connectivity and is deadlock-free.

- Logic in SBBM is much simpler than in bLBDR allowing for further savings with a more compact and efficient mechanism. In fact, the area (as we will see) is reduced by 34% when both mechanisms are compared. Compared to table-based mechanisms the gap is even greater for area and latency. These savings are accounted for the implementation of the routing engine within the router in the NoC. Although this delay will not drive the main latency of the message within the network, it will enable the implementation of fast and compact NoC routers. Also, virtual channels are not required as other mechanisms do, so getting more savings.

- SBBM uses the full set of routing bits of the foundations like the $uLBDR$ mechanism (including the $R_{nn}$, $R_{ee}$, $R_{ww}$ and $R_{ss}$ bits).

On the other hand, SBBM shares the same base signals and expands the configuration bits used in bLBDR, thus, ensuring a perfect downward compatibility. For the sake of explanation, the SBBM routing module is described in several parts. First we describe how the tree for broadcasting packets is formed. Second, we explain how the mechanism works. Finally, the required logic is shown.

One of the down sides of SBBM mechanism (as compared to bLBDR) is that it requires a complete connectivity layer to define the broadcast operation, thus leaving less room for effective region definitions. This requirement comes with a large (as we will see) increment in coverage and effectiveness, thus is not a major problem. Indeed, if we rely in eight connectivity layers, with the support of SBBM instead of bLBDR, we downgrade the effective number of connectivity layers down to four. Obviously, mixed alternatives can be thought, for instance, having both mechanisms and using only one connectivity

[1] A routing algorithm able to be computed on any topology.

layer for SBBM for the entire chip. However, such improvements are left out of the current work.

### 5.2.1 Broadcast Tree

At the same time the routing and connectivity bits are computed, an acyclic spanning tree for broadcast operations is computed. The tree structure is stored in the connectivity bits. We assume 8 connectivity bits per output port where now four of those layers/bits are used for collective operations.

Let us assume we define an entire region covering all the chip and we need support for broadcast operations. At every router, $C_x[0]$ (connectivity layer 0) is used for unicast actions and $C_x[1]$ (connectivity layer 1) is used for broadcast actions. Initially, $C_x[0]$ bits are computed based on the connectivity pattern derived from the final topology. Those $C_x[0]$ bits are then copied to $C_x[1]$ bits.

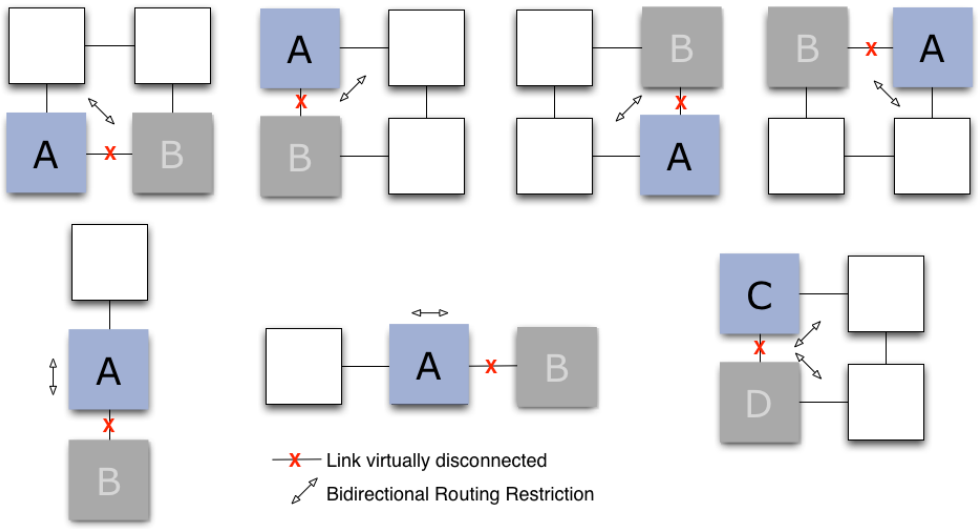
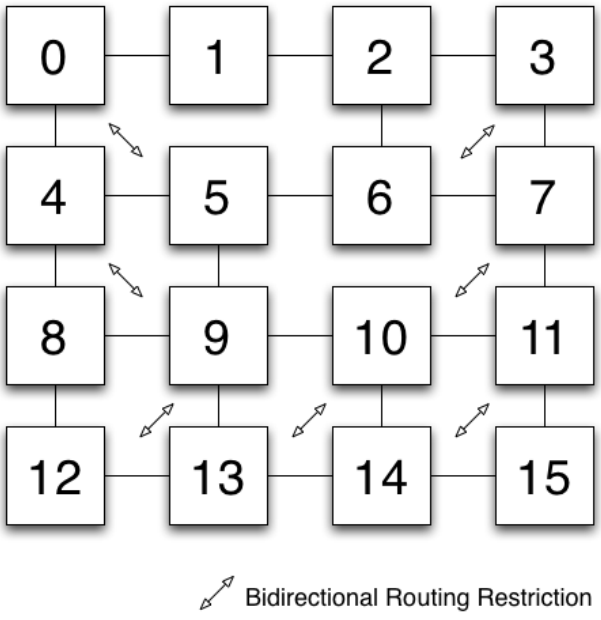



Figure 5.5: Example of how links are virtually disconnected based on the locations of routing restrictions.

Some $C_x[1]$ bits are, then, reset by applying the following rule: If one router has a routing restriction between two links, one of the links is virtually disconnected ($C_x[1]$ bit is reset) in both directions. In Figure 5.5 we can see the six possible cases where a link is virtually disconnected. Although the

figure shows bidirectional restrictions the rule also applies to unidirectional restrictions. As can be seen, if router $A$ has a routing restriction placed between $N$ and $E$ ports, the $E$ port is virtually disconnected between routers $A$ and $B$, meaning that $C_e[1]$ at router $A$ and $C_w[1]$ at router $B$ are reset. If, when processing a routing restriction, one link belongs to two routing restrictions (an example can be seen in Figure 5.5 between routers $C$ and $D$), then the link is virtually disconnected for both routing restrictions, that is, two neighbour routing restrictions end up in only one link being virtually disconnected. Although there are different ways to decide which links to disconnect, we assume the disconnection pattern just described and shown in Figure 5.5.

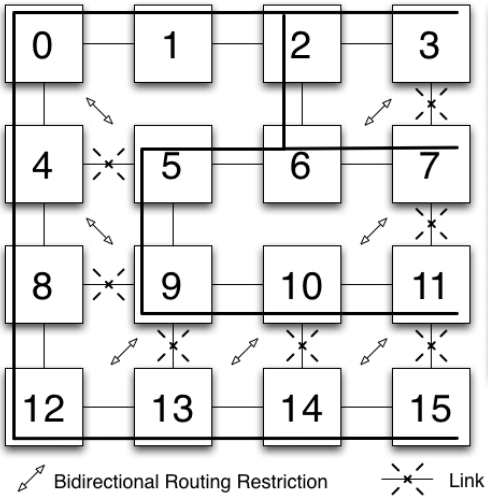



| Router | Cn[0] | Ce[0] | Cw[0] | Cs[0] |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 2 | 0 | 1 | 1 | 1 |
| 3 | 0 | 0 | 1 | 1 |
| 4 | 1 | 1 | 0 | 1 |
| 5 | 0 | 1 | 1 | 1 |
| 6 | 1 | 1 | 1 | 0 |
| 7 | 1 | 0 | 1 | 1 |
| 8 | 1 | 1 | 0 | 1 |
| 9 | 1 | 1 | 1 | 1 |
| 10 | 0 | 1 | 1 | 1 |
| 11 | 1 | 0 | 1 | 1 |
| 12 | 1 | 1 | 0 | 0 |
| 13 | 1 | 1 | 1 | 0 |
| 14 | 1 | 1 | 1 | 0 |
| 15 | 1 | 0 | 1 | 0 |

Figure 5.6: $4 \times 4$ mesh topology with links failed.

Let us explain this operation with the example shown in Figure 5.6. In this topology, two links, one between routers 1 and 5 and the other between routers 6 and 10, have failed. According to this, routing restrictions (bidirectional arrows in the figure) have been computed and so routing bits and connectivity bits (e.g. $C_s[0]$ on router 1 is reset meaning it has no connectivity to the south). Connectivity bits $C_x[0]$ are then copied to $C_x[1]$ and the appropriate links are virtually disconnected (applying the previous rule). We can see

the resulting tree in Figure 5.7. As it is shown, seven links are virtually disconnected (fourteen connectivity bits are reset, those with a grey shadow). Now, the tree is ready for broadcast operations at the region defined (if the region covers all the mesh then is a global tree for the entire chip). Note the built tree is acyclic, thus helping in ensuring broadcast operations will be deadlock-free (we iterate on this later).




| Router | Cn[0] | Ce[0] | Cw[0] | Cs[0] | Cn[1] | Ce[1] | Cw[1] | Cs[1] |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 2 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 3 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 4 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 5 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 6 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 7 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 8 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 9 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 10 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 11 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 12 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 13 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 14 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 15 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |

Figure 5.7: Definition of the broadcast tree.

### 5.2.2 How the Mechanism Works

Let us focus on how the broadcast is performed. When an end node initiates a broadcast operation a packet is created. As said before, a control signal ($B_h$) is used to differentiate broadcast packets from unicast ones. Additionally, four control signals ($B_n$, $B_e$, $B_w$, and $B_s$) are associated to the packet. As with *bLBDR*, these signals travel along with the broadcast packet through control lines (or included in the packet's header).

Control signals are computed on each router but they are used at the next router to decide which output ports can be used for broadcasting the packet. Note this is a key differentiation from bLBDR mechanism. Thus, now in SBBM, the meaning of $B_n$, $B_e$, $B_w$, and $B_s$ signals associated to an incoming packet indicate the output ports the packet needs to be replicated

through. As the first router has to inject the broadcast packet through all possible directions (regardless of the connectivity), the initiating end node sets all the control signals in the local port to one, regardless of the routing and connectivity bits (as done in bLBDR).

We use the same nomenclature as in bLBDR. Broadcast signals for the incoming packet will be referred as $IB_h$, $IB_n$, $IB_e$, $IB_w$, and $IB_s$ regardless of the port identifier the packet comes from. The new signals for packet replicas sent through port $X$ will be labelled as $XB_h$, $XB_n$, $XB_e$, $XB_w$, and $XB_s$.

Upon reception of a broadcast packet, the router may forward the packet through up to four ports (local port is included and U turns are forbidden). The router generates a replica through an output port (e.g. $N$ port) if the corresponding received control signal (e.g. $IB_n$ signal) is set, and the connectivity bit through the output port (e.g. $C_n[1]$ bit) is set. For each replica, control signals are computed again (to be used at the next router) and attached to the packet replica.

### 5.2.3 Logic Implementation

Figure 5.8 shows the SBBM routing logic. In particular, 8 control signals are received and stored: three signals encoding the region of the packet, and five signals with the broadcast info. This unit computes all the output ports that must be used to forward the broadcast packet. To do this, SBBM computes all the broadcast signals. This logic is replicated at each input port. As it can be seen, the complexity at the module is small. The reason for the simplicity is the fact that deadlock-freedom is already guaranteed by the routing bits, and duplicity of packets is prevented by connectivity bits ($C_x[Rid]$ bits).

Indeed, the module just checks the available paths at the next router, which matches with the routing bits present on the current router. As an example, the $NB$ packet signals (used by the broadcast packet that is sent through the north port; see Figure 5.8) are computed as follows:

- The $NB_n$ signal is set if there is no routing restriction between channels $S$ and $N$ at the next router through the north output port, i.e $R_{nn}$ routing bit at the current router. This signal tells the next router that the broadcast packet is allowed to go north.

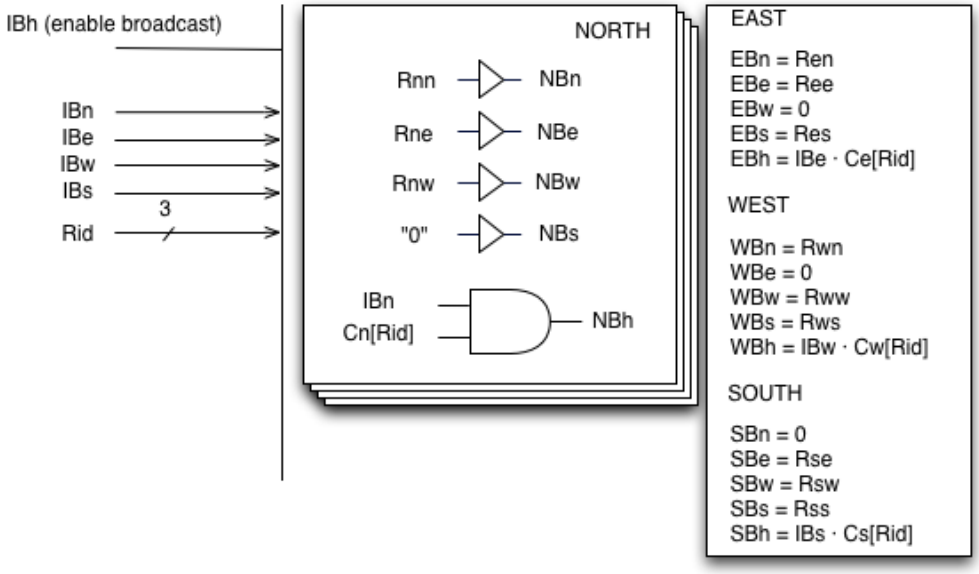



Figure 5.8: SBBM routing logic

- The $NB_e$ signal is set if there is no routing restriction between channels $S$ and $E$ at the next router through the north output port, i.e $R_{ne}$ routing bit at the current router. This signal tells the next router the broadcast packet is allowed to make the turn north-east. The same reasoning is concluded for signal $NB_w$.
- The $NB_s$ signal is reset since $U$ turns are not allowed.
- The $NB_h$ signal is computed by checking the current connectivity bit on the tree-broadcast layer ($C_n[Rid]$) and the $IB_n$ signal. If $NB_h$ is set to zero, then no replica is injected through the $N$ port.

Similar conclusions are obtained for the remaining output ports of the router. It should be noted that every incoming broadcast packet is always routed to the local port (or ports).

### 5.2.4 Detailed Example

Figure 5.9 shows an example of a broadcast action in the example network of Figure 5.7. Router 4 receives the broadcast packet from its local node

and immediately sends two replicas through $N$ and $S$ output ports, the north packet with only the $NB_e$ signal activated to let know router 0 that it must only replicate to the east, and the south packet to router 8 with the $SB_s$ signal activated that tells this router to replicate the packet only to the south. The packet is not replicated through the $E$ port as this link is virtually disconnected ($C_e[1]$ bit is reset), ignoring any incoming signal. Pay special attention at step 4, where router 2 sends a packet to the east, with all signals not activated, but the packet is sent as it must reach router 3. As commented before, $SBBM$ differs from $bLBDR$ in that the signals sent with each replica inform the next router where to send the next replicas (filtered anyway with the connectivity bits in any case). Broadcast actions continue until the last node is reached with this mechanism. Note that the packet is delivered to the local node on every visited router.

## 5.3 Demonstration

As shown in the previous chapter, in Section 4.5, a mechanism based on the routing restrictions methodology is deadlock-free as long as it does not force crossing these restrictions. Routing configuration bits are also used by $bLBDR$ and $SBBM$ mechanisms to prevent crossing the routing restrictions with 1-hop visibility. Taking as a reference Figure 5.2 we can observe that routing bits in $bLBDR$ are used to decide the paths the broadcast packet must cross. As an example $EB_s$ signal is reset (broadcasting through the $S$ port at the router located at the east of the current router) if $R_{es}$ is set to zero (meaning packets can not take east and then south). Therefore routing bits filter routing options of the broadcast packet passing through routing restrictions.

For $SBBM$ the approach is different (see Figure 5.8). Routing bits are used to decide the output port that can be used at the next routers (e.g. $EB_s$ is set if $R_{es}$ is set). Later, connectivity bits (building an acyclic spanning tree) guarantee packets do not form cycles, as the spanning tree is acyclic.

For connectivity in $bLBDR$, such property is guaranteed as long as the topology guarantees minimal path routing and $bLBDR$ is as much flexible as $LBDR$, which already guarantees connectivity.

For $SBBM$, connectivity is guaranteed by the way the spanning tree is

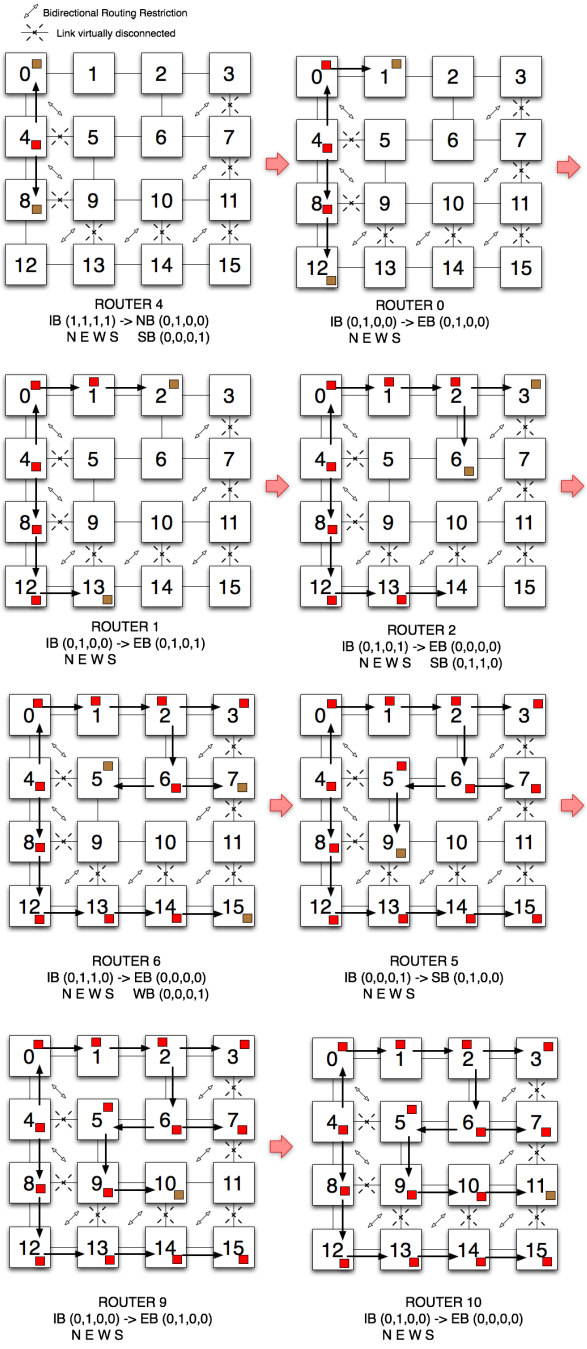



Figure 5.9: Broadcast operation started at router 4.

formed, ensured by the algorithm that computes the spanning tree. Note that links are virtually disconnected in those places when routing restrictions exist. Thus, it is guaranteed that an alternative path exists that overcomes the disconnected link, since the unicast layer is connected and avoids crossing routing restrictions.

Indeed, if any operation crosses a routing restriction or chooses an output port with no connectivity is due to a wrong set of configuration bits. We can see an example in Figure 5.10. Router 0 starts a broadcast operation in $SBBM$ mechanism. If a message arrives at router 5 from router 4 it would be due to the wrong configuration of bits. Connectivity bit $C_e$ from the tree layer at router 4 is reset due to the virtual disconnection, so the routing would not be valid. Even before, at router 0, routing bit $R_{se}$ is reset also due to the routing restriction present at router 4, so when the message sent to the south is generated the $SB_e$ signal is not activated, so again the message can not take this path.

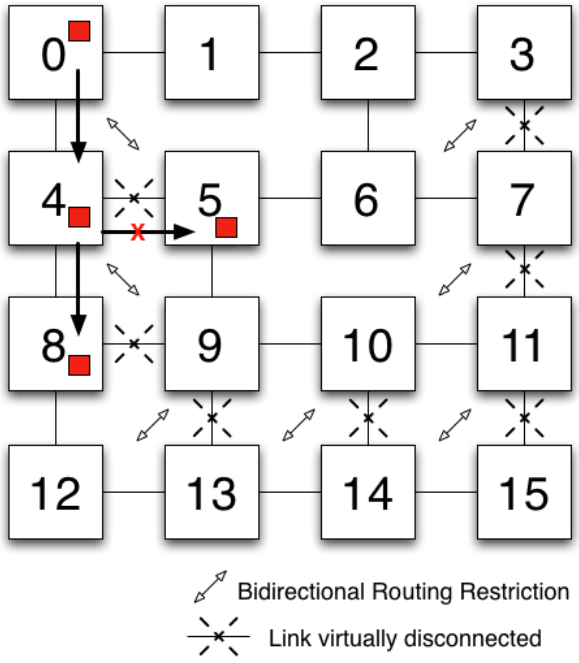



Figure 5.10: Path not suitable.

Multicast and broadcast operations, though, under wormhole switching

are not completely deadlock-free as mentioned in [13]. As explained before for fork operations in *uLBDR* mechanism in Chapter 4, with VCT switching or a switching mechanism that performs at flit level, deadlock scenarios will be avoided.

## 5.4 Evaluation

In this Section we provide evaluation results for *bLBDR* and *SBBM*. First, an evaluation in terms of coverage is shown on a pool of irregular (and regular) topologies derived from a 2D mesh. Second, we analyse the impact of these routing implementations in hardware overhead from the synthesis of the modules. Finally, performance results are shown.

### 5.4.1 Coverage Analysis

In this Section we evaluate the coverage provided by both method on different topologies. Coverage is measured as the percentage of topologies supported from a pool of evaluated topologies. A topology is considered supported if broadcast actions from every node in the network reach all possible destinations and no duplicates are found (no end node receives more than one packet for every broadcast operation). Note that the set of topologies derived from the link variability analysis provided in [49] has been used for this evaluation, as similar as the previous chapter with the unicast routing implementations.

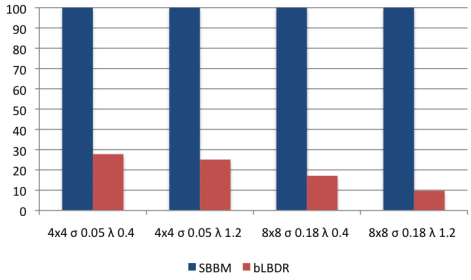



Figure 5.11: Coverage of SBBM and bLBDR.

Figure 5.11 shows the coverage results for *SBBM* and *bLBDR*. As can be seen, coverage in *SBBM* is optimal since all the topologies analysed were

supported by *SBBM*. In all the cases (1423 topologies) *SBBM* successfully implemented a broadcast operation through a valid spanning tree (using the rule described previously in the chapter).

In contrast, *bLBDR*, although efficient in its implementation, does not provide a good coverage. Percentage of supported topologies do not exceed the 30% threshold. The supported topologies are the ones that do not require non-minimal paths (e.g. a 2D mesh with a lower right-hand side submesh removed). As we see, this percentage of cases reduces with network size. Indeed, coverage tends to reduce as irregularity is much more pronounced (as network size increases and as spatial correlation increases). For $8 \times 8$ mesh network coverage is severely reduced to only 10%.

The coverage results are very important, since they do not only determine the suitability of a method on special (irregular) cases. They also determine the complexity of the solution to tolerate such irregularities. For instance, if bLBDR coverage is required to be increased then more logic will be needed. As we will see later, *SBBM* currently covers all the topologies with even less hardware overhead.

In favour of *bLBDR* we can state that it tends to produce shorter broadcast trees, thus incurring in lower latencies. Therefore, if a broadcast mechanism is needed in regular fault-free configurations, then *bLBDR* could be a better option.

### 5.4.2 Area, Latency and Power Breakdown

In this section we provide synthesis results for *SBBM* and *bLBDR* on area, latency and power requirements. To this end, *SBBM* and *bLBDR* methods have been implemented as modules on SystemC. In both cases they represent only the routing/connectivity bits and the logic for the computation of broadcast signals on every router. Therefore, special treatments for multicast in buffer, flow control, and scheduling is not considered. In all the broadcast methods these issues are equivalent and thus do not lead to differences between the methods.

Both modules (*SBBM* and *bLBDR*) were synthesized in two steps. First the SystemC model was translated into an RTL Verilog model. Then, the RTL Verilog model was synthesized with Synopsys Physical Compiler (wireload

models in traditional synthesis tools are not trustworthy any more in the context of nanoscale technologies, therefore Physical Compiler performs placement-aware logic synthesis) on a $65nm$ low-power low-Vth industrial technology library. For SBBM, $R_{xx}$ routing bits ($R_{nn}$, $R_{ee}$, $R_{ww}$, $R_{ss}$) are included on the analysis.

Additionally, we compare the results on area and latency, with the ones from the VCTM method published in [15]. We have to remark that VCTM results were obtained when targeting a $70nm$ technology. This comparison between VCTM, $bLBDR$ and $SBBM$ is done because $SBBM$ also works for regular 2D meshes. We must take into account that the synthesis of both modules is driven by the fact that $SBBM$ and $bLBDR$ methods are orthogonal respect to the router design.

Results are obtained for a $4 \times 4$ 2D mesh NoC, 16 end nodes. Remember that $SBBM$ and $bLBDR$ logic do not increase with the number of destinations as opposed to table-based solutions. Their logic grows only with switch radix size, so we have assumed an standard configuration of five input/output ports: $N$, $E$, $W$, $S$ and a local port attached to a core in the node.

Table 5.1: Area, delay and power evaluations. $4 \times 4$ mesh network.

| | Area ($\mu m2$) | Delay (ns) | Power ($\mu W$) |
|---|---|---|---|
| SBBM (4 ports + local) | 1242.8 | 0.07 | 244.6 |
| bLBDR (4 ports + local) | 1892.8 | 0.08 | 415.3 |
| VCTM - Destination CAM (512 entries) | 24000 | 0.43 | 1800 |
| VCTM - VCT Table (32 entries) | 18000 | 0.87 | 700 |

As we can see in Table 5.1, $SBBM$ achieves much better results when compared to the $bLBDR$ module. In particular, a reduction of 34% in area and a reduction of 50% in power is achieved. In terms of latency, the achievement factor is small, but $SBBM$ still performs better.

Compared to the VCTM solution, savings are much greater. Giving just an example, both tables required for VCTM (at each end node plus router) require an area 33 times larger than the one required for SBBM. In latency, SBBM also outperforms, being its latency just $0.07ns$ as opposed to $0.87ns$ for VCTM.

We have also synthesized $bLBDR$, $SBBM$ and routing tables (a simple

implementation of table-based solution that consists of a bit-based register for 16 destinations along with the logic for decoding the packet header, performing unicast-based collective communication) modules on a simple 5×5 radix router to see how the critical path is affected by these modules. As we can see at Figure 5.12 *SBBM* still outperforms with its logic-based implementation.

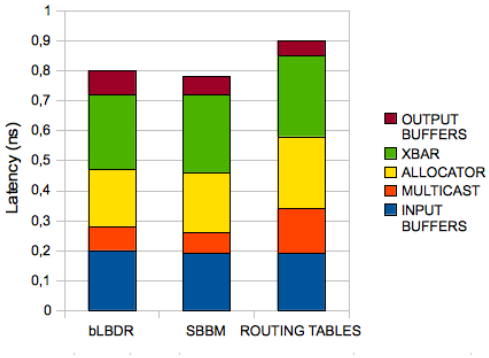



Figure 5.12: Critical path breakdown for different methods.

### 5.4.3 Performance Analysis

We evaluate now the performance of different applications under the behaviour of *bLBDR* and *SBBM* mechanisms. Tests have been run under the GEMS/SIMICS platform [33] with an event driven cycle-accurate on-chip network simulator (described in Chapter 4). Several SPLASH-2 [72] applications (Barnes, FFT, LU and Radix) and Apache [20] application have been tested. A 16-tile (regular 4 × 4 mesh) CMP system has been modelled, each tile including a core processor, a private 2 MB L1 cache, a 4 MB L2 cache bank, a memory controller and the router. Flit size has been set to 3 bytes and virtual cut-through switching is assumed. Both mechanisms are compared to a unicast-based *XY* mechanism (that deals with broadcast/multicast operations by serializing unicast messages). Cache coherency is kept by a directory-based protocol (MOESI_directory). In addition, the token protocol [32] is also evaluated (MOESI_token). As *bLBDR* and *DOR* only allow minimal paths, a network with no failures is used.

Figures 5.13 and 5.13 show the normalized execution time of every applica-

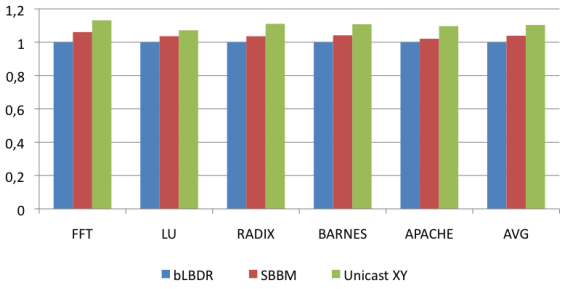



Figure 5.13: Execution time, token-based protocol, normalized results.

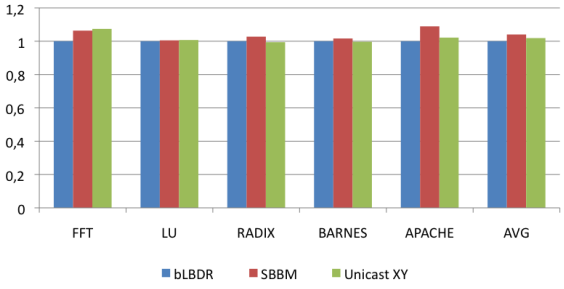



Figure 5.14: Execution time, directory-based protocol, normalized results.

tion. For a token-based implementation, as seen in Figure 5.13, we can appreciate that tree-based approaches (*bLBDR* and *eLBDR*) reduce the execution time between 8% and 13% compared to the unicast-based approach (*XY* routing). Instead, in a directory-based environment, Figure 5.14, the impact is not critical and even in some applications (like Radix), the unicast-based option performs better, but reflected by a slightly small percentage, around 1%. The reason behind this is that with directory-based protocols there is a very low percentage of multicast/broadcast traffic present in the applications. Contrary to this, in token-based protocols collective-communication is required due to the large percentage of broadcast operations. The percentage of collective operations in directory-based protocols is, on average, 0.05%, while in token-based the percentage increases to 11%. Therefore, a tree-based broadcast is beneficial in token-based protocols. When comparing *SBBM* with *bLBDR* the difference on performance is due to the different spanning trees (quality

of the trees) used by both solutions. Anyway, differences are low, and most important, *SBBM* can be used on any irregular topology (100% coverage).

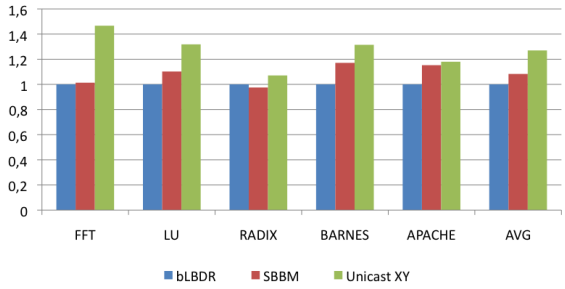



Figure 5.15: Average packet latency, token-based protocol, normalized results.

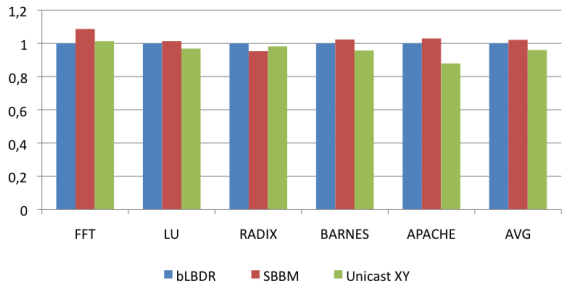



Figure 5.16: Average packet latency, directory-based protocol, normalized results.

Figures 5.15 and 5.15 show normalized average packet latency. Note that using broadcast operations helps in reducing packet latency in token-based protocols (Figure 5.15) where we can find divergence of results of more than 40%, like in FFT, respect to the tree-based methods. In directory-based protocols (Figure 5.16) where the latency results are similar and as the overall impact of broadcast operations present on the running scenario is almost negligible, the unicast-based solution usually performs better.

Finally, Figures 5.17 and 5.17 show the network throughput for both protocols. As expected, throughput is increased when token protocol is used and is heavily penalized when using a unicast-based solution where the serialization of broadcast packets is present. Again, the low percentage of broadcast

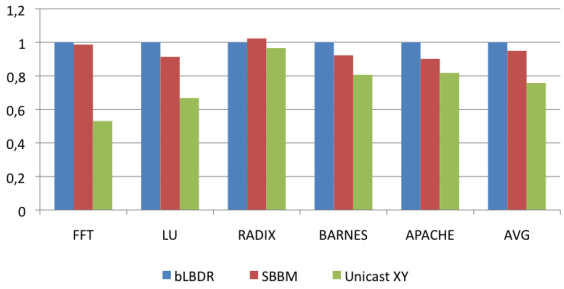



Figure 5.17: Network throughput, token-based protocol, normalized results.

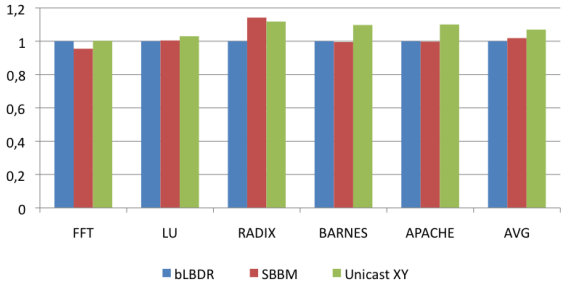



Figure 5.18: Network throughput, directory-based protocol, normalized results.

traffic on the directory-based protocol makes this case having results not quite significant compared when using a token-based protocol.

## 5.5 Gathering Unicast and Broadcast Implementations

In the previous and current chapter, a presentation of unicast and collective communication routing solutions has been made that offer a full coverage on any topology derived from a 2-dimensional mesh. Indeed, *uLBDR* and *SBBM* routing implementations provide support for each type of communication, respectively, with resource costs comparable to other designs that are the best representative on each field in terms of efficiency (like Dimension-Order-Routing), and with an applicable flexibility that routing table-based

implementations show.

The next logical step is to offer an architecture that is able to respond to both needs in communication, while maintaining a compact, simple and flexible implementation. In this section, *eLBDR* (effective Logic-Based Distributed Routing) is presented, to take on such a challenge. We will first describe the architecture that will encompass both *uLBDR* for unicast and *SBBM* for broadcast/multicast communication support. Then, we will evaluate the final method in terms of router implementation and performance impact. Note that coverage is already guaranteed to achieve 100% in both communication traffics.

### 5.5.1 *eLBDR* Architecture

Let us introduce an overall view of the system. As we can see in Figure 5.19, as soon as the packet header is decoded in the input buffer of a router, the first operation is to distinguish whether the packet is meant for unicast or for broadcast routing. As introduced before, a control bit, $IB_h$, is associated with the packet and enables the appropriate module (unicast or broadcast module). *SBBM* additional control signals ($IB_n$, $IB_e$, $IB_w$, and $IB_s$) are used for broadcast operations together with the $R_{id}$ bits that indicate the region the packet belongs to or the tree-based broadcast layer.

Each router has a global register storing all the *eLBDR* bits (routing and connectivity bits). Additionally, each router has a register with an ID (i.e. the coordinates in the 2D mesh). Each input port includes also a local 6-bit register, with the $dr_x$ and $F_x$ bits. The global register is used by both modules, unicast and broadcast, whereas the local register is used only by the unicast module. Unicast module corresponds with the *uLBDR* mechanism as shown in Figure 4.27, and the broadcast module with the *SBBM* mechanism like in Figure 5.8.

Both modules generate the corresponding signals to the arbiter. There is a signal for each output port ($N$, $E$, $W$, $S$ or a local port) from each module. The arbiter configures the crossbar to route the packet to the corresponding output port, or replicates it to several output ports when needed like in the case of fork or broadcast operations. Indeed, both types of operations can be handled by the arbiter in a similar way as a packet can be replicated, at least,

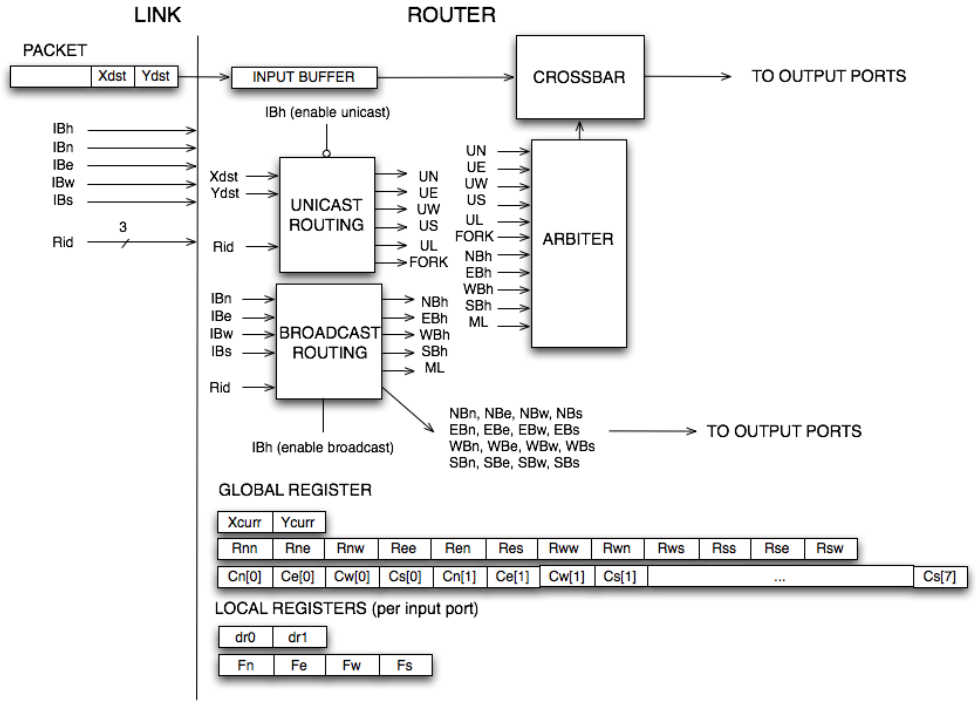



Figure 5.19: A general overview of the $eLBDR$ architecture inside a router.

for two output ports. In fact, the only difference, in case of replicas generated due to a fork operation is that the $IB_h$ signal would be not activated, and this is done by taking into account the $FORK$ signal that arrives to the arbiter. This signal also manages to distinguish whether two $UX$ signals come from the basic $LBDR$ mechanism (remember that $LBDR$ could give two output ports as valid) or are due to a fork operation. This allows the arbiter to manage effectively the priority strategy implemented. Note that the basic mechanism can implement also the basic filtering for simple arbiters described in Chapter 4. In case of deroutes, as this operation only provides one output port, there is no specific action defined at the arbiter.

### 5.5.2 Evaluation

For a complete breakdown evaluation, $eLBDR$ architecture has been integrated in the two router models: the non-pipelined MPSoC router and the pipelined CMP router. Both router models are an evolution of those pre-

sented at Chapter 4 (Section 4.6) to support both types of communications as described before. Note that, as mentioned before, both router models start from a wormhole switching implementation to a virtual cut-through implementation to ensure deadlock-free routing for unicast communication with fork operations involved and collective communication.

As mentioned before, *eLBDR* achieves full coverage. Indeed, as unicast and broadcast modules achieve this objective independently and any message is forwarded to its respective module, it is ensured that no loss in coverage is incurred by *eLBDR*, thus offering 100% coverage. Anyway, *eLBDR* routing implementation was tested against the pool of topology instances used in the previous coverage evaluation sections [49]. Results show that *eLBDR*, as expected, is able to provide full coverage on every topology tested.

### MPSoC Router Overhead

Post-synthesis area and performance results for the routers are shown in Figures 5.20 and 5.21. Routers with just the unicast module (*uLBDR*) and full *eLBDR* (with unicast and broadcast modules) were also synthesized. All routers implement the same amount of buffering (4 slots per input port) and were synthesized for maximum performance.

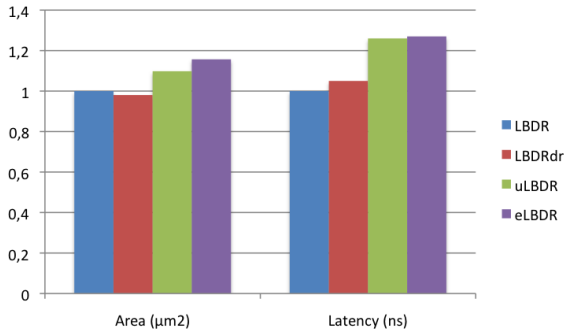



Figure 5.20: MPSoC complete router overhead, normalized results.

Note that *eLBDR* router is only less than 1% slower than the unicast implementation, *uLBDR*, as shown in Figure 5.20, increasing the total area only by a negligible 5% respect to the *uLBDR*. This is due to the fact that

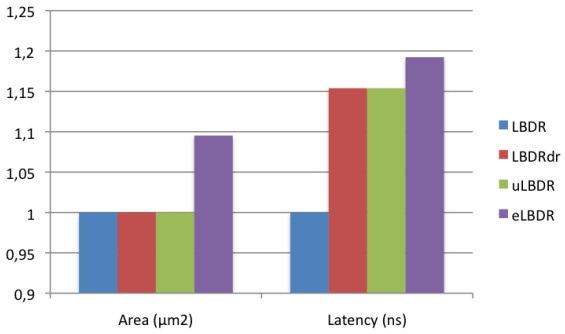



Figure 5.21: MPSoC RT stage overhead, normalized results.

the broadcast module works in parallel with the unicast module, which is more complex from a combinational logic viewpoint. The actual difference between both critical paths is the combinational logic added to couple the unicast and broadcast routing modules together. However, this logic is efficiently handled by the synthesis tool and its impact is minimized. Focusing on the *RT* stage, as seen in Figure 5.21, the area gap between *uLBDR* and *eLBDR* is increased by 9%, due to the addition of the broadcast module (and logic associated). The combinational logics adds a total of 4% more latency respect to the *uLBDR* implementation, but does not impact the critical path of the router.

### CMP Router Overhead

In Figure 5.22 normalized area and delay results are shown for the CMP router with different routing mechanisms (and the switching techniques implemented with them). As can be seen in the figure, for *eLBDR* the support for broadcast operations over *uLBDR* comes with no further penalty in performance and with a marginal increase of area.

Figure 5.23 also shows the area overhead and delay only for the *RT* stage. Again, for *eLBDR* the support of unicast plus broadcast routing sets a minimal impact on both area and delay over *uLBDR*, less than 8% and 3% marginal gaps, respectively. Furthermore, *eLBDR* architecture module does not set the critical path of the router.

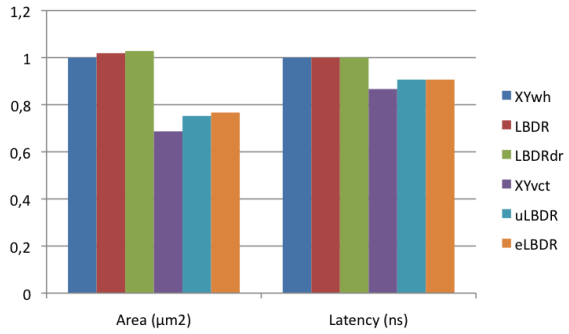



Figure 5.22: CMP complete router overhead, normalized results.

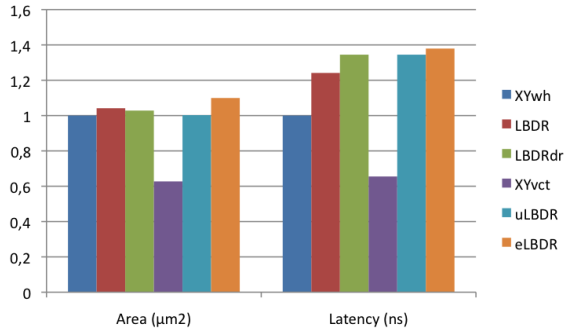



Figure 5.23: CMP RT stage overhead, normalized results.

## 5.6 Conclusions

*bLBDR* and *SBBM* offer broadcast/multicast support specially designed for NoCs, in the sense that its implementation cost is minimal. *SBBM* offers a solution that achieves a full coverage (100%) of any irregular topology derived from an initial 2D mesh structure, while *bLBDR* covers roughly a 30% of the total instances. *bLBDR* only provides minimal path support, but, with *SBBM*, half of the connectivity layers are reserved for the tree computation and obtains slightly lower results in overall performance due to the spanning tree.

*SBBM* does not follow the typical approach. Also, its main driver is the coverage that supports for irregular topologies derived from 2D mesh structures. This support is very convenient if not mandatory to tolerate manufac-

turing defects, support for coherency protocols, virtualization, partitionability, traffic isolation, and collective communication in CMPs. $SBBM$ follows the trend initiated by $bLBDR$ were the mechanism can be viewed as a region-based multicast routing mechanism, where regions are predefined based on the current set of applications being executed on top of the chip. The mechanism, however, can be adapted to support dynamic formation of regions by changing the connectivity bits, and thus the shape of the regions. Bear in mind, that we address manufacturing defects and application-level virtualization granularity. So $SBBM$ (and $bLBDR$) requires that routing/connectivity bits are to be set and changed on power on and on the event of new applications.

In this chapter we have also presented a logic-based routing layer for on-chip networks (either CMPs or MPSoCs) that aims to effectively support any irregular (and regular) topology derived from a 2D mesh without the need for routing tables. $eLBDR$ supports both unicast and collective communication operations. As new challenges on the deep sub-micron level design for NoCs arise, like manufacturing defects, support for application-level parallelism or power-aware techniques, and the need for collective communication, designers must make an effort on implementing new ideas while achieving an affordable trade-off between NoC designs and coverage. The architecture proposed offers support for a fault-tolerant routing layer in manufacturing environments where failures are becoming more and more frequent. This is achieved by effectively supporting a full coverage with a marginal impact on router design. In fact, $eLBDR$ is able to cover all the failure cases on a 2D mesh with marginal impact on router area and delay.

# Chapter 6

# Conclusions

*"So close, and yet so far away."*

Anonymous.

In this final chapter, we present the conclusions of this dissertation, the contributions to the research domain and a brief list of research directions that will be addressed in the future. Finally, we also expose the results in terms of scientific publications and other contributions derived from the work presented in this dissertation.

## 6.1 Conclusions

The following is a list of conclusions extracted from the current dissertation. These conclusions helped to obtain the contributions and scientific publications that are at the core of the document.

- **Current high-performance multicore solutions pledge for tile-based designs.** Tile-based design is gaining momentum for newer Chips Multiprocessor (CMPs) and Multiprocessor System-on-Chips (MPSoCs) solutions. As the expected trend is to include more and more cores inside a chip, these solutions rely on networks-on-chip (NoCs) to handle all the communication traffic between cores.

- **2-Dimensional mesh topologies are appealing for CMPs and MPSoCs.** Manufacturers prefer 2-dimensional mesh topologies due to their simplicity for routing purposes and that they fit very well the chip layout. Although other topologies are also interesting, from the point of view of performance, design tools are not suitable (e.g. fat-tree topologies) or the tile-based design enforce an homogeneous and regular structure of the network, this aiming for a 2-D mesh.

- **It is imperative to find NoC solutions that offer at the same flexibility and tight savings in area, power consumption, and latency, that are critical key aspects in the NoC domain.** Table-based routing solutions offer excellent flexibility, but as they are usually implemented with memory structures, they suffer from poor scalability, and are resource-hungry (area, power consumption, and latency). Logic-based implementations like Dimension-Order-Routing (DOR), offer, on the contrary, excellent results in area, delay and power-awareness, but they lack flexibility and can not support non-minimal path routing. It is imperative to find solutions that are both efficient as logic-based routing algorithms and flexible as routing tables.

- **Future challenges in CMPs and MPSoC will demand dynamic mechanisms in the chip so to adapt to such challenges.** Challenges identified in the document are (1) manufacturing defects where yield will become an important issue, (2) virtualization techniques to effectively use the chip when different applications are run on top of it, (3) aggressive power saving techniques that allow any set of the network components to be switched (with no interactions with the remaining network), and (4) support for efficient DVFS domains which will avoid interferences between such domains.

- **Routing restrictions are an efficient representation of instances of routing algorithms applied to topologies.** Indeed, a straightforward view of the design for the further implementation is desirable. This compact representation of the acyclic CDG over the network to avoid deadlock scenarios is suitable for the on-chip network domain as it helps to implement simpler designs. Indeed, this is seen as a way to tackle

the problem of efficient routing implementations that are at the same level of flexibility of routing tables but with comparable overhead costs of logic-based implementations.

The previous conclusions from the dissertation put the research performed in perspective so to obtain the intended goals. In the next section the specific contributions of the current dissertation are highlighted.

## 6.2 Contributions

The overall contribution of the dissertation is the design of an efficient unicast/broadcast mechanism able to route messages in any of the possible configurations of the network that started with an initial 2-D mesh structure. The mechanisms are able to tolerate any possible configuration (100% coverage) resulting from the challenges identified, with no performance degradation and with costs similar to logic-based implementations of the routing algorithm, like the DOR routing. Most important, the solution is scalable in the sense that the same logic and configuration bits are required regardless of the network/system size. This overall achievement has been obtained with a step by step procedure described next:

- **Configuration bits are a compact yet powerful foundation to aid routing implementations.** The objective of routing ($R_{xy}$) and connectivity ($C_x$) bits is to provide a simple translation from the viewpoint of each router of the neighbouring conditions of routing restrictions and connectivity of the topology. They enable support for the incoming challenges in networks-on-chip: fault-tolerance, power consumption issues, and virtualization-enabled systems. The aim is that routing implementations operate with the same degree of flexibility that routing tables would offer. Configuration bits also offer good scalability, as they do not grow with system size.

- **Logic-based distributed routing: Unicast routing with minimal path support.** LBDR mechanism is a logic-based routing implementation capable to support a vast range of routing algorithms (that feature

deadlock-freedom and full connectivity between each pair of end nodes) in 2-dimensional mesh topologies where each end node can communicate to another end node through a minimal path. LBDR is based on the foundations aforementioned, the configuration bits methodology, so it benefits from the same properties: great scalability, and important savings in area, latency and power consumption respect to table-based routing implementations and similar savings to DOR-based implementations.

- **Deroutes and forks extensions, enabling a unicast implementation to offer full coverage and non-minimal path support.** Universal LBDR (uLBDR) routing implementation handles, with a minimal addition of configuration bits to each router, any irregular pattern configuration of the 2-D mesh topology, providing support for the challenges mentioned before with a compact mechanism. Indeed, uLBDR presents, although slightly more complex than the basic mechanism, better results and large savings than table-based routing implementations.

- **Broadcast/multicast communication.** Two mechanisms developed, broadcast LBDR (bLBDR) and Signal Bit-Based Multicast (SBBM). bLBDR is a broadcast/multicast logic-based implementation that works with the same foundations of the unicast solutions able to offer tree-based broadcast operations at user-defined (totally configurable) region level in the same conditions as the unicast counterpart, LBDR. SBBM is the evolution of bLBDR, with a different approach but reusing the same resources, to offer non-minimal path routing, like in uLBDR, but for broadcast purposes. Results show that SBBM offers 100% coverage on irregular scenarios, whereas bLBDR features the same properties, but restricted to minimal path routing. SBBM presents better savings in area, latency and power consumption than bLBDR, but bLBDR is slightly better in performance results due to the way the different spanning of the broadcast trees are computed for each mechanism. It is a tradeoff that must be considered depending on the conditions of each implementation scenario.

- **An architecture able to offer both types of communication,**

**unicast and broadcast/multicast, while maintaining the same properties of the basic mechanisms.** Effective LBDR (eLBDR) gathers the philosophy of uLBDR and SBBM to offer a complete solution ready for NoCs, at is has been implemented on two real router environments: CMP and MPSoCS. As the previous solutions, eLBDR handles inter-core communication with large flexibility (like in table-based routing implementations) and presenting excellent savings in network resources.

## 6.3 Future Work

In this Section we highlight possible future work directions coming from this dissertation. The following is a list of possible directions:

- **Extend the support to other topologies.** 2-dimensional meshes dominate the current panorama of on-chip network oriented design, but other topologies may also be applicable to NoCs due to different properties they could offer, including highly irregular topologies. Interesting topologies to analyze are 3D meshes (due to the 3D stacking concept), and 2D torus networks (due to its higher bisection bandwidth and reduced diameter).

- **Dynamic reconfiguration of the bits.** We have assumed in this dissertation and our research that any change in the network that generates a new set of configuration bits is made offline by the computation bit algorithm. Extending this methodology to support dynamic reconfiguration is mandatory for transient faults and other related scenarios to offer better performance. However, dealing with the proper order of computing bits and distributing bits in the network is challenging since deadlocks in the transitory state may be induced.

- **Simplification and improvement of the routing mechanisms.** Although the routing implementations presented in this dissertation offer excellent results, we aim for better improvements and savings in the logic designs and foundations.

## 6.4 Publications related to this work

The following list enumerates the papers related with this dissertation that have been published, or are under review process, in specialized conferences or journals. We outline for each contribution the novelties that are part of this dissertation.

- 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," *The 4nd IEEE International Symposium on Networks-on-Chip (NOCS)*, **Best paper award**, 2010.

- 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 review process for a *special issue in IEEE Transactions on Computed Aided Design.*

We presented in this conference the ultimate unicast solution uLBDR, introducing the fork and deroutes operation as the way to provide full coverage. Multicast/Broadcast support is not provided. The second contribution is an extended version that includes power consumption estimations and evaluations with a token-based protocol.

- S. Rodrigo, C. Hernández, J. Flich, F. Silla, J. Duato, S. Medardoni, D. Bertozzi, A. Mejía, and D. Dai., "Yield-oriented Evaluation Methodology of Network-on-Chip Routing Implementations," *International Symposium on System-on-Chip (SOC)*, 2009.

We performed in this paper the first coverage analysis with LBDR (unicast mechanism) with the inclusion of the deroute bits. However, the major contribution in the paper was the linking between routing algorithms and routing implementations. Indeed, we provided a methodology to test different routing instances until the LBDR mechanism succeeds in providing coverage to the topology. Initial router designs are also presented in the paper.

- S. Rodrigo, J. Flich, J. Duato, and M. Hummel, "Efficient Unicast and Multicast Support for CMPs," *The 41st Annual IEEE/ACM Interna-*

*tional Symposium on Microarchitecture (MICRO41)*, 2008. Awarded by HiPEAC as significant contribution.

In this paper we presented the bLBDR mechanism able to broadcast minimal-path regions. The overlapped region concept was also introduced.

- J. Flich, S. Rodrigo, and J. Duato, "LBDR: Efficient Routing Implementation in NoCs," *2nd Workshop in Interconnection Network Architectures: On-Chip, Multi-Chip (INA-OCMC), held in conjunction with the 3rd International Conference on High-Performance Embedded Architectures and Compilers (HiPEAC)*, 2008.

- J. Flich, S. Rodrigo, and J. Duato, "An Efficient Implementation of Distributed Routing Algorithms for NoCs," *The 2nd IEEE International Symposium on Networks-on-Chip (NOCS)*, 2008

- S. Rodrigo, S. Medardoni, J. Flich, J. Duato and D. Bertozzi, "An Efficient Implementation of Distributed Routing Algorithms for NoCs," published in the *special issue on Networks-on-Chip of IET Computers & Digital Techniques*, Volume 3, Number 5, pages 460–475, 2008.

In these previous papers we introduced the LBDR mechanism with no deroutes and forks. We also provided insights how the mechanism can be adapted to two-phase arbiters and how concentrated meshes can be used with the mechanism.

- S. Rodrigo, J. Flich, A. Roca, S. Medardoni, D. Bertozzi, F. Silla, and J. Duato, "Fault-Tolerant Routing for Next Generation Multicore Chips," in review process in *IEEE Transactions and Computers*.

In this paper we presented the overall unicast/multicast/broadcast mechanism (eLBDR), providing two router solutions with the mechanism.

- S. Rodrigo, J. Flich, and J. Duato, "Addressing Collective Communication in CMPs facing new challenges," *XXI Jornadas de Paralelismo*, 2010.

- S. Rodrigo, J. Flich, and J. Duato, "Logic Tree-Based Broadcast Support for CMPs," *XX Jornadas de Paralelismo*, 2009.

- S. Rodrigo, J. Flich, and J. Duato, "Una Implementacion Eficiente de Algoritmos de Encaminamiento Distribuido para Redes dentro del Chip," *XIX Jornadas de Paralelismo*, 2008.

- S. Rodrigo, J. Flich, J. Duato, and D. Bertozzi, "Assessing the implementation trade-offs of logic-based distributed routing for Networks-on-Chip," *4th International Summer School on Advanced Computer Architecture and Compilation for Embedded Systems (ACACES, HiPEAC)*, 2008.

These papers are summaries of the work done in the dissertation and belong to conferences with non peer-review systems.

# Bibliography

[1] P. Abad, V. Puente, and J. A. Gregorio. MRR: Enabling fully adaptive multicast routing for CMP interconnection networks. In *15th International Symposium on High-Performance Computer Architecture*, pages 355–366, 2009.

[2] C. Bobda, A. Ahmadinia, M. Majer, J. Teich, S. P. Fekete, and J. Van der Veen. DyNoC: A dynamic infrastructure for communication in dynamically reconfigurable devices. In *15th International Conference on Field-Programmable Logic and Application*, 2005.

[3] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny. Routing table minimization for irregular mesh NoCs. In *DATE '07: Proceedings of the Conference on Design, Automation and Test in Europe*, pages 942–947, San Jose, CA, USA, 2007. EDA Consortium.

[4] S. Borkar, R. Cohn, G. Cox, S. Gleason, and T. Gross. iWarp: an integrated solution of high-speed parallel computing. In *Supercomputing '88: Proceedings of the 1988 ACM/IEEE Conference on Supercomputing*, pages 330–339, Los Alamitos, CA, USA, 1988. IEEE Computer Society Press.

[5] L. Cherkasova, V. Kotov, and T. Rokicki. Designing fibre channel fabrics. In *ICCD '95: Proceedings of the 1995 International Conference on Computer Design*, page 346, Washington, DC, USA, 1995. IEEE Computer Society.

[6] Faraday Technology Corp. UMC free library – 90nm IPs. Available at `http://freelibrary.faraday-tech.com/ips/90library.html`.

[7] Intel Corp. The single-chip cloud computer. Available at `http://techresearch.intel.com/articles/Tera-Scale/1421.htm`.

[8] Intel Corp. Teraflops research chip. Available at `http://techresearch.intel.com/articles/Tera-Scale/1449.htm`.

[9] Tilera Corp. Tilera tile multicore processors. Available at `http://www.tilera.com/products/processors.php`.

[10] W. J. Dally. Virtual-channel flow control. *IEEE Transactions on Parallel and Distributed Systems*, 3(3):194–205, March 1992.

[11] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[12] M. Dehyadgari, M. Nickray, A. Afzali-Kusha, and Z. Navabi. Evaluation of pseudo adaptive xy routing using an object oriented model for NOCs. In *The 17th International Conference on Microelectronics*, 2005.

[13] J. Duato, S. Yalamanchili, and Ni. L. M. *Interconnection Networks: An Engineering Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[14] N. D. Enricht Jerger, L. Peh, and M. H. Lipasti. Virtual circuit tree multicasting: A case for on-chip hardware multicast support. In *International Symposium on Computer Architecture (ISCA-35)*, 2008.

[15] N. D. Enricht Jerger, L. Peh, and M. H. Lipasti. Virtual tree coherence: Leveraging regions and in-network multicast trees for scalable cache coherence. In *MICRO 41: Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture*, pages 35–46, Washington, DC, USA, 2008. IEEE Computer Society.

[16] J. Flich, M. P. Malumbres, P. López, and J. Duato. Performance evaluation of a new routing strategy for irregular networks with source routing. In *ICS '00: Proceedings of the 14th International Conference on Supercomputing*, pages 34–43, New York, NY, USA, 2000. ACM.

[17] J. Flich, A. Mejía, P. López, and J. Duato. Region-based routing: An efficient routing mechanism to tackle unreliable hardware in network on chips. In *NOCS '07: Proceedings of the First International Symposium on Networks-on-Chip*, pages 183–194, Washington, DC, USA, 2007. IEEE Computer Society.

[18] J. Flich, S. Rodrigo, and J. Duato. An efficient implementation of distributed routing algorithms for NoCs. In *NOCS '08: Proceedings of the 2bd ACM/IEEE International Symposium on Networks-on-Chip*, pages 87–96, Washington, DC, USA, 2008. IEEE Computer Society.

[19] A. Gara, M. A. Blumrich, D. Chen, G. L.-T. Chiu, P. Coteus, M. E. Giampapa, R. A. Haring, P. Heidelberger, D. Hoenicke, G. V. Kopcsay, T. A. Liebsch, M. Ohmacht, B. D. Steinmacher-Burow, T. Takken, and P. Vranas. Overview of the Blue Gene/L system architecture. *IBM J. Res. Dev.*, 49(2):195–212, 2005.

[20] GEMS-Wiki. Workload specific details. Available at `http://www.cs.wisc.edu/gems/doc/gems-wiki/moin.cgi/Workload_scripts`.

[21] M. E. Gómez, P. López, and J. Duato. A memory-effective routing strategy for regular interconnection networks. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, page 41.2, Washington, DC, USA, 2005. IEEE Computer Society.

[22] M. E. Gómez, N. A. Nordbotten, J. Flich, P. López, A. Robles, J. Duato, T. Skeie, and O. Lysne. A routing methodology for achieving fault tolerance in direct networks. *IEEE Trans. Comput.*, 55(4):400–415, 2006.

[23] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy. Introduction to the CELL multiprocessor. *IBM Journal of Research and Development*, 49(4/5):589–604, 2005.

[24] A. Kohler and M. Radetzki. Fault-tolerant architecture and deflection routing for degradable NoC switches. In *NOCS '09: Proceedings of the 2009 3rd ACM/IEEE International Symposium on Networks-on-Chip*, pages 22–31, Washington, DC, USA, 2009. IEEE Computer Society.

[25] M. Koibuchi, A. Funahashi, A. Jouraku, and H. Amano. L-Turn routing: An adaptive routing in irregular networks. In *ICPP '02: Proceedings of the 2001 International Conference on Parallel Processing*, pages 383–392, Washington, DC, USA, 2001. IEEE Computer Society.

[26] M. Koibuchi, A. Jouraku, K. Watanabe, and H. Amano. Descending layers routing: A deadlock-free deterministic routing using virtual channels in system area networks with irregular topologies. *ICPP '03: Proceedings of the 2003 International Conference on Parallel Processing*, 0:527, 2003.

[27] M. Koibuchi, H. Matsutani, H. Amano, and T. M. Pinkston. A lightweight fault-tolerant mechanism for network-on-chip. In *NOCS '08: Proceedings of the Second ACM/IEEE International Symposium on Networks-on-Chip*, pages 13–22, Washington, DC, USA, 2008. IEEE Computer Society.

[28] J. Laudon and D. Lenoski. The SGI origin: a ccNUMA highly scalable server. *SIGARCH Computer Architecture News*, 25(2):241–251, 1997.

[29] I. Loi, F. Angiolini, and L. Benini. Synthesis of low-overhead configurable source routing tables for network interfaces. In *DATE '09: Proceedings of the Conference on Design, Automation and Test in Europe*, pages 262–267, 2009.

[30] O. Lysne, T. Skeie, S. A. Reinemo, and I. Theiss. Layered routing in irregular networks. *IEEE Transactions on Parallel and Distributed Systems*, 17(1):51–65, 2006.

[31] S. Manolache, P. Eles, and Z. Peng. Fault and energy-aware communication mapping with guaranteed latency for applications implemented on NoC. In *DAC '05: Proceedings of the 42nd Annual Design Automation Conference*, pages 266–269, New York, NY, USA, 2005. ACM.

[32] M. M. K. Martin, M. D. Hill, and D. A. Wood. Token coherence: decoupling performance and correctness. In *ISCA '03: Proceedings of the 30th Annual International Symposium on Computer Architecture*, pages 182–193, New York, NY, USA, 2003. ACM.

[33] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood. Multifacet's general execution-driven multiprocessor simulator (gems) toolset. *SIGARCH Computer Architecture News*, 33(4):92–99, 2005.

[34] M. R. Marty and M. D. Hill. Virtual hierarchies to support server consolidation. In *ISCA '07: Proceedings of the 34th Annual International Symposium on Computer Architecture*, pages 46–56, New York, NY, USA, 2007. ACM.

[35] A. Mejía, J. Flich, J. Duato, S. A. Reinemo, and T. Skeie. Segment-based routing: an efficient fault-tolerant routing algorithm for meshes and tori. *International Parallel and Distributed Processing Symposium*, 0:84, 2006.

[36] Nangate. The nangate open cell library, 45nm freepdk. Available at `https://www.si2.org/openeda.si2.org/projects/nangatelib`.

[37] E. Nilsson, M. Millberg, J. Öberg, and A. Jantsch. Load distribution with the proximity congestion awareness in a network on chip. In *DATE '03: Proceedings of the Conference on Design, Automation and Test in Europe*, page 11126, Washington, DC, USA, 2003. IEEE Computer Society.

[38] J.L. Nuñez Yanez, D. Edwards, and A.M. Coppola. Adaptive routing strategies for fault-tolerant on-chip networks in dynamically reconfigurable systems. *IET Computers and Digital Techniques*, 2(3):184–198, 2008.

[39] J. Öberg. Clocking strategies for networks-on-chip. *Networks on chip*, pages 153–172, 2003.

[40] University of Catania (Italy). Noxim, the NoC simulator. Available at `http://noxim.sourceforge.net`.

[41] M. Palesi, R. Holsmark, S. Kumar, and V. Catania. Application specific routing algorithms for networks on chip. *IEEE Transactions on Parallel and Distributed Systems*, 20(3):316–330, 2009.

[42] M. Palesi, S. Kumar, and R. Holsmark. A method for router table compression for application specific routing. In *SAMOS VI Workshop in Mesh Topology NoC Architectures*, pages 373–384, 2006.

[43] M. Pirretti, G. M. Link, R. R. Brooks, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. Fault tolerant algorithms for network-on-chip interconnect. *IEEE Computer Society Annual Symposium on VLSI*, 0:46, 2004.

[44] V. Puente, J. A. Gregorio, F. Vallejo, and R. Beivide. Immunet: A cheap and robust fault-tolerant packet routing mechanism. *SIGARCH Computer Architecture News*, 32(2):198, 2004.

[45] A. Pullini, F. Angiolini, S. Murali, D. Atienza, G. De Micheli, and L. Benini. Bringing NoCs to 65 nm. *IEEE Micro*, 27(5):75–85, 2007.

[46] W. Qiao and L. M. Ni. Adaptive routing in irregular networks using cut-through switches. In *Proceedings of the International Conference on Parallel Processing*, pages 52–60, 1996.

[47] S. Rodrigo, J. Flich, J. Duato, and M. Hummel. Efficient unicast and multicast support for CMPs. In *MICRO 41: Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture*, pages 364–375, Washington, DC, USA, 2008. IEEE Computer Society.

[48] 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 *NOCS '10: Proceedings of the 4th ACM/IEEE International Symposium on Networks-on-Chip*, pages 25–32, 2010.

[49] S. Rodrigo, C. Hernández, J. Flich, F. Silla, J. Duato, S. Medardoni, D. Bertozzi, A. Mejía, and D. Dai. Yield-oriented evaluation methodology of network-on-chip routing implementations. In *SOC'09: Proceedings of the 11th International Conference on System-on-chip*, pages 100–105, Piscataway, NJ, USA, 2009. IEEE Press.

[50] S. Rodrigo, S. Medardoni, J. Flich, D. Bertozzi, and J. Duato. Efficient implementation of distributed routing algorithms for NoCs. *IET Computers and Digital Techniques*, 3(5):460–475, 2009.

[51] F. A. Samman, T. Hollstein, and M. Glesner. Multicast parallel pipeline router architecture for network-on-chip. In *DATE '08: Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1396–1401, New York, NY, USA, 2008. ACM.

[52] F. A. Samman, T. Hollstein, and M. Glesner. Planar adaptive router microarchitecture for tree-based multicast network-on-chip. In *NoCArc '08: International Workshop on Network on Chip Architectures*, 2008.

[53] J. Sancho, A. Robles, and J. Duato. A new methodology to compute deadlock-free routing tables for irregular networks. In Babak Falsafi and Mario Lauria, editors, *Network-Based Parallel Computing. Communication, Architecture, and Applications*, volume 1797 of *Lecture Notes in Computer Science*, pages 45–60. Springer Berlin/Heidelberg, 2000.

[54] J. C. Sancho, A. Robles, J. Flich, P. López, and J. Duato. Effective methodology for deadlock-free minimal routing in infiniband networks. In *ICPP '02: Proceedings of the 2002 International Conference on Parallel Processing*, page 409, Washington, DC, USA, 2002. IEEE Computer Society.

[55] M. D. Schroeder, A. D. Birrell, M. Burrows, H. Murray, R. M. Needham, T. L. Rodeheffer, E. H. Satterthwaite, and C. P. Thacker. Autonet: a high-speed, self-configuring local area network using point-to-point links. *IEEE Journal on Selected Areas in Communications*, 9, 1991.

[56] D. Seo, A. Ali, W. Lim, N. Rafique, and M. Thottethodi. Near-optimal worst-case throughput routing for two-dimensional mesh networks. *SIGARCH Computer Architecture News*, 33(2):432–443, 2005.

[57] D. Seo, A. Ali, W. Lim, N. Rafique, and M. Thottethodi. Near-optimal worst-case throughput routing for two-dimensional mesh networks. In *ISCA '05: Proceedings of the 32nd Annual International Symposium on Computer Architecture*, pages 432–443, Washington, DC, USA, 2005. IEEE Computer Society.

[58] L. Shang, L. Peh, A. Kumar, and N. K. Jha. Thermal modeling, characterization and management of on-chip networks. In *MICRO 37: Proceedings*

*of the 37th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 67–78, Washington, DC, USA, 2004. IEEE Computer Society.

[59] E. S. Shin, V.t J. Mooney III, and G. F. Riley. Round-robin arbiter design and generation. In *ISSS '02: Proceedings of the 15th International Symposium on System Synthesis*, pages 243–248, New York, NY, USA, 2002. ACM.

[60] F. Silla and J. Duato. Improving the efficiency of adaptive routing in networks with irregular topology. In *In Proceedings of the 1997 International Conference on High Performance computing*, 1997.

[61] T. Skeie, O. Lysne, and I. Theiss. Layered shortest path (LASH) routing in irregular system area networks. In *IPDPS '02: Proceedings of the 16th International Parallel and Distributed Processing Symposium*, page 194, Washington, DC, USA, 2002. IEEE Computer Society.

[62] T. Skeie, F. O. Sem-Jacobsen, S. Rodrigo, J. Flich, D. Bertozzi, and S. Medardoni. Flexible DOR routing for virtualization of multicore chips. In *SOC'09: Proceedings of the 11th International Conference on System-on-chip*, pages 73–76, Piscataway, NJ, USA, 2009. IEEE Press.

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

[64] T. R. Sullivan, H.and Bashkow. A large scale, homogeneous, fully distributed parallel machine, i. In *ISCA '77: Proceedings of the 4th Annual Symposium on Computer Architecture*, pages 105–117, New York, NY, USA, 1977. ACM.

[65] A. S. Tanenbaum. *Computer Networks*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2003.

[66] P. Teehan, M. Greenstreet, and G. Lemieux. A survey and taxonomy of GALS design styles. *IEEE Design and Test of Computers*, 24(5):418–428, 2007.

[67] B. Towles and W. J. Dally. Worst-case traffic for oblivious routing functions. In *SPAA '02: Proceedings of the fourteenth Annual ACM Symposium on Parallel algorithms and architectures*, pages 1–8, New York, NY, USA, 2002. ACM.

[68] J. Van Leeuwen and R. B. Tan. Interval Routing. *The Computer Journal*, 30(4):298–307, 1987.

[69] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar. An 80-tile 1.28TFLOPS network-on-chip in 65nm CMOS. In *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International*, pages 98–589, 2007.

[70] L. Wang, Y. Jin, H. Kim, and E. J. Kim. Recursive partitioning multicast: A bandwidth-efficient routing for networks-on-chip. In *NOCS '09: Proceedings of the 2009 3rd ACM/IEEE International Symposium on Networks-on-Chip*, pages 64–73, Washington, DC, USA, 2009. IEEE Computer Society.

[71] P. T. Wolkotte, G. J. M. Smit, G. K. Rauwerda, and L. T. Smit. An energy-efficient reconfigurable circuit-switched network-on-chip. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 3*, page 155.1, Washington, DC, USA, 2005. IEEE Computer Society.

[72] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: characterization and methodological considerations. In *ISCA '95: Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 24–36, New York, NY, USA, 1995. ACM.