LBDR: An efficient unicast routing support for CMPs

Author: Samuel Rodrigo Mocholí
Advisor: José Flich Cardo
email: srodrigo@gap.upv.es

December 12, 2008

Abstract

The roadmap for high-performance computing it is currently switching to multi-core architectures. Industry has shifted to the multi-core paradigm as single-core processors are reaching the power consumption wall. The solution is to put multiple and simple processors in the same chip fabric becoming chip multiprocessors (CMPs). An efficient interconnect layer for CMP architectures is needed to connect all the cores. Networks-on-chip (NoCs) are the key components of these architectures, and they have to deal with the communication scalability challenge while meeting tight power, area and latency design constraints.

2D mesh topologies are usually preferred by designers of general purpose NoCs. However, manufacturing faults may break their regularity. Moreover, resource management frameworks may require the segmentation of the network into irregular regions under virtualization or power-awareness scenarios. Under these conditions, efficient routing becomes a challenge. Although on the off-chip domain the use of routing tables at switches is flexible, in the on-chip domain it does not scale in terms of latency and area due to its memory requirements.

LBDR (Logic-Based Distributed Routing) is proposed as a new routing method that removes the need for routing tables at all. LBDR enables the implementation of many routing algorithms on most of the practical topologies we may find in the near future in a multi-core system. From an initial topology and routing algorithm, a set of three bits per switch/output port is computed. Evaluation results show that, by using a small logic, LBDR mimics the performance of routing algorithms when implemented with routing tables, both in regular and irregular topologies.

This work also explores LBDR implementation in a real NoC switch, proving its smooth integration in the architecture and its negligible hardware and performance overhead.

Contents

Chapter 1

Introduction

1.1 The New Era of CMPs

As the performance and power scalability of monolithic microprocessor cores is running into physical barriers, industry is shifting to multi-core architectures for designing high performance microprocessors. Major processor manufacturers, like Intel, offer processing devices with a small number of cores inside the chip, currently from two to eight cores per chip, and this is becoming mainstream for a variety of markets (desktop/server/embedded). But in the coming years, this trend is expected to keep giving birth processor architectures that manage a lot of cores inside the same chip (see Figure 1.1). As an example, the Teraflops research chip from Intel has recently been announced with 80 integrated cores [4].

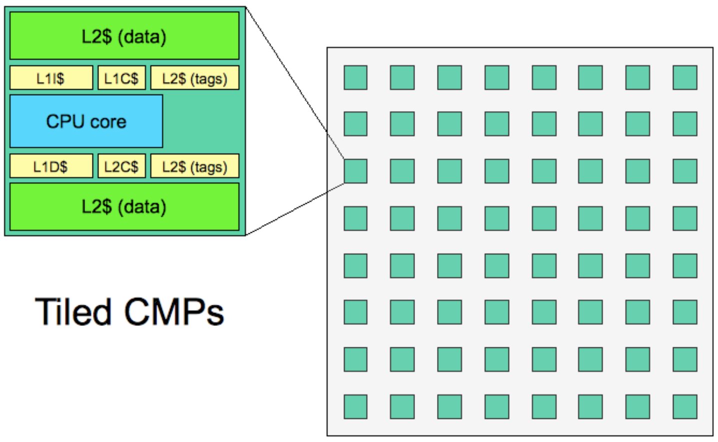


Figure 1.1: An example of many cores in the same chip, every tile has a core.

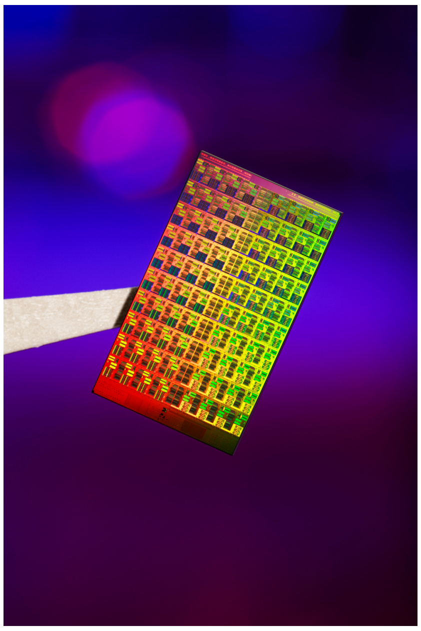


Figure 1.2: Intel's 80 core teraflops research chip magnified.

Chip architectures with such a large number of cores require a high-performance on-chip interconnect for efficient communication between cores and with cache blocks and/or memory controllers. Current chip implementations are based on bus or ring structures (e.g. the Cell multiprocessor [5] jointly developed by Sony Computer Entertainment, Toshiba, and IBM). However, as the number of cores increases, such interconnect fabrics become the bottleneck of the system, as they do not scale. For highly integrated systems, on-chip interconnects (Networks-on-Chip, NoCs) are likely to provide the needed bandwidth and meet the stringent latency requirements. See an example in Figure 1.3.

Besides the use of NoCs in chip multiprocessors (CMPs), there are other application domains that also benefit from a NoC. Clear examples are the TRIPS [6], RAW [7] and Wavescalar [8] architectures, relying on the concept of operand network. The on-chip network in this case is used to exchange operands between functional units. The TRIPS operand network leverages a 5-ary 2D mesh topology.

1.2 Context and Motivation

A lot of research has been undertaken on off-chip networks and so many mechanisms, techniques and methods can be applied to NoCs as well. How-

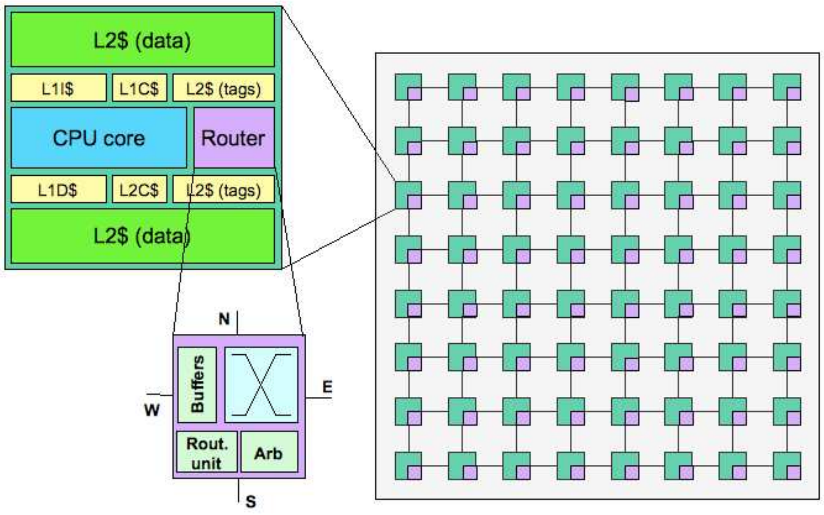


Figure 1.3: Interconnecting all the cores with a NoC.

ever, new physical constraints appear in NoCs, which are not a primary concern in the off-chip domain. In particular, NoC designers must not overlook area, power and latency requirements. To meet the design constraints, NoC designers, encourage the use of a particular scenario.

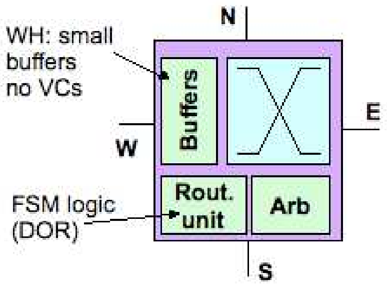


Figure 1.4: Current trends when designing a tile from a NoC.

A 2D mesh topology is usually preferred for the chip layout because of its regularity and matching with the 2D silicon surface. This is the case of the Teraflops research chip, that in addition to the compute element, each core contains a 5-port router. These are connected in a 2D mesh network. As seen in Figure 1.4 every router is connected to other routers with the *North*, *East*, *West* and *South* input/output ports (plus the local port that connects to the core and other devices inside the tile).

Network performance is traditionally measured in terms of packet latency and network throughput. Within the context of a NoC, ultra-low latencies are typically required, so every stage must be carefully performance-optimized. This is the reason why, when designing the routing layer, logic-based routing (e.g., the predominant Dimension-Order-Routing, DOR) is usually the preferred solution in addition to the wormhole switching technique, as wormhole requires minimum area for buffers.

So, a NoC designed with the features mentioned above is a low latency, area and power efficient solution. This would be enough on an ideal world but it is not suitable for the new challenges NoC designers must face.

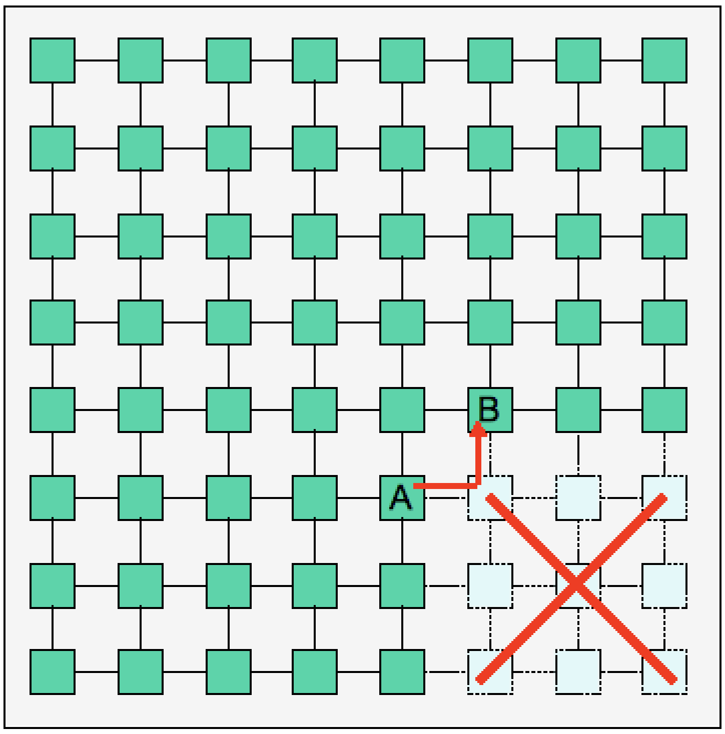


Figure 1.5: Irregular topology result of a failed block of nodes.

Fault-tolerance is becoming a major concern in NoCs and CMPs due to the physical mechanisms that make designing on nanoscale technologies a challenging task. While transient faults (reliability issues) like for example, crosstalk or power supply noise, may be addressed by means of physical or circuit level design techniques [1, 3], manufacturing imperfections resulting in defective IP cores, wires or switches may hamper operation of the whole system. In fact, although a regular NoC topology was selected at design time, defective components may make it irregular (see example in Figure 1.5). Since most of the chip area is still fully functional, it has to be ensured that the network is able to work also with the new irregular topology, which has implications at least on routing design. Yield would be affected if failure is not supported at the network routing level, and with DOR this is not possible.

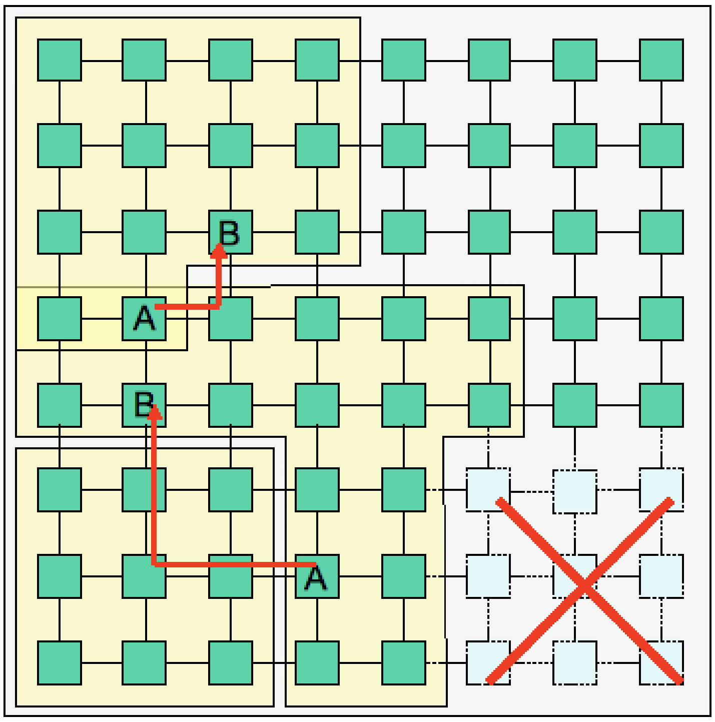


Figure 1.6: A mesh divided into irregular regions.

Besides process imperfections, there are other reasons why a regular network topology may have to be handled as highly irregular.

First, in order to exploit the large number of cores in MPSoCs and due to the fact that applications are not getting enough parallelism, virtualization of the chip is becoming a necessity. In a virtualized system, resources are distributed among different running tasks or applications. Although the virtualization concept is not new, its application to NoCs and CMPs is challenging. The network must guarantee traffic isolation within regions, thus eventually leading to irregular sub-networks within the original 2D mesh. Figure 1.6 shows an example where different regions are used. Again, routing design under such conditions is a challenging task and DOR is not suitable.

Second, power management is becoming an active design topic. Several studies state that around 30% to 40% of the total chip power is due to the interconnect layer so power-aware techniques become excellent solutions to meet this constraint. If at some point, cores and other devices are not needed,

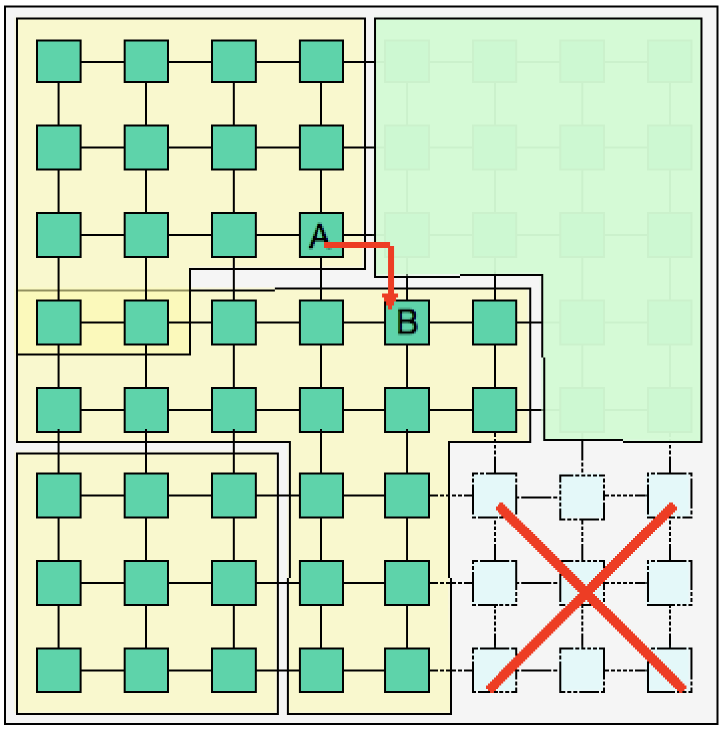


Figure 1.7: Powering down some devices inside the mesh.

they must be powered down. This eventual shutdown leads to irregular regions inside the network and this must be handled again by the network routing level (see an example in Figure 1.7).

In order to find a routing layer capable of handling these challenges with the resuling irregular topologies (or regions) there is a current solution, the use of routing tables. The routing layer can be implemented as source routing or distributed routing. In source routing, the source node grabs the path accessing the routing table matching the destination node and stores it in the packet header. Since the header itself must be transmitted through the network, it consumes network bandwidth. The Intel Teraflops research chip uses source routing.

In distributed routing, in contrast, each switch computes the output port for each input packet, and therefore the next hop to destination. For this purpose, the packet header needs only to contain the destination ID. This can be implemented in different ways. The approach commonly found in regular topologies is the so-called algorithmic routing, which relies on a combinational logic circuit that computes the output port to be used as a function of the current and destination nodes and the status of the output ports. The implementation is very efficient in terms of both area and speed, but the algorithm is specific to the topology and to the routing strategy used on that topology.

To deal with irregular topologies, switches based on forwarding tables were proposed. In this case, there is a table at each switch that stores, for each destination end-node, the output port that must be used. This scheme can be easily extended to support adaptive routing by storing several outputs in each table entry. The main advantage of table-based routing is that any topology and any routing algorithm can be used, including fault-tolerant routing algorithms. However, memories do not scale as much as logic in terms of latency, power and area, thus proving impractical for large NoCs. Imagine that every routing table in every chip has a size of $N-1$ entries and this becomes a bottleneck as the number of cores, N, grows. The size of the routing tables can be possibly reduced in some environments. This is the case of application-specific systems where the communication pattern may be known in advance. However, this is not the case for generic purpose multi-core chips.

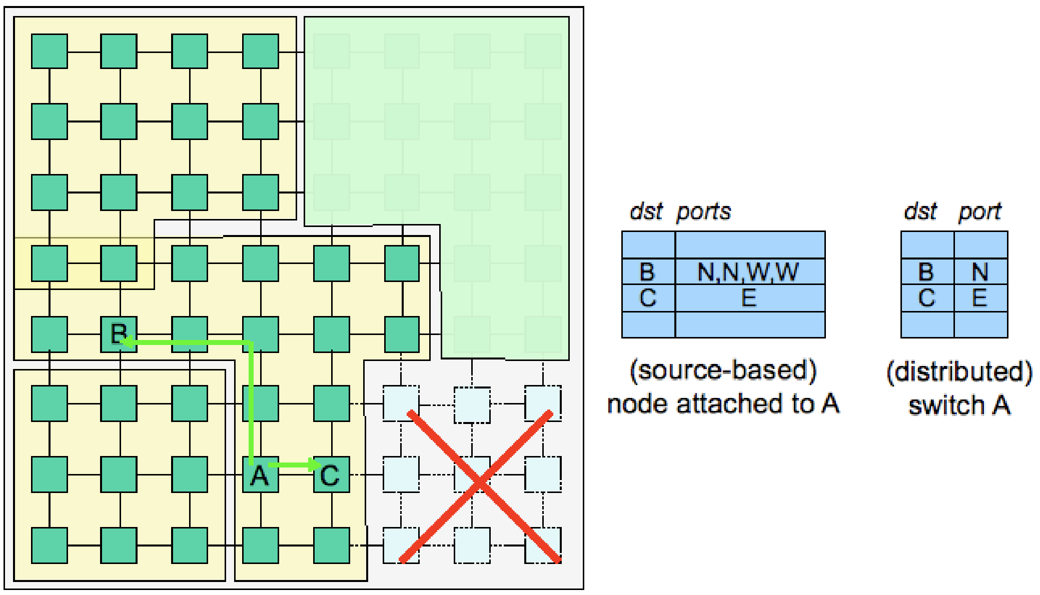


Figure 1.8: Example of routing tables at a certain node.

An example is shown in Figure 1.8. Here we can appreciate a parcial content of the routing tables on node A that store some routing decisions for the routing management layer, being either source or distributed routing. For source routing and destination B, the path $North, North, West, West$ will be stored on the packet header. For the same destination but with distributed routing, the switch at A will route the packet to the $North$ output port.

It would be interesting to find a routing implementation for irregular topologies (or partial 2D meshes) allowing the use of any distributed routing algorithm without the need for routing tables both at endnodes for source-based routing and at switches for distributed routing. In this work such challenge is taken.

The proposal, known as Logic-Based Distributed Routing (LBDR) is a very simple mechanism that removes the routing tables at every switch, enabling the distributed implementation of any routing algorithm on irregular topologies, with a FSM-based (low-latency, power/area efficient) unicast routing implementation. This method relies on 2 flags per output port for routing decisions, a flag per output port to define topology or region definition and connectivity and a small set of logic gates.

To prove the practical feasibility and the effectiveness of LBDR, the method has been implemented it in a realistic network-on-chip switch architecture natively supporting source-based routing. This architecture allows lightweight switch implementations and is, therefore, an ideal reference architecture to assess the implementation overhead of a switch augmented with LBDR capability. Synthesis results on a 65nm technology node point out a number of second-order effects that common assumptions on routing architecture implementations fail to capture. LBDR is enhanced and interfaced to a 2-phase switch arbiter scheme and also integrated into a high radix switch design where more than one end-node is connected to each switch.

The rest of this work is organized as follows. Chapter 2 gives a detailed view of the mechanism and its extensions. Chapter 3 deals out with an implementation of the mechanism on a real NoC switch architecture. On chapter 4 evaluations on performance and area and delay overhead results are presented. Chapter 5 provide citations to related work and finally, conclusions are drawn in chapter 6.

Chapter 2

Logic-Based Distributed Routing

In this chapter the proposed routing mechanism is described, at first illustrating the target system architecture and then delving into the implementation details. Then, an extension of the mechanism is illustrated and finally, a formal demostration is shown.

2.1 System Environment

For the sake of simplicity we focus on networks with no virtual channel requirements, and assume wormhole switching (although the proposed method also works for virtual cut-through switching as well). Messages (or packets in virtual cut-through) are routed with X and Y offsets assuming the X and Y coordinates of the final destination are included in the message header (X_{dst} and Y_{dst}), and each switch knows its X and Y coordinates (through the X_{curr} and Y_{curr} registers at each switch).

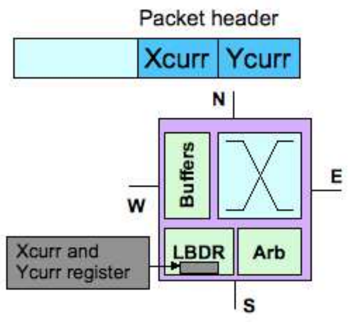


Figure 2.1: Packet header and switch with the coordinates.

Logic-Based Distributed Routing (LBDR) can be applied to a combination of topologies and routing algorithms with some particular characteristics. The following paragraphs describe the conditions topologies and routing algorithms must meet.

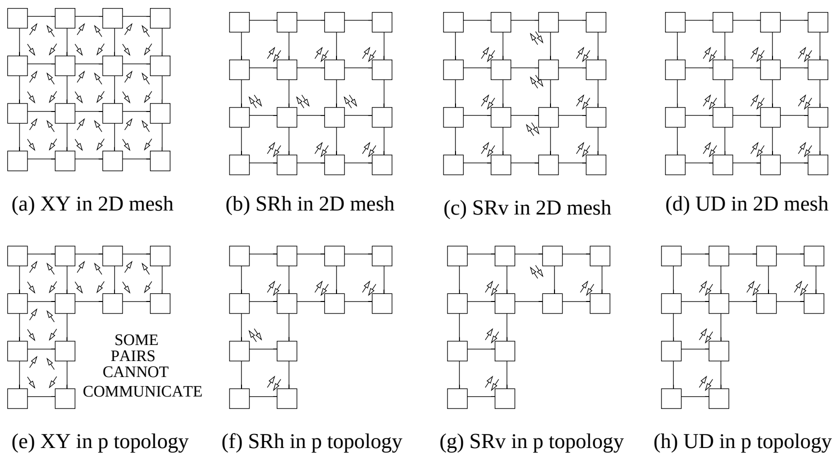


Figure 2.2: Examples of routing algorithms (by their routing restrictions) in 2D mesh and p-shaped irregular topologies.

As stated in Section 1.2 the typical topology of choice for NoCs is the 2D mesh network. However, with the advances of technology and with the appearance of new challenges topologies derive from an initial 2D mesh into a subset of irregular topologies. Topologies shown in Figure 2.2 are examples. It should be noted that all the described irregular topologies share the same property: all the end-nodes (assuming at least one end-node attached to each switch) can communicate with the rest of nodes through any minimal path defined in the original mesh topology. LBDR can be applied to all the topologies that fulfill this property (2D mesh and p-shaped topologies shown on Figure 2.2 would be valid). LBDR is, however, not applicable to topologies where some pairs of end-nodes cannot communicate through a minimal path defined in the original 2-D mesh topology, like in Figure 2.3.

A deterministic (or partially adaptive) routing algorithm without cyclic dependencies among links or buffers can be represented by the set of routing restrictions it imposes. As an example, Figure 2.4 shows the routing restrictions defined by the UD routing algorithms on a 2-D mesh topology. Each arrow indicates a routing restriction. Basically, a routing restriction forbids any packet to use two consecutive channels. So, the final paths for each pair of communicating end-nodes will not pass through any routing restriction. In this work we define a routing restriction as the pair of channels that can not be used in sequence by any packet. For instance, at the last (bottom

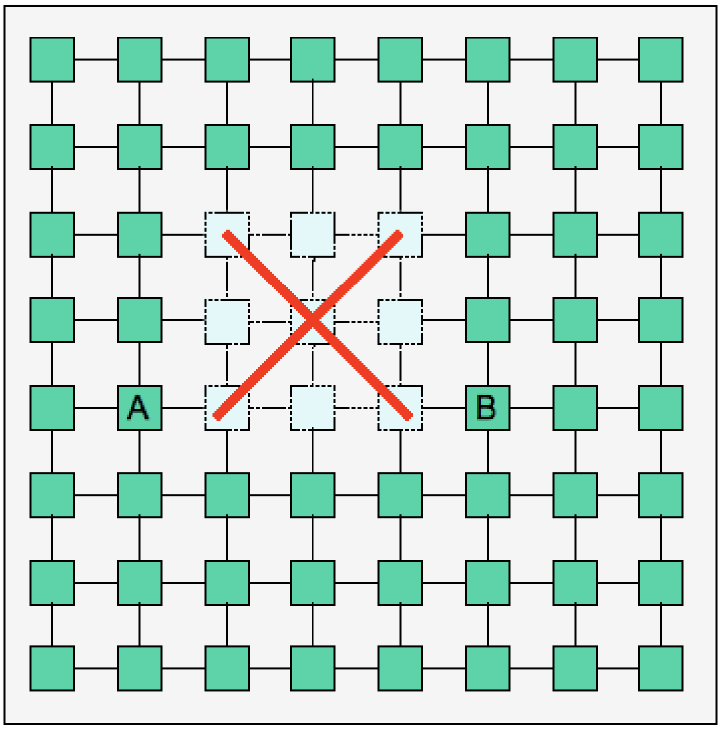


Figure 2.3: Example of topology with non-minimal path between switch A and B.

right-most) switch in Figure 2.4 there are two restrictions: NW and WN[1]. A packet traversing that switch can not use consecutively the N and W channels or viceversa.

In short, LBDR is applicable to any routing algorithm that complies with the following condition: defining the following two sets of channels {N,S} and {E,W}, all the routing restrictions are formed by two channels, each one from a different set. Thus, for instance, restriction EW is not allowed. In other words, routing restrictions forbid only some changes in the direction. Notice that this makes sense since it allows for minimal routing (restrictions like WE, EW, NS, SN, SS, $NN\ldots$ always force the need for non-minimal paths). Notice that for example, the UD routing algorithm pictured at Figure 2.4 follows this requirement, since all the restrictions are WN or NW.

Other routing algorithms like FX [15] and Turn Model [16] also adhere to these conditions, thus they can be implemented with LBDR.

In the case of XY (DOR), the routing algorithm is designed only for the 2D mesh topology and it is unsuitable for non-regular topologies as showed

[1]Channels are labelled N (North), E (East), W (West), and S (South).

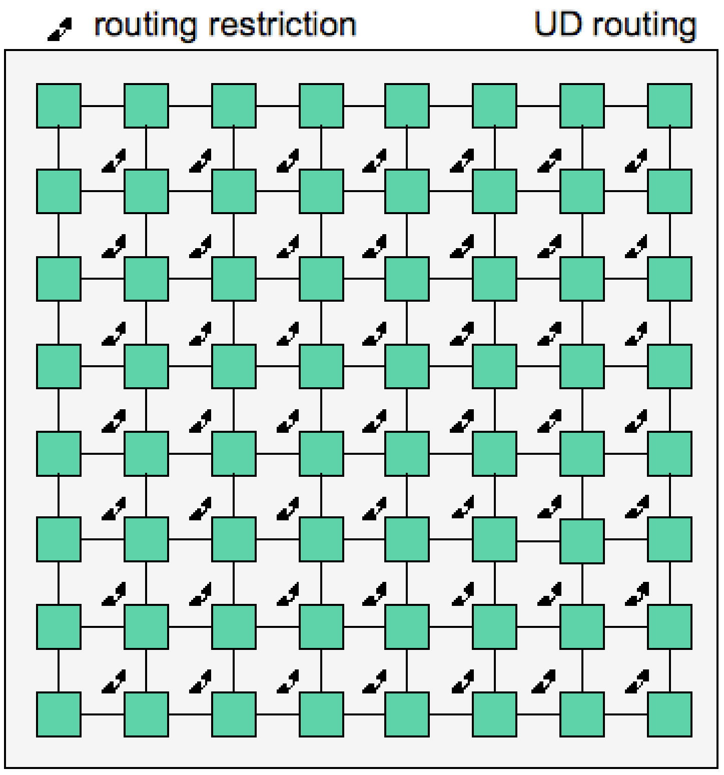


Figure 2.4: Example of a routing algorithm represented by its restrictions.

in Figure 2.2.e because it does not enforce minimal paths in those topologies, but is still supported by LBDR with the full 2D mesh.

There is a very important note that must be remarked. LBDR is a routing layer implementation, not a single routing algorithm description. LBDR works in conjunction and supports the majority of topology-agnostic routing algorithms, like SR_h [12], SR_v [12], and UD (up*/down*) [13]. These routing algorithms must enforce dead-lock freedom and connectivity.

2.2 LBDR Description

Logic-Based Distributed Routing (LBDR) relies on a simple and efficient implementation. Each switch has two sets of bits/flags: routing bits and connectivity bits. Routing bits indicate which routing options can be taken, whereas connectivity bits indicate whether a switch is connected with its neighbours. The value of these bits depends on the topology and the routing algorithm being implemented, and are computed and uploaded to the switches before normal operation (at system boot) or are changed during reconfiguration processes.

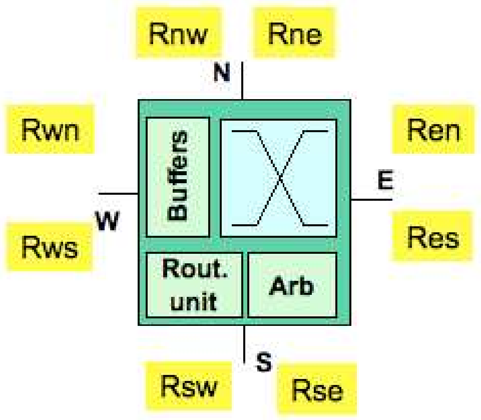


Figure 2.5: Routing bits in a switch.

Routing bits indicate if a change of direction can be made at the next switch (one-hop visibility). Each output port has two routing bits and are referred as R_{xy}. If a R_{xy} bit is set indicates that a packet routed through X port is allowed to go through Y output port at the next switch. For example, at the E output port these bits are labelled R_{en} and R_{es}. They indicate whether packets routed through the E output port may take the N port or S port at the next switch, respectively. For output port N the bits are accordingly labelled R_{ne} and R_{nw}, for output port W R_{wn} and R_{ws}, and for output port S R_{se} and R_{sw} as seen in Figure 2.5.

In Figure 2.6 we can see an example of routing bits in a switch. In this case, R_{ne}, R_{nw}, R_{es}, R_{wn}, R_{ws} and R_{se} define valid directions changes for routing decisions. Instead, R_{en} and R_{sw} would be set to zero, as they represent not valid direction changes at next switches due to routing restrictions.

Regarding the connectivity bits, each output port has a bit, referred to as C_x indicating whether a switch is connected through the x port. Thus, connectivity bits are C_n, C_e, C_w, and C_s (see Figure 2.7). With this set of bits, LBDR offers region definition and traffic isolation.

Figure 2.8 shows an example of all the bits computed for every switch for an irregular topology when using the SR_h [12] routing algorithm.

For the full potential use of the routing and connectivity bits, the switch implements a routing logic, that is divided in two blocks as shown in Figure 2.9. The first block computes the relative position of the packet's destination. For this, two comparators are used and X_{curr} and Y_{curr} are compared with X_{dst} and Y_{dst}. At the output of this logic one or two signals may be active

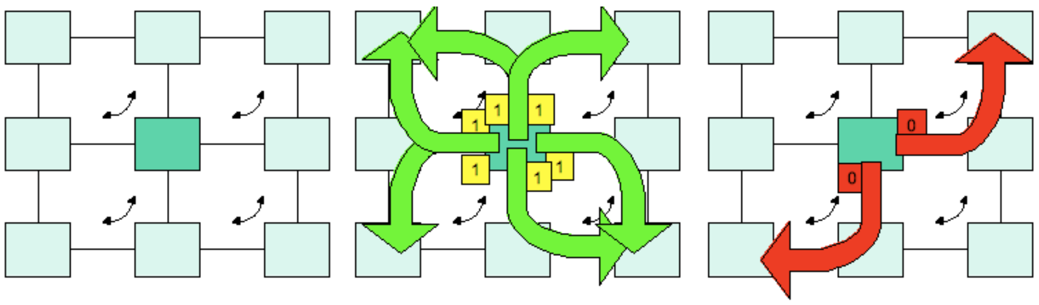


Figure 2.6: An example of routing bits set according to the routing algorithm restrictions.

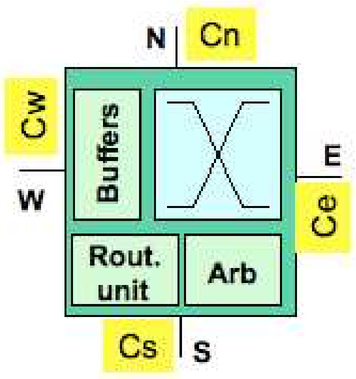


Figure 2.7: Connectivity bits in a switch.

(e.g. if the packet has its destination node in the NW quadrant then N' and W' signals are active). Note also that packets forwarded to the local port are excluded from the routing logic as in this case both current and destination coordinates would be the same.

Once the N', E', W', and S' signals are computed, the second block of the logic comes into play. It consists of four logic units, one for each output port. Each one can be implemented with only two inverters, four AND gates and one OR 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, then signal N'' is computed as a result of operation of these conditions (in Figure 2.9 $X1$, $X2$ and $X3$):

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

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

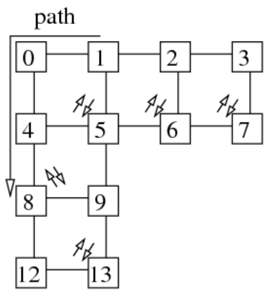


Figure 2.8: Example of routing and connectivity bits with the SR_h routing algorithm.

- The packet's destination is on the NW quadrant and the packet can take the W port at the next switch through the N port ($N' \times W' \times R_{nw}$).

Finally, the connectivity bit C_n is inspected in order to filter the N port with the computed signal N''. If none of the conditions is met, then the N port can not be considered for routing the packet.

Notice that, for example, N and E signals could be active at the same time. In this case, the switch has to choose among them in the arbiter unit, according whether adaptiveness is allowed or the routing algorithm is deterministic.

With the implemented logic and connectivity/routing bits LBDR mimics the behaviour of routing algorithms when using routing tables. As an example, Figure 2.10 shows the path taken from source A to destination B over the NoC. As seen, B is on the NE quadrant from A. At switch A, the first part of the logic will give N and E output ports as possible routing decisions. As we have connectivity to both directions and the routing bits will show there are no routing restrictions on the next direction changes, both N and E are marked as valid. Imagine the switch arbiter chooses N. At the next switch, current and destination coordinates are again compared. As B is still in the NE quadrant, N and E output ports will be marked again as possible routing decisions. But this time, N will not be valid as we have a routing restriction at the switch located to the north, so R_{ne} at this switch will be set to zero, invalidating N. So, E output port, is chosen to route the packet.

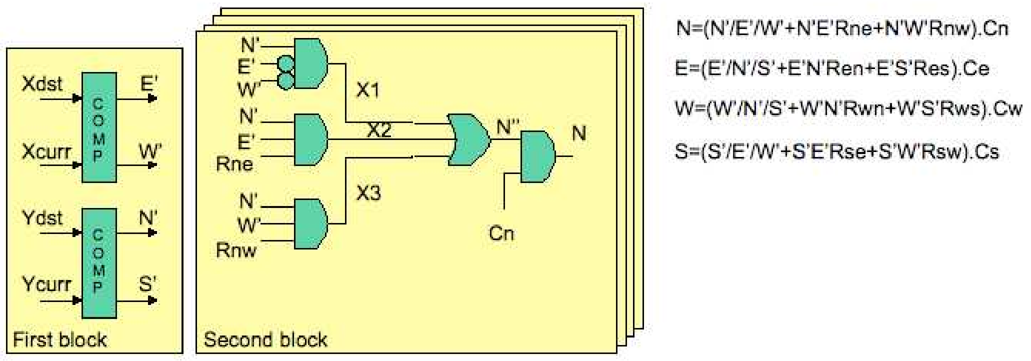

Xdst
Xcurr
COMP
E'
W'
Ydst
Ycurr
COMP
N'
S'
First block
N'
E'
W'
N'
E'
Rne
N'
W'
Rnw
X1
X2
X3
N''
N
Cn
Second block
N=(N'/E'/W'+N'E'Rne+N'W'Rnw).Cn
E=(E'/N'/S'+E'N'Ren+E'S'Res).Ce
W=(W'/N'/S'+W'N'Rwn+W'S'Rws).Cw
S=(S'/E'/W'+S'E'Rse+S'W'Rsw).Cs

Figure 2.9: A full detail on LBDR logic.

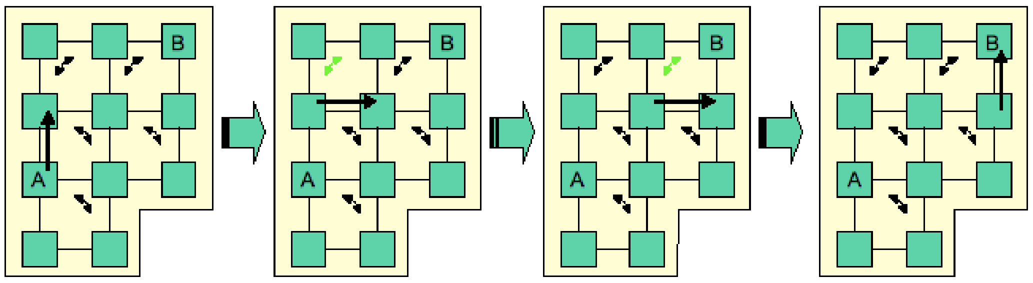


Figure 2.10: An example of routing path taken by a packet with LBDR.

At the next switch we will encounter the same scenario, so again the packet is routed through E output port. Finally, at the next switch, N would be given as is the only valid routing option to reach switch B.

It is important to note that only the bits referring to a routing restriction are set to zero and the remaining ones are set to one, even those that refer to switches not existing in the topology in order the mechanism to work properly. This can be better seen through an example. Imagine the path at Figure 2.8 from switch 13 to switch 7. At switch 13 the signals N' and E' are active because switch 7 is on the NE quadrant (from the point of view of switch 13). In particular, N'' is activated as R_{ne} at switch 13 is set to one, although it does not make sense for routing purposes. However, this allows the packet to being forwarded north, until it reaches switch 5, where it can take the east direction. Notice that output port E will never be taken at switches 13 and 9 due to the connectivity bit C_e set to zero.

2.3 LBDRe: Extended Visibility on Routing Decisions

While performing evaluations there were situations where LBDR showed a performance degradation with the SR_h routing algorithm. An explanation of the behaviour can be seen as an example in Figure 2.8. Now, a packet must be forwarded from switch 1 to switch 8. In this case switch 1 decides to discard output port S because its R_{sw} bit is set to zero due an existing NW restriction at switch 5 through the S output port of switch 1 so the resulting path is 1-0-4-8. However, in this case, a valid path would also be 1-5-9-8. Therefore, LBDR reduces adaptiveness while being conservative. Although LBDR is still working (it always provides a valid set of paths) the performance degradation could be unacceptable (see Section 4.1).

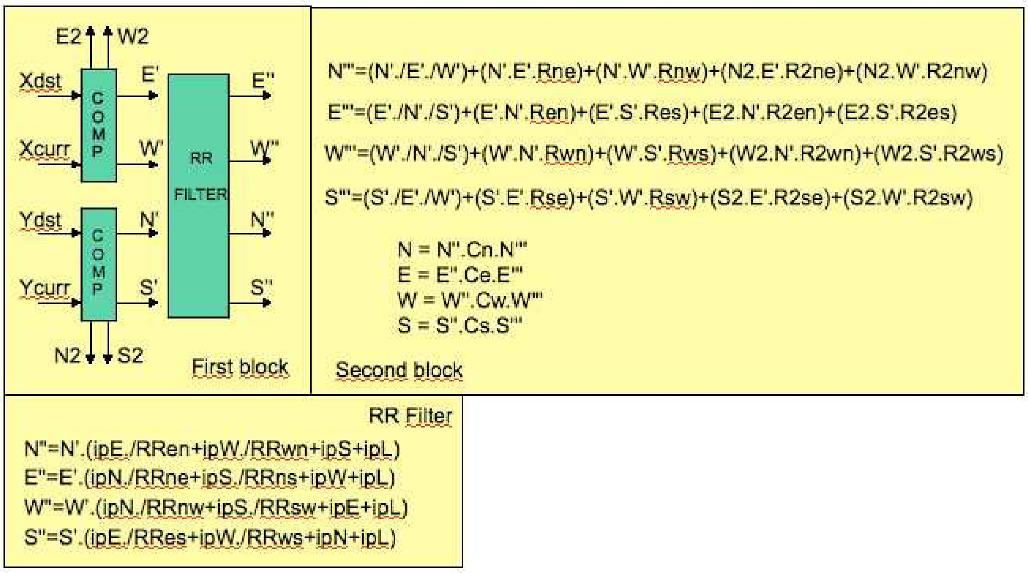

E2
W2
Xdst
Xcurr
COMP
E'
W'
Ydst
Ycurr
COMP
N'
S'
RR FILTER
E''
W''
N''
S''
N2
S2
First block
N'''=(N'./E'./W')+(N'.E'.Rne)+(N'.W'.Rnw)+(N2.E'.R2ne)+(N2.W'.R2nw)
E'''=(E'./N'./S')+(E'.N'.Ren)+(E'.S'.Res)+(E2.N'.R2en)+(E2.S'.R2es)
W'''=(W'./N'./S')+(W'.N'.Rwn)+(W'.S'.Rws)+(W2.N'.R2wn)+(W2.S'.R2ws)
S'''=(S'./E'./W')+(S'.E'.Rse)+(S'.W'.Rsw)+(S2.E'.R2se)+(S2.W'.R2sw)
N = N''.Cn.N'''
E = E''.Ce.E'''
W = W''.Cw.W'''
S = S''.Cs.S'''
Second block
RR Filter
N''=N'.(ipE./RRen+ipW./RRwn+ipS+ipL)
E''=E'.(ipN./RRne+ipS./RRns+ipW+ipL)
W''=W'.(ipN./RRnw+ipS./RRsw+ipE+ipL)
S''=S'.(ipE./RRes+ipW./RRws+ipN+ipL)

Figure 2.11: LBDRe full logic detailed.

In order to fix this, LBDRe comes as an extension to LBDR mechanism to overcome this problem. Figure 2.11 describes this extended LBDR method. As can be seen, the routing and connectivity bits (3 bits per output port) are still maintained, and they are computed in the same way. Four new bits per switch output port are, however, added. Figure 2.12 shows an example of the computed LBDRe bits explained in the next paragraphs for an irregular topology.

The bits labelled $R2_{xy}$ indicate whether the y direction can be taken two hops away from the current switch through the x direction. For example,

Switch	Rne	Rnw	R2ne	R2nw	Ren	Res	R2en	R2es	Rwn	Rws	R2wn	R2ws	Rse	Rsw	R2se	R2sw	RRne	RRnw	RRen	RRes	RRwn	RRws	RRse	RRsw	Cn	Ce	Cw	Cs
0	1	1	0	0	1	1	0	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1
1	1	1	0	0	1	1	0	1	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	1	1	1
2	1	1	0	0	1	1	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
3	1	1	0	0	1	1	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
4	1	1	0	0	0	1	0	0	1	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	1	0	1
5	1	1	0	0	0	1	0	0	1	1	0	0	1	1	0	0	0	1	0	0	1	0	0	0	1	1	1	1
6	1	1	0	0	0	1	0	0	1	1	1	1	1	1	0	0	0	1	0	0	1	0	0	0	1	1	1	0
7	1	1	0	0	1	1	0	0	1	1	1	1	1	1	0	0	0	1	0	0	1	0	0	0	1	0	1	0
8	1	1	1	0	1	1	0	0	1	1	0	0	1	1	0	0	1	0	1	0	0	0	0	0	1	1	0	1
9	1	1	1	1	1	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1
10																												
11																												
12	1	1	1	0	0	1	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0
13	1	1	1	1	1	1	0	0	1	1	0	0	1	1	0	0	0	1	0	0	1	0	0	0	1	0	1	0
14																												
15																												

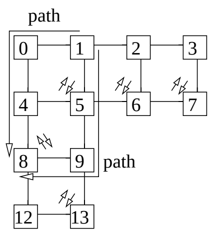


Figure 2.12: Example of computed bits for LBDRe with the SR_h routing algorithm.

$R2_{ne}$ indicates whether a packet is allowed to change direction to E at the switch 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 switch of the routing possibilities two hops away. However, it must be stated that these bits must be not active if they refer to a non-existing switch. For instance, in Figure 2.12 the $R2_{nw}$ bit at switch 4 is not active.

The bits labelled RR_{xy} indicate whether there is a routing restriction between x and y channels at the current switch. 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 switch needs five internal signals ipN, ipE, ipW, ipS and ipL to indicate the incoming port of the packet 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 switch and the packet's destination, 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 active if the packet's destination is at least two hops away in the corresponding direction (if $N2$ is active, then at least two hops must be done in the N direction to get closer to packet's destination). Notice 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 packet's destination is on the same column ($N' \times \overline{E'} \times \overline{W'}$).
- The packet's destination is on the NE quadrant and the packet can take the E port at the next switch through the N port ($N' \times E' \times R_{ne}$).
- The packet's destination is on the NW quadrant and the packet can take the W port at the next switch through the N port ($N' \times W' \times R_{nw}$).
- The packet's destination is on the NE quadrant, the packet's destination is at least two hops away through the N port, and the packet can take the E port at the two-hops neighbour switch through the N port ($N2 \times E' \times R2_{ne}$).
- The packet's destination is on the NW quadrant, the packet's destination is at least two hops away through the N port, and the packet can take the W port at the two-hops neighbour switch 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.

With LBDRe extension to LBDR mechanism, now, the SR_h routing algorithm can be applied with no performance degradation. Figure 2.12 shows two possible paths from source 1 to destination 8. At switch 1, the S output port can now be taken because the $R2_{sw}$ bit is active and the internal $S2$ signal will be activated. Note also that switch 5 has its RR_{nw} bit active, thus avoiding taking the W output port at the current switch, which would lead to an invalid path.

2.4 Deadlock-freedom and Connectivity

In this Section we demonstrate that LBDR is deadlock-free and provides connectivity among all the end-nodes. It must be noted that this can also be applied to LBDRe. As it has been shown before, LBDRe embeds LBDR and therefore it inherits all of its properties.

2.4.1 Deadlock-freedom

LBDR is not restricted to any particular routing algorithm. Instead, it can support any routing algorithm that provides minimal paths for every pair of end-nodes (as we see in Figure 2.2.e XY is a bad choice as it does not provide connectivity in an irregular topology). However, the applied routing algorithm must ensure deadlock-freedom and LBDR has to maintain such property. LBDR computes the routing bits from the routing restrictions defined by the routing algorithm. The algorithm is deadlock free if no packet crosses a forbidden routing restriction. Therefore, LBDR must ensure that no packet crosses any routing restriction defined by the routing algorithm.

Imagine there is a deadlock in the network induced by a set of packets that are requesting buffers in a cyclic manner. In that situation a packet in a switch sw along the cycle is mapped at a buffer in an input port i and is requesting an output port o for which a routing restriction is defined between i and o. Without lose of generality, consider the input port is S and the output port being requested is W. Hence, a SW routing restriction is defined at switch sw.

As routing restrictions are assigned only to links between switches (links connecting end-nodes are excluded), the given packet has previously been forwarded from a previous switch (sw_p). The output port used to forward the packet at sw_p is N. At this switch the routing bit R_{nw} is set to zero (since there is a SW routing restriction at switch sw). Additionally when routing the packet at switch sw_p the signals N' and W' were active as the packet is now requesting output port W at switch sw. Looking at the LBDR logic for output port N, at switch sw_p the N port can not be selected since none of the outputs of the AND gates will be active (x_1=x_2=x_3=0, see Figure 2.9). Indeed, the packet is in the NW direction and the Rnw bit is not active. Therefore, this situation can not be induced and thus LBDR is deadlock-free.

Similar conclusions can be obtained when assuming different sets of forbidden routing restrictions.

2.4.2 Connectivity

To demonstrate that the mechanism provides connectivity we must first highlight that the routing algorithm implemented by LBDR provides minimal paths and connectivity among all the pairs of end-nodes.

Notice that on each hop a packet performs in the network it gets closer to its destination. From the LBDR logic we can also deduce that non-minimal paths are avoided. Each output port is candidate for being selected only if the destination's distance is reduced along that port. For instance, the N port is eligible only if the packet is in the north direction or in the NW or NE quadrants (the signal N' is active).

Consider the case that a pair of end-nodes can not communicate when using LBDR. In this case, although the routing algorithm provides at least one minimal path to reach the destination end-node the LBDR mechanism fails to provide such path. In that situation, there is a point in the network where either LBDR logic provides a non-minimal path or any of the minimal paths are not eligible.

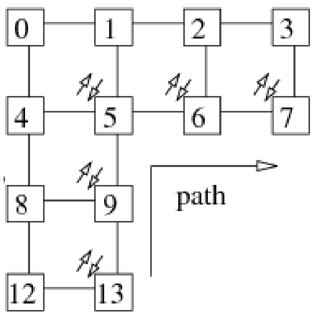


Figure 2.13: Path taken on boundaries at (p) topology .

As an example, consider Figure 2.13 and the routing unit at switch 8. The packet's destination is switch 3, thus is in the NE quadrant. In that situation the W and S ports are not considered by LBDR since the W' and S' signals are not active. In other words, the packet's destination is neither in the W' nor S' directions. Thus, LBDR avoids non-minimal paths. Therefore, only the N and E directions may be considered. In this case N' and E' signals are activated. In that situation, notice that the N port is eligible only if the R_{ne} bit is active and the E port is eligible only if the R_{en} bit is active. Notice that both bits can not be zero at the same time. In that case there would be no connectivity between switches 8 and 5 and thus, the routing algorithm implemented would not guarantee connectivity. As this situation is not assumed by the routing algorithm, it can not happen, and therefore LBDR guarantees connectivity. Therefore, at least one output port (N or E) will be eligible for routing the packet, getting closer to its destination.

However, a subtle case arises in the boundaries of the topology. Figure 2.13 shows a p topology and the packet at switch 13 has its destination at the NE quadrant. Switch 13, however, is at the boundaries of the topology. In this case, switch 13 has its R_{en} and R_{ne} bits active. However, its connectivity bit C_e is not active. In this situation the N port is eligible. Notice that the packet will go north until it reaches switch 5 where it will take either N or E.

Chapter 3

Implementation on a Real NoC Switch architecture

As a result of collaborative work in an internship (at University of Ferrara, Italy), the opportunity to see the impact of LBDR mechanism in a real NoC switch architecture was given.

3.1 XpipesLite Switch Architecture

LBDR has been implemented in the xpipesLite [23] switch architecture, illustrated in Figure 3.1. The switching fabric implements a 2-cycle-latency (one for switch operation and one for traversing the output link), output-queued wormhole-switched router supporting round-robin arbitration on the output ports. The input ports are latched to break the timing path. Allocation of inputs towards specific output ports is handled by an allocator module for each output port (in practice, this is a two phase arbiter). Arbitration is subsequently performed upon receipt of a header flit and output ports are granted until a tail flit arrives. Since the switching fabric natively supports source-based routing, the routing information is attached to the header flit by the network interface, which checks the address against a lookup routing table on the source node.

The length of the routing path field in the header depends on maximum switch radix and maximum hop count in the specific network instance at hand. The switch in Figure 3.1 is an interesting reference architecture to assess LBDR complexity, since most of the complexity of the routing architecture is on burden of the network interface in source-based routing and therefore the switch exhibits minimum complexity. The switch just has to read the target output port from the packet head and to route the entire

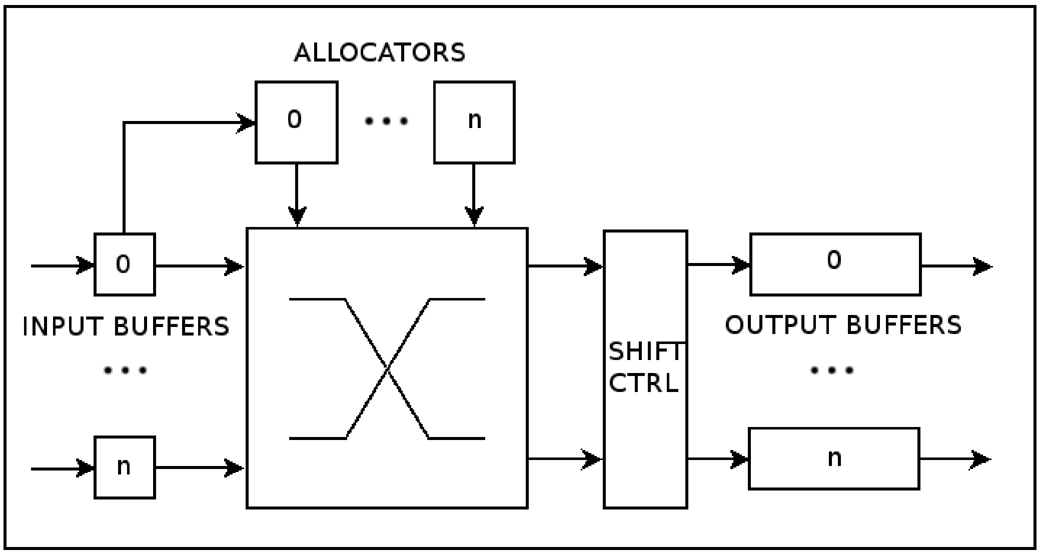


Figure 3.1: XpipesLite switch architecture.

packet accordingly. As an optimization, the xpipesLite switch architecture rotates away the routing information pertaining the current switch in the head flit. This allows positioning of the per-hop routing bits at a fixed offset within the head flits of the packets, thus simplifying switch implementation.

The implementation of LBDR inside the xpipesLite switch architecture is illustrated in Figure 3.2. In the original switch, the allocator checks the target output port from the head flit of the packet and compares it with its own output port ID, thus generating a *match* signal in case of correspondence. In the modified switch variant, this task is offloaded to the allocator since it is on burden of LBDR logic. This time the head flit contains the destination switch coordinates and not routing bits any more. However, the packet length is not increased since in any case the reference architecture already placed this kind of information in the header flit of the packet. LBDR logic is a two stage logic, as explained in Section 2.2, which is illustrated as a single box for each input port in Figure 3.2. It is interesting to observe that LBDR keeps the modular design style of the switch architecture. The output signals from the LBDR modules represent exactly the *match* signals that the allocator used before and which indicate that the packet from a given input port requires a specific output port. The allocator now has to discriminate between competing requests. LBDR logic therefore fits nicely into the XpipesLite switch architecture. In addition, the used two-phase allocator requires input LBDR modules to activate only one routing option each. As an example, for a given input packet the LBDR mechanism can

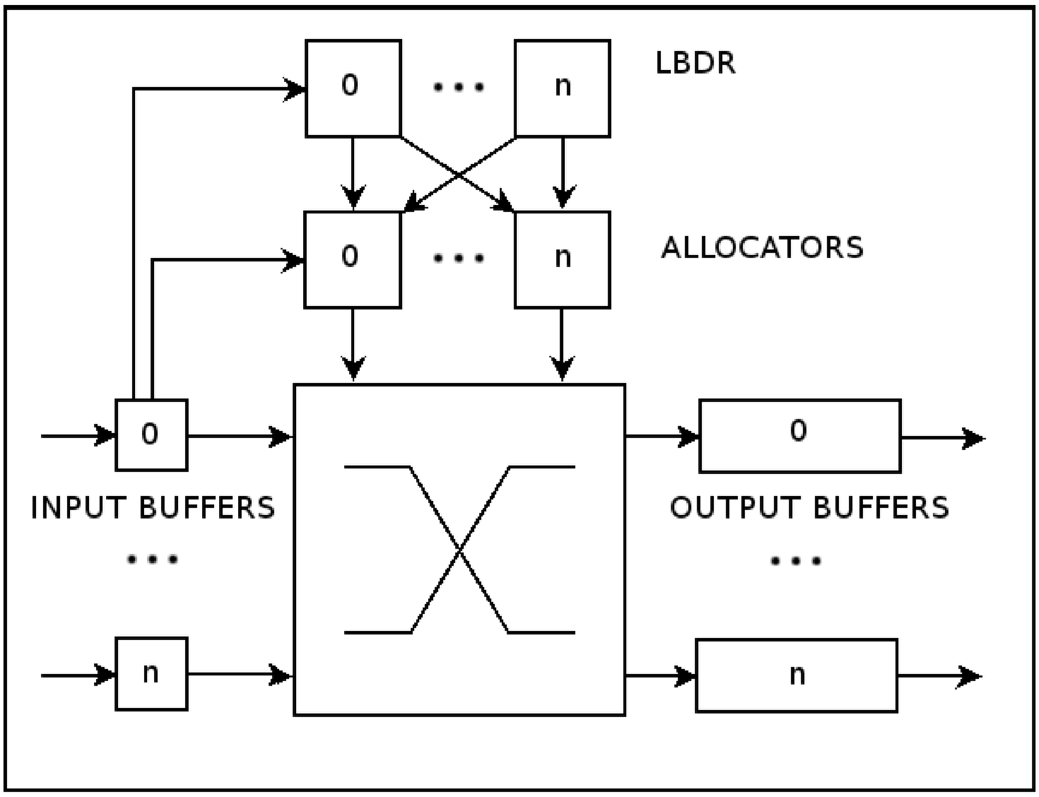


Figure 3.2: XpipesLite switch architecture modified with LBDR implementation.

not provide two valid output ports (for instance N and E). In that case, LBDR should provide only one (either N or E). To solve this problem, a fixed priority scheme described in the next Section is implemented.

3.2 Fixed Priorities and Allocators

LBDR provides in some cases more than one routing option depending on the routing bits. For a packet being routed north-east the logic may provide both output ports N and E as eligible for packet forwarding. It is the responsibility of the switch allocator to select one of the output ports. This leads to an increase in the routing flexibility and a potential increase in network performance.

However, in some NoC designs the switch allocator may be simple and does not allow multiple routing options from the same input port. This is the case of switches designed with a two-phase arbiter where allocators are implemented only at the output boundaries of the switch. In this Section, we extend the LBDR mechanism in order to allow its use on such switch designs. We provide a basic modification to the logic in order to provide only one routing option per input port. This extension, referred to as fixed priorities, relies on each input port and 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 switch.

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 packets 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. So priorities are provided higher 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 3.3 shows the extended LBDR logic providing fixed priorities to the N port.

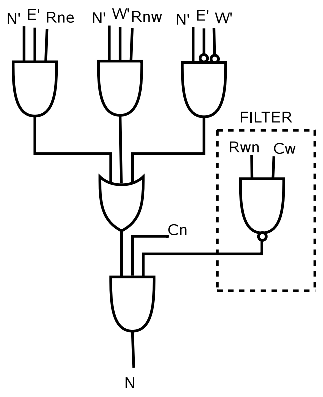


Figure 3.3: Fixed priorities on north output port.

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.

3.3 Extension To More Cores Per Switch

LBDR natively supports a large range of routing algorithms for k-ary 2-mesh topologies. Although widely used, it is well known that this topology scales poorly with the number of nodes and that it incurs a large area and power overhead. Under certain operating conditions, concentrated topologies become attractive [22]. Basically, the idea consists of reducing the number of topology dimensions or (as in our case) of switches in each dimension of a k-ary n-mesh and to increase the number of cores attached to each switch. This way, bisection bandwidth is traded for low latency, area and power.

LBDR was extended to support multiple cores per switch in concentrated mesh topologies. For this purpose, a different labelling scheme for network nodes had to be devised, since LBDR originally required the switch coordinates within the 2D mesh. Now, multiple cores might be associated with the same switch coordinates. The basic idea is that (see Figure 3.4) one local core inherits the same coordinates of the switch, while the other ones have an incremental x coordinate. From a network viewpoint, x coordinates of the switches appear to increase at a coarse granularity, where the granularity is determined by the number of cores attached to each switch. The figure illustrates the case with 4 cores per switch.

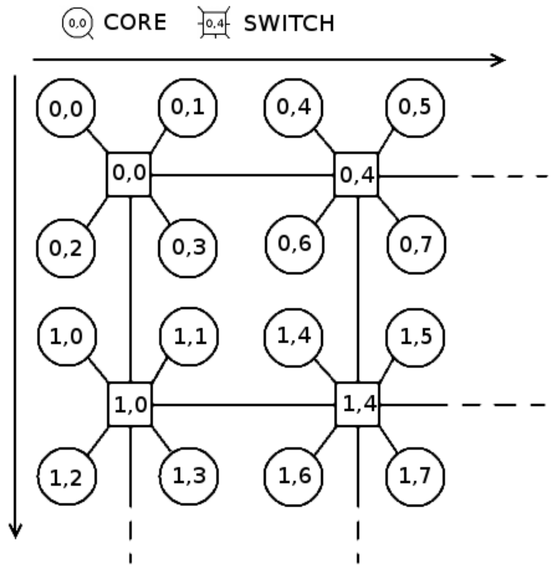


Figure 3.4: LBDR oriented node labelling in concentrated k-ary n-mesh topologies.

This implementation has two advantages: (i) the packet header does not need to be changed since it still carries the destination switch coordinates;

(ii) only the first logic stage of LBDR needs to be slightly extended, while the second one is unaltered.

Modification of the first logic stage of LBDR includes the return of a match signal whenever the target y switch coordinate matches that of the local switch AND the target x coordinate falls within the range of the x coordinates of the local connected cores. In this case, the packet is forwarded to the right local output port. Viceversa, if the packet is headed to another switch, the native LBDR logic can handle this without any modification.

Chapter 4

Evaluations and Results

In this chapter a comprehensive evaluation of LBDR is provided. First, a high-level analysis shows performance achieved by LBDR when compared with competing routing algorithms/mechanisms. Second, synthesis results are provided for the switch architecture described in Section 3 with and without LBDR, pointing out area and critical path delay overheads and accompanied with results in switch architectures for concentrated topologies connecting more cores per switch (Section 3.3). Finally, LBDR is compared with routing tables from an implementation cost and efficiency viewpoint, with a glance at scalability properties.

4.1 Performance

In this section performance achieved by LBDR is evaluated when applied to different topologies/routing algorithms compared with the performance achieved by those routing algorithms when implemented with routing tables. The objective is to check if LBDR and LBDRe get equal performance results compared with routing tables and by how much (and in which circumstances) they lose performance.

NoC simulator Noxim [17] has been used to evaluate LBDR, LBDRe, and table-based routing. In all simulations wormhole switching is assumed, input port buffers are 4-flit deep, and packets are 32-flit long. Flit size is set to one byte. For the transient state, 40K messages are assumed and results are collected after 40K messages are received. XY, UD, and SR_h routing algorithms have been evaluated in an 8×8 mesh and p-shaped topologies (an 8×8 mesh without the bottom-right 4×4 sub-mesh) with uniform traffic. The same overall conclusions have been obtained for all the routing/topology combinations and with uniform, bit-reversal and hot-spot traffic distributions. It has to be noted also that the aim of this evaluation is to check if LBDR (and LBDRe) replaces routing tables with no impact on performance. Achieving a better network throughput depends on the routing algorithm used and not on the way the routing algorithm is implemented.

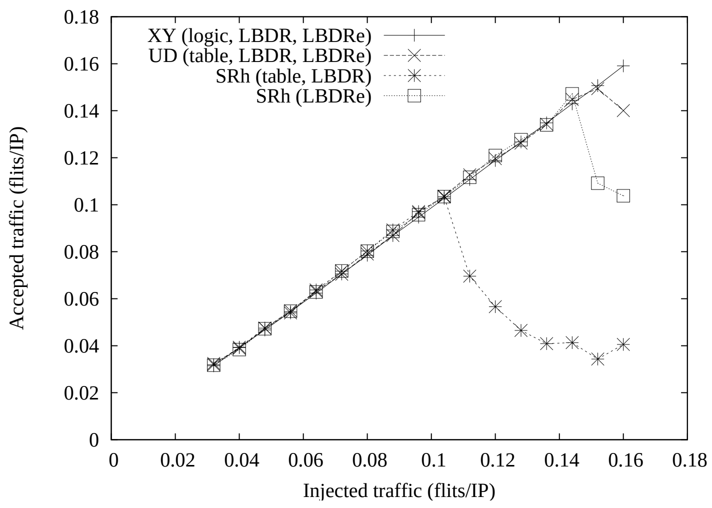


Figure 4.1: Performance achieved for different routing algorithms for the 2D mesh. Uniform traffic distribution.

Figure 4.1 shows the performance (delivered throughput) for uniform traffic and the 2D mesh topology, whereas Figure 4.2 shows the performance for p topology. In all the situations the same basic conclusions can be obtained. First, it can be seen that for XY (in 2D mesh) and UD (in 2D mesh and all the irregular topologies) LBDR mimics the performance achieved with traditional implementation (routing tables). This is achieved because in both cases all the routing restrictions are aligned through the same columns and rows and this permits LBDR to achieve maximum performance.

Second, it can be seen that for SR_h, LBDR achieves different performance numbers depending on the traffic and topology used. In some cases the differences between LBDR and LBDRe may be large (in terms of network throughput) thus aiming to select LBDRe. However, in all the topologies analysed the use of LBDR with a different routing algorithm (UD or SR_h) achieved very close performance numbers to LBDRe. Thus, it is not worth using LBDRe for such marginal performance benefits. It is much more wise to use LBDR as it is a much more compact mechanism.

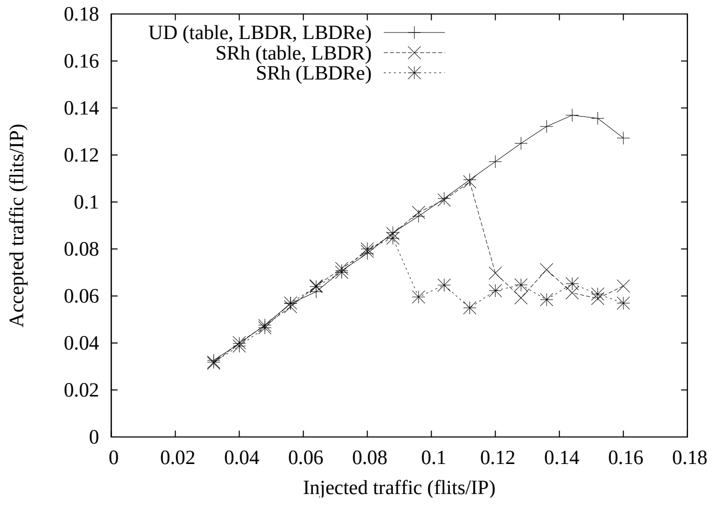


Figure 4.2: Performance achieved for different routing algorithms for a *p*-shaped topology. Uniform traffic distribution.

4.2 Area and Critical Path Delay Analysis

Two 5 × 5 switch architectures were synthetized (source-based routing vs LBDR) with Synopsys Physical Compiler. Since wireload models in traditional synthesis tools are not trustworthy anymore in the context of nanoscale technologies, Physical Compiler performs placement-aware logic synthesis. In practice, after a quick initial logic synthesis based on wireload models, the tool internally attempts a coarse placement of the current netlist. Next, it iteratively optimizes the netlist and the placement, based on the actual wire loads it implies by the current candidate placement. The outcome is a placed netlist that is optimized also accounting for wire delays. We target a 65nm low-power low-Vth technology library available from STMicroelectronics under the CMP project [24].

Critical path results are illustrated in Figure 4.3. Clearly, in the LBDR switch the allocator becomes less complex and some basic functionality has been moved to the LBDR logic. Overall, the synthesis tool has performed a good optimization of the allocator and of the LBDR modules (their total delay is almost the same as that of the old allocator), but has ended up slightly penalizing the control logic of the output buffer on the critical path. Let us recall that input and output buffers also act as flow control stages [25]. Overall, the delay penalty of LBDR is limited to only 5% with respect to source based routing, which is an excellent result.

Area results are even more promising, since the minor degradation of the critical path has been achieved while saving a small percentage of area. Timing and area results clearly allow to see the behaviour of the optimization process of the synthesis tool.

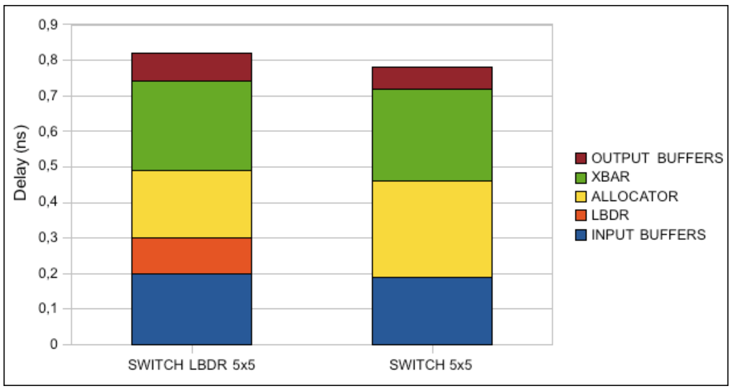


Figure 4.3: Timing physical synthesis results of a 5 × 5 switch.

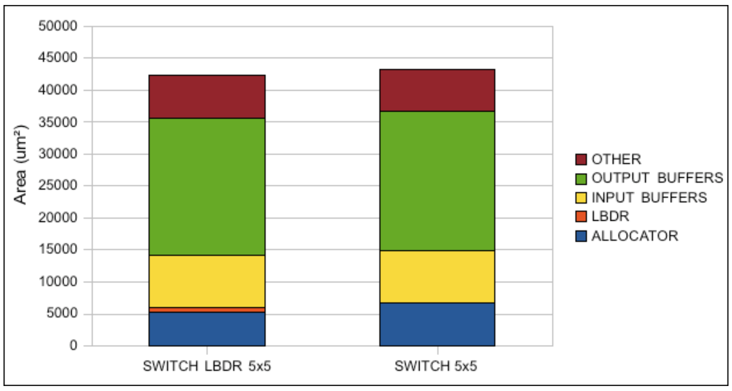
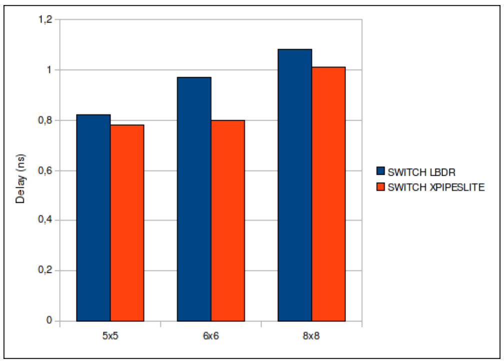
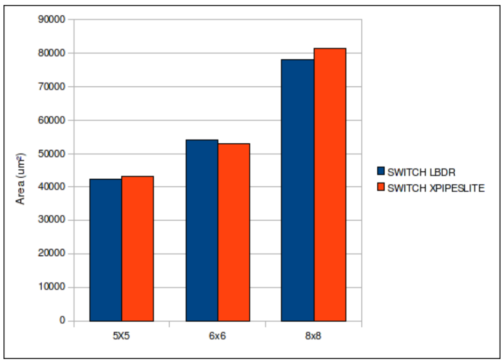


Figure 4.4: Area physical synthesis results of a 5 × 5 switch.

By implementing the extension of more cores per switch, 6 × 6 and 8 × 8 switches were implemented for synthesis for the support of 2 and 4 cores per switch respectively. Comparative timing and area scalability results are illustrated in Figure 4.5(a) and Figure 4.5(b) respectively. While critical path delay in the LBDR switch has been scaled almost linearly by the synthesis tool, some unpredictable optimization has been performed on the xpipesLite 6 × 6 switch. However, the 8 × 8 switch implementation confirms that the timing gap between the two variants is kept limited within 6%. Again, area results are promising for the LBDR switch, which shows a good scalability of this metric even to 8 × 8 switches.



(a) Timing



(b) Area

Figure 4.5: Timing and area scalability results.

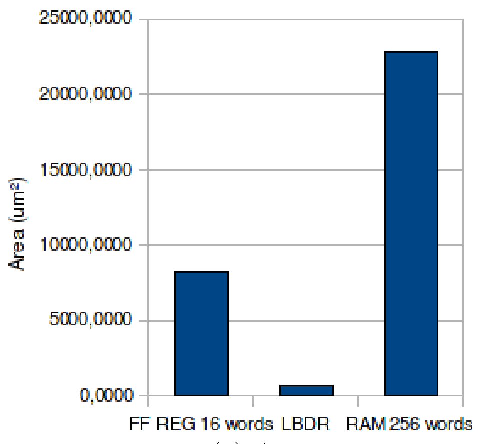
In all cases, the delay and area differences between the two switch variants do not follow a clear scalability trend but have to be assessed case by case, since they are tightly dependent on the optimizations the synthesis heuristic is able to perform on the specific design at hand. This further confirms that the overhead of LBDR is so small that it falls within the unpredictability margin of the synthesis tool behaviour.

4.3 LBDR vs Routing Tables

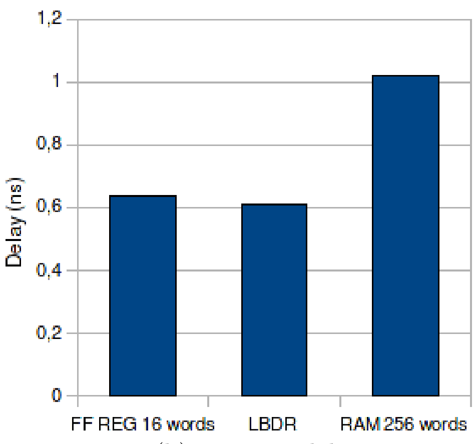
As the main remarked objective in this work is to state that LBDR removes the need for routing tables at switches, so the next step of analysis was clear. In most designs of practical interest, forwarding tables are usually implemented by means of memory macros. This motivates a comparison between LBDR logic implementation and memory macro-based routing tables in terms of area and routing delay.

For the experiments Memaker was used, a memory compiler from Faraday Technology Corporation [2], generating memory macros for a 90 nm UMC process technology. LBDR logic was synthesized again for the new technology library (for the sake of technology-homogeneous comparisons) and its delay contrasted with the access delay of a corresponding memory macro meeting the same routing requirements.

For the evaluation, the following scenario was assumed: switches are placed in a 2D mesh topology, with one computation tile attached to each switch. Each tile consists of a processor core and a memory core, each one



(a) Area



(b) Routing delay

Figure 4.6: Area and routing delay analysis for 16 destinations.

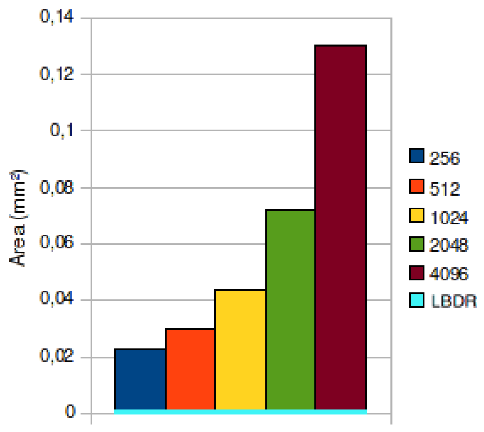
requiring a master and a slave XpipesLite network interface for accessing the network. Thus, switches in the worst case need 8 I/O ports. The total number of destinations is 16 end-nodes in the system at hand (a 4x4 2D mesh).

When routing tables are used for distributed routing, each switch input port has a memory module with a number of words equal to the amount of destinations. Every word is composed of 3 bits, matching the switch radix. Given a destination ID, the switch thus selects the target output port (from 0 to 7) based on table look-up.

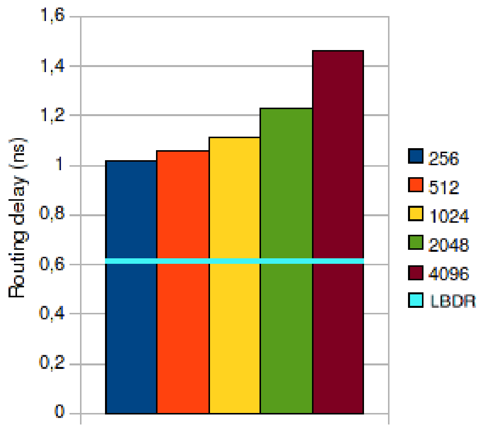
The minimum size in words that Memaker, at the 90nm technology node, can generate is 256 words. For fewer words, Memaker can however infer a single-port register file. We also included this option in the comparison of Figure 4.6, targeting our system under test with 16 destinations. The figure shows that LBDR logic consumes significantly less area while matching the delay of the register-based routing table solution. The oversized memory macro is clearly non-competitive at this system scale.

Figure 4.7 illustrates area and delay scalability of LBDR and memory macro-based routing tables with an increasing number of network destinations. Clearly, while the memory macro suffers from increasing area and delay penalties, LBDR logic complexity does not depend on the number of destinations, hence keeps constant. The area occupied by LBDR modules just grows with switch radix.

When the number of destinations is between 16 and 256, Figures 4.6 and 4.7 suggest that LBDR will be by far the most area effective solution, while at



(a) Area



(b) Routing delay

Figure 4.7: Scalability of LBDR and memory macro-based routing tables when increasing the amount of the destinations.

least small routing delay improvements with respect to register-based routing tables can be conservatively expected.

Chapter 5

Related Work

Some work on the reduction of memory requirements for routing in NoCs already exists. One solution is Interval Routing [9]. With interval routing, sets of destinations requesting the same output ports are grouped. This method is specific for regular topologies. The FIR method [10] is an extension of interval routing to allow different routing algorithms in meshes and tori networks. However, FIR is not applicable to irregular networks. Another solution is named *street-sign routing* [18]. In this method, only the router name of the next turn and the direction of the turn are included in the packet header.

Two solutions for irregular topologies have been proposed [11], [12]. In both cases, the destinations are grouped into regions and regions are coded into switches. A region is coded by the top left-most switch and the bottom right-most switch. Although the number of regions grows logarithmically with the number of failures, the number is unbounded and each region implies a logic. Another solution for routing table minimization is presented in [14]. In this case logic is used for the *regular* case and a deviation routing table is used for routing deviations.

Although different solutions exist, none of them allows the implementation of distributed routing algorithms in irregular topologies with no routing tables and minimum logic.

Chapter 6

Conclusions and Future Work

In this work the LBDR mechanism is presented as a compact mechanism. LBDR allows efficient implementation of most of the existing distributed routing algorithms in regular as well as many irregular NoC topologies. For LBDR only two routing bits and one connectivity bit are required along with a small logic per output port. For LBDRe four more bits are required along with the bits existing in LBDR. LBDR mimics the performance achieved by XY and UD routing algorithms when implemented using routing tables. For more sophisticated routing algorithms, like SR_h, the LBDRe method may be used.

Although not evaluated, we have analysed the extension of LBDR with additional visibility routing bits and obtained no performance gains with any routing algorithm. Therefore, LBDRe, provides enough visibility to extract the full potential of any minimal routing algorithm on an irregular topology. Also, we would like to point that although LBDR loses some performance with some routing algorithms it is much more attractive than LBDRe, due to its simplicity. Also, with a proper routing algorithm (XY and/or UD) the performance penalty is eliminated.

The implementation of LBDR in a real-life network-on-chip switch architecture and a comparative analysis with a lightweight switch for source-based routing shows a delay penalty of only 5% and even a small area saving at the 65nm technology node. In general, its overhead is so small that it falls within the unpredictability margin of the synthesis tool heuristics. Moreover, LBDR was extended to support the connection of multiple cores to the same switch, and the comparative properties with respect to a switch for source-based routing have been proven to scale to higher values of the switch radix.

Finally, by comparing area and routing delay of LBDR logic with those of routing tables, the unmistakable superiority of LBDR has been shown.

Even considering register and memory macro implementation options for routing tables (when applicable by the memory compiler), LBDR always proved the most area-saving solution and already performance-efficient in the worst case scenario. Above all, the constant area and delay of LBDR with an increasing number of destinations in the network makes the point for its better scalability.

The following research will be performed as future work (to get the PhD):

- Support for overlapped regions definition will be incorporated to LBDR.
- Broadcast/multicast support will be analyzed and incorporated to LBDR.
- Fault-tolerance techniques will be analyzed.
- The support for non-minimal paths (enabling the coverage to more topologies) will be evaluated and incorporated to LBDR.

6.1 Contributions

In this Section there are listed previous papers that are the background of this work:

- J. Flich, S. Rodrigo, and J. Duato, 'LBDR: Efficient Routing Implementation in NoCs', Second 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), January, 2008.
- J. Flich, S. Rodrigo, J. Duato, T. Sdring, . G. Solheim, T. Skeie, O. Lysne,, 'On The Potential of NoC Virtualization for Multicore Chips', International Workshop on Multicore Computing Systems (MuCoCoS), held in conjunction with International Conference on Complex, Intelligent and Software Intensive Systems (CISIS), March, 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), April, 2008.
- S. Rodrigo, J. Flich, J. Duato and D. Bertozzi, 'Assessing the implementation trade-offs of logic-based distributed routing for Networks-on-Chip', Fourth International Summer School on Advanced Computer Architecture and Compilation for Embedded Systems (ACACES, HiPEAC), July, 2008.

- S. Rodrigo, J. Flich and J. Duato, 'Una Implementacion Eficiente de Algoritmos de Encaminamiento Distribuido para Redes dentro del Chip', XIX Jornadas de Paralelismo, September, 2008.
- J. Flich, S. Rodrigo, J. Duato, T. Sdring, . G. Solheim, T. Skeie, O. Lysne, 'On The Potential of NoC Virtualization for Multicore Chips', published on journal Scalable Computing: Practice and Experience, Special Issue: Recent Developments in Multi-Core Computing Systems, Volume 9, Number 3, pages 16517, September, 2008.
- S. Rodrigo, J. Flich and J. Duato, 'Towards an Efficient Implementation of Distributed Routing in NoCs', submitted (queued for review) to IEEE Transactions on HiPEAC.
- S. Rodrigo, S. Medardoni, J. Flich, J. Duato and D. Bertozzi, 'An Efficient Implementation of Distributed Routing Algorithms for NoCs', published as best paper in the special issue on Networks-on-Chip of IET Computers I& Digital Techniques.

6.2 Acknowledgements

This work was supported by CONSOLIDER-INGENIO 2010 under Grant CSD2006-00046, by CICYT under Grant TIN2006-15516-C04-01, by Junta de Comunidades de Castilla-La Mancha under Grant PCC08-0078, by the European Commission in part in the context of the HiPEAC Network of Excellence (interconnect cluster) and in part by the GALAXY European Project FP7-ICT-214364.

I would like to thank a lot the kind help, collaboration and advising of Davide Bertozzi and Simone Medardoni for the implementation of LBDR and extensions in the XpipesLite switch architecture during the internship at the University of Ferrara, Italy from May to August, 2008.

Of course, a very big thank you to my advisor, José Flich, for supporting and giving me this great opportunity for my PhD research and being patient and helpful every time I need.

Bibliography

[1] Ernst, D. et al.: 'Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation', The 36th Annual International Symposium on Microarchitecture (MICRO-36), 2003

[2] `http://www.faraday-tech.com/index.html`, accessed November 2008

[3] Bertozzi, D., Benini, L., De Micheli, G.: 'Error control schemes for on-chip communication links: the energy-reliability tradeoff', IEEE Transactions on CAD of Integrated Circuits and Systems, 24, (6), 2005, pp. 818-831

[4] Hoskote, Y., Vangal, S., Singh, A., Borkar, N., and Borkar, S.: 'A 5-GHz Mesh Interconnect for a Teraflops Processor', IEEE Micro Magazine, September-October 2007, pp. 51-61

[5] Kahle, J.A., Day, M.N., Hofstee, H.P., Johns, C.R., Maeurer, T.R., and Shippy, D.: 'Introduction to the Cell multiprocessor', IBM Journal of Research and Development, 49, (4/5), 2005

[6] Sankarakingam, K. et al: 'Exploiting ILP, TLP, and DLP using polymorphism in the TRIPS architecture', The 34th International Symposium on Computer Architecture (ISCA), 2007

[7] Taylor, M.B. et al: 'Scalar operand networks: On-Chip Interconnect for ILP in partitioned architectures', The Ninth International Symposium on High-Performance Computer Architecture (HPCA), 2003

[8] Swanson, S., Michelson, K., Schwerin, A., and Oskin, M.: 'Wavescalar', The 36th Annual International Symposium on Microarchitecture (MICRO-36), 2003

[9] Van Leeuwen, J., and Tan, R.B.: 'Interval routing', The Computer Journal, 30, (4), pp. 298-307

[10] Gómez, M.E. et al: 'A Memory Effective Routing Strategy for Regular Interconnection Networks', International Parallel and Distributed Processing Symposium (IPDPS), 2005

[11] Palesi, M., Kumar, S., and Holsmark, R.: 'A Method for Router Table Compression for Application Specific Routing in Mesh Topology NoC Architecture', International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2006

[12] Flich, J., Mejía, A., López, P., and Duato, J.: 'Region-Based Routing: An Efficient Routing Mechanism to Tackle Unreliable Hardware in Networks on Chip', First ACM/IEEE International Symposium on Networks on Chip (ISNOC), 2007

[13] Gelernter, D.: 'A DAG-based algorithm for prevention of store-and-forward deadlock in packet networks', IEEE Transactions on Computers, 30, (10), October 1981, pp. 709-715

[14] Bolotin, E., Cidon, I., Ginosar, R., and Kolodny, A.: 'Routing Table Minimization for Irregular Mesh NoCs', International Conference on Design, Automation and Test in Europe (DATE), 2007

[15] Sancho, J.C., Robles, A., and Duato, J.: 'A Flexible Routing Schemes for Networks of Workstations', The Third International Symposium on High Performance Computing (ISHPC), 2000

[16] Glass, C., and Ni, L.: 'The Turn Model for Adaptive Routing', The 19th International Symposium on Computer Architecture (ISCA), 1992

[17] Noxim: Network-on-Chip simulator, available at `http://noxim.sourceforge.net`, accessed November 2008

[18] Borkar, S. et al: 'iWarp: An Integrated Solution to High-Speed Parallel Computing', Supercomputing Conference, 1988

[19] Wilton, J.E., and Jouppi, N.P.: 'An enhanced access and cycle time model for on-chip caches', Technical Report, 93/5, Western Research Laboratory, 1993

[20] Shivakumar, P., and Jouppi, N.P.: 'CACTI 3.0: An integrated cache timing, power and area model', Technical Report, 2001/2, Western Research Laboratory, 2001

[21] Toshiba, product guide 2003, available at `http://www.toshiba.com/taec/components/Generic/cmosasic_tc300prodbroch.pdf`, accessed November 2008

[22] Gilabert, F., Medardoni, S., Bertozzi, D., Benini, L., Gomez, M.E., Lopez, P., and Duato, J.: 'Exploring High-Dimensional Topologies for NoC Design Through an Integrated Analysis and Synthesis Framework', Second ACM/IEEE International Symposium on Networks-on-Chip (IS-NOC), pp.107-116, 2008

[23] Stergiou, S. et al: 'Xpipes Lite: A Synthesis Oriented Design Flow For Networks on Chips', International Conference on Design, Automation and Test in Europe (DATE), pp. 1188-1193, 2005

[24] Circuits Multi-Projects, Multi-Project Circuits, `http://cmp.imag.fr`, accessed November 2008

[25] Pullini, A. et al: 'Fault tolerance overhead in network-on-chip flow control schemes', Southern Building Code Congress International (SBCCI), 2005, pp. 224-229