Fault-Tolerant Nostrum NoC on FPGA for the ForSyDe/NoC System Generator Tool Suite

Royal Institute of Technology
School of Information and Communication Technology
Department of Electronic Systems

SALVATOR GKALEA

Master of Science Thesis in System-on-Chip Design
Supervisor: Johnny Öberg, Francesco Robino
Examiner: Ingo Sander

TRITA-ICT-EX-2014:187
Abstract

Moore’s law is the observation that over the years, the transistor density will increase, allowing billions of transistors to be integrated on a single chip. Over the last two decades, Moore’s law has enabled the implementation of complex systems on a single chip (SoCs). The challenge of the System-on-Chip (SoC) era was the demand of an efficient communication mechanism between the growing number of processing cores on the chip. The outcome established a new interconnection scheme (among others, like crossbars, rings, buses) based on the telecommunication networks and the Network-on-Chip (NoC) appeared on the scene.

The NoC has been developed not only to support systems embedded into a single processor, but also to support a set of processors embedded on a single chip. Therefore, the Multi-Processors System on Chip (MPSoC) has arisen, which incorporate processing elements, memories and I/O with a fixed interconnection infrastructure in a complete integrated system. In such systems, the NoC constitutes the backbone of the communication architecture that targets future SoC composed by hundred of processing elements. Besides that, together with the deep sub-micron technology progress, some drawbacks have arisen. The communication efficiency and the reliability of the systems rely on the proper functionality of NoC for on-chip data communication. A NoC must deal with the susceptibility of transistors to failure that indicates the demand for a fault tolerant communication infrastructure. A mechanism that can deal with the existence of different classes of faults (transient, intermittent and permanent [11]) which can occur in the communication network.

In this thesis, different algorithms are investigated that implement fault tolerant techniques for permanent faults in the NoC. The outcome would be to deliver a fault-tolerant mechanism for the NoC System Generator Tool [29] which is a research in Network-on-Chip carried out at the Royal Institute of Technology. It will be explicitly described the fault tolerant algorithm that is implemented in the switch in order to achieve packet rerouting around the faulty communication links.
Acknowledgment

I would like to express my gratitude to my supervisor Johnny Öberg and Francesco Robino who gave me the opportunity to deal with a fascinating project. I would like also to thank and acknowledge to Ingo Sander, his participation to this master thesis as my examiner. Last but not least, I would like to thank my parents for their support all these years and without them I would not be here to write this thesis.
# Contents

Abstract iii

Acknowledgment v

Contents vii

List of Figures ix

List of Tables x

1 Introduction 1
   1.1 Background ................................................. 3
   1.2 Related work .............................................. 4
   1.3 Goal ......................................................... 8

2 Network-on-Chip 9
   2.1 Fundamentals of NoC architecture .............................. 9
       2.1.1 Links .................................................... 9
       2.1.2 Switch .................................................. 10
       2.1.3 Resource Network Interface (RNI) ........................ 12
   2.2 NoC Topologies .............................................. 12
   2.3 Network Flow Control ......................................... 14
       2.3.1 Bufferless Flow Control ................................ 14
       2.3.2 Buffered Flow Control .................................. 15
   2.4 Routing algorithms .......................................... 15
       2.4.1 Source and Distributed routing (i) ......................... 16
       2.4.2 Deterministic and Adaptive routing (ii) .................. 16
       2.4.3 Minimal and Non-Minimal routing (iii) .................... 16
       2.4.4 Deadlock and Livelock ................................... 16
   2.5 Fault-Tolerant Routing ........................................ 17
   2.6 Nostrum NoC ................................................ 18
   2.7 Conclusion .................................................. 19

3 NoC System Generator Tool 21
CONTENTS

3.1 NoC Platform Generation Process ............................................ 21
3.2 Topology ............................................................................... 23
3.3 Messages .............................................................................. 24
  3.3.1 Packets ........................................................................... 24
  3.3.2 Flits ................................................................................. 24
3.4 Routing .................................................................................. 26
  3.4.1 Dimension-Order Routing ................................................. 26
  3.4.2 Deflection Routing ............................................................ 27
3.5 Reference Switch of NSG Tool ................................................... 28
  3.5.1 Receiver unit ..................................................................... 29
  3.5.2 Transmitter unit ............................................................... 29
  3.5.3 Crossbar unit .................................................................... 31
  3.5.4 Conclusion ....................................................................... 32

4 Reconfigurable Fault-Tolerant Switch ............................................ 33
  4.1 Switch v.1 ............................................................................ 33
    4.1.1 Memory & Hardware Logic ............................................ 33
  4.2 Switch Proposal v.2 ............................................................... 35
  4.3 Routing scheme ..................................................................... 36
    4.3.1 Adaptive routing ............................................................ 36
    4.3.2 Routing tables ............................................................... 37
  4.4 Fault-tolerant mechanism and Reconfiguration .......................... 38
  4.5 Prototype .............................................................................. 42

5 Conclusion .................................................................................. 45

6 Future Work .............................................................................. 47

Bibliography .................................................................................. 49
# List of Figures

1.1 Network-on-Chip layers [32] ............................................. 2
1.2 Region definition [26] .................................................. 5
1.3 Dynamic XY Routing algorithm [20] ................................. 6
1.4 Fault-on-Neighbor schematic [15] .................................. 7

2.1 The three building blocks of a NoC node: Links, Switch, RNI .... 10
2.2 A generic switch architecture for 2D NoCs .......................... 11
2.3 A 3x3 2D-Mesh topology ............................................. 12
2.4 A 3x3 Torus topology ................................................ 13
2.5 A binary Fat-tree topology .......................................... 13
2.6 A 3-stage Butterfly topology ...................................... 14
2.7 Buffered and Bufferless flow control .............................. 15
2.8 Nostrum topology [21] ................................................. 18

3.2 Different topologies(ring, 2D-mesh, torus) used by the NSG tool. .. 23
3.3 Node addressing representation .................................... 23
3.4 Message decomposition ............................................. 24
3.5 Packet structure. Up to 128 flits form a packet. .................. 25
3.6 Flit segmentation .................................................. 25
3.7 XY-routing a packet from switch (0,0) to switch (2,2) in a 3x3 2D-mesh. 26
3.8 A situation with packet contention. Deflection routing is performed on the packets A. ......................................... 27
3.9 Schematic of the reference switch of NSG tool. ..................... 28
3.10 The Finite State Machine (FSM) of the crossbar. ................. 31

4.1 Schematic of the switch version 1. ................................ 34
4.2 Schematic of the switch version 2 ................................ 36
4.3 Representation of the hop information in the memories .......... 38
4.4 FSM of the switch version 2 ..................................... 39
4.5 Simulation test for 3x3 2D-Mesh with faulty links ................ 42
4.6 The route of a packet with the presence of faults ................ 42
4.7 The route of the 2nd packet with updated routing tables ........ 43
# List of Tables

1.1 Pros and Cos of Busses vs NoCs [5] ........................................... 3
2.1 Comparison between different fault-tolerant routing techniques ....... 19
3.1 Routing table ........................................................................... 30
4.1 Routing table of switch 4 in a 3x3 mesh .................................... 37
5.1 Area comparison and throughput between 3 different switches. .... 45
Chapter 1

Introduction

Nowadays, complex System-on-Chip (SoC) systems face some communication challenges that need to be handled by the communication infrastructure. These challenges can be summarized as follow [12] [31]:

- **Performance**: high levels of performance, throughput, low latency and synchronization.
- **Scalability**: the easy addition of functional units to the system.
- **Parallelism**: the parallel communication between intellectual properties must be provided.
- **Reusability**: predefined communication platform that can be easily reused to new designs.
- **Quality of Service**: should guarantee the performance & the reliability of the services provided.
- **Reliability and Fault Tolerance**: should provide detection and recovery mechanism for faults in the system.

The Network-on-Chip (NoC) architecture addresses and satisfies most of the above communication requirements. Therefore, it was proposed by the research community, as the communication architecture that targets embedded systems composed by hundred of functional Intellectual Property (IP). This solution overcomes the problems (ex. communication bottleneck) that came with the traditional bus-based architecture and met some of the principal requirements of future systems: reusability, scalability, reliability and low power consumption. In order to achieve that, NoC paradigm breaks the problem of communication between IPs into smaller problems, such as separating computation from communication and treating the interconnect as a protocol stack, where different layers implement different functions of the network.
The NoC technology has emerged significant advantages against traditional hierarchical busses and crossbar interconnect approaches. A major difference between NoC and busses is based on the physical implementation approaches they use. NoCs implement a point-to-point, Globally Asynchronous Locally Synchronous (GALS) approach, while busses use a synchronous and multi-point. Owing to this basic difference, NoC implementation can succeed to sustain higher clock frequencies and higher throughput.

Furthermore, scalability and reusability of the IP blocks are issues that the crossbar architectures cannot resolve. Conversely, the NoC architecture offers layers of abstraction which makes each layer invisible to the others. This creates a scalable switch interconnection in which every layer of abstraction doesn’t involve in the process of any other layer (Figure 1.1). The crossbars also suffer from the restriction of the limited reuse of IP blocks based on a given protocol. On the other hand, the NoC can support mixing IP blocks based on different protocols [3].
1.1. BACKGROUND

<table>
<thead>
<tr>
<th>Bus Pros and Cons</th>
<th>NoC Pros and Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-) Bus timing is difficult</td>
<td>(+) Performance is not degraded when scaling</td>
</tr>
<tr>
<td>(-) Bus arbitration can become a bottleneck</td>
<td>(+) Network wires can be pipelined</td>
</tr>
<tr>
<td>(-) Bandwidth is limited and shared by all units</td>
<td>(+) Reuse of the switch for any network size</td>
</tr>
<tr>
<td>(+) Bus latency is wired-speed once arbiter has granted control</td>
<td>(-) Internal network contention</td>
</tr>
<tr>
<td>(+) Concept is simple</td>
<td>(-) Sophisticated concept</td>
</tr>
</tbody>
</table>

Table 1.1: Pros and Cons of Busses vs NoCs [5]

1.1 Background

Multicore SoC designs are managing enormous inter-core data rates which require high circuit activity. This factor combined with the size, complexity and integration density, make the communication backbone of the system vulnerable. A successful Multicore SoC design must comply with the most of the challenging issues, mentioned in the previous section. Therefore, it must adapt a communication network that provides defined and reliable communication services among all IPs, addressing the reliability and fault tolerant challenge. Though we must learn to build a reliable systems from unreliable components [7].

The main advantage of NoC that is aligned with the fault tolerant issues is the fact that it offers redundant communication alternatives which can be exploit to construct a reliable network, through fault models, error detection, fault tolerance and reconfiguration. These properties of the NoC can be applied for different fault classes, such as transient, intermittent and permanent faults [11], which can occur in four different layers of abstraction. The NoC layers, in Figure 1.1, can be summarized, according to the OSI reference model of communication protocol, to the physical layer, data link layer, network layer and transport layer [32] [28, p.18].

- **Physical layer**: This layer is concerned with the details of transmitting data on a physical medium. It defines the electrical and physical specification of the data connection (ex. number of wires/bits which connect every switch).

- **Data link layer**: This layer is concerned with the reliable data transmission over the physical link. It may encapsulate error detection and correction codes.

- **Network layer**: This layer provides a topology-independent view of the end-to-end communication and includes the switching policy and routing algorithms. It provides the functional and procedural block that transfers arbitrary length of data sequences (ex. switch, crossbar)
CHAPTER 1. INTRODUCTION

- **Transport layer**: This layer is responsible to hide all the information that is underneath and to provide flow control, packet segmentation and messages ordering. It acts as an intermediate agent which wraps and produces an abstraction of the lower layers to the IP block (ex. Resource Network Interface (RNI)).

In each of these NoC layers, fault-tolerant mechanism can be introduced through redundancy that will provide robustness against failures. Redundancy can be classified into three categories:

1. **Spatial redundancy**: Duplicating components of the network.

2. **Temporal redundancy**: Re-execution of a process (ex. computation, data transmission).

3. **Information redundancy**: Adding error correction or fault information to the data.

This short overview of the fault classes, the NoC layers and the redundancy techniques show that, there are as many potential source of faults and errors in future Multicore SoCs, as there are fault tolerant mechanisms for data protection and data lost prevention. Obviously, the fault tolerant research area is extensively large, so this thesis is focusing on the permanent faults that occur in the communication links and is attempting to investigate spatial and information redundancy techniques which are implemented at the Network layer. An extensive case study, based on theses features, has been conducted and it is summarized in the next section.

1.2 Related work

The great feature that comes with the NoC topology is the fact that the interconnect network is a mesh of path redundancies. It means that the communication between two nodes in the network can be achieved with different alternative paths and without the overhead of replicating hardware components. This is accomplished by adapting a fault-tolerant routing algorithm to the Network layer which is responsible for exploring alternative communication paths in the case of faults in the standard path. Based on this assumption, some fault-tolerant routing algorithms have been studied, and they will be analyzed further in the following paragraphs.

An approach that updates regular routing tables is a technique called *Region-Based Routing (RB)* [26]. This schema creates areas that are composed based on a group of destinations at every switch. In every switch, a set of regions are defined according to the restrictions of the underlying routing algorithm and by taking into account the possible input ports used by the packets, the subset of the output ports and the potential destinations that can be reached. This routing information is
1.2. RELATED WORK

stored efficiently in a region-based table and afterwards is being inspected in order to identify which region is suitable for routing the packet and finally extract the set of output ports provided with the matched region. The basic idea is to compute offline in each switch, the regions (the subset of destinations) that can be reached via the same set of output ports. Figure 1.2 illustrates a 2D-Mesh topology with link failures. In a first phase, the algorithm determine a set of possible paths for every pair of nodes. Based on these paths, routing options are calculated which contain information about the input/output port and the destination. These routing options are grouped and merged together, in order to create the regions based on the port/destination similarities. Once routing options and regions are computed, the packets are forwarded based on the region they want to reach and they are redirected from specific output ports in every switch.

Another fault-tolerant routing algorithm is proposed as a Segment-Based Routing (SG) [25]. The fundamental concept of this algorithm is the partitioning of the topology into sub-nets and sub-nets into segments. In each segment, a routing algorithm is applied based on turn restrictions and each segment is independent of each other. This feature allows to apply any local routing restriction in the specific segment independent of the other segments and build up a sub-routing policy for that segment based on the faults in the particular sub-network.

The Dynamic XY Routing (DyXY) [24] and the Dynamic Adaptive Deterministic Routing (DyAD) [20] combine the advantages of both deterministic and adaptive routing schemes to avoid congested switches. The underlying routing algorithm performs re-routing based on the congestion information in every switch. This feature could be extended to avoid faults in the links between the switches and route the packet around the fault. In Figure 1.3 the fault-tolerant routing algorithm is described which implements a Dynamic XY Routing based on the safety of every

Figure 1.2: Region definition [26]
output port combined with congestion information.

A new mechanism that is not using routing tables is described as **Logic-Based Distributed Routing (LBDR)** [17]. The major advantage of this proposal is that implements a small logic circuit that mimics the behavior of routing algorithms implemented with routing tables. The functionality of LBDR is based on a number of routing bits per output port and the information that they are carrying depends on the topology and on the routing algorithm is implemented.

The XY routing algorithm [33] can be extended to use the adaptability of the odd-even turn model [10] as suggested by Wu [36]. This fault-tolerant algorithm reroutes packets around faulty rectangular regions (special convex or concave shapes) based on the XY routing policy. However, it may need to deactivate healthy nodes in order to create these rectangular faulty blocks. The basic idea behind this schema is that when a packet reaches a faulty block, it will be rerouted around the block clockwise or counterclockwise based on certain routing restrictions [36].

The Re-configurable Routing Algorithm [38] makes use of cycle-free contour sur-
1.2. RELATED WORK

rounding by faulty router, so it can reroute the packet properly. The contours must not be overlapped and therefore the faulty switches should have a considerable big distance between each other. In order to achieve cycle-free contour, the algorithm must deactivate all the switches that are in this faulty region.

Based on a fine-grained functional fault model, error-detecting circuitry and distributed on-line fault diagnosis, the status of the switch is extracted [23]. With the methods above, *crossbar faults* model connection failure from an incoming port to an out-coming port, making use of CRC-protected data packets to identify faults and error counters to determine transient from permanent faults. The underlying routing algorithm can be adapted to reroute the packets avoiding faulty link in the path.

A fault-tolerant Source Routing for Network-on-Chip (SRN) has been proposed by Kim and Kim [22] which implements two mechanism, of *route discovery* and *route maintenance* to allow nodes to discover and maintain source routes to arbitrary destinations. The procedure in this proposal is to invoke the *route discovery* protocol to explore a new route destination in case there aren’t any in the route cache for a particular destination and *route maintenance* protocol which is responsible to confirm the packet’s arrival by the next hop along the source route.

A defect routing algorithm can be easily upgraded to achieve fault-tolerance at the cost of some hardware overhead. Fault-on-Neighbor Deflection Routing (FoN) [15] is a deflection aware routing algorithm which exploits the two hops fault information to avoid broken links. In the 2-hop fault information mechanism (Figure 1.4), every switch receives the status of each four neighbors and also collects the status of its three neighbors and transmits it to the fourth. The FoN is applicable to fault regions constrained by convex and concave shapes.

The FoN can be easily extended to use the property of 2-hop fault information together with reinforcement learning, known as Q-learning [35]. The Fault-Tolerant Deflection Routing (FTDR) [16] implements routing tables which contains infor-
information about the hop distance to every other switch and based on the numbers of hops to the destination, it makes routing decision using only local information. In this implementation, every switch exchange also, except from the 2-hop fault information, Q-values that represent the hop distance of a particular packet to its destination.

1.3 Goal

The basic idea in this project is to develop a prototype of a Fault-Tolerant NoC for the Network-on-Chip System Generator (NSG) Tool Suite [29]. This will be accomplished by designing a new switch for the NSG tool that allows dynamic re-programming of the router tables to avoid live-locks in the presence of link failures. The switch must adapt a fault-tolerant routing algorithm based on the study that was contacted, and the outcome would demonstrate the correct functionality of the system prototype on an FPGA together with test programs running on the processors.

- Implementation of the deflection routing policy of the reference switch using routing tables.
- Implementation of a fault-tolerant routing policy based on the Q-learning approach using routing tables.
- Area comparison between the implementation with and without routing tables of the reference switch.
- Modification in the second implementation in order to be reconfigured the routing tables by the RNI.
- Prototype and verify the system on the DE-115 Development and Education FPGA Board [1].
Chapter 2

Network-on-Chip

Today’s Multi Processor System-on-Chip (MPSoC) requires intensive parallel communication between the cores, which demand maximum bandwidth, low latency and low power consumption. A solution to this communication bottleneck is a modular and scalable communication architecture embedded into the SoC that will provide support for the integration of heterogeneous and homogeneous cores based on a defined network boundary concept. This new interconnect schema is called Network-on-Chip and it will be one of the basic communication cornerstones of future systems. In this chapter, the principles of the intergraded communication system will be analyzed from the perspective of the NoC design. In addition, the inherent redundancy, which is one of the features of NoC, will exploit in order to tolerate failures on the communication medium.

2.1 Fundamentals of NoC architecture

The NoC architecture is established on three elementary units which are illustrated in Figure 2.1 which compose a Node. The first unit is composed by the links which consist the physical medium that connects the nodes and enables the communication between them. The second unit is the switch which constitutes the brain of the communication protocol which is responsible for the correct routing of the packets into the network. The switch receives, decodes and forward, to a particular output port the packets according to embodied information. The last elementary unit is the Resource Network Interface (RNI) which makes the hardware abstraction layer available to the IP cores. In other words, it is responsible for the logic connection between the network and the processing elements [12, p.11].

2.1.1 Links

Every two switches in the NoC are connected directly to each other through a communication link. This link is composed by a physical set of wires, which in group of two sets compose a full-duplex communication system or usually a point-
CHAPTER 2. NETWORK-ON-CHIP

Figure 2.1: The three building blocks of a NoC node: Links, Switch, RNI

to-point system. This system defines the synchronization protocol between the two switches. There are two mainstream categories to represent the communication protocol, such as synchronous or asynchronous links (ex. Globally Asynchronous Locally Synchronous (GALS) [18]). Conclusively, the links, as the principal components that manage the physical transmission over the medium, largely define the performance and the power consumption of a NoC.

The links are implemented in the physical layer as illustrated in Figure 1.1. In this layer, the definition of the packet must be declared and consist of atomic control flow units, called flits (it will be analyzed further in Section 3). In addition, a flit can be divided into smaller units of information or physical transfer digits called phits. The advantage of the packet’s disintegration is to achieve low overhead and efficient resource utilization [13, p.224].

2.1.2 Switch

A typical NoC switch consists of a set of input ports, a set of output ports, a switching matrix and local connection to the IP core (see Figure 2.2). In the 'heart' of the switch exists the control logic block, the switch matrix that implements some of the flow control policies in order to redirect the incoming packets to the correct output ports properly. The links that are connected to the switch can be considered to be unidirectional, bidirectional or serial. The architecture presented in Figure 2.2 can be extended to support 2D or 3D NoCs, simply by adding extra buffer for the incoming and out-coming port of the rest of the layers. It is also important
2.1. FUNDAMENTALS OF NOC ARCHITECTURE

Figure 2.2: A generic switch architecture for 2D NoCs

to mention some definitions that specify the behavior of the switch and play a dominant strategic role for moving the data through the NoC.

- **Flow Control policy:** This policy controls the movement of the packets through the network. The control policy can be centralized or distributed. In the distributed policy every switch can make its own routing decisions. In the centralized policy, the node which injects the packet into the network, defines also the routing decisions that need to be taken from all the other nodes through the routing path of the packet. Another approach related to the flow control is the Virtual Channel (VC) that multiplexes a physical link into many logical. Currently, there are three basic packet-switching techniques: store-and-forward, cut-through and wormhole switching.

- **Routing algorithm:** is the logic of packet routing over the NoC. Every routing algorithm can be classified into, deterministic, non-deterministic, adaptive, static, dynamic, minimal and non-minimal according to the characteristics of the algorithm.

- **Switching:** There are two basic switching policies, the circuit switching and the packet-based switching. The major difference between them is the way of transmitting the packet every time. For the first, it needs to reserve the physical path between source and destination, and transmit all the packet at once, in contrast with the second that needs only to transmit a flit at a time.

- **Buffering policy:** It is related to the unit that stores information in the switch. The number and the size of the buffers is an important factor that affects the performance and the power consumption of the whole system.
CHAPTER 2. NETWORK-ON-CHIP

All these features of the switch reveal a huge design space where there are many combination and tactics to implement a strategy which will meet the system requirements in terms of performance, latency and power consumption.

The flow control policy and buffer policy will be discussed further in Section 2.3, while in Section 2.4 routing algorithms will be analyzed.

2.1.3 Resource Network Interface (RNI)

This unit distinguishes the communication process from the computation process. It acts as logic adapter between the switch and the IP core (ex. CPU, Memory, audio core etc). The RNI module is responsible to compose/decompose the data packets into/from the underlying communication network(see Figure 2.1).

2.2 NoC Topologies

The topology of the network refers to the interconnection structure and placement of all the NoC’s Nodes(refer Figure 2.1 for the term Node) and channels which can be modeled as a graph. The connection between the switches can be direct or indirect [13, 18]. In the case of direct topology every switch attached to a IP core forms a Node and every Node is direct connected to its neighbors, in contrast with the indirect topology where there are also Nodes and simple switches that aren’t connected to any IP core and they just propagate the packets through the network.

![Figure 2.3: A 3x3 2D-Mesh topology](image)

Some of the classical NoC topologies are analyzed in the next paragraph.
2.2. NOC TOPOLOGIES

- **Mesh**: this topology organizes the nodes into a $N$ rows x $N$ columns grid which includes multiple paths between nodes, fault tolerance in link failure and it is easy to expand (Figure 2.3). Routers and resources can be addressed as x-y coordinates in a mesh; all links have the same length and area grows linearly with the number of nodes.

![Figure 2.4: A 3x3 Torus topology](image)

- **Torus**: is the expanded version of a Mesh, and it also called k-ary n-cube. The only difference is that it uses long wrap-around link to connect the two end nodes in the same row or column (Figure 2.4). Torus network provides better path diversity than the mesh network and has more minimal routes.

![Figure 2.5: A binary Fat-tree topology](image)

- **Fat tree**: is a representation of an indirect network, in which the leafs are the IP cores, the computational units, and every node has access to the immediate
networks beneath. Main problem is the bottleneck that might occur in the root Node (Figure 2.5).

![Figure 2.6: A 3-stage Butterfly topology](image)

- **Butterfly:** this kind of networks can be uni- or bidirectional. In a unidirectional butterfly network that contains eight input and output cores, and three stages of routers that each includes four switches. (Figure 2.6).

### 2.3 Network Flow Control

The *Flow Control* module inside the switch determines and regulates the transmission of the data into the network. It defines how the network's resources (buffer capacity, channel bandwidth) are assigned to each packet that is traversing the network (Figure 2.7).

#### 2.3.1 Bufferless Flow Control

This case describes the simplest definition of flow control mechanism which doesn’t use temporary storage buffering for incoming/outcoming packets (Figure 2.7b). For this reason, the arbitration function must deal with the situation that a packet didn’t get its requested destination and therefore it must be disposed with a *misroute or drop* action [13, p.225]. This mechanism applies to networks that offer sufficient path diversity, in order for the packet to reach its destination. The main advantage of this technique is based on the bufferless system that eliminates *deadlock* situation in the network but in the same time it reduces the throughput of the network by the misroutering options.
2.3.2 Buffered Flow Control

Buffered flow control provides an efficient mechanism to decouple the allocation of adjacent channels [13, p.233]. Adapting a buffer to the switch (Figure 2.7a) supports the temporary multi-storage of packets in the case the destination channel is occupied and delay the allocation of this channel until it is ready without complications.

There are three main flow control methods that can be combined with any routing algorithm.

1. **Store-and-Forward flow control**: The packets traverse the network in one piece so every node must have received and stored to the buffer the whole packet before transmit it to the next node. The buffer size that must be allocated must be as large as the packet’s length. The main disadvantage of this technique is that introduce high latency.

2. **Cut-through flow control**: reduces the serialization latency at each hop by transmitting the packets as soon as the header is received without waiting for the entire packet to be received in every node.

3. **Wormhole flow control**: It behaves as cut-through flow control but with the only difference that manipulates communication data as flits. This makes efficient use of the buffers as it allocates a small number of buffer flits for the transmission of a packet.

![Figure 2.7: Buffered and Buffereless flow control](image)

(a) Buffered flow control. Packets can be stored internally in the switch.

(b) Bufferless flow control. Only one packet per input port can be stored.

2.4 Routing algorithms

Routing algorithms compute and predetermined the walkthrough taken by a packet from the source to the destination node. It is also responsible for the performance of the network as it affects the load balance across the network. In addition, it is
involved in the modulation of the latency by forming the path length that should follow the packets into the network. Every routing algorithm can be described from:

i where the routing decisions are computed in the network.

ii how a particular path is selected from a set of possible paths.

iii if the path length to the destination affects the routing decision.

2.4.1 Source and Distributed routing (i)

In the source routing, the path that a packet will follow is predetermined by the source node. The source node decides from its routing table a set of routers to be forwarded, and that information is injected into the packet in order to inform the next nodes in which particular path must be routed. The main advantages of this routing is lying on the fact that is topology agnostic and fast in forwarding packets.

In contrast with the source routing, the distributed routing computes the path on the fly. This means that every node that receives a packet makes its routing decisions based on its routing table. It also uses the routing table storage efficiently, as it needs only to use the local information of its surrounding nodes.

2.4.2 Deterministic and Adaptive routing (ii)

The deterministic routing choose to send a packet over the same way which is calculated from the relative position of the source and destination node. The most common deterministic algorithm is the dimension-order algorithm which routes the packets in increasing order, starting from the X direction and afterwards in Y direction.

The adaptive routing takes advantages of the information about the status of the network and calculates the path based on this information. For that reason, multiple paths can be used for the transmission of a packet that makes it suitable for unreliable networks in order to avoid broken links.

2.4.3 Minimal and Non-Minimal routing (iii)

Minimal routing search and select the path from the source to destination that contains the minimum number of hops. Non-minimal routing algorithm doesn’t care about the shortest path that is available and are usually used to avoid disconnected paths in a network with failures.

2.4.4 Deadlock and Livelock

Some problems arise with lossless flow control and are known as deadlock and livelock. The issue of deadlock appears when packets are occupying the resources of the system, resulting in others packets to not make progress. This can be illustrated as a circular wait queue in which a packet A is waiting for a packet B to
2.5. FAULT-TOLERANT ROUTING

free the resources. The deadlock avoidance can be achieved with any bufferless flow control implementation. In the buffered flow control, some specific policies in the routing algorithm must be taken into account (ex. based on turn models), in order to be characterized as deadlock-free. Another solution is the deadlock recovery which monitors, detects and act to restore the normal functionality of the network. The livelock issue arises when packets cannot reach the destination, and they are on continuous movement into the network. Primary responsible for this kind of situation is the appliance of a non-minimal routing algorithm. A hop-counter embodied to the packet’s header could resolve this issue by granting priority to the packets that have large hop-counter.

2.5 Fault-Tolerant Routing

By its nature, NoC offers inherently hardware redundancy as multi-paths exists between two switches. This feature is utilized by the fault-tolerant routing algorithms to overcome the transient and permanent faults that occur in the links. They are trying to increase the robustness of NoC by routing packets around faulty links and by avoiding the possibility to drop a packet, caused by a malfunction link. The drawback is noticed in the performance overhead that introduces due to the fact of misrouting that leads to congestion.

Fault-tolerant routing takes advantage of some information redundancy techniques to address faults [32]

- **Stochastic communication** [7] [8] approach replicates and routes packets through a set of different paths. Flooding technique implements this approach by replicating every incoming packets and send it out from every output port.

- **Fault Regions** are a fault block model which identifies faulty areas in the network and modify the routing policy to reroute the packets around the borders of these regions. Every switch has knowledge about the fault status of its neighbors. This approach has been extended to support rectangular regions [6] and more complex shape [34].

- **Fault Lookahead** introduces extra information links between the switches in order to identify faulty links and modify the routing decision accordingly.

- **Distributed Distance Vector** attempts to apply global exploration of the network by exchanging fault information in a distributed fashion. Every switch keeps track of the routing table that contains a combination of information about the latency of a packet to every neighbor’s switch.

Furthermore, the fault-tolerant routing must be explored further in order to propose a new schema, not only by encountering the faults, but by also proposing a diagnosis mechanism that will recognize the existence of fault.
2.6 Nostrum NoC

NoC architecture is an important area in the research community. The research group of KTH has proposed a novel communication infrastructure for NoC, called Nostrum [19] [21] [27].

![Nostrum topology](image)

Figure 2.8: Nostrum topology [21]

The Nostrum architecture characteristics that address communication issues from the physical to the application layer:

- Implementation of a regular 2D mesh topology.
- Every switch is directly connected to each of its four neighbors.
- Each Resource is connected to each switch through which it is sending packets and communicates with the rest of the network.
- Physical parameters, predictability of performance and power, and clocking schemes can be controlled due to the regularity of the mesh topology.
- The switches implement a bufferless adaptive deflection routing which keeps the size overhead of the switch small.
- It makes use of synchronous clock domain. In addition, pseudo-synchronous or mesochronous clocking can be utilized with the cost of extra hardware in the switches.
- It offers best-effort traffic and guaranteed latency traffic.
2.7 Conclusion

This chapter was about the fundamentals of a NoC system. The reader is now ready to understand the basic differences between the routing algorithms that have been presented and analyzed in Section 1 (Related work). The following table describes main features of each algorithm. It is also assumed that every routing algorithm mentioned in the Table 2.1 is deadlock-free.

<table>
<thead>
<tr>
<th>Technic</th>
<th>Routing</th>
<th>Computation process</th>
<th>Topology</th>
<th>Flow Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>RB [26]</td>
<td>Distributed</td>
<td>$O(N^2 \times Ports \times 2^N)$</td>
<td>Irregular</td>
<td>wormhole</td>
</tr>
<tr>
<td>SG [25]</td>
<td>Credit-based</td>
<td>$O(N^2)$</td>
<td>Irregular</td>
<td>cut-through</td>
</tr>
<tr>
<td>DyXY [24]</td>
<td>XY-Routing</td>
<td>n/a</td>
<td>Irregular</td>
<td>buffered</td>
</tr>
<tr>
<td>LBDR [17]</td>
<td>Distributed</td>
<td>n/a</td>
<td>Irregular</td>
<td>wormhole</td>
</tr>
<tr>
<td>Reconfigurable routing [38]</td>
<td>Deterministic</td>
<td>n/a</td>
<td>Rectangles</td>
<td>packet-switched</td>
</tr>
<tr>
<td>SRN [22]</td>
<td>XY-Routing</td>
<td>n/a</td>
<td>Based on obstacles</td>
<td>n/a</td>
</tr>
<tr>
<td>Fault-tolerant with deflection [23]</td>
<td>Cost-based Deflection</td>
<td>n/a</td>
<td>Regular</td>
<td>bufferless</td>
</tr>
<tr>
<td>FoN [15]</td>
<td>Deflection</td>
<td>500MHz</td>
<td>Based on convex and concave shapes</td>
<td>bufferless</td>
</tr>
<tr>
<td>FTDR [16]</td>
<td>Table-based Deflection</td>
<td>400MHz</td>
<td>Irregular</td>
<td>bufferless</td>
</tr>
</tbody>
</table>

Table 2.1: Comparison between different fault-tolerant routing techniques

In order to have a mechanism that ensures the correct functionality of a NoC in the presence of faulty links, a general definition must be described. The definition that derives from the techniques, mentioned in Table 2.1 is that the fault-tolerant mechanism should comply with the following conditions:

- It must be topology agnostic. It doesn’t need to care about the underlying topology of the network and should adapt on any change of it.

- It should use a bufferless flow control in order to succeed deadlock-free routing.
• A distributed and table-based routing should be advantageous when dealing with reformation of the topology caused by link failures, achieving minimal routing.

Based on these assumptions a fault-tolerant routing mechanism is proposed and is deeply analyzed in Section 4. This mechanism will also replace the existing routing algorithm for fault-free topologies which is used by the Network-on-Chip System Generator (NSG) (see Section 3).
Chapter 3

NoC System Generator Tool

MPSoC platforms are becoming more and more complex, using even more cores on a single chip, pushing the complexity of the whole system to its limits. This brings over the situation where programming and debugging, for that kind of system, would be very hard. A possible solution to this problem would be a fast prototyping tool to generate and program arbitrarily large heterogeneous or homogeneous multi-core systems. The *Network-on-Chip System Generator (NSG) Tool*, presented in [30] [29], is a design flow prototyping tool which can generate and program NoC-based MPSoC for Altera/Xilinx FPGAs. The *Nostrum NoC* [21] architecture and design methodology, which provides a stable and reliable structure for communication in NoC architecture, compose the basic backbone communication infrastructure in which every generated system, by the tool, is based on. Apart from the hardware implementation, the tool automatically creates the distributed memory model and the processors with the device drivers and the application files.

3.1 NoC Platform Generation Process

The NSG requires two kinds of description files in order to generate the desired system. The first input file is the *system description file* in Extensible Markup Language (XML) format which describes all the properties for the system that can be generated. Such as parameters for the topology, size and dimension of the NoC, the IP cores that are connected to every node and the mapping of the software process network to the nodes.
CHAPTER 3. NOC SYSTEM GENERATOR TOOL

The second file is the process description file written in C programming language and contains the system-level description of the system which describes the process network and the functionality of each process mapped to a node of the system. The Figure 3.1 illustrates the dependencies and the generation flow of the system.

After successful generation, the NSG tool will create the following output files.

- **Hardware Description Files** which consists of VHDL files that describe the NoC interconnection with the processing elements, memories and I/O for a target FPGA vendor.

- **Software Project** which contains C-files for the functionality mapped to each processing element, a scheduler to synchronize the process network which is running on the nodes, the system description headers and the device drivers which are the hardware abstraction layer for the processes.
3.2. TOPOLOGY

3.2 Topology

There are many topologies that can be implemented by a NoC, such as mesh, torus, butterfly, fat tree and in different layers (1D, 2D, 3D). Currently, the NSG tool can generate 1D, 2D, 3D Mesh and Torus topologies. There are two methods that are used to address any node in these topologies, the relative addressing technique (Figure 3.3b) in which the address of its node is specified by indicating its distance from a base address, and the absolute addressing technique (Figure 3.3a) in which the actual address of a node is specified by a unique number. In Figure 3.3 the representation of every node is given by the coordinates (X,Y), where .

The NSG tool, in this version, can provide up to 8x8 mesh with four layers, in other words 8x8x4 3D-mesh.
3.3 Messages

Networks provide the essential mechanisms to distributed messages over a group of connected nodes. The message is a contiguous group of data (bits). The data carried by a packet-switched network form the network packet. A packet contains two kinds of data: control information data which are necessary for the delivery of the packet, representing routing information (e.g., source, destination, hop-counter, error detection codes, etc.) and the payload which is the data that the packet is carrying on behalf of the application. In addition, packets can be divided into smaller units, Flow Control Digit (flit) and Physical Transfer Digit (phit).

![Message decomposition](image)

Figure 3.4: Message decomposition

3.3.1 Packets

The NSG tool defines that the size of a physical channel in the network is bounded to a flit, therefore one phit equals to one flit. By default, a packet size is constrained up to 128 flits and the size of every flit is 64 bits (this depends on the configuration of the target description XML file).

Every packet starts with two individual flits following by data flits. As depicted in Figure 3.5, the first flit that is sent out, is the Setup Flit which contains valuable information about how to be processed and about the re-composition of the packet.

3.3.2 Flits

As described in the previous section, a flit consists from 64 bits, where 32 bits are the payload, the actual data for the processes, and the rest are routing information. Every flit can be divided into logic segments that represent useful information.
3.3. MESSAGES

Figure 3.5: Packet structure. Up to 128 flits form a packet.

about the type of the flit, routing information etc. A precise analysis of the flit’s segmentation is demonstrated in Figure 3.6.

Figure 3.6: Flit segmentation

- **Type**: A flit can represent 4 types. Empty=0-, Valid=1-, Setup=11, Data=10. Size=2 bits.
- **Flit_ID**: The unique identifier of ever flit. Size=7 bits.
- **PID**: The process ID which sent the flit. Size=7 bits.
- **HC**: The hop counter(ex.dedicated for lifelock avoidance). Size=8 bits.
- **NS**: This represents the destination X-coordinate(column) of the target node. Size=3 bits.
- **EW**: This represents the destination Y-coordinate(row) of the target node. Size=3 bits.
• **UD**: This represents the destination Z-coordinate(layer) of the target node. Size=3 bits.

• **Payload**: The actual data sent by the process. Size=32 bits.

### 3.4 Routing

Routing algorithms discover and determine a path from a source node to a destination node in a particular topology. The NSG tool, in order to comply with the specification of the Nostrum NoC (Section 2.6), utilizes a deterministic and minimal routing algorithm. The simplest and inexpensive to implement (in terms of area) routing algorithms are deterministic which keeps the area overhead of the switch low. Furthermore, in networks without the ability to store the packets into buffers (such as Nostrum NoC), a modification in the policy of the routing algorithm is mandatory, in order to avoid dropping the packet due to congestion.

For those reasons, the switch of NSG tool implements a combination of the Dimension-Order routing for its simplicity and the Deflection routing to eliminate the need of buffering packets. The following two subsections describe in brief these two policies.

#### 3.4.1 Dimension-Order Routing

*XY routing* [37] [9] is considered to be a type of Dimension-Order routing which belongs to the category of distributed deterministic routing algorithms. This is a typical turn algorithm based on the minimal path discovery, and it is commonly used in mesh and torus topologies.

![Figure 3.7: XY-routing a packet from switch (0,0) to switch (2,2) in a 3x3 2D-mesh.](image)

In a 2D mesh, each switch can be identified from its coordinates (x[=column], y[=row]) (Figure 3.3). Every switch decomposes the incoming packet and analyzes the header flit to identify the destination switch address for the packet. The routing algorithm performs a comparison between the destination switch address (Dx, Dy)
3.4. ROUTING

and the current switch address (Cx, Cy). It first computes, if the Dx=Cx and Dy=Cy in order to forward the packet to the resource. If it is not true, then it checks if Dx ≠ Cx and choose to route the packet to the East port if Cx<Dx or to West port if Cx>Dx. In the case of Cx=Dx, the packet is already aligned in the preferred row, and now it must be examined for the vertical axis. Therefore, the previous procedure is repeated again for the Cy against Dy accordingly, in order to choose the North or South port. An example of the XY-routing algorithm is illustrated in Figure 3.7 where the switch (0,0) injects into the network a packet with the destination address (2,2). Firstly, the packet follows the horizontal axis until Cx is aligned with the Dx, and then it runs on the vertical axis to arrive at the Dy coordinate.

Figure 3.8: A situation with packet contention. Deflection routing is performed on the packets A.

3.4.2 Deflection Routing

A network with bufferless flow control must deal with the issue when two or more packets may arrive at a node and desire to be forward to the same output port. In such a situation, there is a contention between the packets, and a misrouting technique must be performed.

The deflection routing [4] [14], as known as hot-potato routing, implements a form of packet routing due to packet contention, in which two or more packets wish the same output channel and the switch forward one of them to the ’correct’ output channel, deflecting the others away from the preferred port and consequently away from their destination. Due to the absence of buffers and queues at intermediate nodes, the packets must be handled and processed immediately by the switch which has already precomputed alternative output directions for every packet in case of contention. It is also guaranteed that, if a switch has equal number of input and output channels, every incoming packets has at least one alternative way out direc-
tion from the switch. Thus, the packets, instead of being stored and waiting in the switch, are always moving into the network, eliminating the problem of deadlock. However, livelock is still a potential issue that must be avoided by some specific deflection rules [14].

Figure 3.8 depicts the contention that occurs between packet A and packet B. Both of these packets have the same destination address (1,2). When they arrive at the node (1,0), they wish both to be forwarded from the North output port. This condition enables the hot-potato routing to forward normally the packet B to its preferred output channel and deflect the packet A to the East output port.

### 3.5 Reference Switch of NSG Tool

The NSG tool implements a reference switch, based on the two routing policies which were mentioned in Section 3.4.1 and Section 3.4.2. It consists of three main components (Figure 3.9): the receiver(recv), the crossbar(FSM) and the transmitter(xmitter). In order to model the behavior of this digital unit, VHDL was used as the main hardware description language to describe the particular digital system on FPGAs.

![Figure 3.9: Schematic of the reference switch of NSG tool.](image-url)
3.5. REFERENCE SWITCH OF NSG TOOL

3.5.1 Receiver unit

The receiver is the functional unit that is responsible to receive flits from the network through the input channels, store them into temporary buffers, decode the destination headers of the flits, forward the whole flit to the crossbar and create the preferred output direction matrix of each incoming port, based on the flit’s headers, for the crossbar (Figure 3.9).

There are seven instances of the unit receiver in the switch, each of these is assigned to an input port (North, South, East, West, Up, Down, Resource), abbreviated as N, S, E, W, U, D, R accordingly. The current version of the switch can support 1D, 2D and 3D Mesh topologies.

Every instance of the receiver stores, into a 64-bit buffer, the incoming flit and immediately proceeds to the decoding of the Type and Destination address segment (NS, EW, UD) (see Figure 3.6). The Type segment will inform the switch if the flit is valid to be analyzed further. If it is valid, then the destination address is extracted, and it is available to the routing algorithm to make routing decisions.

The routing decisions are based on the deflection algorithm and XY-routing mentioned in Subsection 3.4.2/3.4.1 and it is implemented in the receiver as a hard-coded predefined routing table (Table 3.1). The destination address of each flit is translated according to the absolute addressing policy (Figure 3.3) and it is compared against the coordinates of the current switch to identify the preferred output direction matrix.

As presented in Table 3.1, the receivers examine the Dx (X-coordinate of the destination address - NS/1st column), the Dy (Y-coordinate of the destination address - EW/2nd column) and the Dz (Z-coordinate of the destination header - UD/3rd column) with the current coordinates (row, column, layer) of the switch to conclude to an ordered list of desired output ports (switch matrix - 4th column) that the flit wished to be forwarded. The output of every transmitter is a list of directions with order of preference (N, S, W, E, U, D, R) that wants to be redirected. This means that every flit has, not only one output direction, but 5 more alternative paths and the crossbar can choose from 6 possible directions instead of dropping the flit in case that a desired output port is occupied.

This implementation gives to the switch the ability to assign to every incoming packet at least one output port to be transmitted. This is accomplished through the crossbar unit (Figure 3.9), which controls and assigns connections between the incoming channels to the correct outcoming channels based on the switch matrices that have been produced by the receivers.

3.5.2 Transmitter unit

The transmitter (xmitter) is also a functional unit that buffers the flits from the crossbar unit and it is responsible for injecting them into the network. Each switch contains seven instances of the transmitter unit which control the output channels. This unit does not make any routing decision, instead, together with the receiver...
### Table 3.1: Routing table

<table>
<thead>
<tr>
<th>Dx</th>
<th>Dy</th>
<th>Dz</th>
<th>Switch matrix</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>NS &gt; row</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EW &gt; column</td>
<td>UD &gt; layer</td>
<td>N, E, U, S, W, D</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>EW &lt; column</td>
<td>UD &lt; layer</td>
<td>N, E, D, S, W, U</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>EW = column</td>
<td>UD = layer</td>
<td>N, E, S, W, U, D</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td><strong>NS &lt; row</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EW &gt; column</td>
<td>UD &gt; layer</td>
<td>S, E, U, N, W, D</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>EW &lt; column</td>
<td>UD &lt; layer</td>
<td>S, E, D, N, W, U</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>EW = column</td>
<td>UD = layer</td>
<td>S, E, N, W, U, D</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td><strong>NS = row</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EW &gt; column</td>
<td>UD &gt; layer</td>
<td>E, U, N, W, S, D</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>EW &lt; column</td>
<td>UD &lt; layer</td>
<td>E, D, N, W, S, U</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>EW = column</td>
<td>UD = layer</td>
<td>E, N, W, S, U, D</td>
<td>19</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.1: Routing table
3.5. REFERENCE SWITCH OF NSG TOOL

units, handle the flow control of the incoming/outcoming packet in/out of the switch (Figure 3.9).

The receiving and transmitting processes, which is performed by the transmitter/receiver accordingly, are triggered by the control unit of the crossbar. The time needed for the crossbar to process all the flits and make the multiplexing between input and output ports, defines the throughput of the switch and hence the performance of the network.

3.5.3 Crossbar unit

The crossbar component consist of the control unit of the switch. It is associated with the network, through the receivers(recv) and transmitters(xmitter) which it controls by triggering specific control signals. This unit is responsible to handle the generated switch matrices by the receivers(recv) and produce a new switch matrix which will be the base for the multiplexing assignment between input and output port. To achieve that, it implements a FSM which creates the new matrix and control also the signals that enable the transmitter/receiver units.

![Figure 3.10: The Finite State Machine (FSM) of the crossbar.](image)

In the FSM of the crossbar unit, illustrated in Figure 3.10, there are 4 states of elaboration. The first two states retrieve the data from the generated switch matrices and create a new list of output ports that are correlated with the input ports. The last two states control the enable signals for the receiver/transmitter units and also the signals to the RNI (Figure 3.10).

- **State 0.** The preferred switch matrices from the seven receivers are collected and are stored internally to the crossbar unit for further investigation. The FSM has a fixed priority of serving (N->S->E->W->U->D) and assigning the input to the output channels. The primary procedure investigates if an output direction from the switch matrix of a receiver could be assigned to this particular receiver (input channel). If another input channel occupies the
chosen output direction from the list, then it examines the second preferred alternative direction and so on. Finally, it has been through all the switch matrices generated by the receivers and has created a new switch matrix (called nxt_switch_matrix) which contains information about the multiplexing between input and output ports.

- **State 1.** The nxt_switch_matrix is examined to detect if there is any link between an input port and the Resource (R) port in order to inform the RNI that a flit has been sent to it. This is accomplished by triggering a special signal called write_R. In the case that, the RNI has sent to the switch a flit, an acknowledgment must be performed to notify the RNI that the flit has been processed properly. The special signal for that purpose is the write_R.

- **State 2.** The RNI requires that the duration of the write_R signal must be at least two clock cycle in order to be detectable.

- **State 3.** In the last state; the crossbar has completed the assignments of input to output ports, and it is ready to receive/transmit new flits. Therefore, it activates the receiver unit, to accept new flit, and the transmitter to forward the old flits.

### 3.5.4 Conclusion

The crossbar unit can perform and complete the multiplexing process in 4 clock cycles. This duration could be reduced to three clock cycles if the RNI could identify the write_R signal in one clock cycle. In addition, the routing decision executed by the switch are predefined and cannot be change in any time, meaning that in the presence of broken communication links, the switch will try to transmit packets from the faulty links and therefore the packets are going to be dropped. Although this implementation produces a small, in terms of area and fast switch for fault-free NoC, it does not support the correct functionality of switching in case of permanent faults.

Furthermore, the implementation of the routing table in the receiver units adds extra logic that could easily be replaced by memories that will contain that information. This approach removes the routing information that is hard-coded in the switch and places it to a reconfigure component which can be easily updated or modified.
Chapter 4

Reconfigurable Fault-Tolerant Switch

In this chapter, we will describe how the reference switch discussed in Section 3.5, has been updated to support an adaptive routing policy in case of broken communication links. The routing tables have been implemented in memories which, later, can be reconfigured from the RNI, therefore two versions have been implemented. The version 1 will use the same routing policy with the reference switch, but the implementation of the routing table will be different. In the version 2, the switch will adopt a new routing policy and implementation which is based on distance vectors and fault-tolerant technique and distribution of fault information mechanism based on the Q-learning algorithm [35].

4.1 Switch v.1

The goal of the v.1 implementation is to investigate if the utilization of the FPGA resources (memory, LEs, etc) increases or decreases by eliminating the presence of a hard-coded routing table in the receivers. Table 3.1 describes how the receiver chooses a switch matrix among the 27 possibilities. This process was implemented by describing every comparison with hardware logic. Instead of having hardware logic to describe information, memory elements could be used.

It is observed that with a particular routing algorithm (deflection routing), the number of switch matrices that can exist is fixed and independent of the network’s size. All the routing information ,(refer to the 4rd column in Table 3.1 ), can be stored to a memory and can be retrieved, later on, by a hash function that will be the only hardware logic to decide which switch matrix will be generated.

4.1.1 Memory & Hardware Logic

The switch matrix contains information about the desired output direction of the receiver, and it is described as a list of maximum seven elements. This list associates each receiver with an ordered list of preferred transmitters. In the switch, there are 7 transmitters, so every output direction can be coded with 3 bits and a switch
matrix (7 elements) with 21 bits. Thus, a memory with 27 lines by 21 bits each line can be used to store the routing table. The memory, depending on the address that is delivered by the receiver, returns a specific memory line which represents the ordered list of output directions. In the switch, there are seven memories, each of them is associated with a receiver unit (Figure 4.1). The output of the memories goes directly to the crossbar in order to start the input/output assignment process as it was described in Section 3.5.3.

Now that the routing table has been extracted from the receivers and has been placed into memories, a new hardware logic is needed to correlate the destination address of the flit with an index in the memory. The memory contains exactly the same sequence of data with this indexing, as it is presented in the 4th and 5th column of Table 3.1. The sample code 4.1 describes the function that generates the index address for the memory and this logic is placed into the receivers.

For example, if a receiver (with current coordinates row, column, layer = 0,0,0) gets a flit with destination address (NS, EW, UD)=(2,2,0), it will generate a switch matrix (N, E, S, W, U, D) based on reference hard-logic process. The new logic will produce, index_ns=0, index_ew=0 and index_ud=2. So we have for the address \(\text{index} = 0 \times 9 + 0 \times 3 + 2 = 2\) which is the index for the memory with the same result, as with the old logic.

The memory has been implemented as an asynchronous ROM which contains a constant array of arrays. Each of those arrays contains seven values of 3 bits each value. Write operations are not supported in the current version.

Another approach would be to reduce the number of the memories from 7 to 1,
in order to minimize the overall size but the drawback would be to introduce extra
logic and extra clock cycles and states into the crossbar unit in order to manage 7
different requests from the receivers.

Listing 4.1: Hash function to compute the memory address

```vhdl
index_ns <= 0 when NS > row else 1 when NS < row else 2 ;
index_ew <= 0 when EW > row else 1 when EW < row else 2 ;
index_ud <= 0 when UD > row else 1 when UD < row else 2 ;
address <= 27 when (( reset = ’1’) or ( valid = ’0’)) else (
    (index_ns*9) + (index_ew*3) + index_ud ) ;
```

4.2 Switch Proposal v.2

The presence of faulty communication links into a regular network form a new
topology in which low cost routing techniques such as XY routing is inefficient. An
efficient way to face this issue is generality. A routing chip that implements a dis-
tributed, adaptive and table-based [13, p.203] routing algorithm based on distance
vectors, it is a promising solution for irregular topologies.

- **Distributed Routing**: As the packets travel across the network, the routing
decision are made in every switch by the routing function. In this schema,
the header of the packet contains only the destination address of the target
switch.

- **Adaptive Routing**: It can be combined with the Distributed routing to make
use of the information about the channel status of its neighbors to avoid faulty
regions. It can be divided into two main parts, the routing function and the
output selection function. The routing function produces a set of possible
output channels for a particular packet. The output selection function selects
one free output channel from the set, using local status information.

- **Table-based Routing**: It provides the means for the routing function to
decide among several possible directions, making use of per-hop network state
information [13, p.208]. By reprogramming the contents of the table, this
routing can be applied to different topologies.
• **Distance Vectors Routing**: Every switch maintains a vector containing the distances to all other switches and can also distribute that vector to its immediate neighbors.

![Figure 4.2: Schematic of the switch version 2](image)

### 4.3 Routing scheme

#### 4.3.1 Adaptive routing

The basic idea is to implement an adaptive routing algorithm based on the reinforcement learning approach [35]. The Q-routing algorithm learns a routing policy and makes routing decisions using only local information about the number of 'hops' that a packet needs to travel to the destination node. Every node that implements this technique has an initial routing table with the time required to every other switch. Suppose that $Q_x(d, y)$ is the time needed for a packet to travel from node $x$ to $d$ through the node $y$. When node $y$ receives the packet, it immediately transmits back to node $x$ the estimated delivery time, for this packet, from node $y$ to node $d$. This can be described as following:

$$ t = \min Q_y(d, z) $$

where $z$ is neighbor of $y$. So, node $x$ can revise its estimated delivery time to node
4.3. ROUTING SCHEME

d as:

\[ \Delta Q_x(d, y) = \eta(q + s + t - Q_x(d, y)) \]

The parameter \( \eta \) declares the "learning rate" which means that a factor of 0 will make no learning, \( q \) is the time spending a packet into the \( x \)'s buffer queue and \( s \) is the time needed for the packet to travel from node \( x \) to node \( y \). This formula can be simplified by bounding the parameter \( \eta = 1 \) which makes the reinforcement learning continuous, the parameter \( q=0 \), as there aren’t buffer queues in the switch and the \( s=1 \) because it can be assumed that the transmission time between two nodes is equal to 1 hop. Therefore, we can conclude to \( Q_x(d, y) = 1 + Q_y(d, z) \) [35] [16].

4.3.2 Routing tables

The \( Q \)-routing requires that the nodes be supplied with local information about the hop distance to every other node. As it was described in the previous section, it is assumed that the transmission time for the \( Q \)-routing algorithm is equal to 1 hop distance. This information can be represented by a 2D array and for a given topology, such as \( n \times n \) mesh, there will exist \( n \times n \) entries in the routing table. In addition, the routing table describes the hop distance from every output port of a particular switch to all the other switches. There are also special hop distance values, such as the value \( 0 \) which declares that the packet must be forwarded to the Resource and the value \( 255 \) which declares that the communication link of the particular output direction is broken (or not used) and therefore the switch cannot forward the packet to that specific direction.

<table>
<thead>
<tr>
<th>Routing table</th>
<th>North</th>
<th>South</th>
<th>East</th>
<th>West</th>
<th>Up</th>
<th>Down</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch 0</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>Switch 1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>Switch 2</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>Switch 3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td><strong>Switch 4</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>Switch 5</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>Switch 6</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>Switch 7</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>Switch 8</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>255</td>
<td>255</td>
</tr>
</tbody>
</table>

Table 4.1: Routing table of switch 4 in a 3x3 mesh

For example, in a \( 3 \times 3 \) mesh topologies, there will be nine entries in the routing table, as illustrated in Table 4.1. The initial routing table for every switch is fixed, but it can be updated with the \( Q \)-routing algorithm.
4.4 Fault-tolerant mechanism and Reconfiguration

The reference switch discussed in Section 3.5, has been modified in order to meet the requirements of a reconfigurable fault-tolerant switch. The schematic of the switch version 2 can be viewed in Figure 4.2. Significant modifications have been done in the type and number of the memories, the FSM and some additional input/output signals.

![Figure 4.3: Representation of the hop information in the memories](image)

The memories have been implemented as a 64-bit Dual-port RAM which allows any independent read or write operation in the same clock cycle. With this configuration, only 4 Dual-port RAMs will be needed for the switch. Each memory element will be connected with a pair of inputs channels (N/S, E/W, U/D, R).

The information will be coded as it is depicted in Figure 4.3. Every line in the memory represents the estimated hop distances from the output ports to a specific destination and every segment in the line of the memory represents the hop distance for this particular output port to the destination. The size of each segment is 8 bits. Thus the maximum hop distance that can be represented is 255.

Special cases has been decided to be when a hop distance equals 0 or 255. In the first case, the destination address of the packet matches with the current address of the switch and in the second case, the particular output channel is marked as no-link connection or faulty. By observing the Table 4.1, which depicts the routing table of the switch 4 in a 3×3 2D-Mesh, we notice that the two last columns are full of hops=255 which means that there aren’t connections to other layers as we have a 2D topology. Moreover, if switch 4 receives a packet with destination address of switch 4, the routing table will return hops=0 which declares that the packet should not be forwarded to another switch and must be delivered to the Resource of the current switch.

The FSM has also been modified to comply with the Q-routing policy. The new FSM encounters 7 states (refer to 4.4) in which it processes the switch matrices, update the memories if needed, makes the multiplexing and send out the hop distances to the neighbors. Refer to Figure 4.2 for the units or signals mentioned in the next paragraphs.

- **State 0**
4.4. FAULT-TOLERANT MECHANISM AND RECONFIGURATION

Figure 4.4: FSM of the switch version 2

The packets have just arrived at the receivers, and the headers are been analyzed. The destination addresses will be extracted from the header of the packet and these are the index addresses for the memories (absolute addressing technique). Every receiver inspects the header of the packet to determine if there is a valid bit in order to activate the corresponding memory for read operation. It also informs the FSM, if there is a special packet (with Type='01') which contains information to update the routing table. This update packet targets the switch and it is not forwarded further into the network.

- **State 1**

  The memories, which have been activated, produce a set of hop-distance arrays (every hop-distance array is associated with the input channel through the memory). The hop-distance arrays are loaded from the RAMs and are stored internally in the FSM. In this point, 6 new signals are used (fault_from[5:0]) which correspond to the link status of each communication link. If there is a broken link, for example, in the North(direction=0) channel then the fault_from(0) will be enabled. Consequently, this signal will mask every hop-distance in the array that represents the hop-distance for the North output port and will overwrite it with the value 255.

  In this State, there is the possibility to have a special packet that can arrive only from the Resource. This packet is identified by the Type='01' and is tagged as "update" packet. The update packet contains a hop-distance array that is going to update a specific line in the routing table. The update packet can support, for the time being, only 2D-Mesh topology and is structured as a typical packet with the only difference that the payload contains the hop values that will be stored into the routing table.

- **State 2**

  39
Once the hop-distance arrays are available, two processes are initiated in this State. The first process iterates every hop-distance array and retrieves the minimum hop that exists in the particular table. After that, the \( Q_{\text{values_to[direction]}} \) signal is set to this minimum hop in order to inform the neighbor, who have sent the packet, for the estimated hop-distance of the packet that is sent. The second process starts a sort procedure one every hop-distance array based on the hops. The final output of this process would be a new sorted array with incremental order of hops and directions which are associated with these hops.

- **State 3**

  The \( Q_{\text{value_to}} \) signal, triggered in the previous state, is associated and presented in the \( Q_{\text{value_from}} \) signal. This signal is used from the sender of a packet to check if the output direction that was chosen to forward this packet, has a hop distance equal to \( Q_{\text{value_from[direction]}} + 1 \), based on the formula discussed in Section 4.3.1. If there is a mismatch in the hop for a particular direction, it means that the available routes to that destination have changed, and it needs to update the local routing table with the new information.

  In this state, the finale switch matrix is created based on the sorted hop-arrays. All the hop-distance (output directions) arrays are now sorted. The multiplexing process starts to serve the input channels with a fixed priority (N->S->E->W) and try to assign to each input channel, an output direction. Every time, it iterates every hop-distance array to find a free output path. If it exists then it assigned that input channel to a particular direction. If the output direction for this hop is occupied, then it goes to the next hop which is associated with another direction and repeats the steps above. Finally, a switch matrix will be created which will connect the input to output port. In this State, the \( \text{read}_R \) signal is triggered if we have a connection from the Resource port to an output port, in order to notify the RNI that the packet has been transmitted.

- **State 4-5**

  Based on the 2D-Mesh topology, there can be maximum four \( Q_{\text{value_from}} \) that differ from the hop values in the local routing table. In this situation, the memories need to be updated with the new values. This is accomplished by making use of the dual-port property of the RAMs, which permits to have two simultaneously write operation on the same clock cycle. So, for four values, 2 States are needed.
4.4. FAULT-TOLERANT MECHANISM AND RECONFIGURATION

If we have a connection from an input port to the Resource port, then the $write_R$ signal is enabled to inform the RNI that a packet has been stored into buffers and must be proceeded. These states are also reserved for future work. For example, to support 3D-Mesh topologies.

- **State 6**

  As in the switch version 1, the $recv_enable$ and $xmit_enable$ signals are triggered to activate the receiver/transmitter unit to buffer new packets.
4.5 Prototype

The switch version 2 has been implemented, using the hardware description language VHDL. For validation purposes, ModelSim-6.5e Altera was used to simulate a complete system. The test involves the generation of a 3×3 2D-Mesh with fixed faulty communication links. As shown in Figure 4.5, the link presented with red color are broken. All the switches do not have knowledge about these faults in the network. This test demonstrates what will happen when switch 0 wants to send a packet to switch 8.

Figure 4.5: Simulation test for 3x3 2D-Mesh with faulty links.

After running the simulation, the path that has followed the packet is depicted in Figure 4.6.

Figure 4.6: The route of a packet with the presence of faults.

The signals selected in the waveform are the connection channels of every switch into the network. We can observe that the packets was routed from switch 0 -> switch 3 -> switch 6 -> switch 7 -> switch 4 -> switch 7 -> switch 4 -> switch 7 -> switch 4 -> switch 1 -> switch 4 -> switch 1 -> switch 2 -> switch 5 ->
4.5. PROTOTYPE

Figure 4.7: The route of the 2nd packet with updated routing tables.

switch 8. The bouncing between the nodes is caused because of the Q-routing that explores and updates the routing table of every switch. After this learning period has been completed, it can be noticed that the second packet follow this path without bouncing, and this is because the routing tables have been updated with the correct Q-values. This can be verified in Figure 4.7.
Chapter 5

Conclusion

The switch proposal 2 has been tested and successfully met the requirements of a fault-tolerant switch which can route the packets from a set of alternative paths, in the presence of permanent faults in the communication links. This switch makes routing decision based on the local information and can deliver packets in any node of a faulty network, as far as, the network remains connected.

For comparison purpose, the reference switch, switch version 1 and switch version have been synthesized in Quartus II for the Cyclone IV E FPGA family, in order to compare the size overhead of different switch proposals (Table 5.1).

<table>
<thead>
<tr>
<th></th>
<th>Ref. Switch</th>
<th>Switch v.1</th>
<th>Switch v.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total logic elements</td>
<td>2,817</td>
<td>2,790</td>
<td>3808</td>
</tr>
<tr>
<td>Total registers</td>
<td>820</td>
<td>822</td>
<td>830</td>
</tr>
<tr>
<td>Memory</td>
<td>-</td>
<td>4032</td>
<td>1152</td>
</tr>
<tr>
<td>Throughput</td>
<td>1pkt/4cc</td>
<td>1pkt/4cc</td>
<td>1pkt/7cc</td>
</tr>
</tbody>
</table>

Table 5.1: Area comparison and throughput between 3 different switches.

The switch version 1 offers a same solution, in terms of area, compared with the reference switch which both implement the same functionality. The absence of a hard-coded routing table doesn’t reduce significantly the area of the switch. The fault tolerance approach come with a small drawback of area overhead. As it can be part of the NSG tool, the switch version 2 can be preferred, in case the environment in which the system is operating, is harmful because of electromagnetic fields. In that case, the system must ensure the proper functionality, even there are faults somewhere into the system. The first two solutions give a low-cost switch but with limited functionalities.

It must be mentioned that the size of the memories for the switch version 2 is proportional with the number of the switches into the network. For example, a switch in a NxN Mesh will need a memory with a size of NxNx64b. The difference between the size of memory in switch v.1 and switch v.2 is due to the fact that the
version 1 implements 7 single-port ROMs. In contrast with the switch version 2 which uses 4 dual-port RAMs.
Chapter 6

Future Work

There are several issues that need to be solved in order to have a complete schema of a fault-tolerant switch. The most important is the need for a fault detection mechanism which together with the fault-tolerant routing, will protect and ensure the reliability of the system. The fault detection mechanism is an important unit that decides for the presence of faults into the network, and the fault-tolerant routing is the one that avoids these faults. So there is an interdependent relation between these two mechanisms.

Another advantage, that can be exploited, is the hop information provided by the routing tables in every switch. This information combined with the implementation of an internal soft virtual channel into the switch can provide a predictable system. After a learning period, all the information for the hop distance to every node into the network will be available to all the switches. The main issue is to ensure that the message, we want to deliver in a predicted time, is going to follow the minimal path through the network. The first packet could contain a set up message, that will inform all the switches in the path that needs to be served first among all the other incoming packets. The switches must set up an internal virtual channel (ex. reorder the way it serves every incoming port) between the incoming and outgoing port that the packets used. It will also assign a priority to this port in order to serve it first. If there is a second setup packet, then the switch will add to the priority list of incoming ports, the port used by this packet and also make the inter-virtual channel. This may not provide the minimal path, but it will guarantee the predicted delivery time.

The fault-tolerant switch implementation should be upgraded in order to support 3D-Mesh. The current version relies only on the 2D topology. The extended version will also eliminate the possibility of livelocks by making use of the different layers.

It must be also verified the correct functionality of the system in case of faults that occurs on-the-fly. In the presented system, the faults are manually introduced
at the start up of the system and it could not be injected during the operation. In addition, the correct functionality of the update packets must ensured through on-the-fly tests.
Bibliography


BIBLIOGRAPHY


[33] Tanwar, V. Xy routing algorithm.


