University of Ferrara

Engineering Department of the University of Ferrara

Doctorate Degree in Science of Engineering

Coordinator: Prof. Stefano Trillo

Cycle: XXVI

# CROSS-LAYER DESIGN, OPTIMIZATION AND PROTOTYPING OF NoCs FOR THE NEXT GENERATION OF HOMOGENEOUS MANY-CORE SYSTEMS

ING-INF/01

Candidate:
Hervé Tatenguem Fankem

Advisor:
Prof. Davide Bertozzi

Academic Year 2011/2013

## Acknowledgements

First of all I would like to thank my advisor Prof. Davide Bertozzi. He has taught me, how to do a good research. I appreciate all his contributions of time, ideas, and funding to make my Ph.D. experience productive and stimulating. The joy and enthusiasm he has for his research was motivational for me, even during tough times in the Ph.D. pursuit. The members of the MPSoCs group have contributed immensely to my personal and professional skills. The group has been a source of friendships as well as good advice and collaboration. I am especially grateful to Daniele, Alessandro, Luca, Alberto, Marco and Gabriele. I would also like to acknowledge Jann Raik and the Tallinn Institute of Technology where I spent my internship. For this dissertation I would like to thank my reading committee members for their time, interest, and helpful comments. I gratefully acknowledge the funding sources that made my Ph.D. work possible. I was funded by the E.U. NaNoC and vIrtical Projects. My time at UNIFE was made enjoyable in large part due to the many friends and groups that became a part of my life. I am grateful for time spent with colleagues and friends, for our memorable trips into the cities. Lastly, I would like to thank my family for all their love and encouragement. For my parents who raised me with a love of science and supported me in all my pursuits. For the presence of my brothers/sister and most of all for my, supportive, encouraging, patient and loving Reine Flore whose faithful support me during my Ph.D.

Thank you.
Hervé Tatenguem Fankem

# Contents

# List of Figures

# List of Tables

# Abstract

This thesis provides a whole set of design methods to enable and manage the runtime heterogeneity of features-rich industry-ready Tile-Based Network-on-Chips at different abstraction layers (Architecture Design, Network Assembling, Testing of NoC, Runtime Operation). The key idea is to maintain the functionalities of the original layers, and to improve the performance of architectures by allowing, joint optimization and layer coordinations. In general purpose systems, we address the microarchitectural challenges by co-designing and co-optimizing feature-rich architectures. In application-specific NoCs, we emphasize the event notification, so that the platform is continuously under control. At the network assembly level, this thesis proposes a Hold Time Robustness technique, to tackle the hold time issue in synchronous NoCs. At the network architectural level, the choice of a suitable synchronization paradigm requires a boost of synthesis flow as well as the coexistence with the DVFS. On one hand this implies the coexistence of mesochronous synchronizers in the network with dual-clock FIFOs at network boundaries. On the other hand, dual-clock FIFOs may be placed across inter-switch links hence removing the need for mesochronous synchronizers. This thesis will study the implications of the above approaches both on the design flow and on the performance and power quality metrics of the network. Once the many-core system is composed together, the issue of testing it arises. This thesis takes on this challenge and engineers various testing infrastructures. At the upper abstraction layer, the thesis addresses the issue of managing the fully operational system and proposes a congestion management technique named HACS. Moreover, some of the ideas of this thesis will undergo an FPGA prototyping. Finally, we provide some features for emerging technology by characterizing the power consumption of Optical NoC Interfaces.

# Introduction

Today, many-core embedded systems are moving towards the integration of thousands of cores on a single chip. However, as the number of cores integrated into a chip increases, the on-chip communication tends to become the performance bottleneck and power hungry. To cope with the increasing demand for high performance systems, many-core designs rely on integrated network-on chips and exploit parallelism to make programs run faster. Indeed, among all feasible solutions that have been proposed to cope with the on-chip communication infrastructure, Network-on-Chips (NoCs) are the most viable solution that lead to meet the performance and design productivity requirements of a complex on-chip communication infrastructure. On one hand, NoCs provide an infrastructure for better modularity, scalability, fault-tolerance, and higher bandwidth compared to traditional infrastructures. On the other hand, developing applications using the full power of NoC-based many-core embedded systems is not trivial and requires parallel programs. Moreover, the programs to be run on the many-core chip are of varying degree parallelized and they may have different characteristics regarding processing needs, memory space and bandwidth. Above all, the efficient exploitation of the abundant hardware resources will progressively go through the sharing of such resources among a large number of concurrently executing applications. **The focus is therefore on how to manage the resources in many-core chips in response to an increasingly complex and resource-sharing workload, and how to optimize cooperation between system design layers.** There are two main examples of such resource management concern. On one hand, each core or cluster or cores will be operated at different voltages and frequencies for the sake of optimal execution and ultimately of power management. On the other hand, such splitting will indirectly relieve the clock distribution problem in large chips, which cannot be performed any more under the fully synchronous assumption. The above examples confirm that design layers are not isolated in manycore design, but have deep cross-layer implications that should be co-optimized together.

Following the same cross-layer vision, the future applications will not be able to assume that the underlying manycore computation and communication fabrics are working in their entirety. In fact, there will be an increasing role of manufacturing faults on system integrity, which calls for the relentless development of testing strategies. To ensure the required quality and reliability of such complex integrated circuits before supplying them to final users, extensive manufacturing tests need to be conducted and (absolute novelty) the associated test cost may soon account for the same share of the total production cost, as the ITRS documents start to point out.
When we combine the above (apparently different) issues together, it becomes evident that their compound effect consists of turning a fully regular and homogeneous manycore platform into a runtime heterogeneous one. In fact, the homogeneous design by construction undergoes a differentiation of operating conditions, and suffer from the regularity-breaking effect of manufacturing faults. This consideration is at the foundation of this work.

**To address all these challenges, this thesis provides a whole set of design methods to enable and manage the runtime heterogeneity of features-rich industry-ready Tile-Based Network-on-Chips at different abstraction layers (1st layer: Architecture Design, 2nd layer: Network Assembling, 3rd layer: Testing of NoC, 4th layer: Runtime Operation).**
The key idea consists to maintain the functionalities associated to the original layers, and to improve the performance of architectures by allowing interaction, joint optimization and coordination among the layers (cross-layer design).

At the architecture layer, the manycore management challenge fundamentally means the extension of current NoC architectures towards increased flexibility, reconfigurability and /or notification capability. These terms assume a different meaning depending on the NoC application domain. In general purpose systems, the microarchitectural challenges consist of augmenting regular tile-based NoCs into systems capable of runtime reconfiguration of the routing function, of transient fault notification, and of some form of fault-tolerance capability. This motivates the effort presented in this thesis: bringing state-of-the-art NoC architectures into the next generation of industry-ready NoCs. This essentially goes through the design of feature-rich architectures where the different features are co-designed and co-optimized together. Vice versa, in application-specific NoCs the reconfigurability requirement is (and will be) still far away, while instead the emphasis will be on event notification, so that the platform is continuously under control.

At the network assembly level, the large size of manycore systems and the unpredictability of the underlying silicon technology raise unprecedented compositional challenges, which are fundamentally physical design and design technology issues.

For example, clock variability causes timing issues, as the clock skew affects the timing in two ways: setup-time wise and hold-time wise.

While setup-time issues cause the system to function at reduced performance, hold-time violations may render a system dysfunctional.

The above problems statistically show up at switch boundaries, since switches are separated apart in real layouts. Therefore, it is at the switch boundaries (that is, inter-switch communication) that the above issues should be mainly addressed. This thesis proposes a HTR (Hold Time Robustness) technique, that leads to tackle the hold time issue in synchronous NoCs links. At the network architectural level, manycore design pose the challenge of the choice of a suitable synchronization paradigm. On one hand, such large systems may break the fully synchronous assumption by means of mesochronous clocking. This paradigm today requires a boost of synthesis flow as well as the coexistence with the DVFS (Dynamic Voltage and Frequency Scaling) requirement. Ultimately, this implies the coexistence of mesochronous synchronizers in the network with dual-clock FIFOs at network boundaries. On the other hand, the penetration of dual-clock FIFOs in the design may be much deeper, that is, they may be placed across inter-switch links hence removing the need for mesochronous synchronizers. This approach has fundamentally different implications both on the design flow and on the performance and power quality metrics of the network. They will be all studied in this thesis.

Once the manycore system is composed together, the issue of testing it arises. This thesis takes on this challenge and engineers a testing infrastructure, respectively on top of a general purpose and an application specific NoCs. Moreover, an ultra-low latency NoCs testing infrastructure is designed for "online testing" that cannot afford high testing cycles.
At the upper abstraction layer, the thesis addresses the issue of managing the fully operational system delivered to the end user. At this level, among all the possible runtime management issues, we focus on the congestion management problem in two different scenarios: In the first scenario, we have **multiple physical networks** (one global network and one local network composed of two virtual-channels). In the second scenario, **one physical network** of 3 VCs or 2 VCs is considered in an attempt to collapse the network. In this scenario, local and global traffics interfere not only on links but also on switches. To overcome the above issue, this thesis proposes a congestion management technique named HACS (Head-of-line Avoidance Congestion Skip-ahead), a head-of-line blocking observation mechanism that allows buffered packets to bypass the packet that is at the head of the queue. Moreover, some of the ideas of this thesis will undergo an FPGA prototyping. In fact to validate the industrial-ready NoC, **this thesis reports on the prototyping of a 16-core homogeneous multi-core processor with a faul-tolerant, runtime reconfigurable and dynamically virtualizable on-chip network**. The prototyped system will validate the NoC capability of boot-time testing and configuration, transient or intermittent fault-detection and runtime reconfiguration of the routing function. Finally, we provide some features for emerging technology. My contribution was a key enabler to facilitate the power characterization of Optical NoC Interfaces, in such a way to be able to look forward emerging optical interconnect technology. Overall, the thesis is a comprehensive contribution to the advance in the field of manycore NoC-based system design.

# Chapter 1

# Overview of Two Architectural Variants of a Mesh

## 1 Introduction

The digital design convergence, together with the new usage models of mobile devices, are raising the clear need for new requirements such as flexible partitioning, runtime adaptivity and reliability. The above trend has direct implications on the design of the underlying on-chip network, which becomes not only the system integration framework, but also the control framework executing hypervisor commands, or reacting to runtime operating conditions. The ultimate challenge for the NoC is to co-design these features together. This chapter takes on this challenge and illustrates two design experiences of a NoC switch architecture.

## 2 First Architectural Variant of a Mesh

### 2.1 Baseline Architecture

The first architectural variant (see fig.5.1) proposed in this chapter is a major extension of the baseline ×pipesLite switch [82], which targets the embedded computing domain with a very lightweight architecture.
The considered ×pipesLite variant implements logic-based distributed routing (LBDR): each switch has simple combinational logic that computes target

output ports from packet destinations and local switch coordinates. By means of 26 configuration bits for each switch (indicating switch port connectivity, routing restrictions, and deroutes), the routing function can be reconfigured at runtime[84]. The straightforward yet overly expensive way to make the baseline switch fault-tolerant is through Triple Modular Redundancy. The only advantage is that the TMR architecture can afford keeping the native STALL/GO flow control unmodified.

## 2.2 Basic Design Choices for the New Switch

The proposed switch architecture is designed to be the basic building block of a reconfigurable and fault-tolerant NoC. Reconfiguration is achieved by means of a global controller implemented in software, which requires command execution support in hardware. A dual network is therefore designed to exchange control information between switches and the global controller. Reconfigurability is implemented as runtime modification of the routing function, in order to provide not only flexible network partitioning when several applications are concurrently executed, but also to avoid faulty links/regions of the network. This latter functionality requires that points of failure are first detected, both at boot and run time, then notified to the system manager, that triggers the reconfiguration accordingly.

#### Fault-Tolerance

Whether a fault-tolerance switching strategy should affect the flow control protocol or not is a major design choice with high impact on the overall switch architecture. The work in[87] derives error recovery strategies for the same NoC switch both from a flow control protocol with error notification capability (NACK/GO) and from another one lacking this support (STALL/GO). Data retransmission is used in the former case, while the latter one can only rely on error correction. It has been shown that NACK/GO potentially results in shorter critical path, more conservative area and lower peak power, at the cost of a slight average power overhead. This led us to opt for NACK/GO for the proposed switch architecture (Figure 5.1). The proposed solution targets single event upsets (SEUs). In the data path, detectors trigger flit retransmissions from the sender buffer, which is preceded by correction of the stored flit in case it were corrupted in the buffer. On the control path, FSMs are triplicated to avoid their permanent misalignment, while routing and arbitration logic is just doubled, since dual-rail checkers (DRCs) can trigger retransmissions from the input buffer upon mismatch detection.

#### Notification Interface

In this chapter, we opt for a centralized approach to network control: a global manager is in charge of network reconfiguration decisions as an effect of fault-tolerance, power management or virtualization strategies. In order to address the need for control signaling between network nodes and the global manager, we revert to the dual communication infrastructure proposed in[89], where the main NoC is extended with a ring which connects all the switches of the main NoC together. The ring implementation implies the extension of each switch with a simple routing primitive, which is an oversimplified version of an input buffered switch.

#### Reconfigurability

The reconfiguration mechanism of the routing function in the presence of background traffic should provide deadlock freedom during the transition from one routing algorithm to another, when extra dependencies may arise and lead to deadlock. To cope with this issue, the first switch variant leverages Overlapped Static Reconfigurations (OSR), a technique which avoids draining network traffic[88]. OSR was first proposed for off-chip networks, and its customization for a much more resource-constrained on-chip setting, named OSR-Lite, has been performed in[83]. The basic principle is the following: if packets with the old routing function are guaranteed to never go behind packets using the new routing function, then no deadlock cycles can occur. In OSR this is achieved by triggering a token that separates old packets from new ones. The token advances through the network hop by hop, following the channel dependency graph of the old routing function, and progressively drains the network from old packets, allowing new packets to enter the network at routers where the token already passed.

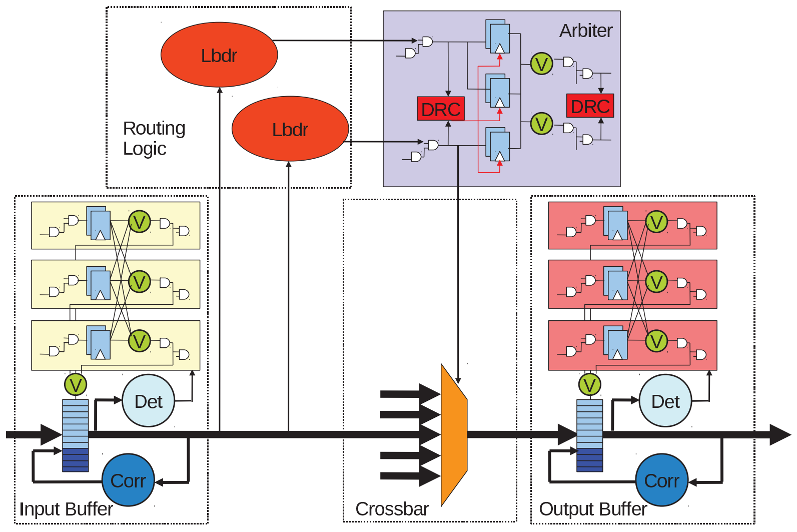


Figure 1.1: First variant: NACK/GO switch architecture.

## 2.3 Experimental Results

All the logic synthesis runs performed in this work have been carried out by means of a low-power standard-Vth 40nm Infineon technology library.

### Complexity Breakdown: Area Results

The following experiment points out both the complexity gap between the native xpipesLite switch and the feature-rich extended one, and the area increment that each integrated switch feature contributes. Normalized area results are shown in figure 5.4, where features are incrementally added to the baseline switch. This and its TMR extension are reported as reference design points. Fault-tolerance is clearly the highest-impact feature. A non-negligible area contribution comes in fact from detector and corrector modules and accounts for almost 13% of the total area. When the reconfiguration mechanism is integrated into the NACK/GO switch, an 11% of area overhead is introduced. The notification system (TMR-protected dual network) results lightweight (5% area overhead) since it takes advantages of the diagnosis logic already made available for fault-tolerance purposes. Finally, the switch capable of built-in self-test and self-diagnosis brings a 27% of area overhead (Chapter 5 describes the testing architecture on top of the first variant), which is the second major source of complexity after fault-tolerance. When we consider a baseline TMR switch which implements only fault-tolerance on top of a baseline xpipesLite switch, we can see that the proposed switch (rightmost bar in the plot) provides many more features at comparable area footprint.

### Complexity Breakdown: Delay Results

In order to evaluate the effects of each additional feature on the switch propagation delay, we performed a 5x5 switch synthesis for maximum performance for all the 5 incremental solutions under test. Results are reported in Fig.5.5. The fault-tolerant NACK/GO switch, the switch with OSR-Lite mechanism and the switch with notification system achieved a similar maximum operating speed. Finally, the testing framework degraded by 13% the maximum performance of the NACK/GO switch. The performance of the switch is limited by the test-wrappers placed on the critical path. Considering the TMR solution, this is around 30% slower than the baseline switch while the proposed switch delivers far more functionalities at the cost of a longer critical path (+13%).

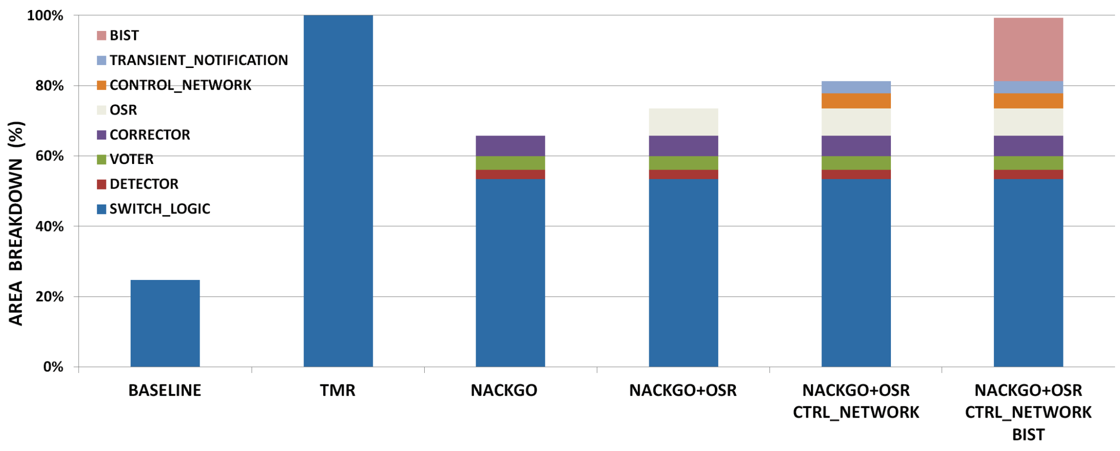


Figure 1.2: Area analysis.

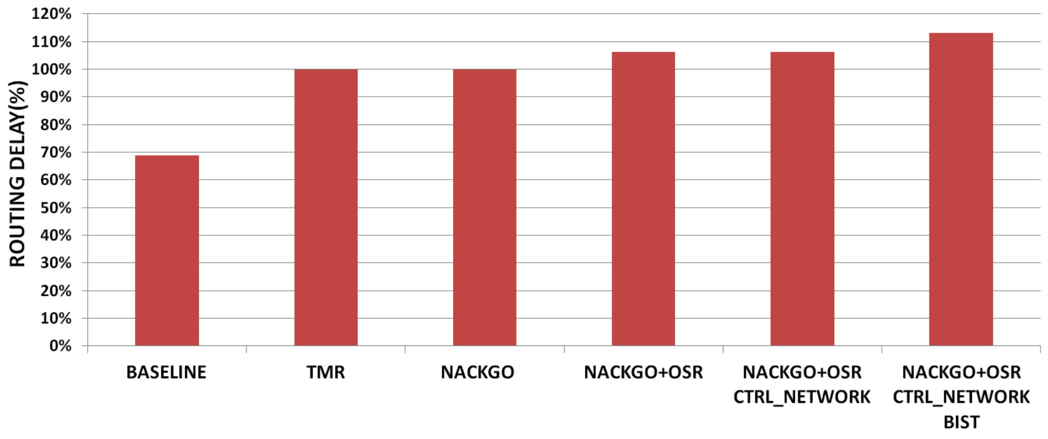


Figure 1.3: Routing delay analysis.

## 3 Second Architectural Variant of a Mesh

The second variant is a parameterized $n \times m$ (n: number of input ports, m: number of output ports) source based routing 2-stage switch, augmented with fault-tolerance provisions. The scheme of the switch architecture is depicted in figure 6.2 and is composed of the following main blocks:
- A fault tolerant input buffer of two slots with triplicated control logic and endowed with voters.
- A fault tolerant output buffer of six slots with the same characteristics of the input buffer.
- A fault tolerant arbiter, triplicated and endowed with voters.
- A Path-Shift module and a Crossbar.
- Some comparators are placed in specific places for runtime diagnosis and to notify the global manager.

The next section describes the behaviour of each block of the switch:

### 3.1 Tightly Coupled Dc_FIFO

Synchronization interfaces, such as dual-clock FIFOs, are typically instantiated as external blocks with respect to the module they are connected with. This "loose coupling" of synchronizers with respect to NoC components implies several drawbacks. First, the FIFO module introduces additional communication latency in the intercommunication link. As a result, provisions

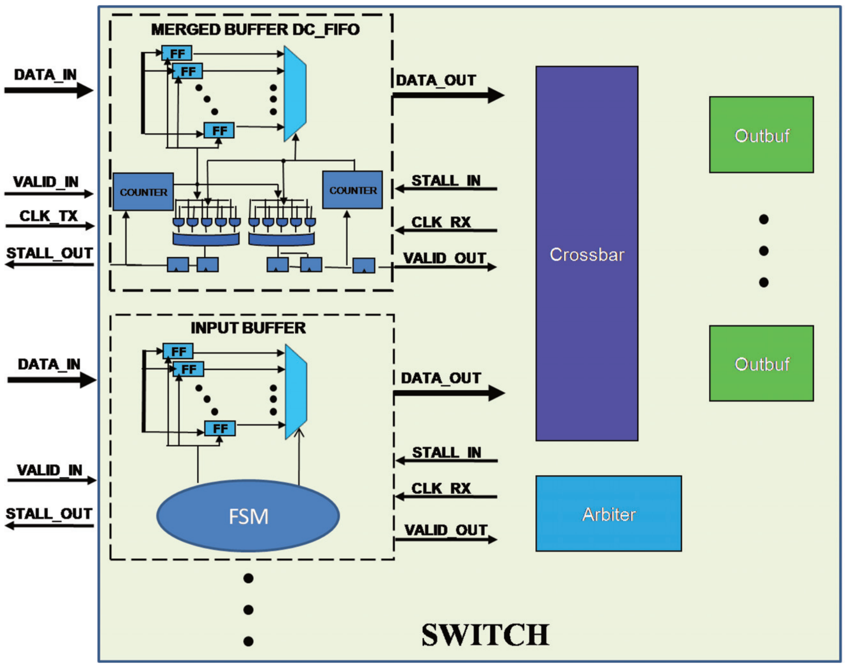


Figure 1.4: Dual-Clock FIFO integration into one input port of the switch architecture.

must be normally made since the flow control signal may arrive multiple clock cycles after the destination module decides to halt the source module. The problem can be addressed by reserving space in the destination buffer, thus incurring a significant area and power overhead, or by enhancing the dual-clock FIFO with flow control capability.

In the EU-funded GALAXY project, the aforementioned problem was tackled by merging the dual-clock FIFO with the switch input buffer, thus coming up with a unique architecture block in charge of buffering, synchronization and flow control, and sharing buffering resources for all of these tasks. The GALAXY project has also showed that this design principle, which we denote as "tight coupling" of synchronizer with the NoC, can be applied to dual-clock FIFOs in a straightforward way. For this reason, the second variant switch can optionally replace its input buffer for a fully synchronous environment with a dual-clock FIFO for a multi-synchronous environment, as illustrated in Figure 6.1. In all cases, functional correctness is guaranteed.

A similar functionality can be easily implemented also in the first variant switch.

## 3.2 Input/Output Buffers

Input and output buffers are much simpler than the ones of the first switch variant, since they do not have to handle the Nack/go flow control protocol but rather the simpler stall/go one. The input buffer is sized with two slots, which is the minimum amount of resources needed not to lose data during stall activation. It was 3 with Nack/go. The output buffer can be arbitrarily sized for performance buffering. As previously mentioned, control logic of input and output buffers is triplicated for fault tolerance and endowed with voters. An additional voter is placed on top of the data-path registers with the purpose of voting the outputs from the three instances of the buffer control logic. The voted output drives the read and write pointers of FIFO data registers.

## 3.3 Probing System

At the same time, probes inserted in front of each voter sniff their inputs and inform (through a comparator and an OR gate) the global manager about possible malfunctioning of each of the replicated branches. We find it important that the manager can keep this kind of information under control, so to be aware of a possible degradation of the fault-tolerance capability of the architecture. The OR gate collects the outputs of the comparators associated with each voting stage, as well as a notification signal from the correction sub-system denoting whether correction actions have been performed or not. Through the OR tree, a global notification of malfunctioning is achieved for each switch and notified to the global controller via a star interconnection topology.

## 3.4 Path Shifting

The Path-Shift module is composed of the following blocks:
- A demultiplexer, immediately inserted after the output of the pipeline stage. It is composed of two inputs (data input and select input) and two outputs (one for head flits and the other one for payload/tail flits).
- A Shifter and an encoder placed along the path followed by head flits.
- A 2x1 multiplexer

When a new flit arrives in front of the Path-shift module, we need to identify the flit type, i.e., whether it is a head flit or not.
For doing this, the select input port of the mux/demux is directly controlled by the first bit of the input flit.
In fact, this bit is set to ”1” for a head flit and to ”0” for payload/tail flit.
So, when the input flit is a tail or a payload, path shifting is bypassed.
On the contrary, when a head flit arrives, we need to shift the routing information so that each switch can always find in the same position its target output port. Alternatively, we would need to embody in the packet the indication of how many hops the packet has already gone through. This way, the switch would have to point every time to a different location in the packet head. After shifting the address bits, checkbits are not meaningful any more and need to be recomputed by the encoder before the flit can move on.

### 3.5 Control Path

Arbitration is performed with a round robin arbiter with triplicated control logic. Each instance of the arbiter is endowed with voters for self-correction; additional voters are located on top of crossbar multiplexers for reconvergence of the control path to the control inputs of the data path. Similarly to the first switch variant, a new arbiter state is saved only after voting it, to make sure that triplicated FSMs do not get misaligned as an effect of errors. This would compromise reliability of the control path for future transactions.

## 4 Experimental Results

This section describes the experimental results of the second variant of a 5x5 switch synthesized at the target speed of 500MHz with the 40nm low-power SVT Infineon technology library. Input buffers are assumed to be fully synchronous.
Figure 6.4 shows the area overhead of the proposed switch (rightmost bar) with respect to an intermediate implementation without any testing support and to a baseline TMR extension of the xPipeslite switch. It can be observed that area overhead for testing amounts to only 12.96% (Chapter 6 describes

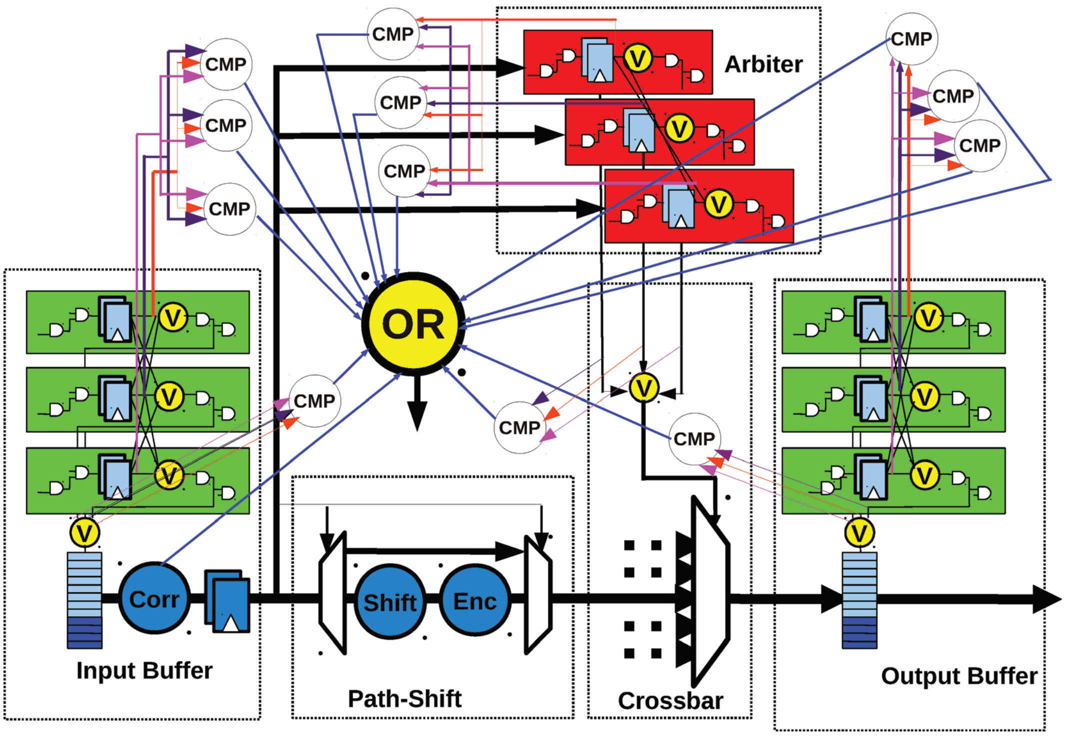


Figure 1.5: The Second architectural variant at a glance.

the testing architecture on top of the second variant). At the same time, more functionalities and provisions than the TMR switch are delivered at a much lower area footprint. Figure 6.5 illustrates the area breakdown of the testing logic. The major contributor to the testing logic comes from the MISRs used to perform the diagnosis and counts for ∼8.50%. The remaining part of the overhead is spread among the wrappers and the LFSRs used as test patterns generators.

Last but not least, when replacing the input buffer with a dual_clock FIFO, in practice there is no area overhead provided we keep the number of buffer slots the same. Actual buffer sizing then depends on network-level requirements such as the speed ratio between sender and receiver as well as the needed throughput across a multi-synchronous link[52].

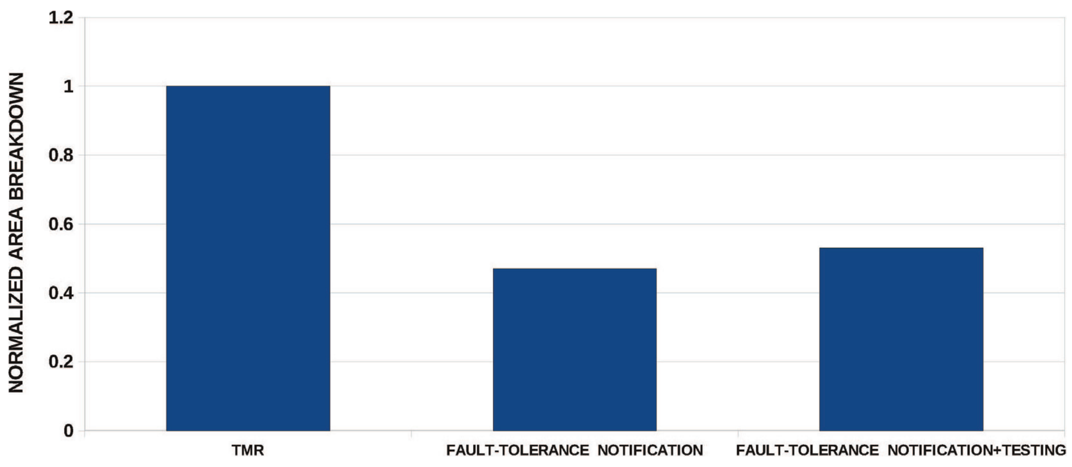


Figure 1.6: Area overhead @500MHz.

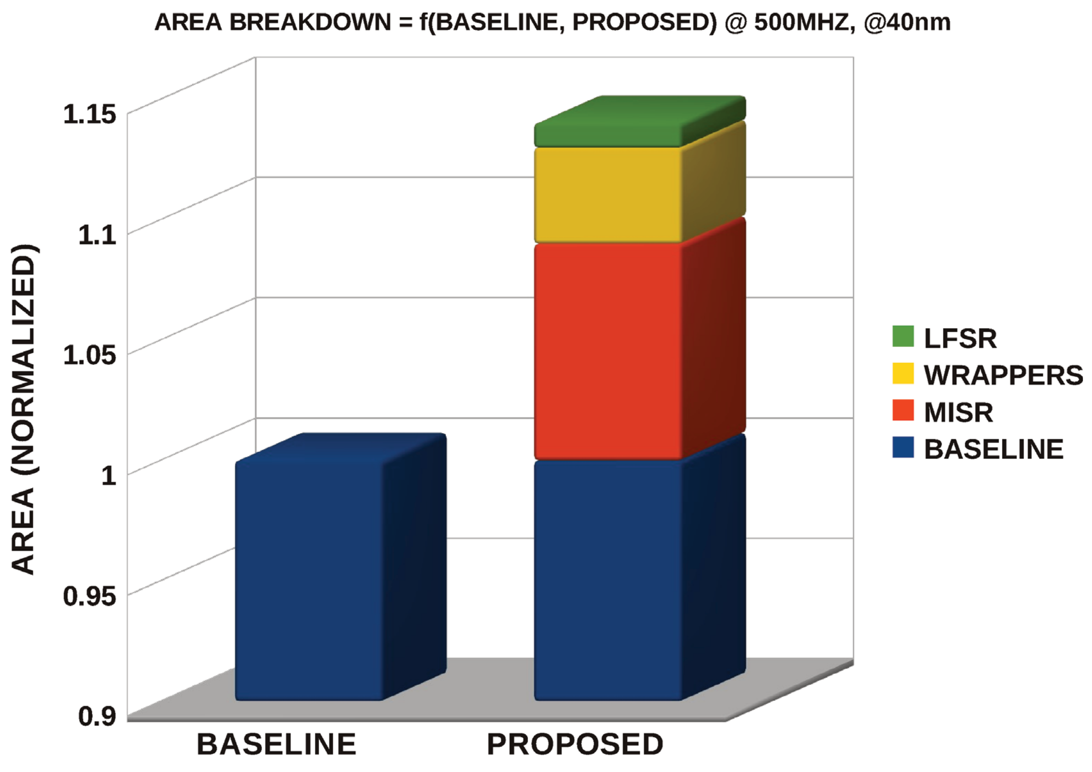


Figure 1.7: Testing overhead and area Breakdown.

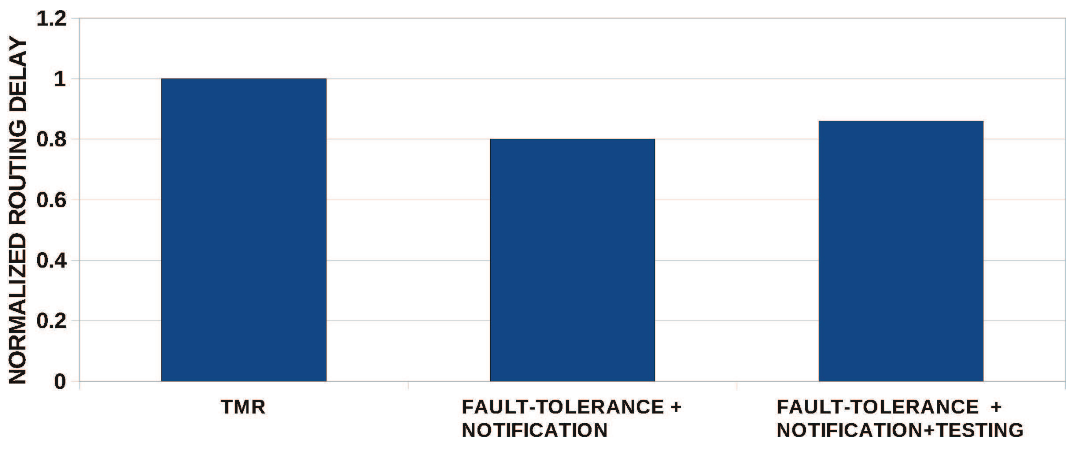


Figure 1.8: Normalized routing delay @Max Performance.

## 5 Conclusions

In this chapter, we present two architectural variants of a mesh endowed with fault-tolerance, notification infrastructure and overlapped static reconfiguration capability. We showed the major step in design complexity with respect to a state-of-the-art switch for low-to medium-end embedded systems, arising from the more aggressive requirements on switch functionality. At the same time, we showed that more functionality than TMR can be delivered within the same area budget, but with a non-negligible speed penalty.

# Chapter 2

# Synthesis Flow

## 1 Design flow

The synchronous flow regards the optimization of system-level hold-time margins, to improve system robustness to timing variability. In synchronous systems, the implementation of a global clock distribution network has increasing adverse effects on timing, as systems scale up in size and transistors scale down in geometries.

Smaller geometries mean higher variability, while larger gate count means greater non-shared depth of the clock tree, between the different system-level regions of the chip.

Clock variability causes timing issues, as the clock skew affects the timing in two ways: setup-time wise and hold-time wise.

While setup-time issues cause the system to function at reduced performance, hold-time violations may render a system dysfunctional. As clock variability is a statistical effect, the timing degradation relates to yield: fewer chips will function properly.

Today, the normal way to improve hold-time margin in a circuit involves insertion of delay buffers at the data path end points. This affects also the setup-time margin of the path.

In this chapter, the compress hold time prototype tool to optimize a circuit for hold-time, is presented. The tool includes innovative algorithms for insertion of hold-time buffers not only at the end points but in any branches of the data paths of a design. An algorithm that fixes hold-time without worsening setup-

time slack is also implemented (Further details are covered by confidentiality and Non Disclosure Agreement).
Maintaining positive setup-time slack is important in order to ensure headroom for successful backend convergence. The tool is successfully demonstrated on three state-of-the-art IC designs, and will be applied to NoC-based systems. It is shown how the tool is able to fix hold-time violations that are not fixable with a standard end point buffer insertion approach, and to fix hold-times without worsening total positive setup-time slack.
This was achieved by the compress_hold_time tool automatically inserting delay buffers deep within the circuit, in non-setup-time critical branches of the data paths. A mixed-approach provides the best of both worlds, resulting in significantly better results than standard approaches.

## 1.1 Introduction

More than 99% of all digital ICs are implemented in a synchronous manner. A synchronous system is characterized by having a synchronously clocked timing reference signal. In synchronous systems the implementation of a global clock distribution network has increasing adverse effects on timing, as systems scale up in gate count and transistors scale down in geometries. Smaller geometries mean higher variability in the logic gates, while larger system size means greater non-shared depth of the clock tree, between the different regions of the chip. Clock timing variability, defined as variability in the clock arrival time at different clock tree sink points, is thus increasing in advanced IC designs due to two factors.

**i.** Increasing on-chip variation (OCV) in clock logic gates, due to scaling down fabrication technologies into nano-scale geometries.

**ii.** Increasing non-common clock tree path levels between system-level blocks, due to design gate count scaling up into giga-scale.

Figure 2.1 illustrates this challenge. The non-common clock path is deeper due to a larger clock tree, while the variability at each clock tree node is increasing. As a direct result, the timing variability, between clock tree sink points, increases. Clock timing variability causes data path timing issues, as

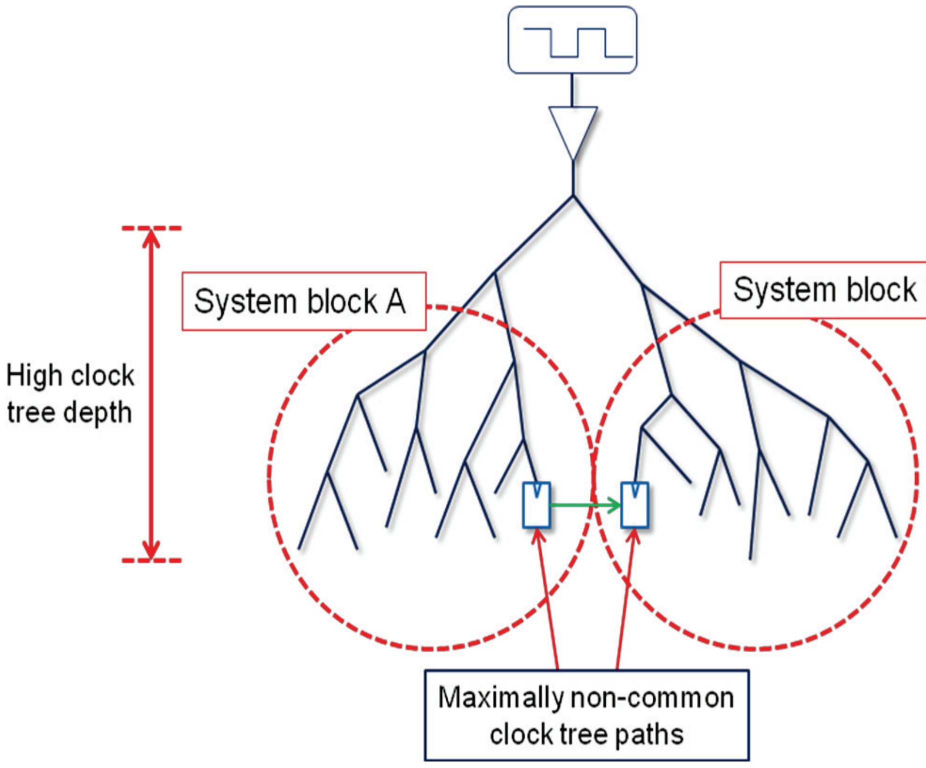


Figure 2.1: Increasing clock tree depth and increasing on-chip variability causes increasing system-level timing uncertainty.

the clock skew affects the data path in two ways: setup-time wise and hold-time wise. While diminishing setup-time margins will reduce design performance, hold-time issues will render circuits dysfunctional. As clock variability is a statistical effect, the timing degradation relates to yield, as fewer chips will function properly. Figure 2 shows how performance/timing related issues constitute the fastest increasing yield degradation factor in scaling IC technologies.

Today, the normal way to fix hold-time violations in a circuit involves insertion of delay buffers at the data path end points. This also affects the setup-time slack of the circuit. As a result, it is not always possible to fix hold-time violations without causing setup-time violations in the process. But even fixing hold-time violations in a manner that does not directly violate setup-time may also represents a degradation of the circuit, if the positive setup-time slack is reduced. This represents a major drawback in existing tools. Maintaining positive setup-time slack is important in order to ensure headroom for successful backend convergence. The tool functionality significantly extends state-of-the-art in hold-time optimization, in that it enables a major increase in flexibility and performance in the hold-time optimization process, by providing advanced new algorithms for intelligent hold-time buffer insertion. The tool will be applied to NoC-based designs in this chapter. It is shown how hold-time violations that cannot be resolved using end point buffer insertion can be resolved, with a reduced penalty on total positive setup-time slack and no impact on worst positive setup-time slack. In the prototype tool, a focus on system-level communication channels has been implemented by allowing hold-time fixing to occur only on data paths between system-level partitions in a design, as arbitrarily specified by the user.

# 2 IC Timing

Timing is by far the most important design parameter in IC design today. The timing of a circuit determines its performance as well as its robustness to fabrication variability.

## 2.1 Timing Margins

Circuit timing revolves around two main concepts:

**i.** Hold-time margin

**ii.** Setup-time margin

Figure 2.2 illustrates these two timing concepts. In the figure, data indicates a data arrival point, e.g. the data input of a flip flop or similar state-holding element. The hold-time margin is the time after an active clock edge during which the data value from the previous clock cycle must retain its state. This is to ensure that the internal state of the state-holding element is completely stable before the cell input starts to change. A hold-time violation will occur if a data path is very fast, so that new data arrives from another flip flop a very short time after the clock has toggled. The setup-time margin is the time before an active clock edge during which the data value from the

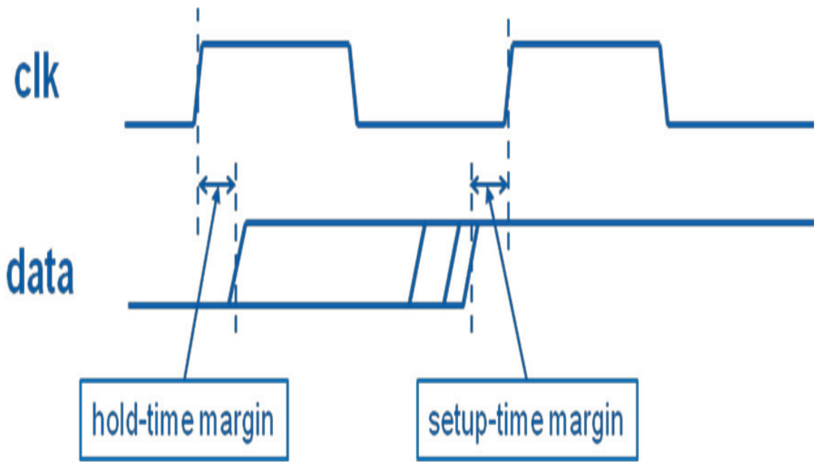


Figure 2.2: Increasing clock tree depth and increasing on-chip variability causes increasing system-level timing uncertainty.

previous clock cycle must attain its final state. This is to ensure that the internal state of the state-holding element is completely stable before the clock input toggles. A setup-time violation will occur if a data path is very slow, so that new data arrives from another flip flop too long time after the clock has toggled, i.e. too short time before the following clock event. Hold-time and setup-time margins are both influenced by variability in the clock timing. If the clock arrival time of the transmitting flip flop and receiving flip flop are skewed relative to each other, the margins can either be increased or diminished. Skew in one direction improves setup- time slack while worsening hold-time slack, skew in the other direction improves hold-time slack while worsening setup-time slack. Since the nature of variability is that the direction of the skew is unknown, in order to accommodate the worst-case scenario, both the setup- and hold- time margins must be higher than the worst-case clock skew. In order to ensure that hold-time issues do not occur, safety margins can be added. While such margins increase the reliability and manufacturability they also tend to limit the performance of the circuits, by taking an increasingly conservative view of circuit timing.

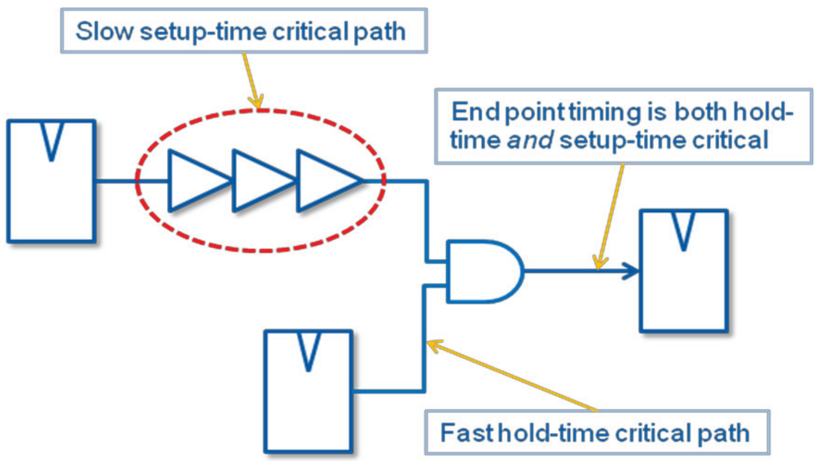

Slow setup-time critical path
End point timing is both hold-time *and* setup-time critical
Fast hold-time critical path

Figure 2.3: Increasing clock tree depth and increasing on-chip variability causes increasing system-level timing uncertainty.

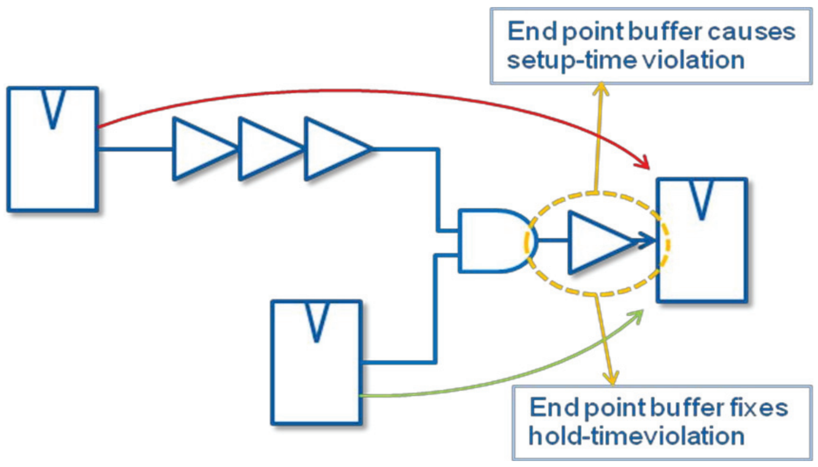

End point buffer causes setup-time violation
End point buffer fixes hold-timeviolation

Figure 2.4: Increasing clock tree depth and increasing on-chip variability causes increasing system-level timing uncertainty.

## 3 Fixing hold-time violations

Hold-time violations can be fixed by inserting extra delay in the data path. This effectively improves the hold-time margin by slowing down the data signals. The normal approach to fix hold-time violations today is by inserting buffers directly at the data path end points, as an ECO design flow step.

This approach is simple and works well, if a corresponding positive setup-time slack is available. The data path leading to the input of a flip flop can be complex however, and as shown in Figure 2.3, the end point timing may be both hold-time critical and setup-time critical. This occurs if there are both fast and slow paths leading to the end point. While inserting buffers at the data path end point slows the signal down and improves the hold-time margin, it meanwhile borrows setup-time, worsening the setup-time margin. This is illustrated in Figure 2.4. This is not desirable for a number of reasons. Firstly, if the end point is setup-time critical, an end point buffer cannot be inserted without causing a setup-time violation. This is not acceptable, as setup-time determines the performance of a circuit. Secondly, even if a degree of positive setup-time margin exists, it is not wishful to reduce the positive setup-time slack, as this is often used to ensure a margin for timing closure later in the design flow.

## 4 Requirements for an IC timing optimization tool

Timing of modern IC designs, and integrating into mainstream IC design flows, is a complex task. Developing a timing optimization tool, there are a number of advanced basic requirements to functionality and standard format compliancy. To import and analyse a circuit, the following functionality is required:

**i)** Import of

**a.** Liberty cell libraries

**b.** SDC timing constraint commands

**c.** SDF cell delay information

**d.** Gate-level Verilog netlist

**ii)** Support for complex, multi-clock and clock gated/muxed architectures.

**iii)** Support for multiple timing modes.

Apart from the development of the actual hold-time buffer insertion algorithms, a major part of the work performed in achieving the deliverable described herein was focused on expanding and maturing existing FloorDirector STA capabilities, integrating the hold-time buffer insertion algorithms into the STA engine, and implementing support for the required netlist modification and Verilog export.

# 5 The Compress_Hold_Time Tool

A prototype tool for improving design hold-time margins, known as compress_hold_time, has been developed, and is embedded into Teklatechâs FloorDirector IC design framework. The tool takes advantage of FloorDirectorâs built-in import and export functionality as well as Static Timing Analysis (STA) engine. Insertion of hold-time buffers in arbitrary branches of the data paths, and insertion of hold-time buffers with no effect on setup-time slack, constitutes the two key innovations in this chapter. The tool is fully configurable, and an arbitrary level of robustness of hold-time can be achieved. The impact on setup-time can be limited to the minimum level inherent in the data path structure of the circuit.

## 5.1 Hold-time buffer insertion

The compress_hold_time prototype tool provides advanced functionality for optimizing hold- time issues, fixing hold-time violations and improving existing hold-time margins, moving state-of-the-art significantly forward. Multimode timing is an integral part of the algorithms, and timing validated concurrently across multiple timing scenarios. The tool analyses the netlist and timing of a design, and based on a parameter setup.
Hold-time is optimized by slowing down certain branches of design data paths, by inserting buffers. While existing tools do this by inserting buffers at the data path end points, compress_hold_time takes a much more advanced approach. While simple end point buffer insertion is also possible using compress_hold_time, the tool implements a number of algorithms. A more advanced algorithm implemented as part of compress_hold_time works by identifying non-setup-time critical branches of hold-time critical data paths, and automatically inserting hold-time buffers deep within the circuit. While not

being limited to inserting hold-time buffers at the data path end points, compress_hold_time is thus able to improve hold-time slack with little or no effect on end point setup-time slack. The algorithm is also able to fix hold-time violations that are not possible to fix by end point buffer insertion alone. This is demonstrated in the results section. Finally, compress_hold_time is particularly useful for NoC-based designs, in which it may be particularly useful to optimize hold-time margins in system-level paths more aggressively, in order to improve robustness to system-level clock variability. Functionality for partitioning a design exists in FloorDirector, and compress_hold_time can take this partitioning into account when optimizing. The optimization can be set to include only paths between partitions in the optimization.

# 6 Results

## 6.1 Hold Time Robustness:Fine-tuning the flow on a 2D mesh

This section aims at the validation of two hold time robustness techniques applied on top of the same baseline NoC (4x4 mesh). The baseline NoC makes use of the xpipeLite switch as its basic building block. This experiment was a fine tuning experiment of the hold-time improvement flow on the simple test case of a 2D mesh. The first technique named here HTR1, improves the hold time while keeping constant the setup time. On the contrary the second one (named here HTR2) allows to improve the hold time by reducing the setup time. In the experiments we conducted, the hold time robustness techniques have been applied only to NoC links. The reason for this is that NoC routers are relatively small objects, therefore it is not difficult to enforce a tight skew constraint inside them. Vice versa, interconnected switches may be well far apart, therefore there the ultimate skew is far more unpredictable and the need for hold time optimizations arise. For hold time robustness, this means that safe margins against later possible degradations during the place&route step and even later as an effect of process variations have to be enforced. We will hereafter compare the two techniques mentioned above (HTR1 and HTR2) with respect to the 4x4 mesh devoided of any hold time optimization.

The min-max synthesis we conducted was made by using the following 40nm libraries:
- ucstarlib lpsvt 12t Pslow V090 T125
- ucstarlib lpsvt 12t Pfast V121 Tm30

Once the netlist was generated by the logic synthesis tool, we applied the HTR1/HTR2 techniques on top of the same netlist and we measured respectively:
- The worst and the total hold/setup time slacks on the links (see tables 2.1 and 2.2)
- The total hold/setup time slacks on the whole design (see table 2.3)
- The area overhead (see table 2.4)

Table 2.1 contains the worst hold/setup time slack on the link of each of the three designs mentionded above. From left to right, these designs are respectively the unoptimized 4x4 mesh (traditional synthesis flow), vs. the HRT1- and HTR2-optimized netlists. The second line of this table contains the worst hold time slack on the link, measured in the best case library. As we can see on line 2 of that table, the HTR1 hold time slack results to be greater than that of the 4x4 mesh (25% of improvement), moreover the HTR2 hold time slack is 40% greater than that of the HTR1 one. The third line of the same table contains instead the worst hold time slack on the link, measured in the worst case library. In this latter case, the HTR1 technique improves the hold time slack by 40% with respect to the 4x4 mesh while that of the HTR2 one results to be roughly 90.4% greater than that of the HTR1 one. As regards the setup time (see line 4 of table 8), they are almost all equal. More precisely, the 4x4 mesh and the HTR1 have the same setup time while that of the HTR2 is a little bit smaller than the other values. Indeed these measurements are in perfect agreement with expected results, hence represent a perfect calibration of the NaNoC flow for hold time robustness. On one hand, the HTR1 technique allows to improve the hold time slack while keeping the setup time slack constant (0.48ns vs 0.48ns). On the other hand, the HTR2 one allows to improve the hold time slack by reducing the setup time slack (only 2% of reduction).

The informations contained in table 2.2 are similar to those of table 2.1. Here instead of measuring the worst slack on the link, we measured the total worst slack over all links. In this case, the HTR1/HTR2 hold time slacks measured in the best case library (see line 2) result to be respectively 61% and 62.4% greater than those of the 4x4 mesh. When measurements are made in the worst case library (see line 3), the same HTR1/HTR2 hold time slacks result to be respectively 104.8% and 108.8% greater than those of the 4x4 mesh. Finally, the overall setup time slack degradation of HTR2 with respect to that of the HTR1 one is only 3.7%. Table 2.3 contains measurements about the total worst hold/setup time slack on the whole design. As expected, the HTR1 technique allows to improve the overall hold time slack (see lines 2 and 3) with respect to the 4x4 mesh while keeping the setup time slack almost constant (see line 4). As regards HTR2 technique, it allows a better hold time slack improvement than the HTR1 one (13.4% vs 13.0% in the âbest case libraryâ and 20.9% vs 24.6% in the âworst case libraryâ) but at the expense of the setup time slack degradation (2.26% vs 0.25%). Table 2.4 shows the area cost of the 4x4 mesh and the HTR1/HTR2 tech- niques. The total area (expressed in %) has been normalized with respect to that of the 4x4 mesh. From line 5 of table 2.4, it appears that both techniques (HTR1/HTR2) require almost the same area overhead, 13% for the HTR1 and 12% for the HTR2 one. Moreover, when considering the area breakdown, it appears that all this overhead comes from combinatorial cells added to improve the hold time (see line 2 of table 2.4). On the other hand the non combinatorial area remains constant in all the three cases (see line 3 of table 2.4).

| Worst slack on link | 4x4 mesh | HTR1 | HTR2 |
|---|---|---|---|
| Hold time bc_lib | 0.08ns | 0.10ns | 0.14ns |
| Hold time wc_lib | 0.15ns | 0.21ns | 0.40ns |
| Setup time wc_lib | 0.48ns | 0.48ns | 0.47ns |

Table 2.1: Worst slack on links: unoptimized 4x4 mesh vs. HTR1 vs. HTR2.

## 7 Conclusion

As shown in the results section, this was successfully achieved. It was shown how the newly developed algorithms achieve better results on both total

| Worst slack on link | 4x4 mesh | HTR1 | HTR2 |
|---|---|---|---|
| Hold time bc_lib | 899.98ns | 1449.2ns | 1462.34ns |
| Hold time wc_lib | 2078.97ns | 4258.17ns | 4342.15ns |
| Setup time wc_lib | 5931.52ns | 5997.22ns | 5773.8ns |

Table 2.2: Total slack on links: unoptimized 4x4 mesh vs. HTR1 vs. HTR2.

| Worst slack on whole design | 4x4 mesh | HTR1 | HTR2 |
|---|---|---|---|
| Hold time bc_lib | 4144.77ns | 4685.48ns | 4699.57ns |
| Hold time wc_lib | 10362.65ns | 12533.35ns | 12918.75ns |
| Setup time wc_lib | 26021.3ns | 26087ns | 25431.8ns |

Table 2.3: Total slack on whole design: unoptimized 4x4 mesh vs. HTR1 vs. HTR2.

| Area | 4x4 mesh | HTR1 | HTR2 |
|---|---|---|---|
| Combinatorial Area | 81937um2 | 106355um2 | 104720um2 |
| Non Combinatorial Area | 104282um2 | 104282um2 | 104282um2 |
| Total Area | 186219um2 | 210637um2 | 209002um2 |
| Total Area (%) | 100% | 113% | 112% |

Table 2.4: Total Area: 4x4 mesh vs. HTR1 vs. HTR2.

number of hold-time violated end points resolved and total positive setup-time slack reduction. The target was to improve the block-to-block hold-time robustness to 25% of the clock cycle for any circuit. The compress_hold_time prototype tool provides full flexibility, and the level of robustness required can be specified arbitrarily. Together with the capabilities to optimize without borrowing setup-time slack, a solution to any given level of robustness can be achieved with a minimal impact on setup-time slack, as per the limitations inherent to the structure of the data path [1].

[1]This chapter has included contents that are referred to a cooperative and interdisciplinary work where furher details are in[74].

# Chapter 3

# Contrasting Multi-Synchronous MPSoC Design Styles for Fine-Grained Clock Domain Partitioning: the Full-HD Video Playback Case Study

## 1 Abstract

Fine-grained (per-core) multi-synchronous systems calls for new clocking strategies and new architecture design techniques. This chapter compares two fundamental multi-synchronous implementation variants based on the extensive use of dual-clock FIFOs vs mesochronous synchronizers respectively. The architecture-homogeneous experimental setting, the cost-effective merging of synchronizers with NoC switch buffers, the sharing of as many physical synthesis steps as possible between the two architectures and the requirements of a realistic full-HD video playback application are the key innovations of this study.

## 2 Introduction

Pioneer research on GALS systems envisions the use of clockless interconnect fabrics bridging synchronous islands with each other [27, 28] in a multiprocessor system-on-chip (MPSoC). A few chip demonstrators prove the viability of this solution [24, 25, 26], yet they have not resulted in the widespread adoption of asynchronous NoCs in the industrial arena. The reason is that the gap between asynchronous handshaking techniques and current design toolflows is still too large and in most cases uneconomical to bridge. As an example, they require unconventional circuits such as Muller C-elements that are usually unavailable in standard cell libraries. Moreover, asynchronous logic is not well supported by mainstream CAD tools. Even in those cases where physical design falls within reach of such tools, this is done with a lot of manual intervention and disabling fundamental tool optimization features not to violate specific timing constraints of asynchronous circuits [40], hence resulting in largely unoptimized designs. In this context, the best solution found so far for prototype design and fabrication consists of implementing routers and GALS interfaces as hard macros using ad-hoc design styles [26]. Hard macros should be viewed in this case more as a way of working around the lack of proper design and verification tools rather than an aggressive optimization technique. In fact, area of these designs remains consistently larger than fully synchronous NoC counterparts (1.8× in [26]). Regardless of the design toolflow, it should be observed that as RC propagation delay of on-chip interconnects degrades in nanoscale technologies the handshaking operations in asynchronous NoCs start to last a considerable amount of time thus significantly affecting communication performance.

*The above landscape calls for a more evolutionary and cost-effective solution in the direction of a progressive relaxation of synchronization assumptions in nanoscale MPSoCs.*

Common design practice consists of implementing clock domain crossings by means of dual-clock FIFOs. However, this solution is expensive in terms of buffering resources in the FIFO itself (needed to absorb the clock speed difference) and of FIFO crossing latency, which can be of several clock cycles [32, 34]. This overhead is likely to worsen in the future to counter the degradation of the resolution time constant of synchronizers [51].

Dual-clock FIFOs can be used to build up DVFS (Dynamic Voltage and Frequency Scaling)-enabled systems with fine spatial locality by following the architectural template in Fig.4.1, hereafter denoted as *plain multi-synchronous.* They are instantiated at every switch port thus implementing clock domain crossing for inter-switch communication. Each switch belongs to the clock domain of its associated IP core. This solution replicates at a larger scale the overhead of the FIFO synchronizers and introduces routing delay unpredictability. In fact, network packets might have to cross low-speed clock domains on their way to destination, and the spatial distribution of such performance bottlenecks might change at runtime depending on the use case. On the other hand, this architecture template removes the need for a global clock tree, hence potentially resulting in better scalability and lower sensitivity to technology constraints.

*Since network-level implications of extensively using dual-clock FIFOs for clock domain crossings are still largely unexplored, current design practice consists of conservatively using these components only for coarse-grained system partitioning in order to keep the overhead affordable.* It is however not clear whether the architectural template in Fig.4.1 is viable for cost-effective and fine-grained multi-synchronous MPSoCs like those in [30].

One promising synchronization technique that is fully compliant with the multi-synchronous paradigm is mesochronous clocking. Mesochronous synchronizers allow a reliable communication between synchronous blocks derived from a master clock (hence sharing the same frequency) but suffering from arbitrary phase offset. This could be the case of a NoC inferred as an independent clock domain, as illustrated in Fig.4.2 and hereafter denoted as the *mesochronous NoC*. Given the chip-wide extension of the network domain, clock distribution might be unbalanced and the different switches might receive the same clock signal but with a different phase offset. Constraining such offset in the top-level clock tree might be either overly power expensive or even infeasible for large nanoscale designs. Hence, mesochronous synchronizers might be used to retime the data and transfer it reliably from one switch to another. Even this architecture requires dual-clock FIFOs to decouple the network from the clock domains of the individual IP cores, however they end up being instantiated only at network boundaries, while inside the network more cost-effective mesochronous synchronizers (area-, power- and latency-wise) are used.

*While it can be easily demonstrated that mesochronous synchronizers are less costly than dual-clock FIFOs when considering these synchronizers in isolation, their integration within an entire platform might question this conclusion* since a number of typically overlooked effects come into play. First, the synchronizer might have to be augmented to enforce timing margins for the layout constraints of the design at hand (e.g., length of specific links). Second, further complexity might be needed to implement clock gating for the case of idleness. Third, requirements for more buffer slots might be posed to connected NoC switches to enable full throughput operation. Fourth, while the full-empty protocol of a dual-clock FIFO directly matches a stall/go flow control protocol in the network, augmenting mesochronous synchronizers with flow control capability is not equally straightforward. Above all, the main differentiating factor between the architectures in Fig.4.2 and Fig.4.1 is the presence of a global and independent clock domain for the NoC.

As a result, identifying the most efficient implementation of multi-synchronous NoCs is non-trivial and requires careful consideration of the application domain, of the system architecture and of physical synthesis effects. Conclusions cannot be clearly drawn by assessing synchronization interfaces in isolation. This chapter takes the network-level perspective and aims at quantifying design quality metrics of the two architectural templates with layout awareness.

For the sake of fair comparison, we implemented the two multi-synchronous NoC variants with the same library of NoC components (the xpipes library) and brought them through the same physical synthesis process, apart from the few design steps that are solution-specific. Frequency settings of IP cores and of the network (in the mesochronous case) strongly impact relative performance and power figures, in addition to dictating constraints for the physical synthesis. Since these settings are tightly application-dependent, we implemented an important case study for future mobile devices: full-HD video playback. This enabled us to set the operating conditions for the frequency islands in the NoCs and to simulate realistic communication bandwidths between the cores.

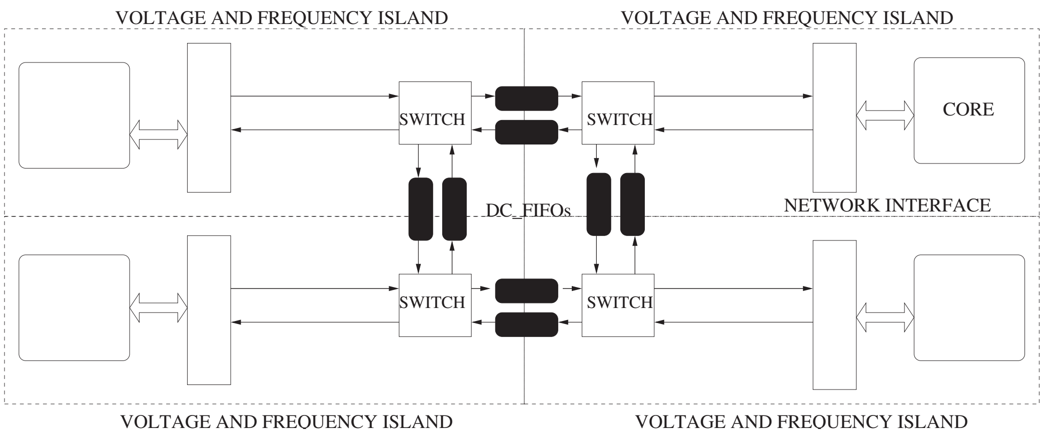

VOLTAGE AND FREQUENCY ISLAND
VOLTAGE AND FREQUENCY ISLAND
SWITCH
SWITCH
CORE
DC_FIFOs
NETWORK INTERFACE
SWITCH
SWITCH
VOLTAGE AND FREQUENCY ISLAND
VOLTAGE AND FREQUENCY ISLAND

Figure 3.1: Plain multi-synchronous architecture based on dual-clock FIFOs.

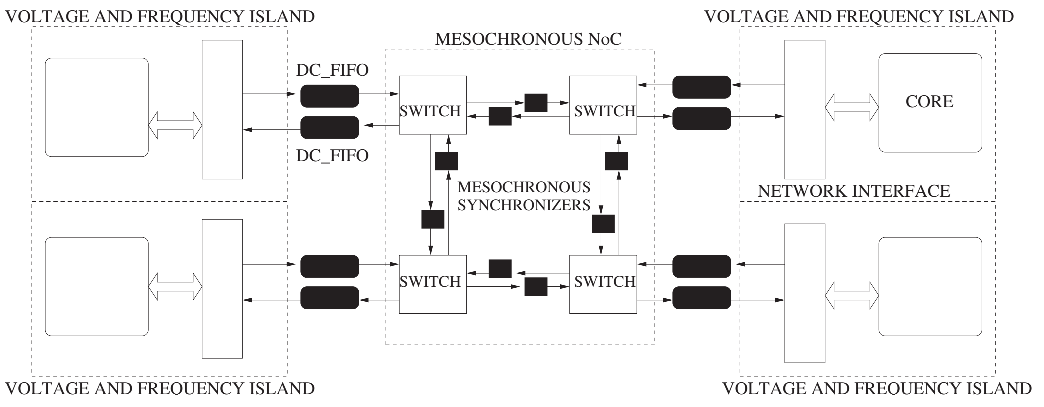

VOLTAGE AND FREQUENCY ISLAND
VOLTAGE AND FREQUENCY ISLAND
MESOCHRONOUS NoC
DC_FIFO
SWITCH
SWITCH
CORE
DC_FIFO
MESOCHRONOUS SYNCHRONIZERS
NETWORK INTERFACE
SWITCH
SWITCH
VOLTAGE AND FREQUENCY ISLAND
VOLTAGE AND FREQUENCY ISLAND

Figure 3.2: Mesochronous synchronization in a multi-synchronous architecture.

## 3 Related Work

Many works are focused on asynchronous interconnection networks for GALS systems, eliminating the need for global clock distribution. The CHAIN interconnect [39], the ANOC architecture [41, 42], the prototype GALS NoC in [43], the RASP network [44] and the mesh-of-tree topology in [38] are relevant examples thereof. Mesochronous clocking is a milder approach to the relaxation of synchronization assumptions in MPSoCs. A common design method of mesochronous synchronizers consists of delaying either data or the clock signal to sample data reliably [45, 46] and/or to use a phase detector circuit

[36].
Delay-line based synchronizers are mostly suitable for full custom designs. A different approach within reach of SoCs is proposed in [45, 35]: it employs cyclic write and read pointers with a certain initial spread to allow collision-free write and read operations. [53, 34] employ similar approaches while also synchronizing back-pressure signals. [53] investigates tight coupling of mesochronous synchronizers with NoC switches. Dual-clock FIFOs are the intuitive way of decoupling clock domains from each other, however they incur large area, power and latency overhead, thus motivating research efforts to mitigate their cost [33, 31, 32].
The dual-clock FIFO architecture in [52] borrows the token ring solution for FSMs from [32] and the asynchronous comparison of pointers from [31]. Above all, it is integrated inside NoC switches serving as multi-purpose input buffer.
In this chapter, we borrow and/or adapt synchronizer architectures from previous work in [52, 53], where design issues are discussed for the synchronizers in isolation. In contrast, the focus of this work is on the network level. Very few previous works share the same abstraction level. The most relevant one is [37], where the multi-synchronous DSPIN network is contrasted with the fully asynchronous ASPIN solution with synthetic traffic. However, there is no exploration of the design points for multi-synchronous systems.

## 4 Synchronizer Architecture

Synchronizer selection and tuning was made based on the following guidelines for the sake of fair comparison:

1. Compliance with a standard cell design flow to facilitate application to the embedded computing domain.
2. Implementation of the source synchronous link design style, where synchronizers receive a regular NoC link, carrying data, flow control commands and a copy of the clock signal of the sender used as a strobe signal. This style is currently the most mature one for synchronizer-intensive designs.

3. Suitability for a *tight coupling* design style, where synchronizers are not placed as external blocks in front of NoC switches, but rather integrated with the input buffer of downstream switches. This way, the same buffer slots of the synchronizer can be reused for switch performance buffering and for flow control purposes, thus coming up with a cost-effective implementation. With tight coupling, latency is significantly reduced as well, since additional stages are removed from the link and collapsed into the switch input buffer. The reader should refer to [54] for an overview of the benefits of tightly coupled synchronizers, which we take for granted in this work.
4. Same buffer switching policy: unused buffer slots because of network idleness or lack of congestion should not be clocked. This choice will then enable a fair comparison of idle power between architecture variants.

The considered dual-clock FIFO architecture is illustrated in Fig.3.3(a).
The size of this latter is parameterizable. Based on [52], at least 5 slots are required in order to support arbitrary frequency ratios between sender and receiver clocks. *Full* and *empty* detectors signal fullness and emptiness conditions of the FIFO. These detectors perform an asynchronous comparisons between the FIFO *write* and *read* pointers that are generated in clock domains which are asynchronous to each other.
For this reason, 2-stage brute force synchronizers are used to synchronize deassertion of the full signal in the sender domain and of the empty signal in the receiver domain, as showed in Fig.3.3(a). Further details can be found in [52].
In this chapter we also consider the finding in [51] that the resolution time constant of synchronizers keeps degrading as feature sizes shrink, therefore more stages will need to be cascaded in brute force synchronizers in order to achieve MTBFs (Mean Time Between Failures) of practical utility. To model such requirements, we augment the dual-clock FIFO with 4-stage brute force synchronizers (instead of 2) and increase the number of data FIFO slots to 7 to preserve full-throughput operation in this case. In the experimental results, both the 5 slot and the 7 slot FIFO variants will be considered to account for the trend pointed by [51].

The dual-clock FIFO can be readily used for tightly coupled NoC-synchronizer design, since it can directly serve as a switch input buffer. In fact, its empty signal can be easily conditioned (see right-hand side of Fig.3.3(a)) and changed into the valid signal for the switch arbiter. Based on it, the FIFO is admitted to an arbitration round. Once a connection is established between an input and output buffer of the switch, FIFO transmission can be stopped via the stall/go flow control signal $RX\_stall$.

The baseline mesochronous synchronizer architecture is borrowed from [53] and illustrated in Fig.3.3(b).

The rationale is to temporarily store incoming information in one of the front-end registers, using the incoming clock wire to avoid any timing problem related to the clock phase offset. Once the information stored in the front-end registers is stable, it can then be read, processed and sampled by the target clock domain.

Flow control is implemented by means of the stall/go signal, which freezes synchronizer counters to prevent buffer overflow in the downstream block. While this signal is already in synch with the back-end counter, it should be synchronized with the transmitter clock before feeding the front-end counter. A simple 1-bit synchronizer is instantiated for this purpose. This synchronizer is replicated again in front of the upstream switch since it is demonstrated in [53] that this solution gives rise to a larger timing margin for link delay.

The data path synchronizer can be easily coupled with the switch. For this purpose, its output directly feeds the switch arbitration logic and its internal crossbar. In practice, it serves as switch retiming and input buffer stage, in addition to its native synchronization task. Unlike the (way more complex) data path synchronizer, the replicated 1-bit control path synchronizer cannot be integrated into the upstream switch, an approach which we denote as *hybrid coupling* and which we follow in our implementation.

Finally, for the sake of fair comparison, we augmented the mesochronous synchronizer of [53] with the same clock gating policy of the dual-clock FIFO. Therefore, when there is no valid data in the synchronizer front-end, counters are frozen and data buffers are not clocked. The back-end counter is stopped after the valid signal is synchronized with the receiver clock domain by means of another 1-bit synchronizer, as illustrated in Fig.3.3(b).

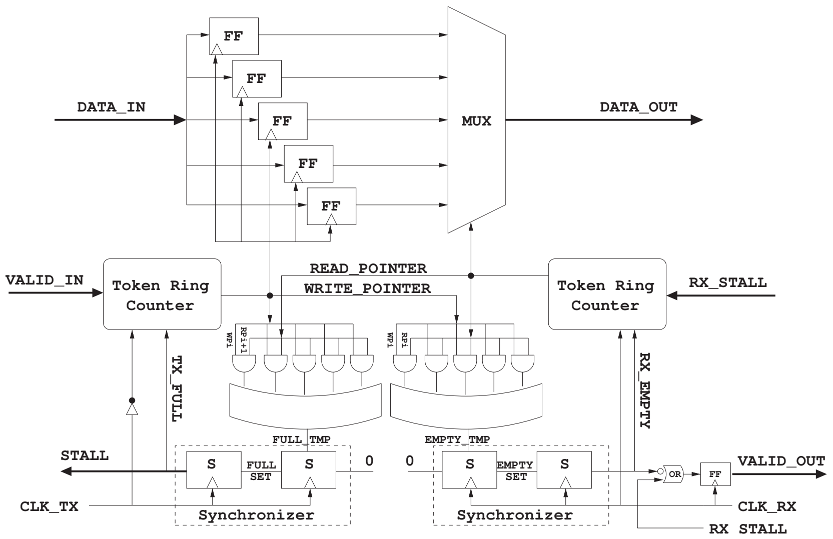
Both the dual-clock FIFO and the mesochronous synchronizer are instantiated as input buffers in the switches of the xpipesLite NoC to implement the architecture variants in Fig.4.2 and Fig.4.1. Unlike the picture, it is worth recalling that there are no synchronizers placed in NoC links, since they are all collapsed in the input buffers of downstream switches or network interfaces (except for the 1-bit control-path synchronizer in mesochronous links). The operating speed of the network or of its switches is now needed, but this information is tightly application-dependent. Next section describes the use case considered in this chapter.

## 5 Full-HD Video Playback Requirements

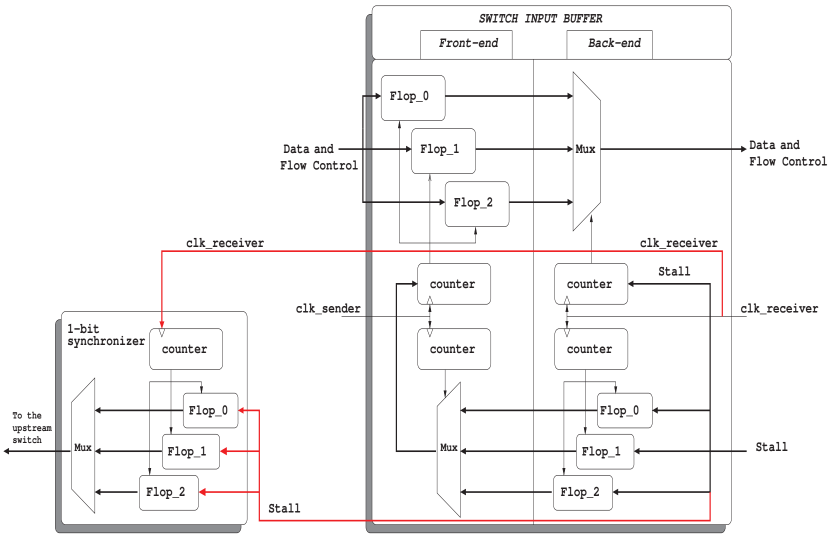
Extrapolating the usage scenarios of existing smart phones, one can imagine that in some years from now, the video playback and capture capabilities will not be limited to QVGA or WQVGA resolutions. With new mobile devices having bigger display sizes, at least 1024x768 resolutions have to be expected, but maybe even full HD-TV resolution. To provide a relevant experimental setting for the architectures under test, communication requirements of CPU, hardware accelerators and memory for video playback have been scaled up to the high-end HD-TV resolution, thus addressing the most challenging scenario foreseeable in the next few years. Extrapolated communication bandwidth requirements, based on the industrial experience of some of the authors, are illustrated in Fig.3.4.
The operating speed of the IP cores depends on the bitwidth and on the architecture of the core implemented in the real platform. Therefore, it was only possible to identify a possible range of speeds for each core depending on industrial IP core libraries and their projected future development. Table 3.1 lists possible min-max values of practical interest in the years to come.

### 5.1 System configuration

Future high-end mobile computing platforms will be most likely hierarchical. At top level, a number of heterogeneous components will be interconnected by a communication infrastructure, which is likely to have an irregular structure.



(a) Dual-clock FIFO architecture.



(b) Mesochronous synchronizer architecture.

Figure 3.3: Architecture of synchronizers.

One component of this system will most likely be a multi-core programmable accelerator consisting of a regular array fabric of computation tiles. Alternatively, the entire system might only consist of such a regular fabric, like the product in [48] for embedded computing.

The focus of this chapter is therefore on a 4x4 grid of identical general purpose

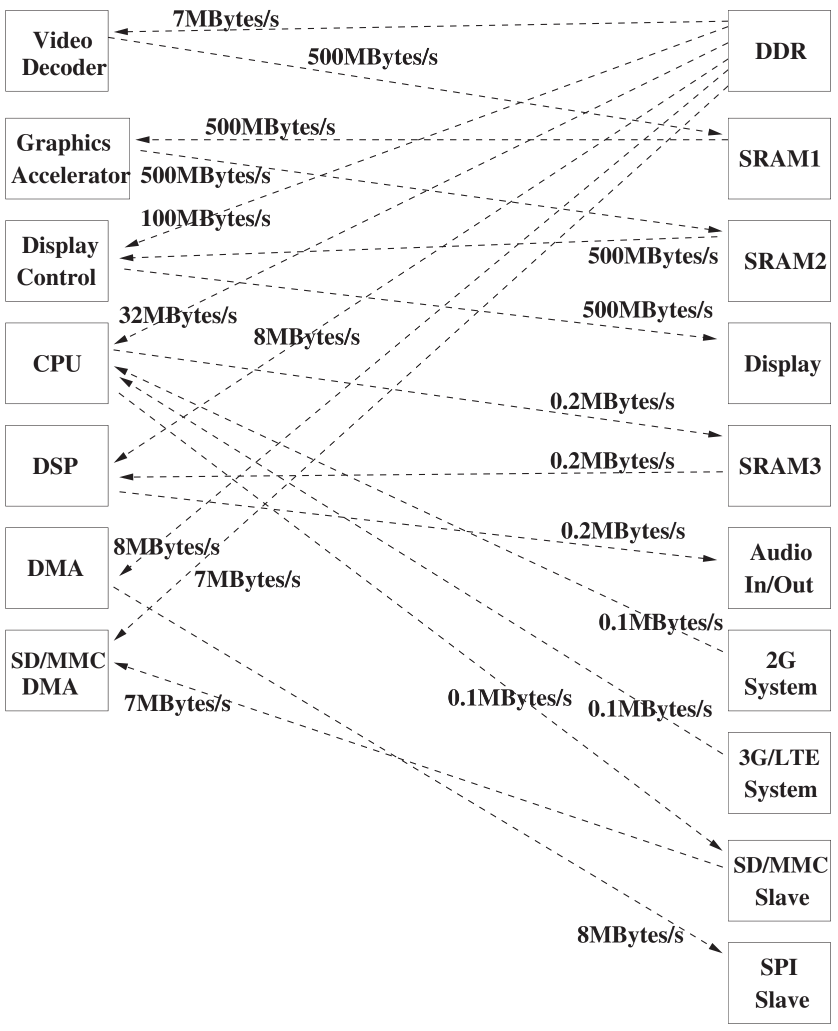


Figure 3.4: Communication bandwidth requirements for full-HD video playback: 1920x1080 pixel, 60 frames/s, true color.

processor cores onto which the requirements of Fig.3.4 and Table 3.1 are mapped.

At runtime, the video playback use case might be activated by mapping tasks to cores and by configuring those cores to run at the speed that optimizes execution of their associated task.

Without lack of generality and for the sake of fair comparison, we assume that the frequencies in Table 3.1 are the speeds of the processor cores in the homogeneous MPSoC.

We mapped the tasks to the 16 nodes of the 4x4 2D mesh network with the minimum-path mapping algorithm in [49], which optimizes hop delay with priority for the most communication-hungry cores. Dimension-order routing is assumed. An additional constraint was introduced: I/O peripheral controllers (such as DDR or SPI controllers) had to be mapped on the periphery of the chip, thus leaving central locations only for computation or storage tasks.

For the pre-computed mapping, three possible frequency settings were chosen for the cores within the bounds in Table 3.1. They are denoted in the same table as $Arch.1$, $Arch.2$ and $Arch.3$.

In $Arch.1$ the maximum speeds were chosen for almost all cores assuming 500 MHz to be the maximum possible clock speed in the system. $Arch.1$ is an aggressive scenario where high-speed IP cores are available and a lot of pressure is put on the on-chip interconnect performance.

In order to highlight a key design concern in multi-synchronous systems, settings were then changed as in $Arch.2$ to reflect the case where throughput-intensive information flows are exchanged between fast cores but are routed through slow intermediate cores. $Arch.2$ tests robustness of architectures under test to this unfortunate interrelation between operating speeds and routing paths.

Finally, we artificially extended the case study to derive more general results by selecting the same operating speed of 400 MHz for all cores ($Arch.3$). This could be more easily the case in an homogeneous MPSoC with a coarse-grained power management, or running an application which requires similar frequency settings for its cores in spite of their non-homogeneous communication requirements.

## 6 Design Flow

System architectures were at first modeled in RTL-equivalent cycle-accurate SystemC. OCP traffic generators were set to run at the speeds in Table 3.1 and programmed to inject traffic based on the bandwidth requirements in Fig.3.4.

At this level, the mesochronous NoC can be modeled and simulated as a

| IP core | Min (MHz) | Max (MHz) | Arch.1 (MHz) | Arch.2 (MHz) | Arch.3 (MHz) |
|---|---|---|---|---|---|
| Video Decoder | 100 | 500 | 300 | 500 | 400 |
| DDR | 266 | 400 | 400 | 400 | 400 |
| Graphics Accelerator | 100 | 300 | 300 | 100 | 400 |
| SRAM 1 | 200 | 500 | 500 | 500 | 400 |
| Display Control | 100 | 500 | 300 | 500 | 400 |
| SRAM2 | 200 | 300 | 300 | 300 | 400 |
| CPU | 300 | 1000 | 500 | 500 | 400 |
| Display (ext. interface) | 250 | 1000 | 500 | 300 | 400 |
| DSP | 100 | 300 | 300 | 300 | 400 |
| SRAM3 | 100 | 300 | 300 | 300 | 400 |
| Audio In/Out | 10 | 100 | 100 | 100 | 400 |
| 2G System | 100 | 150 | 150 | 150 | 400 |
| DMA | 100 | 200 | 200 | 200 | 400 |
| 3G/LTE System | 100 | 200 | 200 | 200 | 400 |
| SD/MMC DMA | 100 | 200 | 200 | 200 | 400 |
| SD/MMC Slave | 100 | 200 | 200 | 200 | 400 |
| SPI Slave | 64 | 128 | 128 | 128 | 400 |

Table 3.1: Range for IP core speed and chosen settings.

synchronous one, since as demonstrated in [53] with the tight (or hybrid) coupling design style latency (in clock cycles) and throughput of the networks are the same.[1] SystemC functional simulation aims at deriving the equivalent speed of the mesochronous NoC that enables it to match the video playback performance of the dual-clock FIFO-based platform.

We exploited an industrial 65nm technology library for the physical synthesis, using Synopsys Physical Compiler and Cadence SoCEncounter for logic synthesis and for place&route respectively.

For both platforms, a hierarchical bottom-up approach was taken. With this approach, it is possible to synthesize, place and route separately each switch and then to assemble them together to build up the entire system at the top level of the hierarchy. The target synthesis frequency of the switches in both platforms under test is 500 MHz, i.e., the worst case speed every switch should be able to support. In fact, we consider video playback as just one use-case running on top of the network. Other use-cases might require different speed settings for the switches of the plain multi-synchronous platform or

[1]Latency in real elapsed time is slightly different, however positive and negative skews across switch-to-switch links offset each other thus making the difference with a synchronous NoC irrelevant. Again, this holds for a tightly coupled design style.

for the NoC of the mesochronous platform, which we assume to be upper bounded at 500 MHz.

A hierarchical clock tree synthesis [47] was performed for the mesochronous NoC. Local clock trees of the switches were synthesized with a tight skew constraint of 5% of the target clock period. In contrast, the top-level clock tree was inferred with a relaxed skew constraint of 60% of the clock period, since timing of switch-to-switch communications is safeguarded by mesochronous synchronizers.

Both platforms under test make use of source-synchronous links. In order to avoid any timing misalignment between the data and the source synchronous clock wires, we used the automatic bundled routing feature of the routing tool. This proved sufficient for timing closure in our case. In more challenging scenarios, the same result can be achieved with some scripting effort like in [55].

Finally, it is worth mentioning that we used Synopsys PrimeTimePX to measure power on post-layout netlists with full annotation of switching activity from Verilog functional simulation. When measuring mesochronous NoC power, the operating speed of the NoC was provided by the SystemC functional simulation so to compare networks that provide the same application-perceived performance.

## 7 Experimental Results

### 7.1 Area results

We first of all comment on the area results after place&route, which are reported in Fig.3.5 and normalized to the area of a plain multi-synchronous platform with 7 slot dual-clock FIFOs.

For the baseline 5 slot FIFO implementations (two leftmost bars), the gap between the mesochronous and the plain multi-synchronous NoC is not large (a 3.53% lower area), since both mesochronous synchronizers and dual-clock FIFOs are merged with the NoC architecture, where they replace input buffers. With a loosely coupled approach, mesochronous synchronizers would have required an oversizing of switch input buffers for full throughput operation

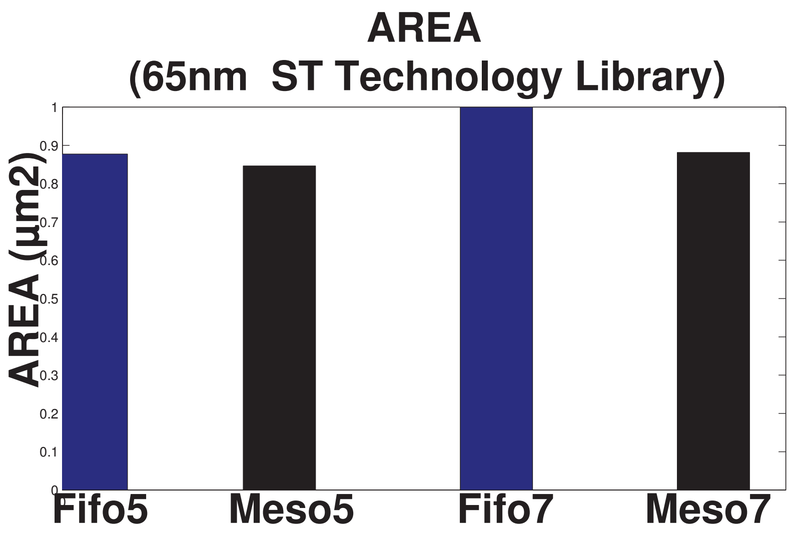
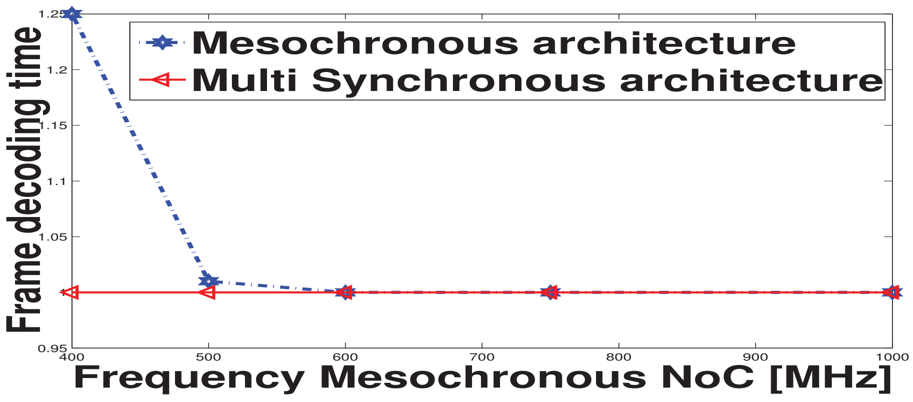


Figure 3.5: Area comparison between multi-synchronous NoC implementation variants when varying the FIFO depth.

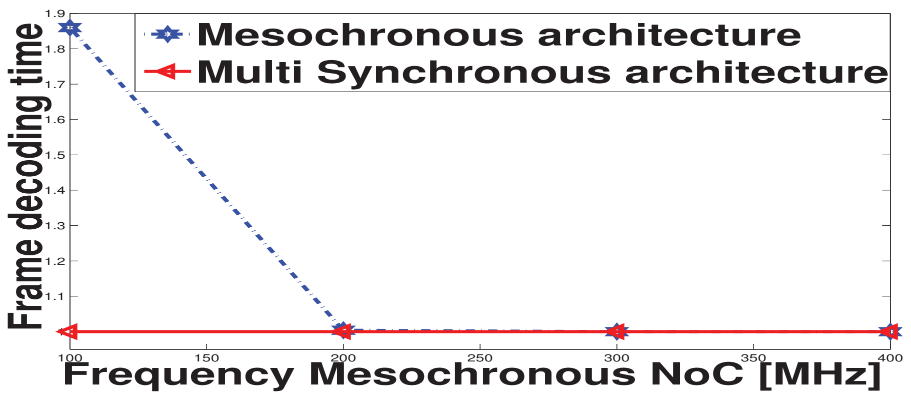
which would have offset their inherently lower number of buffer slots with respect to dual-clock FIFOs. On the other hand, such augmented input buffers do not dominate NoC area since other components (especially the output buffers, sized to 6 slots in this architecture) play a major role.
However, even for short term chip implementations a 5 slot FIFO does not guarantee a reasonable yield (see for instance [56]). An oversizing to 7 slots is advisable. When re-synthesizing both platforms under these assumptions, the two rightmost bars in Fig.3.5 point out an increased area gap: the mesochronous NoC saves around 12% area. For larger systems (e.g., 64 cores) this gap can only increase, since the plain multi-synchronous platform makes larger use of dual-clock FIFOs than the mesochronous one.

## 7.2 Setting the speed of the mesochronous NoC

We now address the speed setting of the mesochronous NoC, which is preliminary to power measurements.
By injecting the same traffic into the plain multi-synchronous and the mesochronous NoCs based on the frequency settings of *Arch*.1 and the bandwidth require-



(a) Arch1



(b) Arch2

Figure 3.6: Determining the speed of the mesochronous NoC for the *Arch*.1 and *Arch*.2 settings.

ments of the video playback application, we measured the frame decoding time from the SystemC functional simulation. Results are reported in Fig.3.6(a) as a function of the speed of the mesochronous NoC.

As we can see, execution time decreases, at first steeply and then more gradually, as the frequency of the mesochronous NoC increases, until reaching a saturation point. The intersection of the two curves returns the smallest frequency of the mesochronous NoC which allows the same performance of the plain multi-synchronous solution. Area and power consumption are tightly dependent on this value.

Clearly, for frequencies above 500 MHz the network is not the performance bottleneck and such a setting would not be cost-effective. In contrast, already at 400 MHz the performance penalty is large, hence suggesting a frequency of 500 MHz as the best choice for this case. This scenario is clearly a worst case for the power efficiency of the mesochronous NoC, since in the multi-synchronous NoC only a few islands operate at the maximum speed of 500 MHz, while the entire mesochronous NoC has to operate at such a speed. This is an effect of $Arch.1$ settings where processor core speeds are high and performance critical packets end up crossing moderate- to high-speed intermediate islands.

Different results were obtained for the $Arch.2$ settings. This time, already at 200 MHz the mesochronous NoC is able to match performance of the plain multi-synchronous one (see Fig.3.6(b)). This is an effect of throughput-intensive packet flows crossing low-speed intermediate islands, a scenario which heavily penalizes the multi-synchronous NoC.

Either $Arch.1$ or $Arch.2$ might occur in practice. Usually, core speeds for task execution on an homogeneous computation fabric are dictated by the application and by the need to lower power consumption of the processor cores. Typically, idle time is exploited to lower the core speed for better power efficiency or to temporarily switch-off the core. Operating conditions for each task are taken and coordinated by the global power management framework. At this level, communication costs are somehow abstracted since many details of the hardware communication infrastructure may not be known in advance, including the clock distribution strategy or the final mapping. In heterogeneous systems, where different IP cores ranging from CPUs to hardware accelerators, I/O devices and memory macros are networked, these speeds even depend on the library of available components. Then, mapping of tasks onto the on-chip network is typically performed based on cost metrics such as hop delay or power cost while meeting constraints such as that of non-exceeding maximum link capacity [49] or the placement of I/O controllers. As a result, there is typically no explicit constraint in synthesis flows avoiding the combination of $Arch.2$ settings with the mapping selected for this work. Even assuming to introduce such a constraint, this would imply to either increase processor core speed (thus increasing its power which might dominate

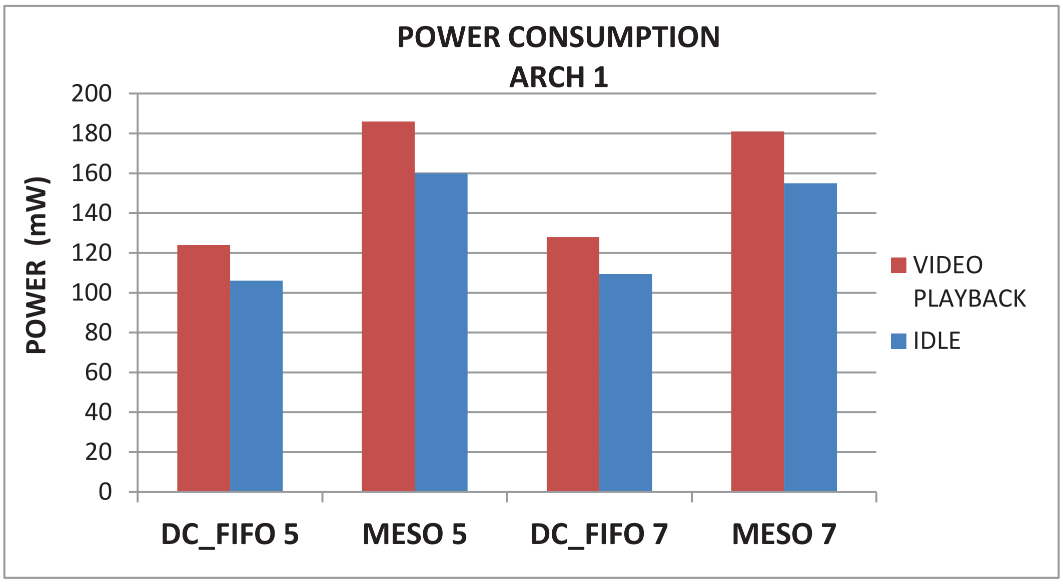


Figure 3.7: Power consumption of *Arch*.1 system configurations.

over network power) or to opt for a non-minimal route (with non-trivial implications on communication performance and/or deadlock avoidance). This chapter leaves the exploration of these optimizations for future work, and searches for architecture design styles able to benefit from current design methodologies.

## 7.3 Power results

Figures 10.2, 10.3 and 10.4 report power consumption comparison between the mesochronous and the plain multi-synchronous platform both in idle and traffic conditions for the three *Arch.x* operating conditions described in Section 5.1.

Given the homogeneity of clock rates in the *Arch*.3 scenario, this latter is considered first and highlights inherent architectural differences and their role in determining power. As Figures 10.2, 10.3 and 10.4 show, this scenario plays in favor of the mesochronous network. The plain multi-synchronous platform results in 21% and 23% power consumption overhead in idle condition and during video playback respectively with respect to the mesochronous counterpart.

The reason lies in the more complex control logic and the higher number of

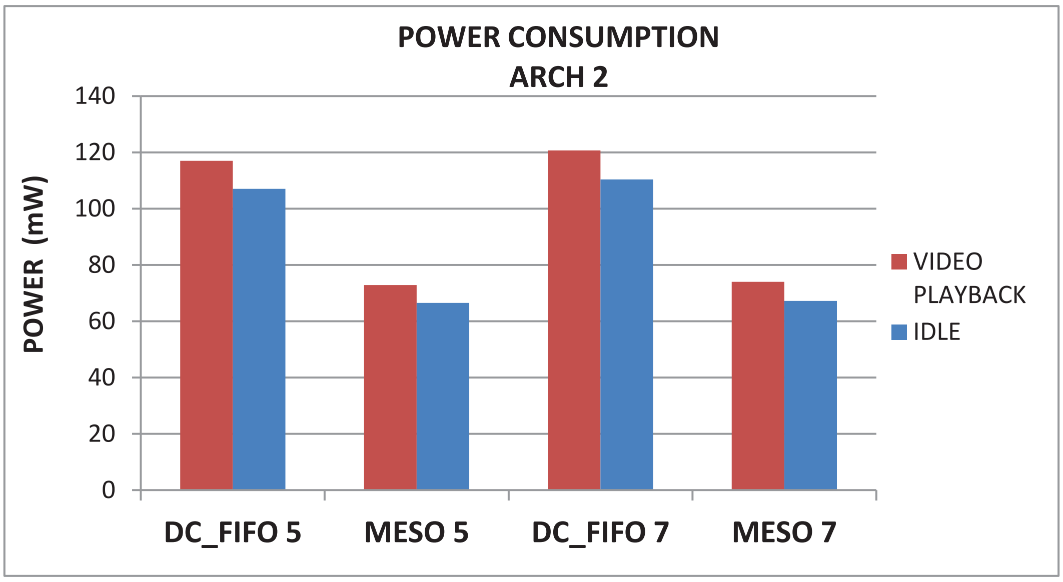


Figure 3.8: Power consumption of *Arch*.2 system configurations.

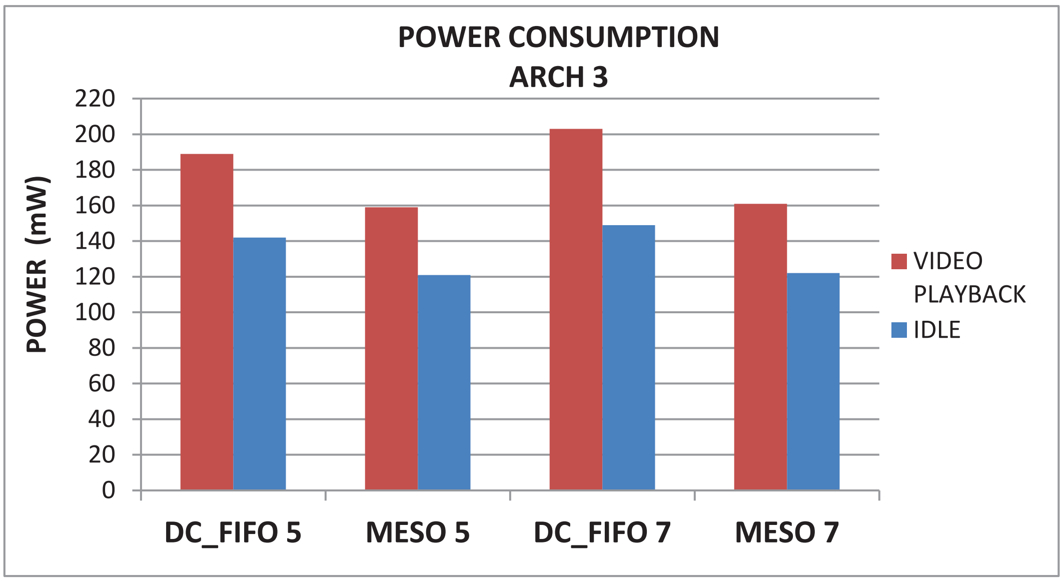


Figure 3.9: Power consumption of *Arch*.3 system configurations.

buffer slots of the dual-clock FIFOs in the plain multi-synchronous platform with the respect to the mesochronous synchronizers in the alternative implementation variant. Interestingly, when moving from 5 to 7 slot dual-clock FIFOs, the plain multi-synchronous platform exhibits a relevant 15mW overhead for the video playback, while the overhead for the mesochronous NoC is marginal. This is an appealing property for future on-chip realizations, where a dual-clock FIFO intensive design will be severely penalized.

On the contrary, the *Arch*.1 scenario adds the runtime configuration of the network in the power balance and plays in favor of the plain multi-synchronous network. In this case, it saves around 32% of power in idle condition and 31% for the video playback with respect to the mesochronous counterpart. This is mainly due to the high operating frequency (500MHz) required by the mesochronous network to match the performance of the alternative architecture. Although mesochronous synchronizers are more lightweight than the dual-clock FIFO synchronizers, the lower average operating frequency of the switches in the plain multi-synchronous platform dominates the final power consumption figures.

This result is however strictly dependent on the interaction between operating speeds of the switches (or of the mesochronous NoC) and packet routing. In fact, the *Arch*.2 scenario provides opposite results because of the low operating speed of the mesochronous NoC (200MHz). This parameter is key to determining power efficiency of the NoCs, thus explaining the significant 65% power consumption overhead of the plain multi-synchronous platform both in idle and in traffic condition for *Arch*.2. When extending the dual-clock FIFO synchronizers to 7 slots, this extension further increases the power gap between the two platform solutions, although to a smaller extent with respect to *Arch*.3 (around 3mW). This is due to the fact that the FIFO slot overhead is offset by the low operating speeds of many clock domains. Also, whether the traffic flows through high-speed or low-speed intermediate hops makes power more or less sensitive to the FIFO size. The same considerations apply to *Arch*.1.

Finally, the mesochronous NoC we are considering features a maximum skew of 60% of the clock period between any two leaves of the top-level clock tree. When iterating the place&route with a tighter 5% skew constraint, we measured a power consumption increment across the inferred mesochronous NoCs ranging from 4 to 6%. This result is in agreement with [50] and denotes a promising option when the system size scales further up.

## 8 Conclusions

Although a mesochronous NoC will be increasingly area efficient with larger integration densities, power efficiency strictly depends on the operating conditions. Dual-clock FIFO based solutions are severely penalized by those mappings that route performance-critical packets across slow intermediate nodes. In this scenario, the mesochronous NoC can easily do a better job. Vice versa, when the combination of routing decisions, mapping strategy and core speed setting is such to put pressure on NoC performance, the mesochronous NoC is forced to work at maximum speed to match the performance of the plain multi-synchronous solution, thus resulting in power overhead.
Based on the performed experiments, we believe that mesochronous NoCs have a room in multi-synchronous systems: they enable packet routing through performance-homogeneous hops and work with traditional design methodologies. Vice versa, dual-clock FIFO intensive architectures will be hardly affordable for fine-grained clock domain partitioning, especially considering the buffer over-provisioning that silicon technologies will require to sustain yield. Nonetheless, in those use cases where only few cores have to run at high clock rates, they are appealing for the reduced operating speeds of many of their clock domains. However, their successful exploitation requires a proper upgrade of the power management and NoC mapping strategies (and, above all, their co-optimization) to work around slow intermediate network hops, although the performance-power trade-off in this case is still unclear and will be the focus of our future work.

# Chapter 4

# Mesochronous NoC Technology for Power-Efficient GALS MPSoCs: Mesochronous vs. Synchronous

## 1 Abstract

MPSoCs are today frequently designed as the composition of multiple voltage/frequency islands, thus calling for a GALS clocking style. In this context, the on-chip interconnection network can be either inferred as a single and independent clock domain or it can be distributed among core's domains. This chapter targets the former scenario, since it results in the homogeneous speed of the NoC switching elements. From a physical design viewpoint, the main issues lie however in the chip-wide extension of the network domain and in the growing uncertainties affecting nanoscale silicon technologies. This chapter proves that partitioning the network into mesochronous domains and merging synchronizers with NoC building blocks, two main advantages can be achieved. First, it is possible to evolve synchronous networks to mesochronous ones with marginal performance and area overhead. Second, the mesochronous NoC exposes more degrees of freedom for power optimization.

# 2 Introduction

Networks-on-chip (NoCs) are proving capable of easing the communication bottleneck arising in multi-core computing platforms [91, 92, 93, 94], thus overcoming the fundamental performance, power and physical design limitations of shared and multi-layer busses. There is today little doubt on the fact that a high-performance and cost-effective NoC can only be designed in 45nm and beyond under a relaxed synchronization assumption [94, 96]. One effective method to address this issue is through the use of globally asynchronous and locally synchronous (GALS) architectures, where the chip is partitioned into multiple independent voltage and frequency domains. Each domain is clocked synchronously while inter-domain communication is achieved through specific interconnect techniques and circuits [95]. The methodology of inter-domain communication is a crucial design point for GALS architectures. One approach currently experimented in GALS NoC prototypes consists of using purely asynchronous clockless handshaking for transferring data words across clock domains [97, 98]. A few chip demonstrators prove the viability of this solution [109, 111], but they have not achieved widespread adoption of asynchronous NoCs in the industrial arena yet. In order to more incrementally evolve current industrial practice, some previous work in [101, 104] has developed synchronizer-based GALS NoC technology. In particular, design techniques merging synchronizers with network building blocks (named the tightly coupled design style) have proved area, power and performance efficient with respect to loosely coupled solutions, where synchronizers are placed as external blocks to NoC switches. All these previous work concerns architecture design space exploration and quality metrics assessment of synchronizer-based communications at the switch level. This chapter builds on these milestones and moves a step forward by taking the network-level perspective. While the migration from fully synchronous parallel systems to GALS systems with voltage/frequency decoupling between IP cores is taken as a matter of fact in this chapter, there are significant GALS NoC architecture variants the designer can still choose from. The first one consists of placing NoC switches in the clock domains of the IP cores they are connected with. In contrast, an alternative solution consists of inferring the on-chip network as an independent clock domain, disjoint from those of the IP cores. In this scenario, dual-clock FIFOs need to be instantiated only at the network boundary, since the network is synchronized by a single and independent clock signal. The homogeneous performance of NoC switches, the fewer amount of dual-clock FIFOs required and the possibility to have an always on system interconnect fabric make this solution more attractive to this chapter. However, the feasibility and efficiency of this solution is now mainly on burden of the physical designer. In fact, he has again to deal with a large synchronous clock domain (the NoC itself) distributed throughout the entire chip. A workaround for this problem consists of inferring the network as a set of mesochronous domains, instead of a global synchronous domain, yet retaining a globally synchronous perspective of the network itself. The granularity of a mesochronous domain can be as fine as a NoC switch, which is the case considered in this chapter. The communication between neighboring switches is then mesochronous as the top-level clock tree might not be equilibrated. This brings the additional advantage that mesochronous synchronizers are typically more lightweight than dual-clock FIFOs for use in switch-to-switch links. This chapter leverages mature mesochronous communication technology to perform a comprehensive crossbenchmarking of a mesochronous NoC with a fully synchronous NoC for use in a GALS system. Both networks share the same baseline MPSoC-oriented NoC architecture for the sake of fair comparison. The tightly coupled design principle is followed for mesochronous links, so that their unique optimization opportunities in the NoC domain are fully exploited. The chapter relies on actual implementations on a 65nm industrial technology library and provides the assessment of a wide range of design quality metrics, some of them of special interest for nanoscale silicon technologies: performance, area, power, and clock tree synthesis efficiency. This way, this chapter can provide useful guidelines for those industrial designers currently committed to the development of next generation NoC-based MPSoCs. Concisely, the main contribution of this chapter can be summarized as the crossbenchmarking of two GALS systems, the former implemented with a **fully synchronous NoC**; the latter leveraging a **mesochronous NoC**. Both systems have been compared from an area and power viewpoint. Since dual-clock FIFOs for connection to network inter-

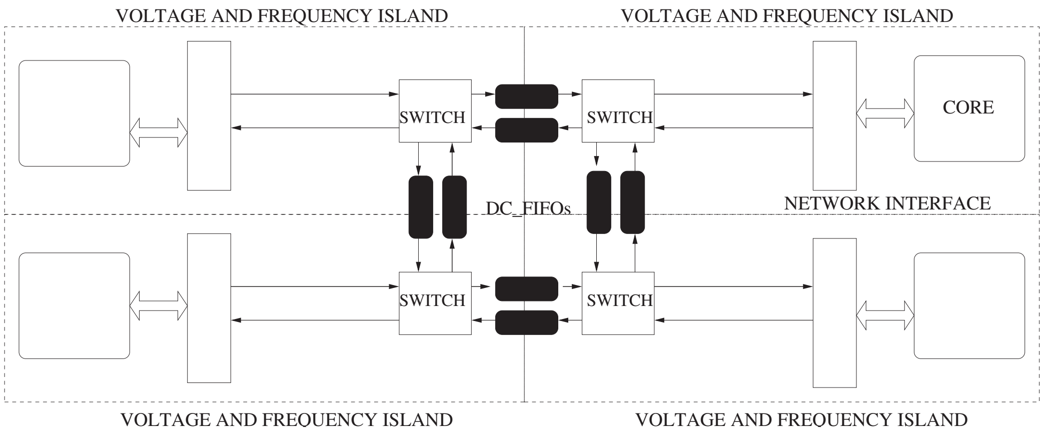

VOLTAGE AND FREQUENCY ISLAND
VOLTAGE AND FREQUENCY ISLAND
SWITCH
SWITCH
CORE
DC_FIFOs
NETWORK INTERFACE
SWITCH
SWITCH
VOLTAGE AND FREQUENCY ISLAND
VOLTAGE AND FREQUENCY ISLAND

Figure 4.1: Plain multi-synchronous architecture based on dual-clock FIFOs.

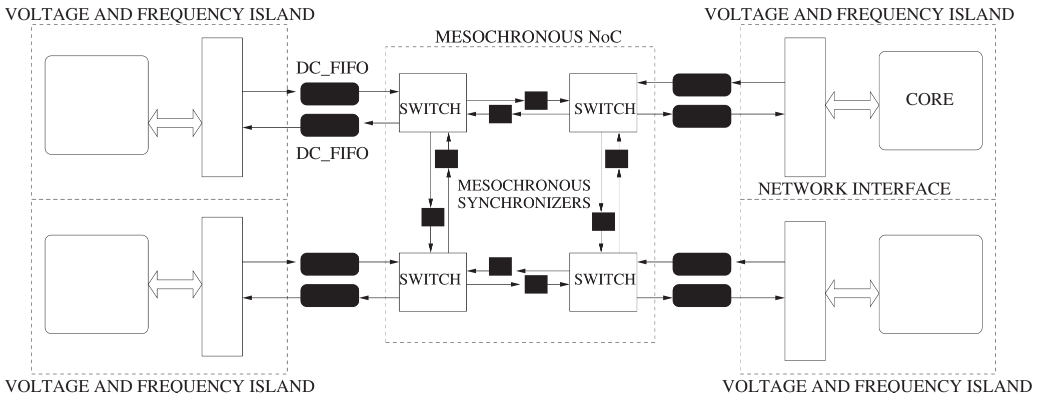

VOLTAGE AND FREQUENCY ISLAND
VOLTAGE AND FREQUENCY ISLAND
MESOCHRONOUS NoC
DC_FIFO
SWITCH
SWITCH
CORE
DC_FIFO
MESOCHRONOUS SYNCHRONIZERS
NETWORK INTERFACE
SWITCH
SWITCH
VOLTAGE AND FREQUENCY ISLAND
VOLTAGE AND FREQUENCY ISLAND

Figure 4.2: Mesochronous synchronization in a multi-synchronous architecture.

faces are common to both solutions, they have not been considered in this work. The remainder of this chapter is organized as follows. Section 3 will present the GALS platforms under analysis whereas Section 4 will review the architecture of the synchronizer block utilized as baseline to build the mesochronous network. Section 5 will describe the method utilized to synthesize the fully synchronous and the mesochrnous GALS system. Section 5 will present a comparison in terms of area, wiring and power overhead. Finally, Section 6 concludes this work with a final discussion and directions for future work.

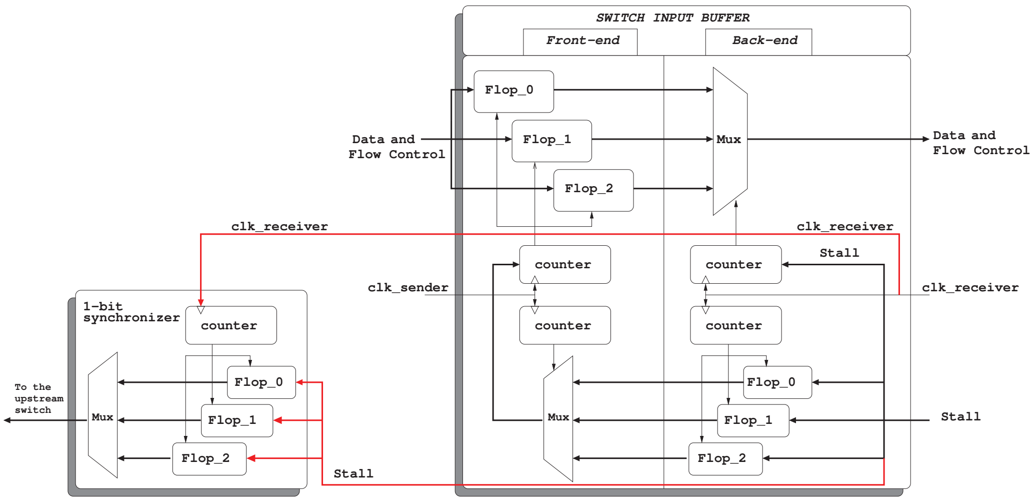


Figure 4.3: Hybrid mesochronous synchronizer architecture.

## 3 Target GALS Architecture

A GALS-based design style fits nicely with the concept of voltage and frequency islands (VFIs), which has been introduced to achieve fine-grain system-level power management and is currently driving the transition of most MP-SoCs to GALS systems. In these systems, if network components belong to the core's VFIs then performance of communication flows would be determined by the slowest domain crossed on the way to destination. Also, in case a VFI is shut down, global connectivity is jeopardized. An alternative solution is illustrated in Fig. 4.1, where the NoC lies in its independent VFI. This way, performance of the whole switching fabric would be homogeneous, with only boundary effects to take care of. Also, the network would be loosely coupled with the cores' VFIs, and each core/cluster of cores could be shutdown without any impact on global network connectivity. The main issue with an independent NoC VFI consists of the feasibility of its clock tree. The reverse scaling of interconnect delays and the growing role of process variations are some of the root causes for this. Even though inferring a global clock tree for the entire network will still be feasible for some time, it will probably come at a significant power cost. Moreover, it is unclear when the difficulty of tightly and globally enforcing the skew constraint will truly become a roadblock. However, a workaround for this problem does exist, as illustrated in Fig.4.2. The network could be inferred as a collection of mesochronous domains, instead of a global synchronous domain, yet retaining a globally synchronous perspective of the network itself. There are several methods to do this. One simple way is to go through a hierarchical clock tree synthesis process. In practice, a local clock tree is synthesized for each mesochronous domain, where the skew constraint is enforced to be as tight as in traditional synchronous designs. Then, a top-level clock tree is synthesized, connecting the leaf trees with the centralized clock source, with a very loose clock skew constraint. This way, many repeaters and buffers, which are used to keep signals in phase, can be removed, reducing power and thermal dissipation of the top-level clock tree. The granularity of a mesochronous domain can be as fine as a NoC switch block. The communication between neighboring switches is then mesochronous as the clock tree is not equilibrated, while the communications between switch and IP cores are fully asynchronous because they belong to different clock domains. Bi-synchronous FIFOs are therefore used to connect the network switches to the network interfaces of the cores, as showed in Fig.4.2. This synchronization paradigm comes with additional advantages. First, it makes a conscious use of area/power-hungry dual-clock FIFOs, which end up being instantiated only at network boundaries. Instead, more compact mesochronous synchronizers are used inside the network, thus minimizing the cost for GALS technology. Finally, unlike fully asynchronous interconnect fabrics, the synchronizer-based source-synchronous GALS architecture illustrated in Fig.4.2 is within reach of current mainstream design toolflows with just incremental effort. Typically, some scripting effort within commercial synthesis frameworks enables these latter to meet the physical requirements of source-synchronous designs [100, 108]. In the rest of this chapter, the architectures in Fig.4.1 and Fig.4.2 will be compared from many viewpoints by means of physical synthesis runs, to quantify when exactly to migrate away from the architecture of Fig.4.1 and the actual overhead of the architecture of Fig.4.2. The xpipesLite NoC architecture [99] is used as baseline experimental setting to implement both GALS platforms. The flow control protocol used by xpipesLite is stall/go: a forward signal, synchronous with data, flags data availability (valid), while a backward signal flags a destination buffer full (stall) or empty (go) condition.

# 4 Hybrid coupling of synchronizer with the NoC

Usually, mesochronous synchronizers are just placed in front of the downstream switch (the loosely coupled design style). This has implications on the size of the switch input buffer as well, which should cover the round trip latency to sustain maximum throughput. Given the large buffering and latency overhead of this approach, we proposed in [103] to bring the synchronizer deeper into the downstream switch, as illustrated in Fig.4.3. The reference synchronizer architecture receives as its inputs a bundle of NoC wires representing a regular NoC link, carrying data and/or flow control commands, and a copy of the clock signal of the sender used as a strobe signal for them. The circuit is composed by a front-end and a back-end. The front-end is driven by the incoming clock signal, and strobes the incoming data and flow control wires onto a set of parallel latches in a rotating fashion, based on a counter. The back-end of the circuit leverages the local clock, and samples data from one of the latches in the front-end thanks to multiplexing logic which is also based on a counter. The rationale is to temporarily store incoming information in one of the front-end latches, using the incoming clock wire to avoid any timing problem related to the clock phase offset. Once the information stored in the latch is stable, it can be read, processed and sampled by the target clock domain. In the architecture of Fig.4.3, the synchronizer output now directly feeds the switch arbitration logic and its internal crossbar, thus materializing the tight coupling concept of the mesochronous synchronizer with the switch architecture. The ultimate consequence is that the mesochronous synchronizer becomes the actual switch input stage, with its latching banks serving both for performance-oriented buffering and synchronization. A side benefit is that the latency of the synchronization stage in front of the switch is removed, since now the synchronizer and the switch input buffer coincide. The buffering overhead in the switch input buffer because of flow control is also removed accordingly. The main change required for the correct operation of the new architecture is to bring the stall/go flow control signal to the front-end and back-end counters of the synchronizer, in order to freeze their operation in case of a stall. While this signal is already in synch with the back-end counter, it should be synchronized with the transmitter clock before feeding the front-end counter. The backward propagating stall/go is then directly synchronized with the transmitter clock available in the front-end by means of a similar but smaller (1-bit) synchronizer. For this architecture solution, only 3 latching banks are needed in the synchronizer front-end, since link latency has been minimized. In practice, only 1 slot more than the input buffer in the fully synchronous switch. A loosely coupled approach would require a 4 slot input buffer and a 3 latch banks synchronizer. As regards the control path, a 1-bit synchronizer is replicated in front of the upstream switch. This synchronizer is not merged with the downstream buffer, since this would give rise to overly tight timing constraints [101]. In contrast, integrating only the data-path synchronizer is denoted as the hybrid coupling and gives more guarantees for timing closure, and is the approach taken hereafter.

## 5 Synthesis of GALS Platforms

Both the synchronous and the mesochronous platforms have been designed to be seamlessly integrated into an industrial design flow using commercial tools for physical synthesis. Only standard cells are used and no full custom components. The reference topology of our experiment is a 4x4 mesh network where each switch is connected to either a core or a memory (of size 1.5mm). As far as the physical synthesis is concerned, the same bottom-up methodology has been utilized for both platforms. Specifically, each network switch has been placed and routed in isolation with a target frequency of 500MHz. The clock tree of each switch has been synthesized with a tight skew constraint of 5% of the target clock period. Once the local clock tree is characterized with its input delay, skew and input capacitance, a macromodel is built in order to be used in the next design step. Furthermore, in order to implement a hierarchical clock tree synthesis, a buffer has been inserted to the input clock pin of each switch block. Once the switches have been placed and routed, they are imported as macro blocks in the main network design along with their libraries detailing both timing and physical characteristics. The next step consists of performing a top-level clock tree synthesis by leveraging the switch macromodels previously extracted. In fact, this model can be used to characterize the bottom clock tree given that these local clock trees will not be modified by the place&route tool. Therefore, in order to preserve the clock tree local to the switches, a **"PreservePin"** tag must be used in the CTS specification file. Please notice that the hierarchical CTS has been used both for the synchronous and the mesochronous platforms, since this is a standard methodology for parallel hardware platforms. The only difference is the skew constraint in the top level clock tree, which can be loosened for the mesochronous design while should be tightly enforced for the synchronous one. Final step of our hierarchical methodology consists of routing the switch-to-switch links and performing parasitics extraction for accurate static-timing analysis and power estimation. Timing closure for both the synchronous and mesochronous NoC has been achieved at 500 MHz by performing exactly the same physical synthesis steps.

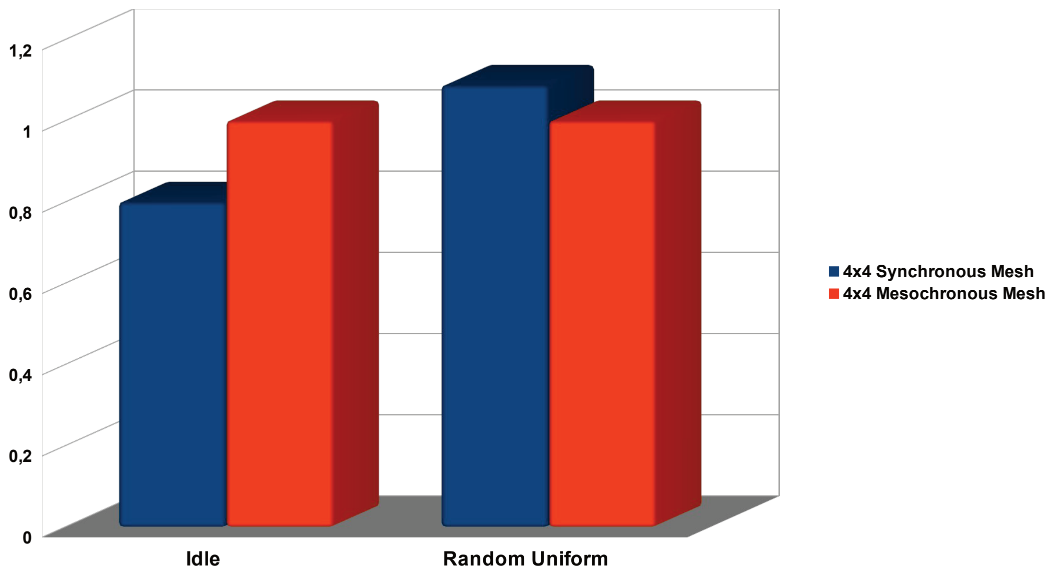


Figure 4.4: Power consumption with no activity and with uniform random traffic (normalized with respect to the mesochronous network).

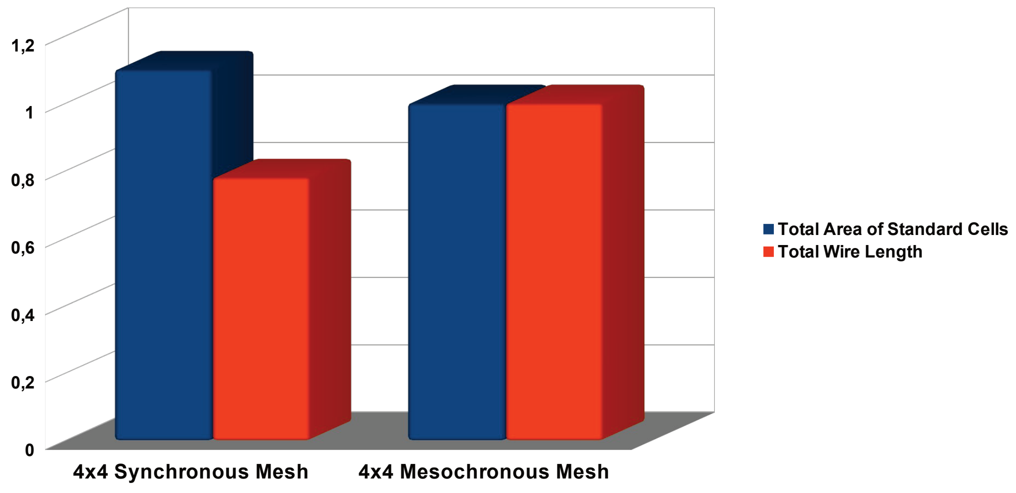


Figure 4.5: Area and wiring intricacy (normalized with respect to the mesochronous network).

## 6 Experimental results

### Area and Wiring Overhead

Figure 4.5 reports post-place&route area and wiring statistics for the architectures under analysis. From an area viewpoint, both systems exhibit the same footprint. More in detail, our baseline architecture (i.e., the fully synchronous mesh) features a 2-slots input and 6-slots output buffers. On the other hand, its mesochronous counter-part has 3-slots input buffer and exactly the same amount of output buffering. Nonetheless, the area overhead is identical. This is due to the fact that synchronization mechanisms, tightly coupled in the input buffer, are implemented through latch banks, which require typically a smaller area footprint compared to the flip-flops adopted in the input buffer of the baseline architecture. The ultimate result is an equal area occupation in both platforms although this comes with a somewhat more challenging testing framework. From the wiring point of view, a 23% net saving is achieved by the fully synchronous platform. The reason lies in the fact that the mesochronous platform features an additional clock wire per output port utilized as strobe signal for data synchronization and a further external single bit synchronizer for backward flow control synchronization instantiated in each of the 48 switchâtoâswitch channels of the network. Last but not least, the slightly more complex network topology contributes to a more complex structure of the clock tree.

**Power analysis**

By leveraging post-layout netlists and back-annotated switching activity, we were able to achieve very accurate power figures. In fact, cycle-accurate simulations have been carried out with uniform random traffic as well as with all the network switches in idle conditions. A value-change-dump file (VCD) has been annotated from the simulations and consequently utilized to carry out a very accurate power estimation with Synopsys PrimeTimePX. Figure 4.4 reports power consumption of both fully synchronous and mesochronous networks. Idle power plays in favor of the fully synchronous network. This is mainly due to the additional switchâtoâswitch clock wire used as strobe signal for data synchronization. This result calls for further evolution of mesochronous NoC technology, to implement a form of clock gating on these lines. On the other hand, when stimulating the networks with a uniform random traffic pattern, the mesochronous Network-on-Chip exhibits a smaller power consumption with respect to the fully synchronous one. The reason lies in the inherent architectural difference between the input buffer of the mesochronous switch and of the synchronous one. In this latter, both flip-flop banks are triggered at each clock cycle. Conversely, latch banks of the mesochronous input buffer are triggered by an enable signal driven by a counter. Since the counter logic enables only a single latch bank at a time, the ultimate register power consumption of the mesochronous input buffer is smaller than its synchronous counterpart. With a mesochronous NoC, an interesting opportunity pointed by [105, 107] is to exploit hierarchical clock tree synthesis to reduce power of the top level clock tree. The tuning knob to materialize power savings is the relaxation of the skew constraint, so that less buffers are instantiated in the top-level tree. We experimented this on the 4x4 mesochronous mesh by incrementally relaxing the skew constraint.

Given the relatively small system size, we constrained the top level tree to be placed and routed outside IP core area, which captures the challenging requirements of many real-life MPSoC designs.

## 7 Conclusions and Discussion

Evolution of MPSoCs to GALS systems is an ongoing process, driven by the immediate need to decouple voltage and frequency of IP cores from each other for power management. However, when a GALS NoC is implemented as an independent clock domain with dual-clock FIFOs at the boundaries, then physical designers have again to deal with a global chip wide clock tree (i.e., the one of the network itself). By capitalizing on mature mesochronous technology, this chapter compares a mesochronous NoC and a fully synchronous NoC (both for use in a GALS system) in a systematic way. The lesson learned from this experimental work can be summarized as follows:

**1.** A fully synchronous NoC can be evolved to a mesochronous NoC at no area and latency overhead because of the hybrid coupling design style.

**2.** During network activity, the mesochronous NoC proves more power-efficient because of the inherent clock gating implemented at its tightly coupled synchronizers in the switch input buffer. In contrast, a 20% higher standby power is incurred because of the transmitted and continuously switching clock signals in source synchronous links. A clock gating technique applied to the switch input buffer of the synchronous NoC and to the source synchronous links of the mesochronous NoC may align power results of the two solutions. In any case, the mesochronous NoC does not feature any significant overhead from a power viewpoint.

**3.** Hierarchical clock tree synthesis in a mesochronous NoC can potentially reduce total power of the top level clock tree at the cost of progressively loosening skew constraints in the same tree. However, power savings achieved in this way are not significant yet, for a number of concurrent reasons. First of all, there is a gap between the required maximum skew and the obtained one, since the CTS tool has been conceived for minimizing skew, and not for increasing it. Therefore, to take full advantage of this effect, CTS tools should be customized accordingly.

**4.** On the other hand, when tight skew constraints are required under challenging physical and timing constraints, the CTS tool is not able to meet the target. In practice, this means that with current CAD tools it will become rapidly impossible to enforce tight skew constraints in the top level clock tree. Under these operating conditions, it is important to have an underlying architecture with inherent skew robustness. In our experiments, mesochronous NoCs prove capable of meeting this requirement[1].

[2]

[1]This chapter has included contents that are referred to a cooperative and interdisciplinary work where furher details are in[50].

[2]This chapter has included contents that are referred to a cooperative and interdisciplinary work where furher details are in[50].

# Chapter 5

# Testing Archicteture on Top of The First Variant of the Mesh

## 1 Abstract

The digital design convergence, together with the new usage models of mobile devices, are raising the clear need for new requirements such as flexible partitioning, runtime adaptivity, reliability. In turn, such feature-rich architectures make the testing challenge more severe. The above trend has direct implications on the design of the underlying on-chip network, which becomes not only the system integration framework, but also the control framework executing hypervisor commands, or reacting to runtime operating conditions. The ultimate challenge for the NoC is to co-design these features together, while taking advantage of cross-feature optimization opportunities. This chapter takes on this challenge and illustrates the design experience of a NoC switch architecture serving as the key enabler for the next generation of reliable and reconfigurable systems.

## 2 Introduction

Network-on-Chip (NoCs) design principles have recently reached a stage where they start to stabilize[78, 79]. Unfortunately, the requirements on the design of an embedded system as a whole are far from stabilizing. More in detail, there is today an umistakable trend toward the implementation of heterogeneous architectures in mobile systems, combining a host CPU with a many-core programmable accelerator[80], or embedded GPGPU[81]. The provision of such acceleration by means of an array fabric of homogeneous processor cores yields unprecedented trade-offs between flexibility and energy efficiency. As a result, efficient exploitation of on-chip accelerators might go through time-multiplexing or time-interleaving of concurrent applications, which would incur large penalties in terms of serialization or context switches, respectively. It is therefore no wonder that partitioning is emerging as the fundamental paradigm for operation of many-core programmable accelerators, in order to pursue the integration of functionality from separate users/virtual machines onto NoC-based many-core architectures. A static partitioning scheme cannot keep up with the increased levels of adaptivity of modern embedded systems, therefore flexible partitioning should be the target, that is, partitions can be arbitrarily set up or tore down at runtime, and their shape may vary over time. A key enabler for flexible partitioning is represented by the ability to rapidly, safely and properly configure the routing mechanism of the underlying on-chip interconnection network, and to dynamically reconfigure it at runtime to expose the maximum level of flexibility to users. However, flexible partitioning and runtime reconfiguration is not the only requirement that fosters the next generation of on-chip networks. Today, due to the increased variability of components and breadth of operating environments, reliability becomes relevant to mainstream applications. Implementation of fault-tolerant communication over the on-chip network is still a challenging task in spite of the extensive record of research works in the field[86], due to the potentially large overhead that may affect the resulting architectures. In addition, a key property that novel NoCs cannot miss is to guarantee a potentially fast path to industry, since NoC deployment is today a reality. An important requirement for this purpose is the efficient testability of candidate NoC architectures. This property is very challenging due to the distributed nature of NoCs and to the difficult controllability and observability of its internal components. When we also consider the pin count limitations of current chips, we derive that NoCs will be most probably tested in the future via built-in self-testing (BIST) strategies. Although there is still ample room to research novel approaches to the specific challenges NoC architectures have to cope with, one key concern is to co-design the solutions to different challenges in an integrated framework. In practice, interdependencies between different NoC features should be detected ahead of time so to avoid the engineering of highly optimized solutions for specific problems, that however coexist inefficiently together in the final switch architecture. This chapter takes on the challenge of engineering a NoC switch for the next generation of feature-rich, highly reconfigurable and reliable systems, with thorough cross-feature optimizations. The novel switch design point features the following key properties:

- The routing function is reconfigurable at runtime, so to enable advanced network management policies such as network partitioning and isolation, virtualization, selective power down of specific regions; the network does not need to be drained for the sake of deadlock-free reconfiguration.
- Reliability protection is guaranteed through fault-tolerant flow control and through an error detection, notification and recovery infrastructure, with specific solutions for different kinds of faults. Single event upsets are detected and fixed on-the-fly, while intermittent faults are detected and addressed via routing path reconfiguration to avoid them.
- A BIST framework is set up targeting 97% coverage of single stuck-at faults of the entire feature-rich NoC switch design. The framework is conceived not only for post-silicon testing, but also for boot-time testing of the device.

The distinctive contributions of this chapter are the following:

- We come up with a feature-rich NoC switch that makes an important breakthrough in state-of-the-art NoC architectures;
- We show the interdependencies and smooth integration requirements between the reconfiguration framework, the fault-tolerance mechanism and the testing strategy in the proposed NoC switch.
- We quantify the steep increase in complexity that the new requirements demand for in the NoC switch architecture.
- We identify which NoC feature impacts switch area and critical path the most through analysis of incremental variants of the switch architecture.

## 3 Baseline Architecture

The switch architecture proposed in this work is a major extension of the baseline ×pipesLite switch [82], which targets the embedded computing domain with a very lightweight architecture.

The considered ×pipesLite variant implements logic-based distributed routing (LBDR): each switch has simple combinational logic that computes target output ports from packet destinations and local switch coordinates. By means of 26 configuration bits for each switch (indicating switch port connectivity, routing restrictions, and deroutes), the routing function can be reconfigured at runtime[84]. The straightforward yet overly expensive way to make the baseline switch fault-tolerant is through Triple Modular Redundancy. The only advantage is that the TMR architecture can afford keeping the native STALL/GO flow control unmodified. The baseline ×pipesLite architecture, as well as its TMR extension, will be used as reference design points in the experimental results.

## 4 Basic Design Choices for the New Switch

The proposed switch architecture is designed to be the basic building block of a reconfigurable and fault-tolerant NoC. Reconfiguration is achieved by means of a global controller implemented in software, which requires command execution support in hardware. A dual network is therefore designed to exchange control information between switches and the global controller. Reconfigurability is implemented as run time modification of the routing function, in order to provide not only flexible network partitioning when several applications are concurrently executed, but also to avoid faulty links/regions of the network. This latter functionality requires that points of failure are first detected, both at boot and run time, then notified to the system manager, that triggers the reconfiguration accordingly. The basic design techniques to deliver the target func- tionality (boot-time testing, fault tolerance, control signalling, and routing function reconfiguration) are hereafter described, while their interdependencies are captured in section 5.

### Fault-Tolerance

Whether a fault-tolerance switching strategy should affect the flow control protocol or not is a major design choice with high impact on the overall switch architecture. The work in[87] derives error recovery strategies for the same NoC switch both from a flow control protocol with error notification capability (NACK/GO) and from another one lacking this support (STALL/GO). Data retransmission is used in the former case, while the latter one can only rely on error correction. It has been shown that NACK/GO potentially results in shorter critical path, more conservative area and lower peak power, at the cost of a slight average power overhead. This led us to opt for NACK/GO for the proposed switch architecture (Figure 5.1). The proposed solution targets single event upsets (SEUs). In the data path, detectors trigger flit retransmissions from the sender buffer, which is preceded by correction of the stored flit in case it were corrupted in the buffer. On the control path, FSMs are triplicated to avoid their permanent misalignment, while routing and arbitration logic is just doubled, since dual-rail checkers (DRCs) can trigger retransmissions from the input buffer upon mismatch detection.

### Notification Interface

In this work, we opt for a centralized approach to network control: a global manager is in charge of network reconfiguration decisions as an effect of fault-tolerance, power management or virtualization strategies. In order to address the need for control signaling between network nodes and the global manager, we revert to the dual communication infrastructure proposed in[89], where the main NoC is extended with a ring which connects all the switches of the main NoC together. The ring implementation implies the extension of each switch with a simple routing primitive, which is an oversimplified version of an input buffered switch.

### Reconfigurability

The reconfiguration mechanism of the routing function in the presence of background traffic should provide deadlock freedom during the transition from one routing algorithm to another, when extra dependencies may arise and lead to deadlock. To cope with this issue, our switch leverages Overlapped Static Reconfigurations (OSR), a technique which avoids draining network traffic[88]. OSR was first proposed for off-chip networks, and its customization for a much more resource-constrained on-chip setting, named OSR-Lite, has been performed in[83]. The basic principle is the following: if packets with the old routing function are guaranteed to never go behind packets using the new routing function, then no deadlock cycles can occur. In OSR this is achieved by triggering a token that separates old packets from new ones. The token advances through the network hop by hop, following the channel dependency graph of the old routing function, and progressively drains the network from old packets, allowing new packets to enter the network at routers where the token already passed.

**Boot-Time Testing Architecture**

This work implements a BIST strategy for the new switch, targeting single stuck-at faults. All network switches are tested and diagnosed in parallel, thus cutting down on test application time and making it independent of network size. We envision a unique 39 bit LFSR per switch that feeds pseudo-random test patterns to every switch port in parallel. Test responses of a switch sub-block are fed to connected sub-blocks, serving as test patterns for them, as much as possible, depending on whether aggregate coverage remains reasonable or not. This approach minimizes test pattern generator (TPG) and test wrapper overhead. Link testing is performed in a cooperative way: test patterns are injected by the upstream switch while diagnosis is performed by the downstream one. The BIST architecture is depicted in Fig.5.2. Test responses are collected by Multiple Input Shift Registers (MISRs). One MISR performs signature analysis of test responses from the link, the input buffer and the LBDR block together with the associated OSR-Lite logic. Another one collects responses from a crossbar mux, an output buffer, a port arbiter and the associated OSR-Lite logic. The testing framework is able to reveal the correct position of faults inside the switch, since a MISR is dedicated to each input and output port.

# 5 Cross-Feature Optimizations

Supporting fault-tolerance, reconfiguration, notification and testing in the same switch architecture gives rise to relevant design inter-dependencies, summarized in Fig.5.3. They are either opportunities to exploit, or requirements to enforce to ensure correctness, as hereafter described.

**A.** The support for dynamic reconfiguration of the routing function suggests an efficient approach to the detection of and recovery from wear-out faults. These latter typically exhibit a progressive onset, consisting of frequent transient faults affecting the same circuit (intermittent faults) [90]. The NACK/GO infrastructure can detect transient faults and notify them (and their location) to the global manager. The global manager can thus monitor the frequency and the location of these events and eventually trigger a reconfiguration to exclude the affected circuits before they become permanently damaged.

**B.** The OSR-Lite mechanism for NACK/GO switches should accurately define when a switch port can migrate to a new routing algorithm. We activate this latter when the token is received, and the last packet of the old routing function has left the port while getting an ACK back from the receiver side for it.

**C.** Fault-tolerance support in the switch has a double effect on testing effectiveness. On one hand it ends up reducing the observability/controllability of signals and hence the testing coverage. Voters and correctors in fact mask errors of associated modules. Moreover, error detectors natively prevent propagation of test responses across the data path upon error detection. Hence, they cannot serve as test patterns for downstream modules, thus limiting fault observability as well as downstream controllability. Therefore, we identified some workarounds to preserve testing coverage. In order to improve observability, voter inputs are brought to diagnosis logic that detects whether there are discrepancies or not. To increase controllability of the data path, we feed it with fully random test patterns from the LFSR, which are with high probability incorrectly encoded (we use BCH code for data protection). Test wrappers prevent the detector from driving the buffer FSM at each cycle, thus enabling from time to time the test pattern to go through the data path. Another wrapper randomly enables the corrector from time to time. When this happens, test patterns for the data path are not provided by the LFSR, but by the corrector itself. On the other hand, fault tolerance has also some benefits on testing, that we exploited. In fact, replicated logic through TMR and DMR provides a built-in support for diagnosis by enabling the comparison of replicated outputs. Similarly, the error notification output of the decoder was brought directly to a MISR for diagnosis.

**D.** Having a testing strategy enables the fault-tolerance framework to take the absence of permanent faults for granted. Having our testing strategy iterated at every system bootstrap increases the confidence level of this assumption. Therefore, at every usage session, the user is informed about whether the fault-tolerance logic has its full recovery capability from transient faults at runtime, or whether it is degraded by permanent faults to some extent.

**E.** By having a notification infrastructure in place, and a global manager for control tasks, implies that switches do not need to notify each other the outcome of testing and diagnosis. This avoids the implementation of complex notification protocols between the switches. Instead, after testing completes at every system bootstrap, per-switch diagnosis bits are notified to the global manager which processes them, gets global visibility, and programs the routing function of the switches accordingly.

**G.** In the absence of a dual control network, overprovisioning of the main NoC would be needed to support the reconfiguration process. For instance, in[88] a dedicated virtual channel is instantiated for this purpose. Vice versa, we use a VC-less main network and a simpler dual network to convey key reconfiguration information.

**F-H.** The reconfiguration and the testing support dictate the packet format on the dual control network. In particular, packets are composed of 2 or 3 flits. The first flit always contains information about the delivery time of the packet (for fault statistics purposes), the type of information (i.e., BIST testing result, reconfiguration bits or transient fault notification), and either the source or the destination switch address. When BIST testing information are delivered to the global manager, then the second and the third flit contain respectively the diagnosis result and its negated for information time redundancy. Differently, reconfiguration packets back to switches require new LBDR programming bits in both the second and the third flit. Finally, online fault notification comes with only two flits, with the second flit reporting the input-output port combination that triggered a transient fault.

**I.** As mentioned in point G, the first system configuration at bootstrap follows the BIST testing procedure, and does not take place in the presence of background traffic. Therefore, we avoid token propagation of OSR-Lite at this stage: and switches are programmed as they get the new routing information through the dual network.

**J.** It is useless to localize faults inside modules that should be entirely disabled regardless of the internal fault position. An LBDR-aware algorithm, executed by the global manager, searching for routes on a given topology, needs only information about the faulty links to generate a compatible routing function. As a consequence, we group switch logic blocks based on the switch input or output port they belong to, and treat port failures as the failure of attached links. Thus, we end up having one diagnosis bit (pass/fail) per switch port/link

**K.** As explained in point A, the switch is able to detect transient errors. However, since we are targeting also intermittent faults, we notify the occurrence of transient faults to the global controller through ad-hoc control packets, which also carry fault localization information.

**L.** The dual control network is a single point of failure, since the correct configuration of the system depends on its capability to safely deliver information from switches to the controller and vice versa. For this purpose, we took the approach in [10] and combined fault-tolerance design techniques with online testing strategies on the dual control network, at the cost of some extra communication latency on it. If a global manager detects that communication with a switch is not fully reliable, the switch is discarded, the topology is modified and a new routing function computed. We architect the dual network under conservative assumptions, so that the probability of

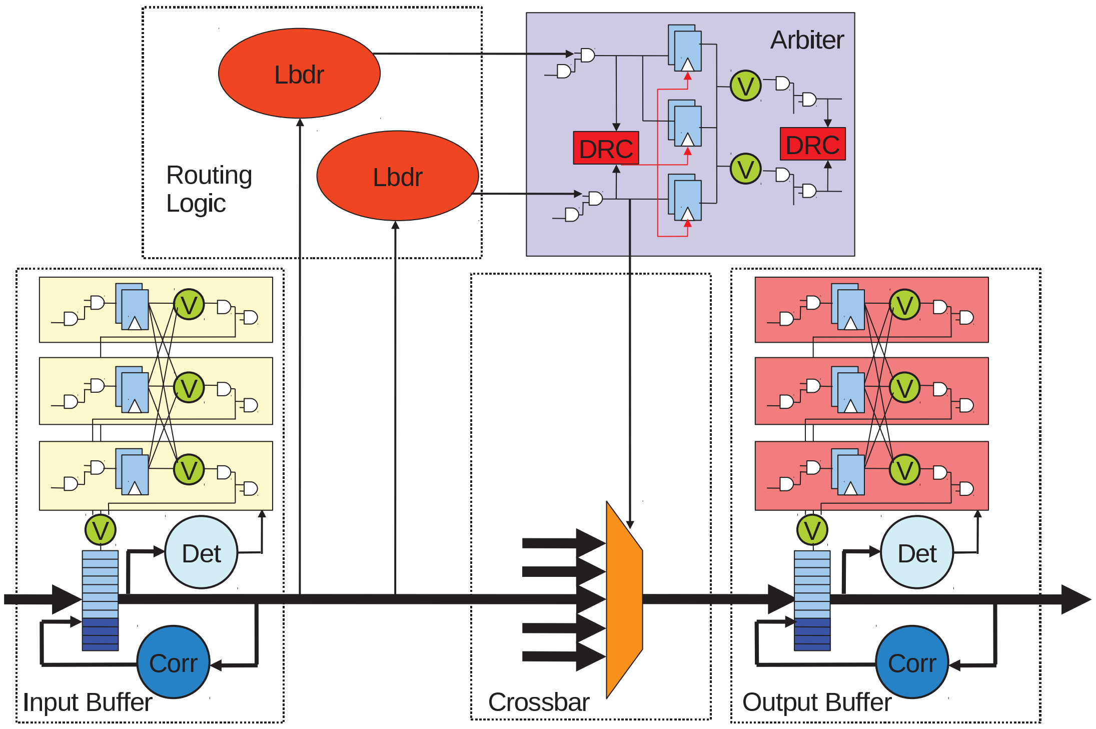


Figure 5.1: NACK/GO switch architecture.

a deviation between wanted and actual routing configuration is marginal.

# 6 Experimental Results

All the logic synthesis runs performed in this work have been carried out by means of a low-power standard-Vth 40nm Infineon technology library.

### Complexity Breakdown: Area Results

The following experiment points out both the complexity gap between the native xpipesLite switch and the feature-rich extended one, and the area increment that each integrated switch feature contributes. Normalized area results are shown in figure , where features are incrementally added to the baseline switch. This and its TMR extension are reported as reference design points. Fault-tolerance is clearly the highest-impact feature. A non-negligible

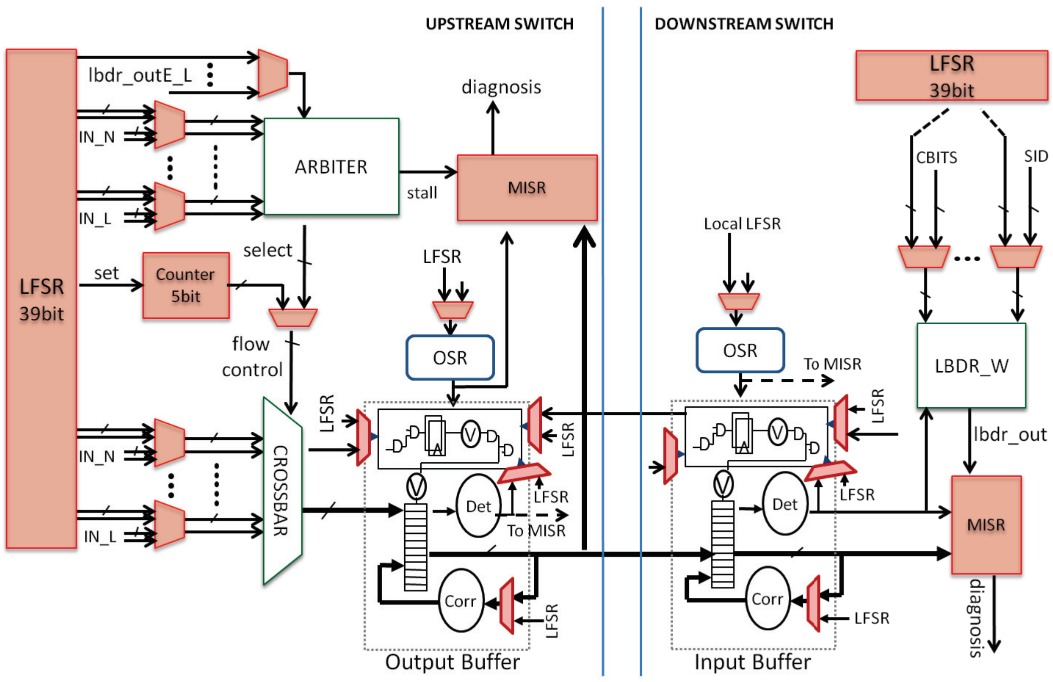


Figure 5.2: BIST-enhanced switch architecture.

area contribution comes in fact from detector and corrector modules and accounts for almost 13% of the total area. When the reconfiguration mechanism is integrated into the NACK/GO switch, an 11% of area overhead is introduced. The notification system (TMR-protected dual network) results lightweight (5% area overhead) since it takes advantages of the diagnosis logic already made available for fault-tolerance purposes. Finally, the switch capable of built-in self-test and self-diagnosis brings a 27% of area overhead, which is the second major source of complexity after fault-tolerance. The area penalty mainly comes from MISRs and test-wrappers. Despite the use of pseudo-random test patterns, which typically save area with respect to deterministic ones in BIST approaches[85], the control logic to be tested is so complex that test wrappers need to penetrate deeper into the switch architecture to improve controllability and observability. However, when we consider a baseline TMR switch which implements only fault-tolerance on top of a baseline xpipesLite switch, we can see that the proposed switch (rightmost bar in the plot) provides many more features at comparable area footprint.

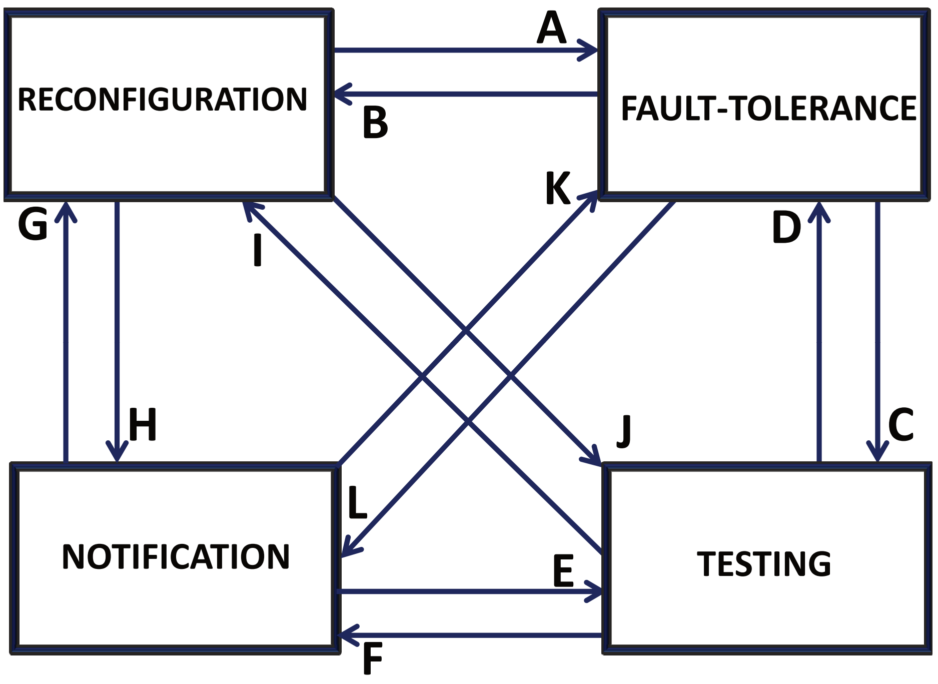


Figure 5.3: Interdependency Diagram between Reconfiguration, Fault-tolerance, Testing and Notification.

**Complexity Breakdown: Delay Results**

In order to evaluate the effects of each additional feature on the switch propagation delay, we performed a 5x5 switch synthesis for maximum performance for all the 5 incremental solutions under test. Results are reported in Fig.5.5. The fault-tolerant NACK/GO switch, the switch with OSR-Lite mechanism and the switch with notification system achieved a similar maximum operating speed. Finally, the testing framework degraded by 13% the maximum performance of the NACK/GO switch. The performance of the switch is limited by the test-wrappers placed on the critical path. Considering the TMR solution, this is around 30% slower than the baseline switch while the proposed switch delivers far more functionalities at the cost of a longer critical path (+13%).

**Coverage for single stuck-at faults**

Table 6 reports the total number of cells associated to each tested module, and the related achieved coverage by testing. This latter was derived

| Switch sub-block | Cells | Coverage |
|---|---|---|
| OSR-Lite | 360 | 95.0% |
| Arbiter | 334 | 96.7% |
| Crossbar | 154 | 97.4% |
| Input Buffer | 1272 | 97.6% |
| Output Buffer | 1855 | 97.4% |
| LBDR | 289 | 91.0% |
| TOT | 4264 | 96.8% |

Table 5.1: Coverage for single stuck-at faults.

by means of an in-house made gate-level fault simulation framework. Worse results are obtained for the OSR-Lite mechanism (â$\frac{1}{4}$95%) and especially for the LBDR (â$\frac{1}{4}$91%), as a direct consequence of their FSMs and combinational logic complexity, respectively. Concerning the testing latency, a network composed of the proposed switches, as assumed so far, would take 10.000 clock cycles for testing, regardless of the network size.

## 7 Conclusions

In this chapter, we propose a fully testable switch architecture endowed with fault-tolerance, notification infrastructure and overlapped static reconfiguration capability. We showed the major step in design complexity with respect

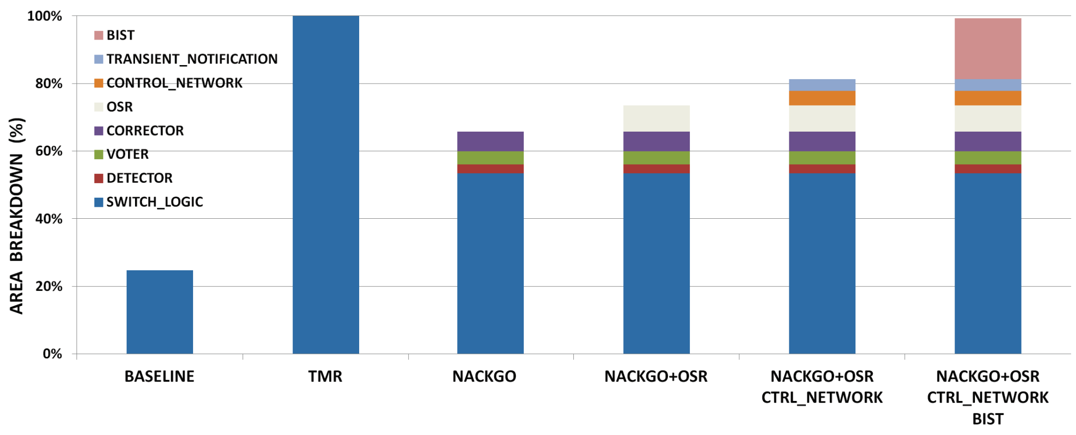


Figure 5.4: Area analysis.

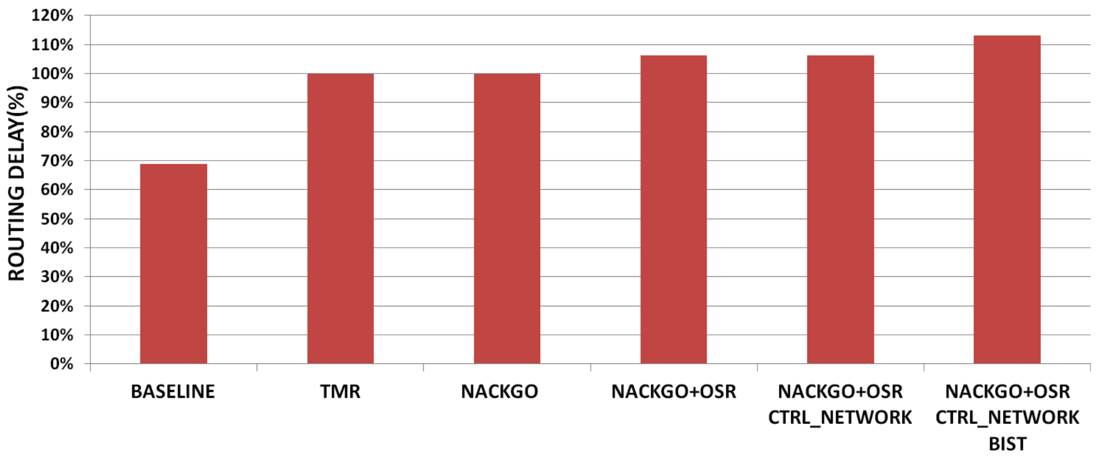


Figure 5.5: Routing delay analysis.

to a state-of-the-art switch for low-to medium-end embedded systems, arising from the more aggressive requirements on switch functionality. At the same time, we showed that more functionality than TMR can be delivered within the same area budget, but with a non-negligible speed penalty. Overall, this chapter detected the interdependencies between the different design features and addressed them all in the coherently integrated final switch microarchitecture.

# Chapter 6

# Testing Architecture on Top of the Second Variant of the Mesh

## 1 Switch Architecture

A parameterized $n \times m$ (n: number of input ports, m: number of output ports) source based routing 2-stage switch has been designed for the application specific computing domain, augmented with fault-tolerance provisions. The scheme of the switch architecture is depicted in figure 6.2 and is composed of the following main blocks:
- A fault tolerant Input buffer of two slots with triplicated control logic and endowed with voters.
- A fault tolerant Output buffer of six slots with the same characteristics of the input buffer.
- A fault tolerant Arbiter, triplicated and endowed with voters.
- A Path-Shift module and a Crossbar.
- Some comparators are placed in specific places for runtime diagnosis and to notify the global manager.

The switch model was generated to be parameterizable to meet the requirements of application-specific topologies. Parameters that can be set include the number of input and output ports and the width of the data portion of the flit. The width of the checkbit portion is derived accordingly.
We will now describe the behaviour of each block of the switch, then we will present the testing architecture with experimental results.

## 1.1 Tightly Coupled Dc_FIFO

Synchronization interfaces, such as dual-clock FIFOs, are typically instantiated as external blocks with respect to the module they are connected with. This âloose couplingâ of synchronizers with respect to NoC components implies several drawbacks. First, the FIFO module introduces additional communication latency in the intercommunication link. As a result, provisions must be normally made since the flow control signal may arrive multiple clock cycles after the destination module decides to halt the source module. The problem can be addressed by reserving space in the destination buffer, thus incurring a significant area and power overhead, or by enhancing the dual-clock FIFO with flow control capability.
In the EU-funded GALAXY project, the aforementioned problem was tackled by merging the dual-clock FIFO with the switch input buffer, thus coming up with a unique architecture block in charge of buffering, synchronization and flow control, and sharing buffering resources for all of these tasks. The GALAXY project has also showed that this design principle, which we denote as âtight couplingâ of synchronizer with the NoC, can be applied to dual-clock FIFOs in a straightforward way. For this reason, the switch can optionally replace its input buffer for a fully synchronous environment with a dual-clock FIFO for a multi-synchronous environment, as illustrated in Figure 6.1. In all cases, functional correctness is guaranteed.
A similar functionality can be easily implemented also in the switch provided the dc_FIFO is extended with the NACK/GO flow-control protocol.

## 1.2 Input/Output Buffers

Input and output buffers are much simpler than the first switch variant, since they do not have to handle the Nack/go flow control protocol but rather the simpler stall/go one. The input buffer is sized with two slots, which is the minimum amount of resources needed not to lose data during stall activation. It was 3 with Nack/go. The output buffer can be arbitrarily sized for performance buffering. As previously mentioned, control logic of input and output buffers is triplicated for fault tolerance and endowed with voters. An additional voter is placed on top of the data-path registers with

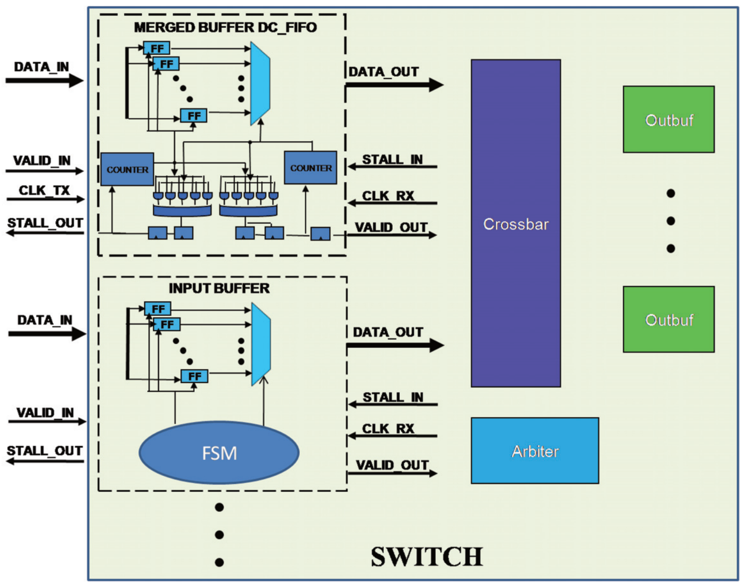


Figure 6.1: Dual-Clock FIFO integration into one input port of the switch architecture.

the purpose of voting the outputs from the three instances of the buffer control logic. The voted output drives the read and write pointers of FIFO data registers.

## 1.3 Probing System

At the same time, probes inserted in front of each voter sniff their inputs and inform (through a comparator and an OR gate) the global manager about possible malfunctioning of each of the replicated branches. We find it important that the manager can keep this kind of information under control, so to be aware of a possible degradation of the fault-tolerance capability of the architecture. The OR gate collects the outputs of the comparators associated with each voting stage, as well as a notification signal from the correction sub-system denoting whether correction actions have been performed or not. Through the OR tree, a global notification of malfunctioning is achieved for each switch and notified to the global controller via a star interconnection topology. In fact, we do not need a fine-grain diagnosis like in the first variant, since if a switch component starts to fail repeatedly, we do not envision any reconfiguration course of action in the system nor it might be possible in the application-specific hardware platform at hand.

## 1.4 Error correction

Error correction was the preferred fault-tolerance strategy given its capability to stretch device lifetime as much as possible in the presence of permanent faults. The Hsiao code was used to implement the corrector. However, differently than the first switch, an encoder needs to be integrated since with source based routing the heat flit needs to be changed. Therefore, parity check bits for the new flit need to be computed. Flit width can have different sizes and this depends on data width and on the number of parity checkbits. For example, for a data width of 32, 48, 64 or 128, their respective compound flit width will consist of 39, 55, 72 and 137 bits. The number of checkbits depends in fact on data width and is composed of 7, 7, 8 and 9 bits for respectively a 32, 48, 64 and 128 bit data width.
These checkbits are computed by means of Hsiao encoder and are appended to the data word before injection into the crossbar, as illustrated in figure 6.2.
A corrector module named "Corr" and located after the data registers of input buffers controls that the received checkbits match the computed ones for data in transit. If checkbits are not the same, this means that the data is corrupted and needs to be fixed. The corrector is able to detect at least two errors and to correct only one. The corrector is also endowed with a one bit output that informs the global manager each time computed checkbits don't match.
After the corrector module, there is a pipeline stage to cut down on the switch critical path due to additional error correction logic,changing critical path delay into latency. The pipelined organization of the switch will enable in the future to replace the current corrector with more powerful ones, without or marginally impacting the clock speed. In the presence of switch pipelining, integration with flow control is critical not to waste data and not to incur throughput penalties. The switch implements a custom solution for this purpose. A stall signal arriving from output ports through port arbiters is brought both to the input buffer and to the single-slot pipeline registers, where it serves as the enable signal. This way, we can avoid using two slot

buffers as pipeline registers, as typically required for cascading stall/go retiming and flow control stages. In fact, the transmission can be frozen directly in the upstream input buffer, while preserving the capability of the pipeline register to store 1 flit. At the same time, there is no performance penalty, since flow resumption is immediate and in case there is no pushing data from input buffers the stall signal is not activated from the output buffers (i.e., we never prevent the pipeline register to get used).
At the output of the pipeline stage, flits are routed across two main paths:
- Towards the Path-Shift Module
- Towards the Arbiter Module

## 1.5 Path Shifting

The Path-Shift module is composed of the following blocks:
- A demultiplexer, immediately inserted after the output of the pipeline stage. It is composed of two inputs (data input and select input) and two outputs (one for head flits and the other one for payload/tail flits).
- A Shifter and an Encoder placed along the path followed by head flits.
- A 2x1 multiplexer
When a new flit arrives in front of the Path-shift module, we need to identify the flit type, i.e., whether it is a head flit or not.
For doing this, the select input port of the mux/demux is directly controlled by the first bit of the input flit.
In fact, this bit is set to ”1” for a head flit and to ”0” for payload/tail flit.
So, when the input flit is a tail or a payload, path shifting is bypassed.
On the contrary, when a head flit arrives, we need to shift the routing information so that each switch can always find in the same position its target output port. Alternatively, we would need to embody in the packet the indication of how many hops the packet has already gone through. This way, the switch would have to point every time to a different location in the packet head. After shifting the address bits, checkbits are not meaningful any more and need to be recomputed by the encoder before the flit can move on.

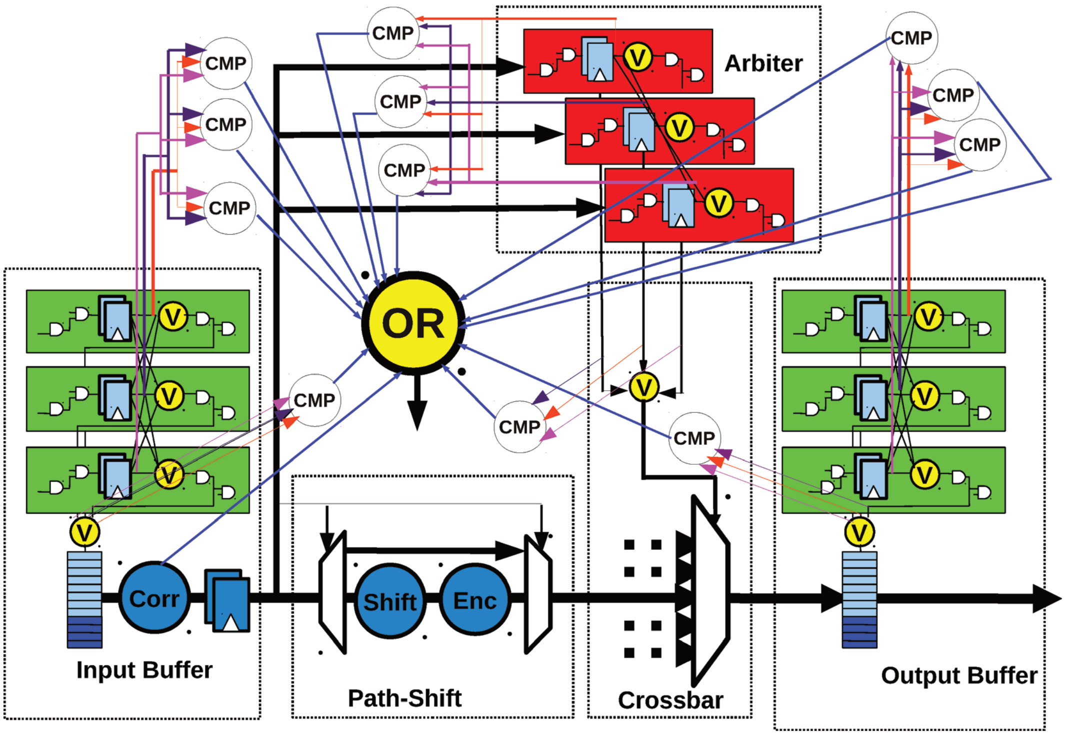


Figure 6.2: The Second architectural Variant of the Mesh at a glance.

### 1.6 Control Path

Arbitration is performed with a round robin arbiter with triplicated control logic. Each instance of the arbiter is endowed with voters for self-correction; additional voters are located on top of crossbar multiplexers for reconvergence of the control path to the control inputs of the data path. Similarly to the first variant switch, a new arbiter state is saved only after voting it, to make sure that triplicated FSMs do not get misaligned as an effect of errors. This would compromise reliability of the control path for future transactions.

## 2 Testing Methodology

The main challenge of testing an application specific system consists of the highly heterogeneous nature of the system itself, where every core (or cluster of cores) might be operated at a different speed. For this purpose, the use of a dual-clock FIFO between IP cores becomes mandatory. However, core speeds are not typically fixed, but they may vary within a range. After consulting

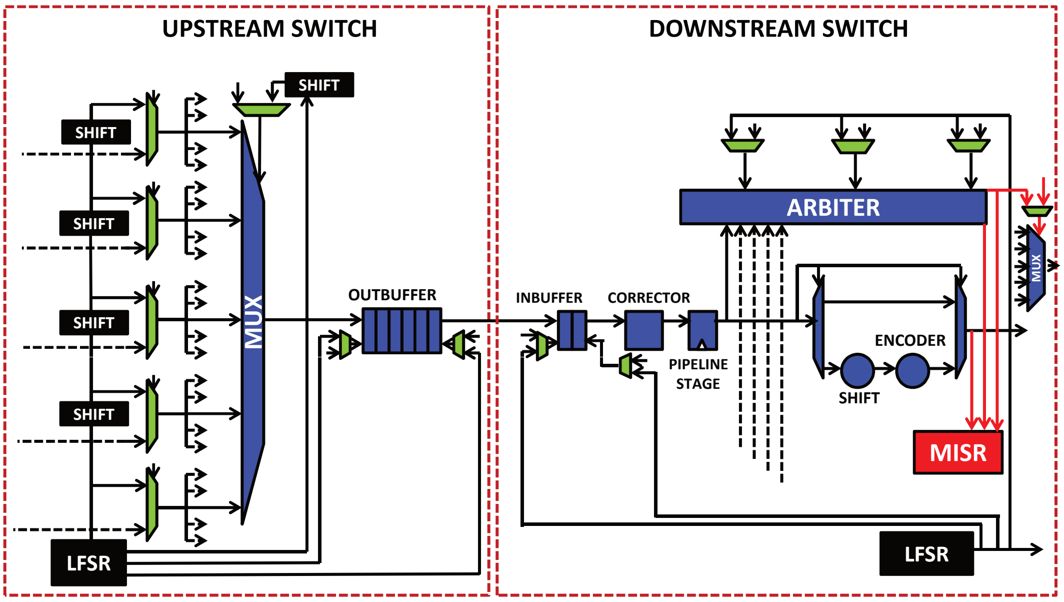


Figure 6.3: Testing Architecture. In green, the test wrapper is pointed out.

with the industrial partners, we agreed on the feasibility of bringing all the cores/clusters to the same speed for the sake of testing. This can be achieved with frequency regulators or with a dedicated low-speed compact PLL for system testing. Under these assumptions, we can assume to be able to test the system under fully synchronous timing, as will be done throughout this deliverable. Alternatively, the NaNoC design platform provided in deliverable D1.4 a methodology to test multi-synchronous links through an asynchronous handshake for the exchange of test patterns between frequency domains.

Due to requirement to limit the amount of area for the embedded testing logic, and considering a milder approach to testing latency optimization, we decided to adopt a testing approach based on pseudo-random test patterns. The approach and the philosophy are pretty similar to the ones used for the first variant switch, except for a few architecture-specific customizations. Diagnosis is again performed with Multiple Input Shift Registers (MISRs). The overall testing architecture is depicted in the picture of figure 6.3.

As we can see, each switch is endowed with one worst case LFSR of 39 bits. The LFSR feeds through a test wrapper all the crossbar multiplexer (”MUX”) inputs with a 6 bits shift between any two adjacent inputs. This is needed to

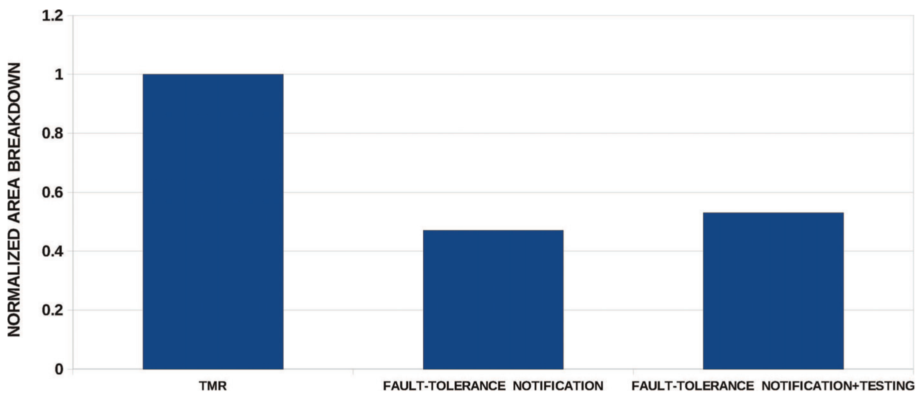


Figure 6.4: Area overhead @500MHz.

get a satisfactory coverage for the multiplexer.
Meanwhile, the same LFSR drives the select input ports of each multiplexer of the crossbar with a 1 bit shift position between each multiplexer, again to stimulate all cells and improve the coverage.
At the same time, the LFSR drives the stall input and the valid input of each input/output buffer, and the local ID flags, busy_in and valid_in of each arbiter.
The data inputs of each arbiter are driven by test vectors from the upstream switch, again implementing the principle of cooperative and parallel testing between NoC switches.
A MISR placed after the arbiter and before the crossbar multiplexer of the downstream performs the diagnosis.

## 3 Experimental Results

This section describes the experimental results of a 5x5 switch (second variant) synthesized at the target speed of 500MHz with the 40nm low-power SVT Infineon technology library. Input buffers are assumed to be fully synchronous.
Figure 6.4 shows the area overhead of the above switch (rightmost bar) with respect to an intermediate implementation without any testing support and to a baseline TMR extension of the xPipeslite switch (see D2.1). It can be

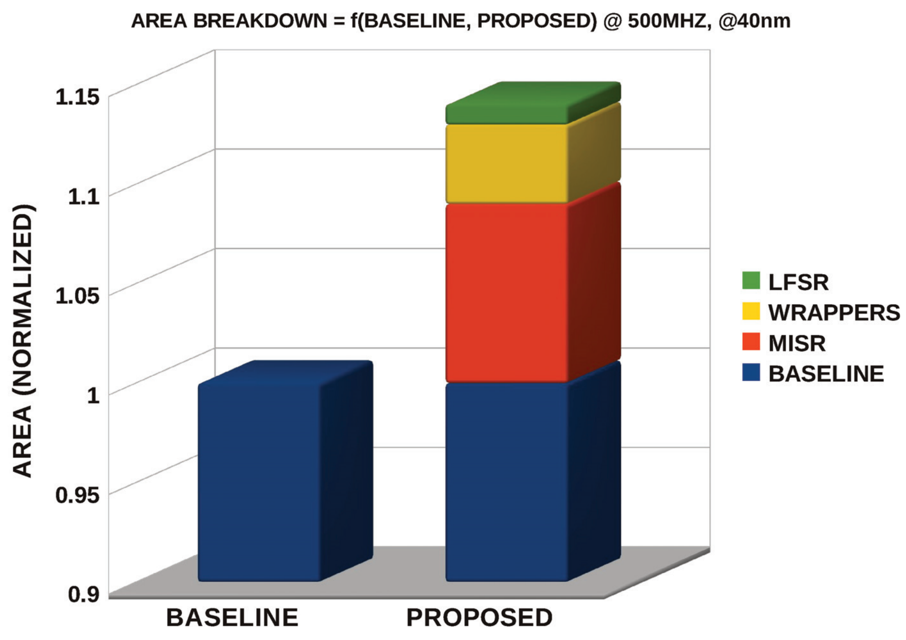


Figure 6.5: Testing overhead and area Breakdown.

observed that area overhead for testing amounts to only 12.96%. At the same time, more functionalities and provisions than the TMR switch are delivered at a much lower area footprint. Figure 6.5 illustrates the area breakdown of the testing logic in the switch. The major contributor to the testing logic comes from the MISRs used to perform the diagnosis and counts for ∼8.50%. The remaining part of the overhead is spread among the wrappers and the LFSRs used as test patterns generators.

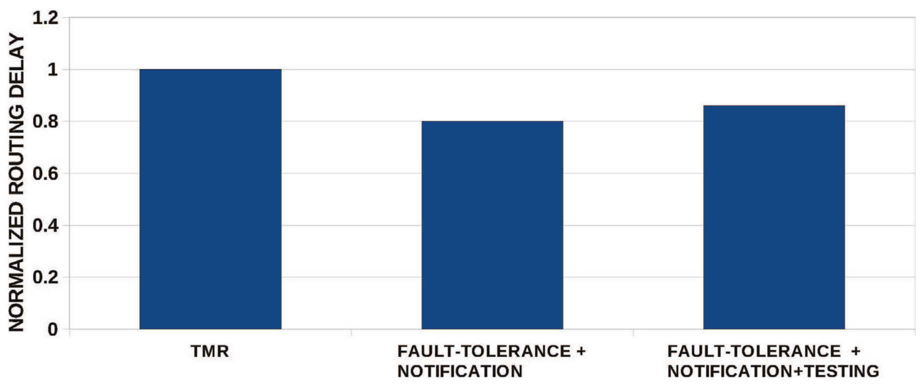


Figure 6.6: Normalized routing delay @Max Performance.

| Switch sub-block | Cells | Coverage |
|---|---|---|
| Arbiter | 393 | 96.43% |
| Multiplexer | 144 | 99% |
| Input Buffer | 803 | 99.5% |
| Output Buffer | 640 | 98.7% |
| Encoder | 136 | 99.6% |
| Shift | 56 | 100% |
| TOT | 2172 | 98.7% |

Table 6.1: Coverage for single stuck-at faults.

The number of cycles required to reach 98.7% coverage of single stuck-at faults in the whole switch is ∼10.000. We found this a satisfactory trade-off given the use of pseudo-random test patterns.
Table 6.1 reports the number of cells and the coverage of each switch sub-blocks. We used the same fault simulation framework as for the first variant switch. The overall data-path can achieve a coverage of ∼99.19%, while the remaining control-path is ∼96.43%; this is due to the complexity of the FSMs inside the arbiter.
Figure 6.6 finally shows the implication of the switch features on its critical path. The testing logic affects the maximum operating speed of the switch. In fact, routing delay becomes ∼8% larger than the one of the switch without the testing logic. Nevertheless, the maximum operating speed achievable is still higher than that of the TMR reference solution.
Last but not least, when replacing the input buffer with a dual_clock FIFO, in practice there is no area overhead provided we keep the number of buffer slots the same. Actual buffer sizing then depends on network-level requirements such as the speed ratio between sender and receiver as well as the needed throughput across a multi-synchronous link[52].

# Chapter 7

# Ultra-Low Latency NoC testing via Pseudo-Random Test Pattern Compaction

## 1 Abstract

This chapter aims at devising an optimized pseudo-random test methodology for NoCs and its architectural support. The guiding principle consists of using a test pattern compaction engine for generating minimal test lengths. We show the application of this principle driven by the objective to minimize test application time, at the cost of test wrapper complexity. The achieved design point results in a reduction of test application time by two orders of magnitude with respect to state-of-the-art test architectures for NoCs exploiting pseudo-random patterns.

## 2 Introduction

All systems-on-chip (SoCs) should be tested for manufacturing defects. Such testing procedure turns out to be particularly challenging for those large scale SoCs making use of a network-on-chip (NoC) as their communication backbone: the controllability/observability of NoC links and sub-blocks is relatively reduced since they are deeply embedded and spread across the chip.

This issue adds up to the newer challenges of testing generic large digital designs in nanoscale technologies. For instance, pin-count limitations restrict the use of I/O pins dedicated for testing. Other concerns regard the use of external testers, which has been a mainstream testing practice so far: lack of scalability of test data volumes and high cost for full clock speed testing. Finally, wear-out mechanisms such as oxide breakdown, electro-migration and mechanical/thermal stress become more prominent in aggressively scaled technology nodes. These breakdown mechanisms occur over time, therefore the methodology and the infrastructure used for production testing should be designed for re-use during the system lifetime as well. This again urges new testing strategies. Built-in Self-Testing (BIST) to some extent overcomes the above problems since test patterns are generated and evaluated on chip. By exploiting the precise knowledge of the architecture and of the circuits under test, a designer can come up with deterministic test patterns, thus potentially resulting into minimized test sequences and superior coverage results. Unfortunately, engineering such handcrafted deterministic patterns is a largely manual and time consuming task which should be performed again in case of technology library migrations or circuit modifications. Moreover, placement and routing of the design will certainly modify the gate-level netlist, thus making coverage expectations not fully trustworthy.

For synthesized logic, which is by far a relevant part of an embedded system, pseudo-random test patterns are frequently used because of their higher flexibility. They potentially result in a lightweight test architecture due to the simplicity of the linear feedback shift registers (LFSRs) and of the multiple input signature registers (MISRs) used to generate test patterns and signatures respectively. The work in [50] proves a 20% area saving in 45nm technology when a BIST architecture for NoCs is fed by pseudo-random patterns rather than by handcrafted deterministic ones. Unfortunately, testing with pseudo-random patterns also typically dominates the total runtime of the BIST: in [12] 200000 cycles are reported for testing the main modules of a NoC switch with such patterns. A reduction of one order of magnitude in testing latency has been proved in [50], however authors there take a hybrid approach, where pseudo-random test patterns are combined with deterministic ones and with architecture-specific DfT optimizations.

In this chapter, we view the flexibility and the reduced test latency requirements as fundamental for efficient NoC testing. On one hand, re-engineering deterministic patterns for each product evolution or technology migration is not cost-effective. On the other hand, overly long test application times are not compatible with the lifetime testing paradigm, where a testing procedure may be run at least at each system bootstrap.
As a result, the main objective of this chapter is to develop an ultra low-latency testing framework for NoCs capable of achieving such unprecedented test application times (below 250 cycles regardless of the network size) without reverting to deterministic patterns. In contrast, we start from pseudo-random patterns and develop a testing methodology and its architectural support in the NoC that preserve the generic and flexible nature of the patterns. Such methodology and test architecture are therefore pretty general in scope and can be applied to any NoC architecture other than the one considered in this chapter. Test set compaction is at the core of our testing framework. In order to minimize the test application time we chose to compact test patterns for combinational logic blocks, where compaction efficiency is more likely to outperform that achievable for sequential circuits. Registers are tested with standard techniques. Combined with the concurrent testing of switch sub-blocks, this approach achieved an order of magnitude lower test application time than current literature.
The trade-off is clearly with the implementation complexity of the test wrapper, which needs to isolate the circuits that are concurrently tested and to connect them to their test pattern generators (TPGs) and response analyzers. This chapter proposes also an optimization strategy to limit such overhead which consists of cascading several circuits under test (test responses of one block become test patterns for the next one) and of exploiting the synergies between them.

## 3 Related Work

As the integration densities keep increasing, on-chip interconnection networks are becoming the reference communication backbone for multi-core computing in many embedded high performance systems [10], [7]. However, defects

still continue to increase and are more prominent in scaled technology. To cope with this high defect rates, many test mechanisms have been proposed. For example, [15] and [19] propose a test mechanism for regular and modular systems like on-chip networks, but they incur a considerable area overhead. In the same way, [18], [16], [20] and [17] have proposed full-scan and boundary scan strategies, but they still have a high area overhead as showed in [14]. An alternative approach to reduce this overhead could be the partial scan technique, however its testing time is the main drawback (tens of thousands of clock cycles).
Another solution consists of applying test patterns from the input/output borders of the network [8]; this approach has been extended in [9] in such a way to support the diagnosis too. However, this approach is limited by the high number of necessary test pins.

[3] proposed a testing framework based on handcrafted deterministic test patterns and exploits the inherent structural redundancy of NoC switches. This deterministic approach leads to one of the fastest testing times reported in the literature for single stuck-at faults. However, this approach requires the in-depth knowledge of the architecture under test and an extensive effort to carefully engineer handcrafted test patterns for it. On the same NoC architecture of [3], the work in [50] implements a test architecture fed by pseudo-random patterns. The achieved design point cuts down on the area overhead by 20% in 45nm technology but provides comparable coverage in one order of magnitude more test application time. On a different switch architecture, [12] reports several tens of thousands of cycles for pseudo-random testing of most parts of the switch, confirming that the use of such patterns inherently plays against testing latency.

In this chapter, on the same NoC architecture of [3] and [50] we provide a new design point, which achieves a high fault coverage with generic test methodology steps and architecture design techniques (i.e., no deterministic patterns). We rely on existing test set compaction tools (namely [4] and [5]) to cut down on testing latency, and elaborate on the test methodology and on the architecture support needed to achieve unprecedented values of test application times. The key challenge this chapter deals with is to identify the most suitable circuits for test set compaction, so to maximize compaction efficiency and minimize test application time. At the same time, the implementation complexity of test wrappers, TPGs and response analyzers are kept under control.

## 4 Testing methodology

The philosophy behind this work is that pseudo-random test patterns are desirable for NoC testing since they can be easily reused and extended across architecture variants and technology migrations. In fact, they save the considerable effort and time to develop handcrafted deterministic patterns for the architecture under test.
On the other hand, we believe that some optimizations with respect to their naive application to NoCs are necessary to reduce test application time. This chapter pushes this consideration to the limit and tackles the challenge of materializing an **ultra-low latency testing framework for NoCs starting from pseudo-random test patterns**. The applied optimizations then retain the generic and flexible nature of the testing methodology by never reverting to deterministic test patterns.
We found test set compaction a suitable architecture-agnostic step to optimize test patterns, where the specific architecture implementation comes into play only in determining the achieved compaction efficiency. In this chapter we applied state-of-the-art compaction tools from the Turbo Tester [23] suite for handling test patterns for the modules of NoC switches. To note that compaction tools have more degrees of freedom for test set optimizations when they are fed by long test sequences that most likely stimulate the same error multiple times. Thus, the choice of pseudo-random test sequences is an ideal target for this case since these latter are commonly longer than deterministic sequences generated by any ATPGs.
In order to obtain minimal test lengths, the following strategy was selected. **First, long pseudo-random test sequences were generated till the obtained coverage for single stuck-at faults was 99%. Then, an efficient static test set compaction tool (Optimize [4]) was run by requesting it to derive a compressed test set tracking the previously obtained coverage for each tested module. Finally, a veri-**

**fication step of the achieved coverage with the new test set was performed. The compacted vectors are then easily hardwired in hardware TPG.**

A key issue for our testing framework was to properly identify the circuit blocks the methodology should be applied to. The guiding principle in making this choice was to achieve the lowest possible testing latency for the NoC as a whole. This was pursued in two ways:

- Testing of NoC switches was engineered in such a way that all switches can be tested in parallel. The key enabler was to implement a cooperation mechanism between switches for testing their inter-switch links. In practice, each switch sends test patterns across its outgoing links and response analysis will be performed in the neighboring switches. The opposite holds for incoming links, which are analyzed locally. At the same time, switch internal blocks are tested in parallel, thus avoiding to create dependencies between testing phases and enabling the maximum testing parallelism.
- Test set compaction for combinational logic is intuitively simpler and potentially more effective than that for sequential circuits. There are a number of reasons for this. First, the compaction algorithm should deal with simple test vectors rather than with test sequences. Secondly, testing sequential logic of switch FSMs presents different requirements with respect to combinational logic testing. In fact sequential logic has fewer inputs than combinational logic but needs more clock cycles to be stimulated, thus requiring a suitable wrapper properly sequencing the outputs of the test generator. For these reasons, our goal of minimizing test application time motivated the choice for splitting each switch sub-block (essentially arbiters, buffers and crossbar) into their combinational logic and registers for the sake of testing. Then, registers were tested with well-known techniques (fundamentally, comparison of test responses or built-in scan-chains for self-testing), while the test set compaction methodology was applied to combinational logic.

The choice of isolating combinational logic for its efficient testing has a relevant impact on FSMs. In fact, their state registers are tested separately and their current state signals feeding the combinational logic should be made

controllable to the TPG. In turn, the outputs and the next state signals from the combinational logic should be made observable to the response analyzer. For testing the registers, we adopted the most convenient standard approach depending on the register type (state registers, configuration registers and data registers).
The above design decisions paved the way for a test architecture for NoCs aiming at competitive coverage of single stuck-at faults and unprecedented test application times with respect to current literature at the cost of test wrapper complexity. However, the choice of pseudo-random test patterns and of their compaction poses the foundation for low-overhead TPGs and response analyzers, thus partially counterbalancing the footprint of the test wrapper. The achieved trade-off and its comparison with state-of-the-art solutions will be quantified in the experimental results section.

## 5 Baseline Switch Architecture

A 5x5 xpipesLite switch [6] has been used in this chapter as the baseline design point without any testing support. The main blocks of this switch are illustrated in Figure 7.1. For each input port, a 2-slot input buffer and a routing module are instantiated. We use logic-based distributed routing [22], which computes target output ports by means of a simple combinational logic for each packet head. In order to provide implementation support for different routing algorithms, the logic is fed by 26 configuration bits per input port. For each output port, a port arbiter and a 6-slot output buffer are instantiated. Also, a 5x1 Multiplexer is placed in front of each output buffer, globally building up the switch crossbar. The switch implements wormhole switching and the stall/go flow control.

## 6 Testing Architecture

Next we present in detail all the changes applied to each block of the architecture. We will analyze respectively the testing scheme of the LBDR, Arbiter, Output buffer, Input buffer and Crossbar. After analyzing each block standalone, we will perform an optimization by merging some test phases

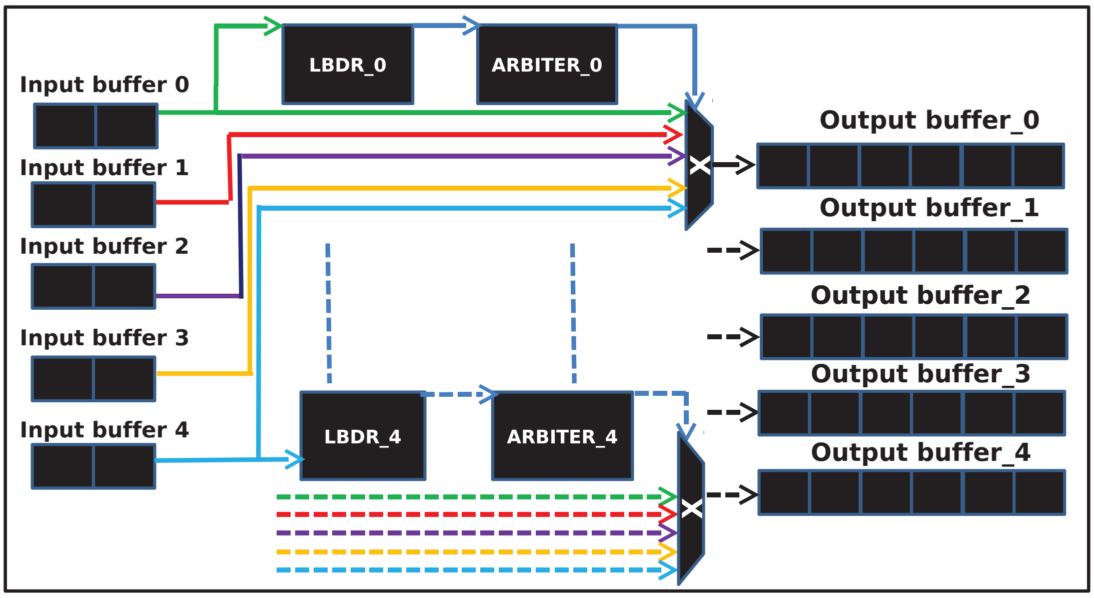


Figure 7.1: Baseline Switch Architecture.

together.

## 6.1 LBDR testing

Each LBDR routing module is split into two main blocks in such a way to be able to apply the key concept of our approach (see Fig. 7.2). The first block is composed of the LBDR configuration registers ($FF\_i$ blocks of fig.7.2), while the second one is the LBDR combinational logic (*Combinational* block of fig.7.2). Combinational logic computes the information contained in the configuration registers. Configuration registers contain information about the routing restrictions of the routing algorithm (Rbits), the connectivity of the switch ports (Cbits), the local switch ID in the topology (Sid) and about deroutes to be taken in some special cases (Dbits) [22]. In test mode, these informations are randomized by the TPG, implementing the compacted vectors. The LBDR logic reads also the destination address (11 bits) from the head flit, which needs again to be randomized in test mode. The compacted vectors codified in the $TPG\_Optimized$ block of fig.7.2 are used to test the combinational block of the LBDR. Before reaching the block under test, the compact test set has to cross first the wrapper placed in front of it. Response analysis is performed by comparators exploiting the output of the 5 switch

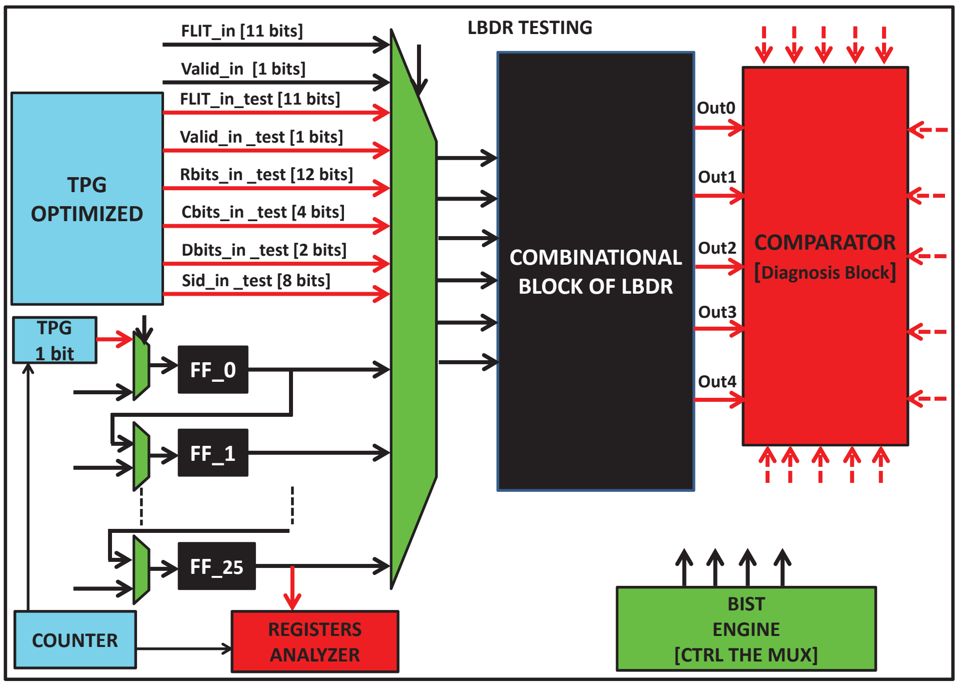


Figure 7.2: Lbdr Testing Architecture.

ports, but nothing prevents from using MISRs. In both cases, the faulty routing module can be easily identified. All switch routing modules share the same TPG.
The scan-chain approach has been adopted for testing the registers of the LBDR. A 1 bit test pattern generator was used to inject a sequence of 0s/1s along the scan chain. A response analyzer, placed at the end of the scan chain, receives and analyzes the bit response after some amount of cycles (depending on the number of registers to cross). The configuration registers of the same LBDR modules are tested in a sequential manner. The testing of the five LBDR configuration registers occurs in parallel; they share the same TPG and counter but have different analyzers.
Finally, a BIST control engine drives the select bits of the test wrapper.

## 6.2 Arbiter testing

We distinguish between combinational logic and state registers also in the arbiter FSM. The testing diagram is depicted in Fig.7.3.
The arbiter state registers are tested in the same way as those of the in-

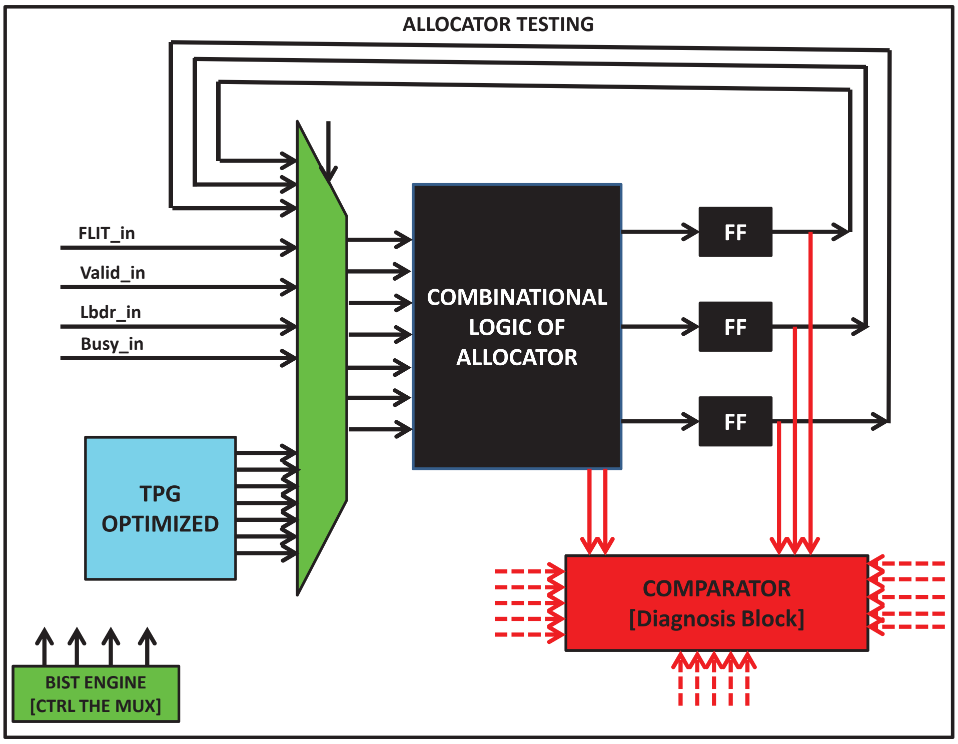

ALLOCATOR TESTING
FLIT_in
Valid_in
Lbdr_in
Busy_in
COMBINATIONAL LOGIC OF ALLOCATOR
FF
FF
FF
TPG OPTIMIZED
COMPARATOR [Diagnosis Block]
BIST ENGINE [CTRL THE MUX]

Figure 7.3: Arbiter Testing Architecture.

put/output buffers (see Section 6.3).

For the combinational block, besides connecting its primary inputs to the optimized TPG, we broke the feedback loop of the FSM in order to increase the controllability of the block. A comparator is again used for response analysis, thus exploiting the multiple instantiation of arbiters in the switch and their concurrent testing.

## 6.3 Output buffer testing

The output buffer belongs to the data path of the switch but it internally consists of an actual data path (data registers with selection input demux and output mux) and of a control FSM driving the read and write pointers of the mux and demux. For the sake of testing, these two internal blocks have been separated. The test architecture is illustrated in Fig.7.4.

The combinational block of the control-path is tested with compacted vectors generated by the Turbo Tester tool. State registers of the FSM could

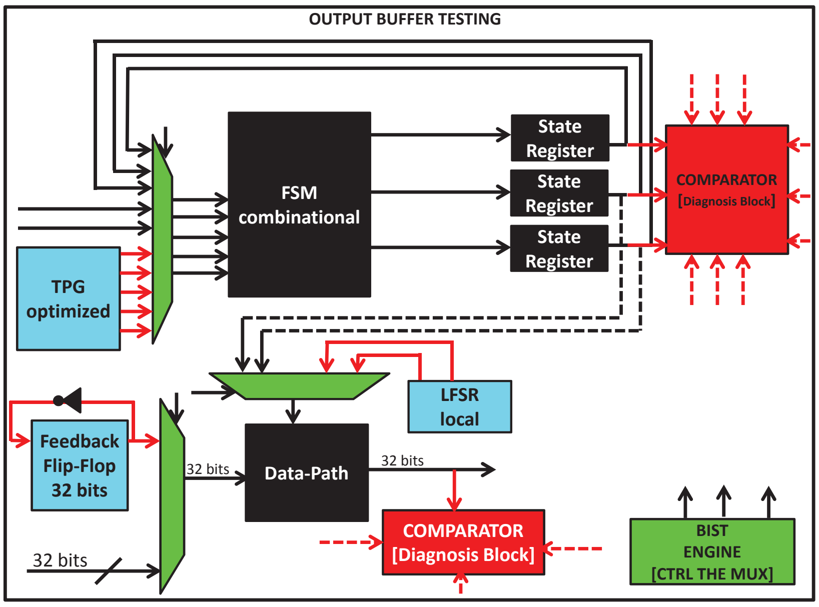


Figure 7.4: Output Buffer Testing Architecture.

be tested as previously illustrated for the LBDR. However, here we present a further opportunity consisting of feeding the combinational logic outputs to the registers and analyzing their outputs by means of cycle-by-cycle comparators with the same outputs from the other switch buffers. Even in this case, full coverage is guaranteed. In order to test the data path registers, we used a 32 bit register where the outputs are inverted and connected back to the inputs, thus generating a sequence of 0s and 1s. Due to the fact that the output signals from the control-path are not random enough, we opted for a small LFSR to drive the read/write pointers of the data path. Both pointers are connected to the same LFSR outputs, so to coherently swap all register banks.

## 6.4 Input buffer testing

The input buffer is tested exactly in the same way as the output buffer. The testing diagram of this element is identical to the testing scheme of output buffer as described in fig.7.4. The only difference is the number of slots (2 for the input buffer and 6 for the output buffer).

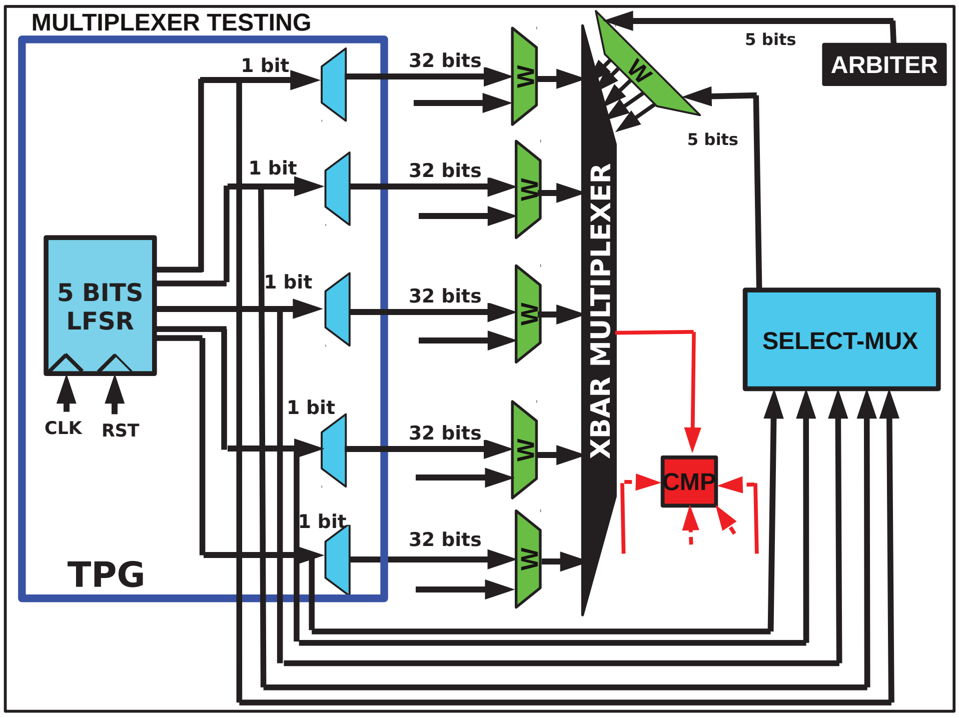


Figure 7.5: Testing Architecture for Crossbar Multiplexers.

## 6.5 Testing Multiplexers of the Crossbar

The testing infrastructure adopted to test the multiplexers of the crossbar is depicted in Fig. 7.5. The TPG is a 5 bits maximal-length LFSR. The 5 bits LFSR generates all the patterns necessary to stimulate all the possible states of the multiplexers in less than 32 cycles. In particular, the relevant patterns for the testing of a 5x1 multiplexer are the following: 10000, 01111, 01000, 10111, 00100, 11011, 00010, 11101, 00001 & 11110. Each pattern generated by the LFSR feeds a block called "select-mux". This latter is able to select the multiplexer input port carrying the logic value that is negated with respect to the other 4 input values. Finally, the diagnosis is performed by means of a comparator exploiting the output of the 5 multiplexers of the crossbar.

## 6.6 Testing Infrastructure Optimization

After testing each block independently, we cascaded the circuits under test exploiting the synergies between them in order to cut down on test wrapper and TPGs complexity.

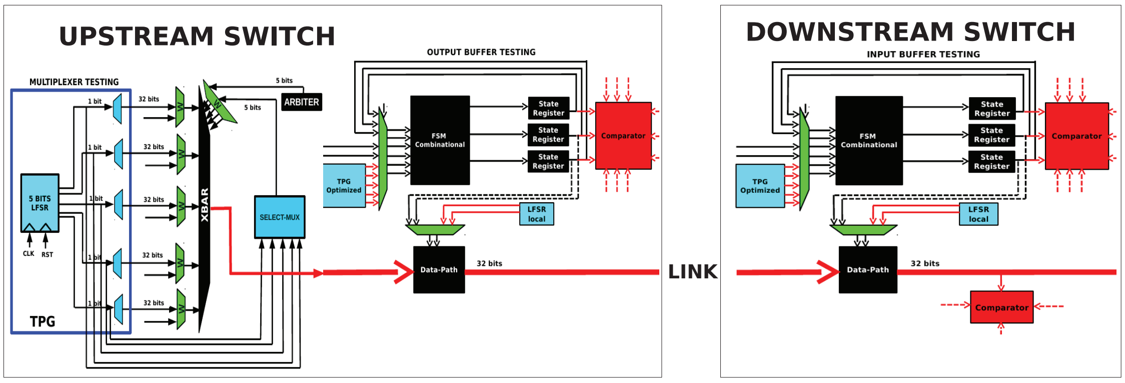


Figure 7.6: Cascaded Testing Architecture.

The cascade is composed by the following modules:

- The Crossbar of the upstream switch
- The Output buffer of the upstream switch
- The Inter-switch link
- The Input buffer of the downstream switch

At the beginning of the cascade, we inserted a 5 bits LFSR (as described in Figure 7.5). Since test responses of one block become test patterns for the next one then the LFSR is responsible for the injection of test vectors for the whole cascade.
Such optimization allowed us to remove the input/output TPGs based on registers previously located in front of the input/output buffer. In the same way, we removed both the comparators located after the multiplexer and the output buffer. Finally, the comparator located after the input buffer in the downstream switch is the only preserved. The cascade testing scheme is presented in Fig.7.6.
To test the communication link, we synchronized the injection rate of some of the test patterns generators. In this specific case, we have three independent TPGs to synchronize:
-The TPG in front of the multiplexer input ports

-The two LFSRs of the read/write pointers of the input/output buffer

The synchronization must be performed in such a way to ensure the maximum coverage of all the blocks that belong to the cascade.
The multiplexer TPG starts to inject the test data as soon as the test mode is selected. The injection length depends on the number of cycles necessary to maximize the coverage of the cascade. Meanwhile, both the LFSRs respectively lying in the output buffer upstream switch and in the input buffer downstream switch are clocked each 31 cycles. Clearly, they must be clocked at least six times (i.e. the number of slots of the output buffer) in order to allow the data tests to cross all the data-path registers.

## 7 Experimental results

This section presents the experimental results for a 5x5 NoC switch synthesized at 600 MHz in a 65nm industrial technology library. For the sake of comparison, two test architecture variants for the same baseline switch are available from previous work and used in this chapter to assess the trade-offs of the new design point proposed by this chapter. Therefore, our approach with optimized pseudo-random patterns is contrasted with the handcrafted deterministic test patterns used in [3] and with the non-compacted pseudo-random patterns used in [50], and above all with their enabling test architectures. In this comparison we do not consider ATPG generated test patterns since it has been demonstrated in [3] that these latter are not competitive with the handcrafted deterministic ones from a coverage viewpoint. This chapter does not consider solutions providing less than 98% coverage on the considered NoC architecture.

### 7.1 Area Overhead

Fig.7.7 illustrates the area overhead of three BIST solutions for the xpipesLite switch (the one of this chapter, the one with handcrafted deterministic patterns [3] and the one with non-compacted pseudo-random patterns [50]) normalized with respect to the deterministic switch. As we can see, around 11%

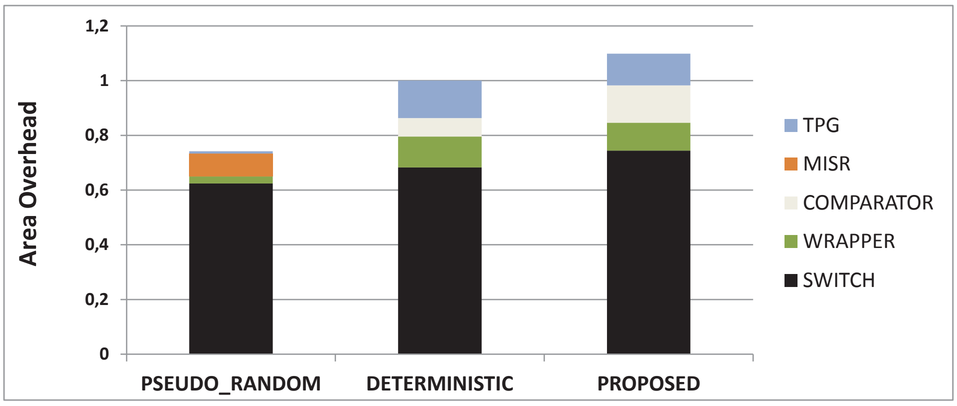


Figure 7.7: Area Overhead: Deterministic [3], Pseudo-Random [50] and Proposed Compacted Pseudo-Random Approach.

of the area overhead of the deterministic approach comes from the wrapper, needed because of the different test phases that this approach requires, in addition to switch sub-block isolation for the sake of testing. Another 7% comes from TPGs, which encode the handcrafted test patterns, and marginally from diagnosis logic and the BIST manager. Finally, the comparator tree used for response analysis takes around 13% of the area.

When the test architecture is reconceived for non-compacted pseudo-random patterns, then the area overhead is reduced by ∼26% in the considered 65nm library. Interestingly, the breakdown is completely different. MISRs are used for response analysis and account for most of the area overhead. LFSRs are extremely compact TPGs while less than 3% of the area is devoted to the test wrapper. In this architecture, block cascading was extensively used (i.e., test responses of some blocks are fed as test patterns to downstream blocks, at least until cascading does not hurt coverage too much) thus cutting down on the test wrapper overhead.

When it comes to the test architecture with test set compaction, the area overhead is 9.8% with respect to the deterministic approach. The proposed solution optimizes pseudo-random patterns thus incurring in a small area overhead, while significantly improving test application time (see section 7.2). Most of the area overhead is due to the multiple instantiation of comparators and to the test wrapper, which needs to provide finer-grain circuit isolation in test mode. The applied optimizations in the test infrastructure proved very effective in reducing the area overhead, to the extent that it closely tracks the overhead of the deterministic test architecture.

## 7.2 Testing time

Table 7.1: Test Application Time Per Block

| **TPGs** | Multiplexer | Output LFSR | Input LFSR |
|---|---|---|---|
| Injection period | 1 cycle | 31 cycles | 31 cycles |
| #Vectors | 239 | 8 (7.7) | 8 (7.7) |
| Total Cycles | 239 | 239 | 239 |

Table 7.1 contains the injection rates of the TPGs used to optimize the cascade of Fig. 7.6. In fact, the TPG located at the beginning of the cascade starts to inject the data test as soon as the test mode signal driven by the BIST-engine is high. As mentioned in Table 7.1, this injection occurs for ∼239 cycles. At the same time, the multiplexer TPG controls the select inputs of the crossbar through the ”select-mux” block. Meanwhile, local LFSRs of the output/input buffers in the upstream/downstream switch control the read/write pointers of data-path registers. These local LFSRs continuously inject patterns after each 31 cycles as mentioned in Table 7.1. Overall, the test application time amounts to 239 cycles.

Table 7.2: Testing Cycles as function of the Testing Approach

| **Testing Technique** | Testing Time (cycles) |
|---|---|
| Compressed Pseudo-Random Testing Approach | 239 |
| Deterministic Testing Approach | 1104 |
| Pseudo-Random Testing Approach | 10000 |

Table 7.2 compares this value with those of the alternative approaches, for the same target coverage of approximately 99%. All approaches implement some form of testing cooperation between neighboring switches and share the same baseline switch architecture. However, test set compaction, proper choice of the granularity of the circuits to test (and consequent compaction efficiency) and merging of test phases make the approach of this chapter the fastest.

Only handcrafted deterministic patterns can somehow approach the testing time of this chapter with around 1104 cycles. Although non-compacted pseudo-random patterns can achieve 96% of coverage in a comparable time with handcrafted deterministic patterns, they take around 10000 cycles to reach around 98% of coverage.

Table 7.3: Test application time and coverage of different testing methods

| | Test Cycle | Coverage |
|---|---|---|
| Our | 239 | 98.3% |
| [3] | 864 - 1104 | 99.3% |
| [2] | 3.88 x $10^2$ - 2.89 x $10^3$ | 97.79% |
| [20] | 4.05 x $10^5$ | 95.20% |
| [13] | 2.74 x $10^3$ | 99.89% |
| [14] | 9.45 x $10^3$ - 3.33 x $10^4$ | 98.93% |
| [21] | 5 x $10^4$ - 1.24 x $10^8$ | N.A. |
| [11] | 320 | 99.33% |
| [12] | 200 x $10^3$ | full (no exact numbers) |

Our test application time compares favorably with previous work, as Table 7.3 shows. Only [2] and [11] are somehow competitive. However, [2] does not test the control path while [11] reports 320 cycles for a 3x3 mesh (made of a simplified switch architecture) which however grow linearly with network size. Also, this latter approach makes additional use of BIST logic for the control path not accounted for in the statistics.

We feel that area overhead is hardly comparable with previous work since whenever numbers are available, features of the testing frameworks are very different (e.g., control path not tested [2], test patterns generated externally [20, 14], diagnosis missing [20, 13, 14, 21], lack of similar test time scalability [8, 11], NoC architecture with overly costly links [13]). Moreover, the impact of synthesis constraints is never discussed.

Table 7.4: Coverage as function of the Testing Approach

| **Testing Technique** | Coverage (%) |
|---|---|
| Compacted Pseudo-Random Testing Approach | 98.3% |
| Deterministic Testing Approach | 99.30% |
| Pseudo-Random Testing Approach | 98.24% |

## 7.3 Coverage

The obtained coverage for single stuck-at faults is illustrated in Table 7.4. The technique proposed in this chapter tracks the coverage of the pseudo-random approach although does far better in terms of latency.

At the same time, our technique removes the burden of deriving handcrafted test patterns and enables the use of test generation and compaction tools, with one order of magnitude lower testing latency.

Table 7.5: Compaction Table

| **Combinational Logic** | Random (#vectors) | Random (coverage) | Compacted (#vectors) | Compacted (coverage) |
|---|---|---|---|---|
| FSM Input | 1000 | 100.00% | 10 | 100.00% |
| FSM Output | 1000 | 100.00% | 17 | 100.00% |
| LBDR | 400000 | 93.04% | 61 | 92.17% |
| ARBITER | 170000 | 99.37% | 42 | 99.37% |

Table 7.5 shows the impact of the compaction tool on the pseudo-random test set generated for each combinational block of the switch control logic. The second and third columns of the table report the number of pseudo-random vectors together with their coverage while the last columns show the number of vectors with the respective coverage once they are compacted by the tool. It is possible to notice that the compaction operation is efficient and the compacted test set tracks the previously obtained coverage for each tested module.

# 8 Conclusions

This chapter presents a testing methodology and architecture support for NoCs that aim at the minimization of test application time, a requirement that well matches the future requirements of lifetime testing frameworks. In fact, while testing latency was not a concern for production testing, it becomes such when the testing procedure is run at system bootstrap and/or at runtime. We demonstrate NoC testing in less than 250 cycles. Above all, we do not achieve this result with handcrafted deterministic test patterns, but rather with an optimization methodology of pseudo-random patterns. The guiding principle is test set compaction, although the low-latency requirement forces a careful selection of the logic to test for best compaction efficiency. The trade-off is therefore between test application time and test wrapper overhead, although the final area footprint tracks that for a test architecture with handcrafted deterministic patterns but with one order of magnitude lower testing latency.

# Chapter 8

# Cost-effective Contention Avoidance in a CMP with Shared Memory Controllers

## 1 Abstract

Efficient CMP utilisation requires virtualisation. This forces multiple applications to contend for the same network resources and memory bandwidth. In this chapter we study the cause and effect of network congestion with respect to traffic local to the applications, and traffic caused by memory access. This reveals that applications close to the memory controller suffer because of congestion caused by memory controller traffic from other applications. We present a simple mechanism to reduce head-of-line blocking in the switches, which efficiently reduces network congestion, increases network performance, and evens out the performance differences between the CMP applications.

## 2 Introduction

The access to the off-chip memory in large chip multiprocessors (CMPs) based on a switched interconnect (NoC) consumes a significant portion of the bandwidth in the on-chip network. Furthermore, this traffic is targeted towards specific areas of the chip where the memory controllers are connected.

Previous studies [57] show that the placement of these memory controller connections have a significant impact on the network load, but the authors do not study how this impacts the performance of the applications themselves and the complex interaction between the local traffic (caused by cache coherency protocols) and the memory controller traffic. The initial intuitive understanding of the effect of memory controller access point placement on application performance is that the applications located closest to the memory controller access points will experience better performance compared to applications allocated further away [70]. However, we show that applications that reside close to the memory controller might be more severely affected by the interplay between local traffic and memory controller traffic.

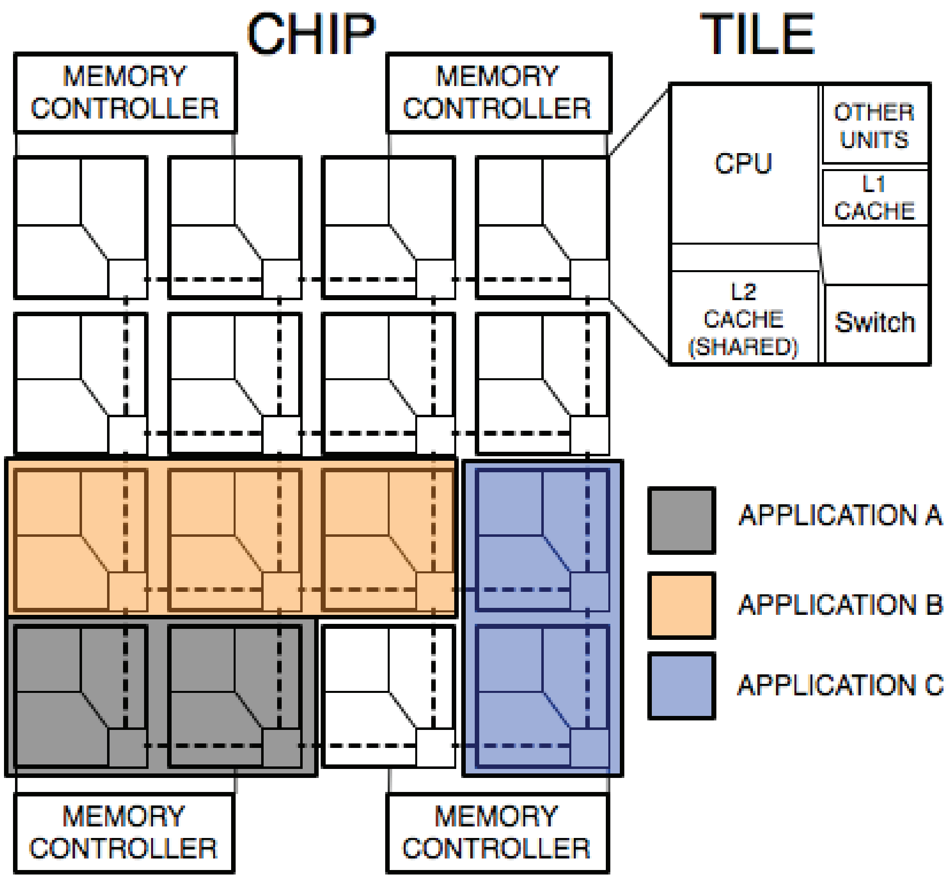


Figure 8.1: CMP tile-based design with dynamic application domains

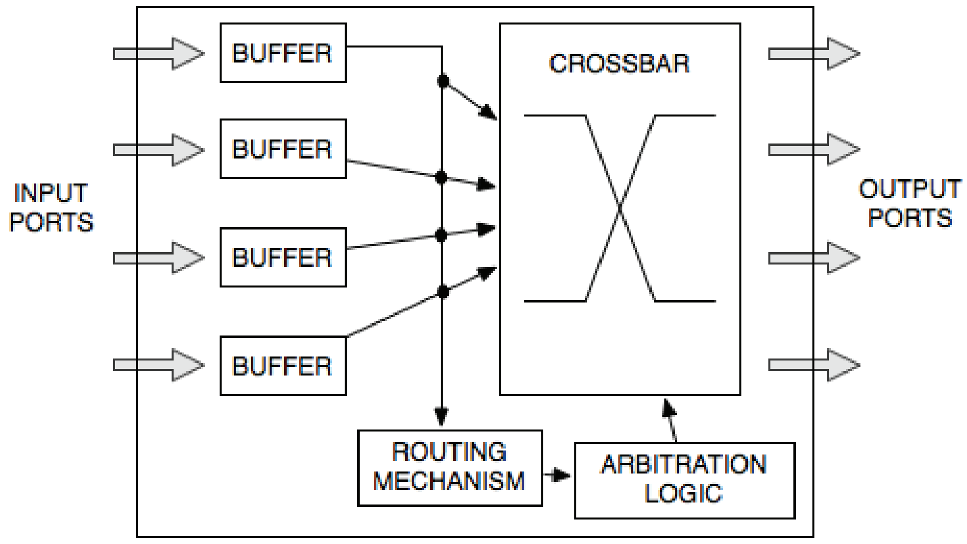


Figure 8.2: Basic switch architecture

To ease development, the tiles in a CMP are usually homogeneous, with a structure as displayed in Figure 8.1. Every tile has a private level I and (usually) a shared level II cache, together with the processing core and a switch to access the on chip network. Off-chip memory (DRAM) is accessed through one or more memory controllers connected to the on-chip network, usually at the edge of the chip, through one or more ports. The network on chip carries cache coherency traffic between the level I and level II caches, and memory access traffic to and from the memory controller. The two traffic types may or may not be divided into two virtual networks (using virtual channels). The interaction between these two traffic types is the core of the issue we study in this chapter. Local traffic from one application (cache coherency traffic) should not interfere with the local traffic from other applications, as application isolation is a core concept of CMP virtualisation [59]. Most applications will, however, be affected by the memory controller traffic from other applications. In this chapter we study how and to what extent the local and memory controller traffic contribute to network congestion and how this affects application performance. Based on this study we present a mechanism to reduce head-of-line blocking, and thus network congestion, both with and without virtual channels. The structure of this chapter is as follows: Section 3 presents the NoC background and the related work. In Section 4 we describe the congestion problem in on-chip networks and the causes behind it, and we present our congestion control solution in Section  5. Next, in Section 6, we detail the evaluation scenario and the results obtained, and finally in Section 7, we present some conclusions and future work.

## 3 NoC Background and Related work

There is significant ongoing research to study how application mapping and basic properties of virtualisation is related to network performance. In [59], the importance of traffic isolation and contiguous application mapping is presented to demonstrate the foundations of virtualisation. Das et al. [70] study application mapping mechanisms, and show that memory intensive applications should be located close to the memory controller, but the authors do not study the impact of application traffic and memory controller traffic on the mapping. To simplify such mapping problems, Abts et al. [57] study alternative memory controller placements by moving the memory controller access points towards the centre of the chip. This breaks the regularity of the chip, both in design and routing, so further study is required before these strategies may realistically be employed. Finally, Sanchez et al. [64] describe how different NoC topologies can impact on application performance, but do not consider the location of the application relative to other applications and the memory controllers in the CMP.
There is a number of solutions in [65, 66, 67] that attempt to reduce the negative effect of shared resources through quality of service (QoS) based on priority schemes. Although all these solutions can alleviate network congestion by prioritising different traffic types, their objective is to differentiate the traffic and they do not focus on the congestion problem itself. As a consequence, there maybe congestion within each traffic class for unpredictable traffic patterns.
A number of solutions for CMP NoCs are presented in [60, 61, 62, 63]. The authors describe mechanisms that collect congestion information from the neighbouring nodes through the routing process and buffer ingress/egress monitoring. The idea is to offer an alternative path to route around a con-

gested area of the chip. However, this assumption will impact negatively creating more congested resources, as it is impossible to avoid the congested region if all the congested traffic has the same target (the memory controller). Van den Brand et al. [2] and Thottethodi et al. [19] rely on a central controller to gather congestion information from the network. Whereas the former uses a guaranteed service traffic class to propagate congestion notifications to the sources, the latter uses a separate control network for this purpose. Neither solution offers great scalability because of the centralisation. There is still an ongoing field of research on many aspects related to congestion management in NoCs for CMPs, but we have found that the basic congestion problem in a virtualised CMP is not well understood. Therefore, our objective with this chapter is to present a study on how congestion problems arise in the event of many concurrent applications with shared resources (memory controllers). This work serves as a motivation and guide in the search for cost-effective resource management solutions, and we present a solution to deal with the congestion problems in these scenarios.

## 4 NoC Congestion

In this section we describe the concept of network on chip contention and how this leads to network congestion. Furthermore, we examine the relationship between the local traffic and memory controller traffic in order to determine how this will affect application performance based on its location on the chip relative to the memory controller access points.

### 4.1 NoC Contention

Whenever a NoC packet enters a switch, it is buffered, and the header information is read to determine the output port for the packet from the switch (see Figure 8.2). The packet must then wait until it reaches the head of the queue and the appropriate output port is available (i.e. not receiving a packet from another port in the switch and not blocked by flow control). NoCs employ *flow control* to ensure that packets are never dropped by guaranteeing that there is buffer space available in the next hop switch before forwarding the packet across the link [68]. Since multiple packets in different input ports in the switch may have the same output port, a given packet might have to wait for several scheduling rounds (output port contention) before it is allowed access to the switch crossbar and can continue. During this time there may be other packets in the same buffer with different output ports that are available. However, these packets cannot proceed because they are blocked by the first packet in the queue. This is known as *head-of-line blocking*.

Together with head-of-line blocking, the flow control mechanism causes congestion trees to build up in the network. Whenever a packet is blocked, it will block packets upstream, gradually expanding the congestion tree branches from the tree root through this back-pressure mechanism. The root is the switch without enough capacity to forward all incoming packets (the place where the packets are first blocked). For the memory controller traffic, congestion tree roots will typically be the cores that are the memory controller access points. Output port contention and head-of-line blocking combined with the back-pressure caused by the flow control mechanism leads to network congestion at high network load, which has a significant impact on the performance of the NoC.

## 4.2 Application Performance Relative to Memory Controller Location

When using virtualisation to support multiple concurrent applications on a CMP, the two traffic types (local traffic and memory controller traffic) may or may not be separated into two different virtual networks using virtual channels. If all the traffic runs on the same virtual channel (i.e. one virtual channel) it is obvious that applications that suffer congested transit memory controller traffic will experience congestion in the local traffic as well. However, using two virtual networks allows a separation of the traffic which reduces the interaction between the two traffic types.

With two virtual channels, each channel is typically guaranteed 50% of the physical channel bandwidth. Consequently, as long as neither of the two traffic types have a demand greater than 50% of the channel bandwidth, there is no significant interaction between the traffic. However, most applications have a larger amount of local traffic than memory controller traffic. Thus, the applications that are located far away from the memory controller and have little transit memory controller traffic, the application is free to use more than 50% of the bandwidth for local traffic. For the applications located closer to the memory controller the amount of transit memory controller traffic increases drastically, which reduces the effective local traffic down to max 50% and may introduce congestion problems for the local application traffic. Consequently, applications located closer to the memory controller will exhibit worse performance. This contradicts previous studies which concluded that applications close to the memory controller had better performance [69, 70], and we clearly see this effect in the evaluation section.

This discussion has shown that even though a large degree of traffic isolation can be achieved using virtual channels, there is still interaction which can adversely affect application performance as we will see in the evaluation section (Section 6). We will also see that not separating the traffic has even more adverse effects on application performance. Efficient resource management in terms of congestion control is therefore required, both to increase overall efficiency of the chip and fairness between the running applications.

## 5 NoC Congestion Control

For congestion management, we propose HACS (Head-of-line Avoidance Congestion Skip-ahead), a head-of-line blocking observation mechanism that allows buffered packets to bypass the packet that is at the head of the queue. The core mechanism is presented in Figure 8.3. Note that this mechanism is supported under virtual cut-through packet switching. Whenever a packet is stalled for a given time period at the head of a buffer, HACS will search further back in the queue for the first packet that is routed to a free output port, because of a different destination, and let this skip to the head of the queue. This effectively reduces head-of-line blocking with the result of reduced congestion.

We now discuss the implementation of HACS in xpipesLite [68]. All packets are assumed to be 4 flits long by padding shorter ones and by splitting longer messages into multiple packets. The network guarantees in-order delivery

of packets headed to the same destination. An arbiter is instantiated for each output port to perform round robin arbitration among all inputs with valid asserted and presenting a head flit. The switch implements the LBDR mechanism [71].
Assume two packets ”A” and ”B” are stored in an input buffer (see Figure 8.3), and let the arbiter of the output port requested by ”A” be stalled (blocked), thus preventing packet forwarding. In the HACS switch, a timer is activated upon snooping such a stall condition. If the stall signal changes during the count-down, the timer will be reset till the generation of the next stall. If the stall is still high at the time-out, the control logic shifts the read pointer to the head flit of the second packet. Before computing the destination of ”B”, the LBDR routing logic saves the destination of ”A” in backup registers (Lbdr-out-A in the figure) for further comparison with the target output of packet ”B”. A XOR comparator compares the two target ports required by ”A” and ”B”. If they are different and the port requested by ”B” is available, the stall goes down and ”B” is forwarded. If not, the read pointer shifts once again to packet ”A” till the stall is deasserted. If the stall signal is removed while performing the target port comparison, the valid signal is driven low to avoid sampling ”B” before ”A”, thus preserving the order on each output port. In this unfortunate and very unlikely case, the switch experiments one (1) cycle overhead before being able to forward ”A”. HACS can also be applied to each virtual layer of a network with virtual channel support. As previously illustrated, 2 VCs may be considered for memory controller traffic separation. In practical terms, we followed the strategy proposed in [72]. Essentially, the HACS switch is replicated twice, while placing a demux in front of each input port and a mux with associated arbiter after each output port. The link is enhanced with a virtual channel identifier and with a flow control signal for each virtual channel. This is done to exploit logic synthesis optimisations for the sake of area efficiency.

## 6 Evaluation

In this section we first describe the simulation environment we used for the evaluations, followed by a discussion of the results obtained, including the

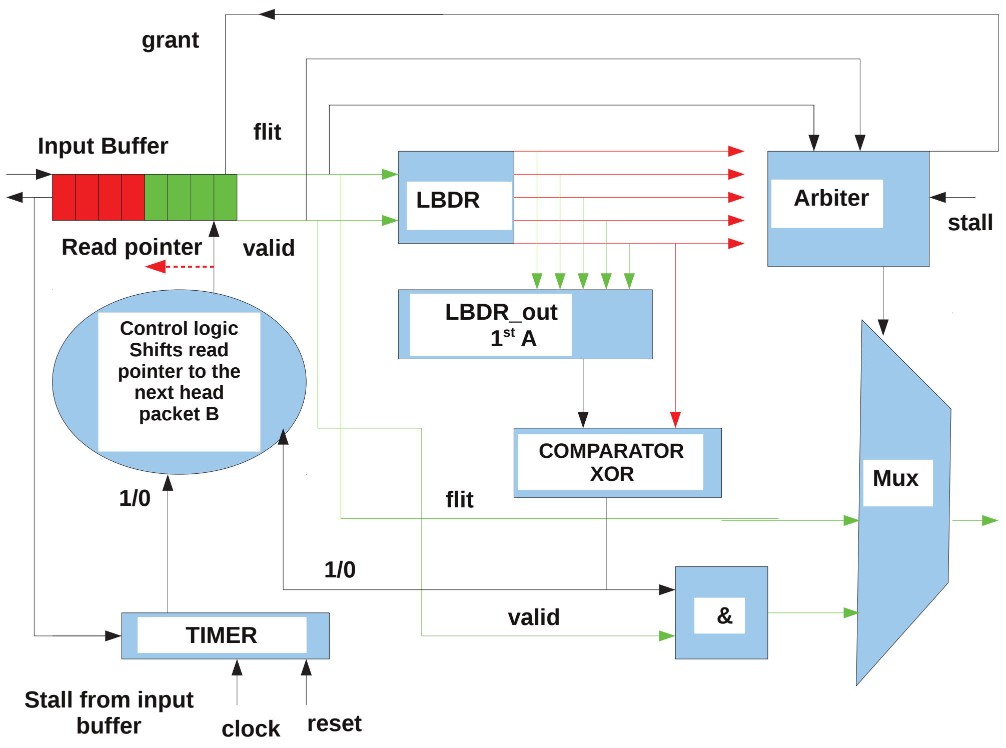


Figure 8.3: Switch architecture

results obtained with HACS and its implementation costs after synthesis.

## 6.1 System configuration

Our simulation framework is a combination of tools chosen to simulate a CMP system as closely as possible. Multi2sim [73] is a simulation framework for heterogeneous computing that allows one or more applications to run on top of it in CMP-like scenarios. It is able to model a complete memory hierarchy system integrated into the CMP and its connection to the respective processor cores. We combined Multi2sim with a cycle-accurate flit-level network-on-chip simulator called gNoCsim (developed by Universidad Politecnica de Valencia, and being used in the NaNoC project [74] by different partners). GNoCsim is able to simulate the network between all the resources in the chip; caches, memory controllers, and processor cores. For the evaluation process, we modelled a CMP that resembles current chip configurations like in Figure 8.1. This configuration implements a tile-based system, and each tile is composed of a processor core, a private L1 cache, a bank of a L2 shared cache, a memory directory bank to be used with the directory-based MOESI cache coherency protocol, and different configurations of memory controllers. Each memory controller is connected to the main memory with 2 channels (each memory controller has two access points). A detailed overview of the chip configuration is shown in Table 8.1.

| Parameter | Configuration | Parameter | Configuration |
|---|---|---|---|
| Core | x86 | Topology | $10 \times 10$ 2-D mesh |
| L1 cache | 16 KBytes Instructions<br>16 KBytes Data<br>Total 32 KBytes per core<br>2 cycles latency<br>2-way associativity<br>64 bytes block size | Routing mechanism | LBDR + SR |
| L2 cache | 256 Kbytes per core<br>20 cycles latency<br>4-way associativity<br>64 bytes block size | Packet switching | Virtual cut-through (VCTlite) [75] |
| Main memory | 1 Gbyte total<br>200 cycles latency | Buffer queue size | 12 flits |
| Coherence protocol | MOESI CMP, directory-based | Flit-size | 8 bytes |

Table 8.1: CMP configuration.

A $10 \times 10$ 2-D regular mesh topology was used for the CMP system. The LBDR [71] mechanism was used for the routing purposes allowing for routing-contained application domains in combination with the Segment-Based Routing algorithm (SR) [76]. Virtual networks are used for different levels of traffic of the memory hierarchy system, implemented as multiple virtual channels (a total of two virtual channels are used) except for Figure 8.8 were no virtual channels are used.

For the evaluations we used a collection of applications from the SPLASH-2 benchmark with the default parameters defined in [77]. The applications are statically mapped to the chip when the experiment is set up. Applications are mapped to completely fill the chip, giving a fair share of cores to each application. Every batch consists of a single application type from the benchmark suite rather than being composed of a collection of mixed applications. This regularity makes it significantly easier to generate relevant statistics and spot trends in the results, such as to get averaged results for the execution time comparison. Running a mix of applications will introduce spikes in the communication, but this will be evened out by the number of applications over time, so the conclusions will still be the same. See Figure 8.4 for an example of the mapping of 32 concurrent applications with 4 memory controllers.

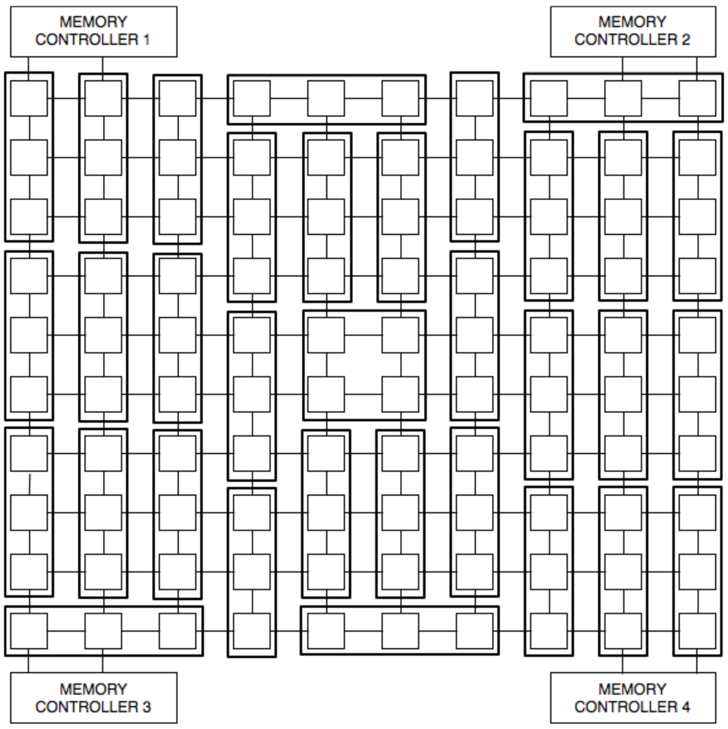


Figure 8.4: 32 concurrent applications mapped on the system

## 6.2 Results

We have evaluated several combinations of number of concurrent applications and memory controllers for a 10 × 10 mesh. Specifically, we have evaluated 6, 8, and 12 applications with one memory controller, 12 and 16 applications with two memory controllers, and 32 applications with four memory controllers. Due to space constraints we report the results for 12 applications with one memory controller and 32 applications with four memory controllers. The general trend from the results is that network performance decreases and unfairness (the difference in runtime based on application location, with two virtual channels) increases as the number of concurrent applications increases for a given number of memory controllers.

We have plotted the execution time distribution for 12 applications (ocean workload) with a single memory controller both with (Figure 8.7) and without (Figure 8.8) virtual channels. The memory controller is located in the uppermost corner. The figure with virtual channels clearly shows how the threads that are located closer to the memory controller have a longer execution time (as much as 7.5% longer than when running alone) than the thread located farther away. The picture is more chaotic without the use of virtual channels. There is no clear unfairness, however, the overall increase in execution time (8.1%) is larger than with virtual channels.

In Figure 8.5 we display the mean squared error between injected and accepted traffic for different applications in the scenario with 32 concurrent applications with 4 memory controllers, with and without HACS implemented. In the figure, HACS2 is allowed to skip ahead the second packet, while HACS3 may skip ahead the second or third packet. The figure shows how the impact of performance degradation due to congestion for the different applications. HACS2 and HACS3 are able to reduce the penalty to only 2.5% in average. Note that there is negligible difference between HACS2 and HACS3. The performance degradation is significantly worse with fewer memory controllers. Figure 8.6 shows network throughput as a function of time for the ocean workload. The uppermost plot is the injected traffic, and the bottom plot is the accepted traffic without congestion control, a clear case of a congested network. HACS2 and HACS3 solutions almost remove all the congestion, handling almost all the injected traffic. The second to bottom line is HACS without virtual channels. This still increases averaged network throughput by around 40%, although the result is poorer than with virtual channels. The designer has to assess the trade-off between performance and implementation costs.

Summarising, the objective of these evaluation cases was to reproduce scenarios that try to reflect current chip configurations, and realistically illustrate the effect of multiple simultaneous applications. The cost/performance tradeoff depends on how much resources are available (in our case, the amount of

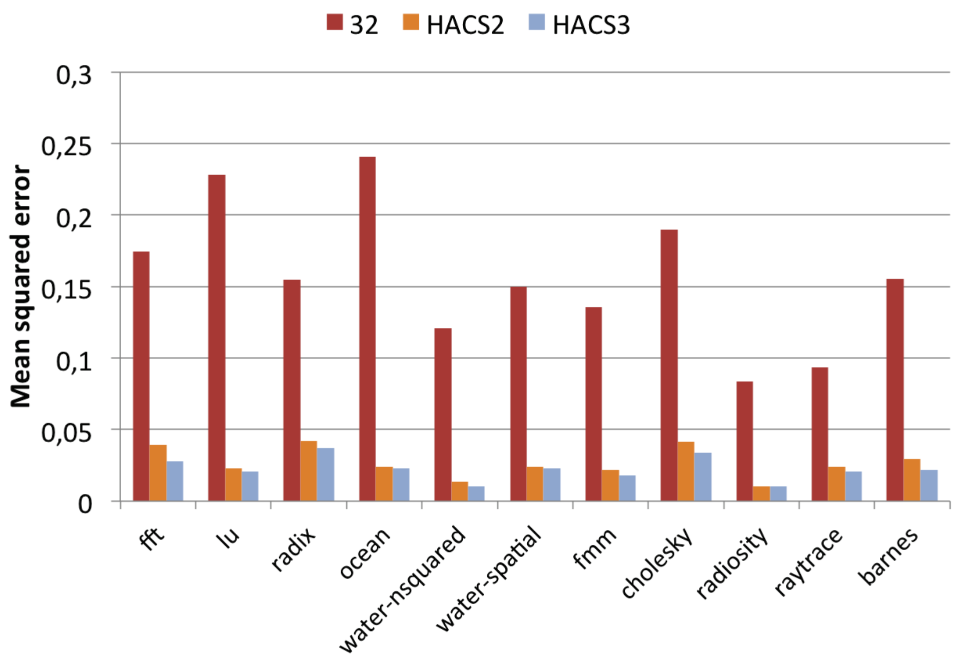


Figure 8.5: 32 concurrent applications mapped on the system (MSE between injected and accepted traffic)

memory controllers) and there is a need for congestion management strategies that can alleviate the problem with minimal impact on the design of the chip. In the next section we evaluate the cost of the congestion control mechanism we have developed.

## 6.3 Hardware breakdown

This subsection characterises area and critical path delay overhead of the HACS switch compared to a baseline one taken from the xpipesLite NoC library [68]. The reference switch implements input buffering, stall/go flow control and virtual cut-through switching. Baseline and HACS switches were synthesised for a target speed of 500 MHz in a 65nm industrial technology library. Normalised post-synthesis area results are illustrated in Figure 8.9. The area of the HACS switch without virtual channels is about 5.34% larger compared to the baseline switch. The implementation with virtual channels results in 2.09x the area of the virtual channel-less switch. Observe in the

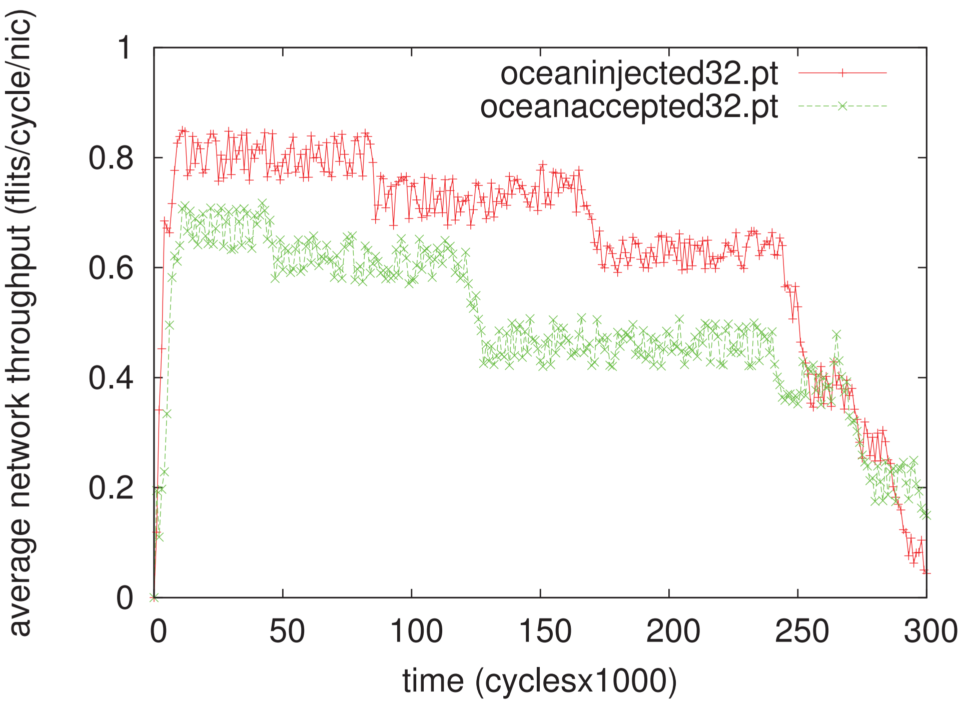


Figure 8.6: 32 concurrent applications mapped on the system (Averaged network throughput, ocean workload)

figure that the baseline input buffer features approximately the same area of the input buffer with the new logic to shift the read and write pointers, thus denoting the marginal impact on control logic. On the other hand, most of the area overhead is due to the timer inserted in the switch and is about 4.09%. This could be improved in future solutions by using buffer thresholds instead of a timer. After synthesis, the critical path of the new switch without VCs was proved to be degraded by less than 1% with respect to the baseline one. The virtual channel implementation contributes an additional 3% of critical path degradation, associated with the arbiters in the switch output ports selecting which virtual channel to move forward.

## 7 Conclusions

We have studied the effects of shared memory access in a CMP with multiple concurrent applications. It is often assumed that network congestion is

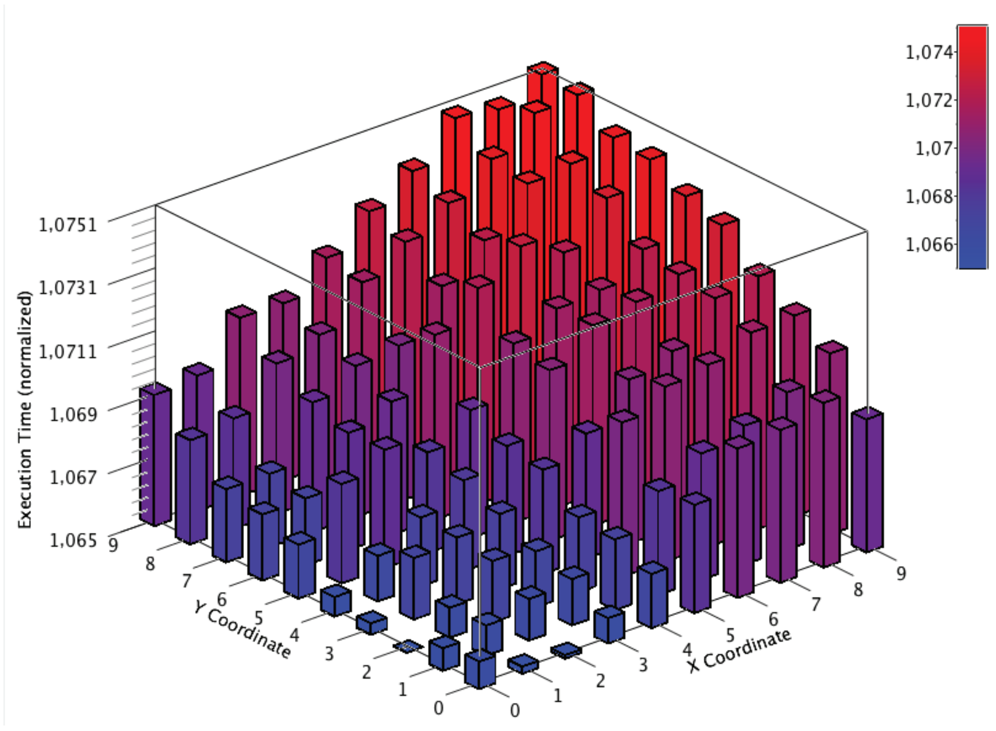


Figure 8.7: Execution time distribution, 1 memory controller, ocean workload (With virtual channels)

not an issue for CMP systems because of the abundant bandwidth in the network on chip. Our evaluations show that network congestion may indeed be a problem when multiple applications access shared memory through the memory controllers available on a typical CMP, and we developed a simple solution, HACS, to remove this congestion. We have observed some effects from our experimental results. First, the hotspots formed by the memory controller traffic lead to network congestion, which can degrade the performance of applications by 15% in average in our scenarios. Second, if there is a high degree of local traffic (which is often the case), the applications allocated close to the memory controller will have less bandwidth for local traffic than applications located further away. The applications closest to the memory controllers are therefore penalised and have longer execution times compared to the others. Further work includes evaluating a wide variety of network controller configurations and workloads.

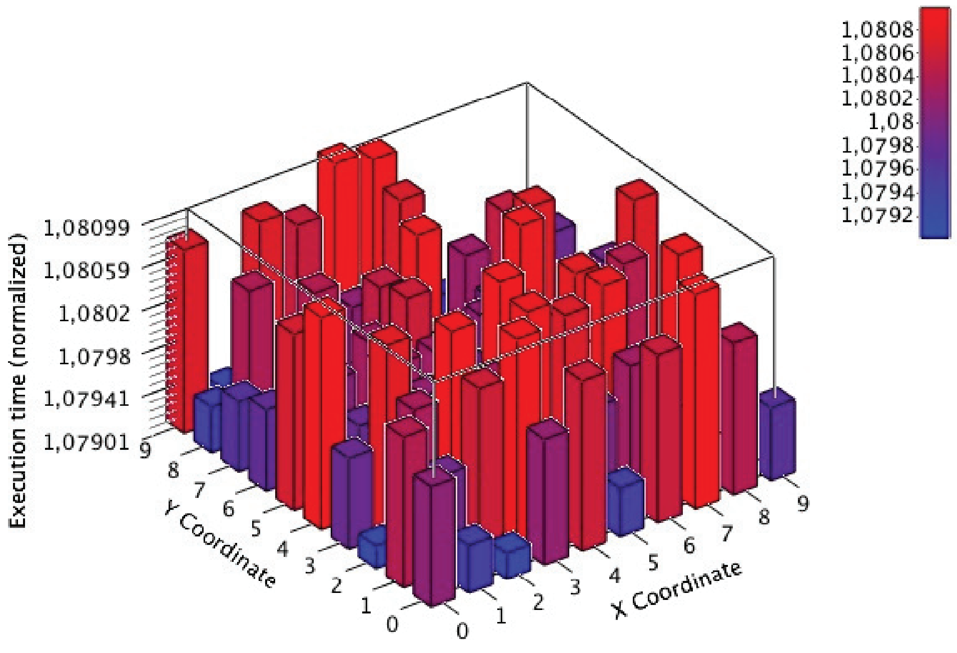


Figure 8.8: Execution time distribution, 1 memory controller, ocean workload (No virtual channels)

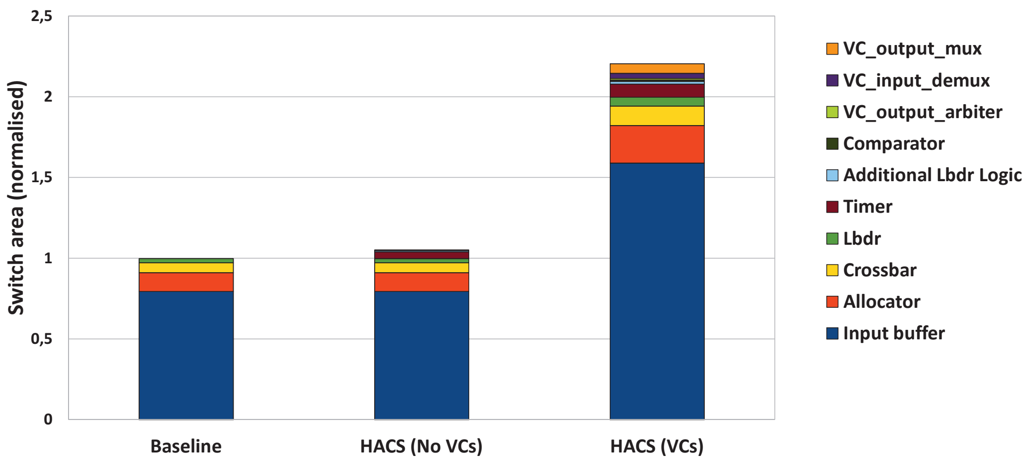


Figure 8.9: Switch area at 500 MHz

# Chapter 9

# Final FPGA Prototyping of Homogeneous Multicores.

## 1 Abstract

This chapter reports about the prototyping of design methods mentioned in the previous chapters on a Xilinx Virtex-7 FPGA. Boot-time testing and configuration, runtime detection of faults, runtime reconfiguration of the routing function, dynamic virtualization of the interconnect fabric are especially validated on the FPGA prototype, where a 4x4 multi-core system has been implemented and managed. The advanced form of platform control is achieved via hardware/software co-design and co-optimization.

## 2 Introduction

NoC design principles have recently reached a stage where they start to stabilize, in correspondence to their industrial uptake. A key property that novel NoCs cannot miss is to guarantee a potentially fast path to industry, since NoC deployment is today a reality. An important requirement for this purpose is the efficient testability of candidate NoC architectures. This property is very challenging due to the distributed nature of NoCs and to the difficult controllability and observability of its internal components. When we also consider the pin count limitations of current chips, we derive that NoCs will be most probably tested in the future via builtin self-testing (BIST) strategies.

Finally, there is an increasing need in embedded systems for implementing multiple functionalities upon a single shared computing platform. The main motivation for this are the constraints set for systems size, power consumption and/or weight. This chapter reports on the first-time prototyping of a Network-on-Chip capable of supporting all of the advanced features described above, and represents a prove of the validation of the industry-ready NoC. The presented prototype builds on the first switch variant mentioned in chapter 1. Then, it validates the (re-) configuration capabilities that preserve safe network operation in the presence of wanted (e.g., virtualization) and unwanted (e.g., manufacturing defects, intermittent faults) effects. The prototyping platform is represented by the Xilinx Virtex-7 evaluation board named VC707, described in section3. The prototyped system implemented inside the FPGA is a homogeneous multicore processor, which resembles programmable hardware accelerators of hierarchical, high-end embedded systems, or basic computation clusters of many-core processors. The validated design methods include:

1. boot-time testing and diagnosis of the 4x4 2D mesh NoC, targeting permanent faults;
2. switch-level and network-level fault-tolerance, targeting transient faults and intermittent faults (i.e., those faults that rapidly anticipate the breakdown of links or switch components;
3. runtime reconfiguration of the network routing function, with logic-based distributed routing as the underlying routing mechanism. The validated reconfiguration procedures are twofold: at boot-time, without background traffic, and at runtime, with background traffic.
4. Dynamic virtualization, i.e., partitioning of the whole NoC into isolated partitions running different applications. As such, this chapter validates:
5. the design methods for supporting NoC static irregularities (routing methods, testing and diagnosis methods);

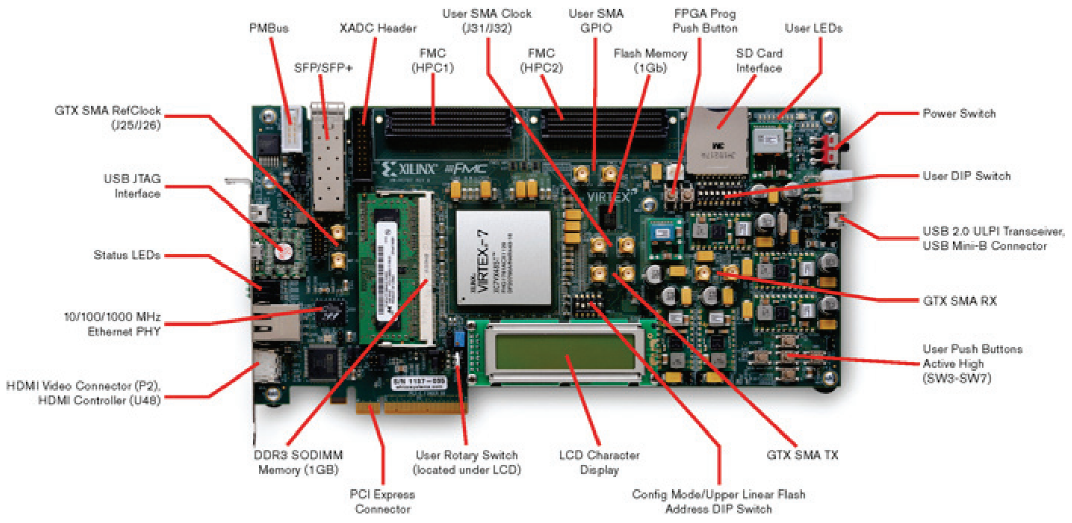


Figure 9.1: VC707 baseline prototyping board.

6. the design methods for supporting dynamically virtualized NoCs (error detection and signaling mechanisms, runtime reconfiguration methods, virtualization methodology);
7. methodologies for seamless integration of NoC topologies within IP cores.

## 3 FPGA Platform

The target system to prototype is overly complex, hence calling for high-end FPGAs and development boards, not to incur integration capacity limits. The Virtex-7 FPGA VC707 Evaluation Kit was selected for that. It is a full-featured, highly-flexible, high-speed serial base platform using the Virtex-7 XC7VX485T-2FFG1761C and includes basic components of hardware, design tools, IP, and pre-verified reference designs for system designs that demand high-performance, serial connectivity and advanced memory interfacing. The included pre-verified reference designs and industry-standard FPGA Mezzanine Connectors (FMC) allow scaling and customization with daughter cards. The XC7VX485T FPGA features 485760 logic cells, 75900 CLB slices, 2800 DSP slices, 37080 kb of block RAM, 14 total I/O banks and 700 max. user I/O. The key features of the evaluation board (see Figure9.1) are as follows:

. GA VC707 Evaluation Kit: ROHS compliant VC707 kit including the XC7VX485T-2FFG1761 FPGA

. Configuration: Onboard JTAG configuration circuitry to enable configuration over USB, JTAG header provided for use with Xilinx download cables such as the Platform Cable USB II, 128MB (1024Mb) Linear BPI Flash for PCIe Configuration, 16MB (128Mb) Quad SPI Flash.

. Memory: 1GB DDR3 SODIMM 800MHz / 1600Mbps, 128MB (1024Mb) Linear BPI Flash for PCIe Configuration, SD Card Slot, 8Kb IIC EEPROM.

. Communication and Networking: GigE Ethernet RGMII/GMII,SGMII, SFP+ transceiver connector, GTX port (TX,RX) with four SMA connectors, UART To USB Bridge, PCI Express x8gen2 Edge Connector (lay out for Gen3).

. Display: HDMI Video OUT, 2 x16 LCD display, 8X LEDs.

. Expansion Connectors: FMC1 - HPC (8 XCVR, 160 single ended or 80 differential, user-defined pins), FMC2 - HPC (8 XCVR, 116 single ended or 58 differential user-defined pins), Vadj supports 1.8V, IIC.

.Clocking: Fixed Oscillator with differential 200MHz output used as the system clock for the FPGA, programmable oscillator with 156.250 MHz as the default output, default frequency targeted for Ethernet applications but oscillator is programmable for many end uses, differential SMA clock input, differential SMA GTX reference clock input, Jitter attenuated clock used to support CPRI/OBSAI applications that perform clock recovery from a user-supplied SFP/SFP+ module.

.Control and I/O: 5X Push Buttons, 8X DIP Switches, Rotary Encoder Switch (3 I/O), AMS FAN Header (2 I/O).

.Power: 12V wall adapter or ATX, Voltage and Current measurement capability.

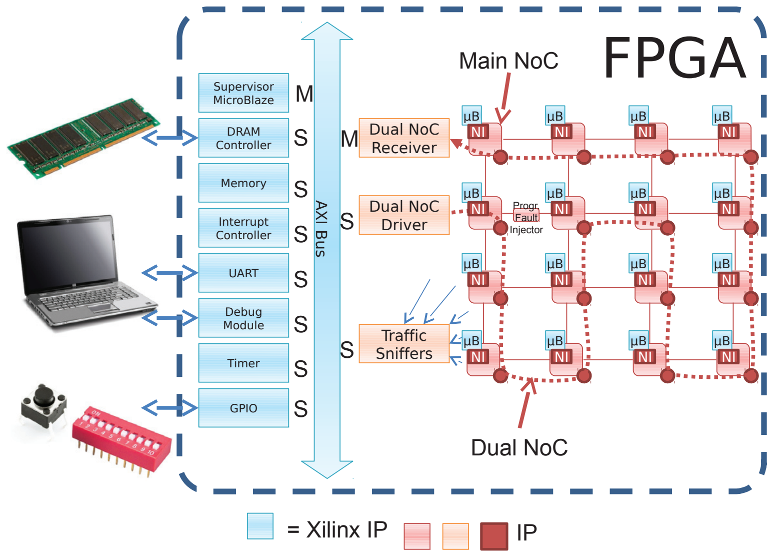


Figure 9.2: FPGA platform overview

. Debug and Analog Input: 8 GPIO Header, 9 pin removable LCD, Analog Mixed Signal (AMS) Port.

## 4 The System Under Test

The high-level view of the design can be found in Figure 9.2. The system comprises a large number of components within the FPGA. As can be seen on the left side of the diagram, a relatively standard Xilinx subsystem is instantiated first; this comprises an AXI interconnect linking together a MicroBlaze (to run the supervision software), a small memory and an external DRAM controller, and several peripheral controllers required to run software on the MicroBlaze and to communicate with a laptop. The right side of the diagram depicts the designed components. This part of the system is the ”Device Under Test” (DUT) of the platform, whose functionality is to be verified. It comprises mainly:

. The main NoC, built as a 4x4 mesh of the first variant switches.

. The dual NoC, built as a chain that follows the topology of the main NoC. The dual NoC is in charge of configuring the main NoC and of collecting status information (e.g. fault detections) from the main NoC.

. At each node of the main NoC (see also Figure 9.3), a MicroBlaze and a memory (by Xilinx) are connected to the switch by means of Network Interfaces.

. Two special blocks, based on AXI NIs, have been designed to connect the dual NoC to the supervision subsystem. These blocks allow the supervision MicroBlaze to receive notifications by the dual NoC, and to reprogram it.

. A sniffer module, monitors traffic along all links of the main NoC mesh, computing link utilization. It is designed so that the supervision subsystem can probe it at regular intervals and transfer its contents towards a userâs laptop.

. A fault injection module has been instantiated along a mesh link. This simple module, connected to a physical button on the FPGA board, provides a method to inject faults on that link to test the platformâs fault tolerance and the NoC reconfiguration capability. To build this platform, we proceed in steps (Figure 9.4). First, we instantiate within Xilinx Platform Studio (XPS) a complete design comprising all the supervision subsystem, the 16 additional MicroBlazes, and the corresponding 16 memories. At this stage, no NoC is instantiated yet. Using XPS for this task allows us to efficiently connect and configure all the Xilinx blocks, and facilitates the instantiation of the toplevel HDL files. Additionally, this makes it possible to subsequently load the applications into all 17 MicroBlazes' memories, and to debug those processor step-by-step, directly through the Xilinx toolchain, which is Eclipse-based. After the first pass of synthesis, however, we remove from the design the Xilinx AXI subsystem which is connecting the 16 additional MicroBlazes and memories, and swap in the NoC (main and dual) in its place. We then proceed to finish the implementation flow within Xilinx ISE by performing mapping, placement and routing, and generating the final bitstream. We leverage some key features of the Virtex 7 board, apart from the FPGA chip. The on-board DRAM is used to provide sufficient space for the software running on the supervision MicroBlaze to work. Physical buttons and switches of the board are connected to an on-chip GPIO controller to allow the user to interact with the platform. Finally, a laptop can be connected to the board by means of two cables to monitor the platform's operation; one cable carries serial port signals (piggybacked onto a USB port) and the other carries JTAG signals (also piggybacked onto a USB port). The former is used to read the board's outputs, while the latter allows for programming the board and interactively debugging the on-FPGA MicroBlazes. Custom-written software runs in three locations of the system: on the supervision MicroBlaze, on the 16 MicroBlazes connected to the main NoC, and on the external laptop.

**.** The software on the supervision MicroBlaze is tasked with oversight of the main NoC and data NoC, with regular polling of the Traffic Sniffers, and with interfacing with the external world through the serial interface.

**.** The 16 MicroBlazes connected to the mesh run micro-benchmarks. These micro-benchmarks have the main role of generating traffic on the mesh, so that the various platform features can be tested. Real functional behaviour was implemented: the nodes perform pipelined matrix multiplications, exchanging data in producer-consumer fashion. More advanced applications could not be implemented due to the lack of I/O interfaces on these nodes and due to lack of memory to instantiate a full C library.

**.** The user's laptop is connected to the board through a JTAG-over-USB cable and a serial-over-USB cable. The former can be leveraged mainly by the Xilinx toolchain, allowing for board programming and debugging. The latter is monitored by a GUI that displays in real-time the platform status and link utilization. This GUI allows the user to analyze the impact of running different software on the system and the behaviour upon fault injection or virtualization implementation.

## 5 Basic components: the on-chip network

A 4x4 mesh with one core and one memory per switch has been chosen as target on-chip network of the FPGA platform. In particular, Figure9.3 represents the basic components instantiated to realize the 4x4 mesh. A MicroBlaze and a memory are connected to each switch through two Network Interfaces. Finally, a sniffer is placed on each bidirectional network link to monitor the network traffic. The sniffers collect information about the traffic crossing the switch-to-switch and NI-to-switch links and deliver such information to the global manager (i.e., the supervision MicroBlaze). Both the NIs and the switches have been designed ad-hoc to support the target on-chip network where fault-tolerance, testing capability and reconfigurability features are guaranteed. Note that the MicroBlaze also includes a directly-connected BRAM of 128kB (not shown in the figure) to store its application software.

### 5.1 The Network Interfaces

We instantiate two types of Network Interfaces NIs: an AXI initiator NI to interface with the MicroBlaze, and an AHB target NI to interface with the memory. This choice was deliberate (e.g., both could have been AXI) to demonstrate interoperability among the two. Due to the relatively simple needs of the MicroBlaze core, which does not support multiple transaction IDs, we save area by instantiating a small AXI initiator NI with support for only one such ID. However, the NI is still supporting all AXI features. Both AXI and AHB NIs, and their interoperability, were extensively tested in RTL and on the FPGA. For integration into the platform, a few tweaks to the NI were needed:

. NIs embed routing tables to statically perform source routing. In this platform, routing is distributed and reconfigurable to work around faults or to enforce virtualization. Therefore, the routing tables are modified to instead encode the XY coordinates of the destination core; these will be processed at the switches. The coordinates are expressed as strings of 9 bits: 4 bits for each coordinate (slightly overprovisioned for a 4x4 mesh) plus one bit to differentiate among the two local cores at each node, i.e. MicroBlaze and memory.

. The input and output buffers of the NIs are extended to support the NACK-GO flow control protocol used by their switches, instead of STALL-GO.

. The AXI initiator NIs are extended with two extra pins, directly connected to FPGA pads, in turn connected to physical switches of the FPGA board. This means that the user, manually flipping those switches, can change the value of two bits inside each NI. The NI in turn exposes these two bits to the MicroBlaze at the reserved address 0x11000000. The MicroBlaze can poll this location to change among operating modes, e.g. staying idle, or executing one of multiple pre-programmed applications. As can be inferred from Figure3, note that in the platform, the 16 MicroBlazes attached to the mesh have no way to communicate with the external world except for this facility.

## 6 Basic components: the supervision subsystem

In order to demonstrate the NoC functionality, a supervision subsystem is required. We choose to instantiate it within Xilinx Platform Studio, and using as many Xilinx IP cores as possible, for convenience; we integrate it with custom designed IP when suitable. The subsystem (see Figure3) includes a Xilinx AXI interconnect, with two masters (a Microblaze and the Dual NoC Receiver) and numerous AXI, AXI Lite and AHB slaves. At the heart of this subsystem is a Microblaze running software. This software is tasked with:

. Probing the status of the NoC, e.g. after BIST and upon fault occurrences.

. In response to the above, configuring or reconfiguring the NoC.

. Awaiting for possible user requests to reconfigure the NoC in a virtualized manner.

. In response to the above, reconfiguring the NoC.

. Polling the link sniffers periodically to monitor activity on the NoC links.

. Transferring key information about the platform's functioning outside the FPGA through the serial port (or, potentially, an Ethernet port).

To perform these actions, multiple support controllers and devices are needed. First of all, since the supervision software and the required underlying C library have a non-negligible footprint, incompatible with on-chip resources, a DRAM controller is advisable to be able to store the software. To support the basic functionality of the Xilinx C library, a timer and an interrupt controller must also be present. (Note that, in contrast, the 16 Microblazes connected to the NoC mesh do not have access to external memory, timer or interrupt controller; this limits the capabilities of the software that can be run on those).
In order to monitor the NoC, it is necessary for the Microblaze to be able to access the dual NoC. This is done via three components plugged to the AXI bus: a Dual NoC Driver, a Dual NoC Receiver, and a memory. The Microblaze can directly write to the Dual NoC Driver, which is a slave on the AXI bus, to program the main NoC. Due to the way the dual NoC was designed, the reverse operation cannot be done with a read to the same device; instead, whenever there is a message requiring attention (e.g., upon BIST completion or fault detection), the dual NoC sends a packet to the Dual NoC Receiver, which converts it into an AXI transaction directed at the on-bus memory (a standard Xilinx core). The Microblaze can periodically poll this memory to check all notifications. To supervise the NoC activity, the Microblaze can also poll the Traffic Sniffers. These blocks can be connected to up to 16 links of the main NoC on one side, and to the AXI bus on the other. For maximum thoroughness, we choose to monitor as many as 80 links of the NoC (almost all, disregarding just a few whose information is redundant), with five Traffic Sniffers in parallel. The sniffers include a counter that is incremented at the passage of any flit; whenever the counter is read by the MicroBlaze, it automatically resets itself. A simple division yields a utilization metric. Finally, the FPGA needs external interfaces. First of all, a GPIO controller allows the Microblaze to periodically check the status of a

few physical buttons and switches on the FPGA board. This allows the user to change operating modes of the platform; for example, we use this feature to instruct the software on the Microblaze to initiate the reconfiguration to get into virtualized application mode. Two extra blocks are used to communicate with the user's computer. A UART controller is an output-only interface that allows the platform to transfer information to the laptop, where the GUI by UPV can visualize it. A debug module, relying on a JTAG-over-USB electrical connection, allows for bidirectional communication: the user can program the supervision Microblaze, step through its software, and check the content of certain on-FPGA registers and memories. Since the debug module allows for monitoring of up to 8 Microblazes, we connect it to the supervision Microblazes and to selected 7 other Microblazes out of the 16 attached to the main NoC mesh.

## 7 Basic components: the reconfiguration algorithm

The supervisor MicroBlaze is constantly monitoring the status of the NoC through the dual NoC. Whenever a notification is received about a fault on a link, if deemed necessary (e.g. unless it is assumed to be a transient), the supervisor triggers the reconfiguration algorithm. This algorithm computes the required changes in the LBDR bits at specific switches in order to migrate from the current routing algorithm to a new one that avoids the use of the notified link. Those bits are encoded and transmitted through the dual NoC together with a triggering notification to switches to launch the reconfiguration process.

## 8 The application

The MicroBlazes have been programmed in order to start their application after the 4x4 mesh is configured, upon flipping a physical switch on the board. The application run by the MicroBlazes is a matrix multiplication consisting of the product of a pair of matrices. The MicroBlazes sequentially forward the results to each other in a pipelined producer-consumer fashion. Each MicroBlaze performs the multiplication of a private matrix and a matrix delivered by the previous MicroBlaze of the sequence. Once the matrix product is computed the resulting matrix is forwarded to the next MicroBlaze. The lack of I/O interfaces and memory does not allow the implementation of more advanced applications. The private matrix (mat private) is stored by each Microblaze into its local (pertaining to its local switch) AHB scratch memory of 4kB.

The AHB memory is used as storage and for inter-processor communication in the application. Indeed the incoming matrix from the previous Microblaze (mat input) is stored in the AHB memory connected to the local switch. Each MicroBlaze stores its matrix product result (mat output) into the AHB memory connected to the switch to which the next MicroBlaze of the sequence is connected. The first MicroBlaze of the pipeline initializes its own local AHB memory before performing the matrix product. Each MicroBlaze has local registers storing the address of the local AHB memory, the addreses of the remote AHB memory of the next MicroBlaze, the position within the pipeline, and the pipeline length.

In order to guarantee the synchronism between the MicroBlazes, custom semaphores are implemented. Interestingly, these are purely software and do not need dedicated hardware support. Such a solution slightly increases the complexity of the code but clearly simplifies the hardware design effort and the area overhead. Of course this approach is possible only since the application is fixed and known upfront; more sophisticated synchronization capabilities would demand hardware-level atomicity support. The goal of these semaphores is to avoid reading the same incoming matrix multiple times, and to avoid overwriting output matrices before the next MicroBlaze has been able to process them. Each MicroBlaze has been enhanced with 4 semaphores. Notice that semaphores to be polled have been placed in the local AHB memory in order to reduce congestion in the network.

## 9 The physical platform implementation

Some steps of the implementation flow described in Figure 9.4 can be parallelized; for example, the initial platform description involves several blocks which can be independently synthesized in parallel. Even after joining all the pieces together, the mapping stage can be run on two threads in the Xilinx toolchain, and the placement and routing in four.
The platform fills the FPGA almost completely, as can be seen in Table 9.1. The left column reports the utilization when the template system generated in XPS is implemented, the right one is for the same system where the NoC, dual NoC and associated designed IP (e.g. sniffers, dual NoC interface blocks, etc.) have been instantiated to replace the simple AXI interconnect. It can be seen that the NoC represents approximately 17% of the FPGAâs sequential resources and 72% of the combinational resources (or a little bit more, since this is the overhead on top of the default bus); it does not occupy any BRAM nor require external pins. Due to development timing constraints, no specific optimizations could be taken to reduce the area of the design; given the large number of blocks and the redundancy features (e.g. triplication of some components, BIST, datapath encoding) built into the NoC, we perceive the resource utilization figures as very positive. Note that triplicated logic in the switch has to be marked with special annotations embedded in the RTL, otherwise the Xilinx synthesis tools would detect it as redundant and prune it away. The design is not aimed at, and not optimized for, high performance. The very high resource utilization features also impose a significant timing overhead as routing necessarily becomes more convoluted and less efficient. We record a maximum operating frequency of 38 MHz; the critical path is in the BIST logic of the switch.

## 10 Validating Built-in Self-Testing and NoC configuration

In order to validate the Built-in Self-Testing implemented in the 4x4 mesh, a permanent failure was forced in the network by hard-wiring to zero a link wire. In this implementation, the failure was injected in the link between switch 11 and 10. However, it could have been freely injected in different locations since the 4x4 mesh has been based on a switch that guarantees around 97% of stuck-at-fault coverage.

As soon as the FPGA board is booted the BIST automatically starts and the switches cooperatively exchange test patterns as shown in Figure 9.5. When the BIST procedure is completed, the BIST managers integrated in each switch send to the dual NoC the diagnosis information related to the switch they belong to. In the FPGA platform under test, the BIST manager of Switch 10 reveals an error on its East input channel where the error has been injected. Thus it notifies the dual NoC, which takes care of delivering all the BIST di- agnosis information to the global manager (i.e., the supervision MicroBlaze). In particular, the diagnosis information crosses the dual NoC and the Dual NoC Receiver before being stored in the memory connected to the supervision subsystem (Figure 9.6). The supervision MicroBlaze has been programmed to periodically check for dual NoC notifications by polling the control bus memory (Figure 9.7). It recognizes when the BIST notification information has been stored in the memory (i.e., the BIST procedure is completed) and it runs the configuration algorithm. Thus, it computes configuration bits able to guarantee deadlock-free routes despite the failed link. The configuration bits are sent to the Dual NoC Driver through the AXI bus. They cross the dual NoC and configure the routing mechanism of each switch (Figure 9.8).

## 11 Validating Fault Detection and NoC Reconfiguration

Once the network has been tested and permanent faults have been detected and tackled by the off-line configuration, the system can be still affected by run-time transient and intermittent faults. Such faults cannot be handled by off-line strategies as they appear and disappear unpredictably. As a result, the network has been designed as fault tolerant to satisfy the high reliability constraints imposed by modern systems. In particular, the fault-tolerant flow control protocol (NACK/GO) is used on the data path to notify error detection and trigger data retransmissions. Although this protocol has been primarily designed to tackle SEUs (Single Event Upset), the system is also able to tackle physical effects such as wear-out. Indeed wear-out effects end up in permanent faults but they are known to have a gradual onset. In practice, frequent transient faults affecting the same circuitry denote the possible onset of a permanent fault. Before this happens, the network routing function could be modified to exclude the affected circuit from communication traffic. NACK/GO lends itself to such a policy, since its retransmission and/or voting events may be notified to the supervision MicroBlaze which may monitor the distribution and frequency of transient faults over time and eventually take the proper course of recovery action. This exact policy is supported and validated by the FPGA platform. Physical buttons and switches of the board are connected to an on-chip GPIO controller to allow the user to interact with the platform. The physical buttons have been leveraged to inject transient faults in the network and validate the above mentioned fault tolerance policy. For this purpose, a fault injection module has been instantiated along the link routed from switch 4 to 5. This module is connected to a physical button on the FPGA board and provides a method to inject transient faults on that link (see Figure 9.9). Since the link may be idle when the button is pressed, the fault injection module integrates a simple FSM that waits until a valid flit is crossing the link to inject the fault, ensuring that actual important information is corrupted. Therefore, every time the button is pushed, the error is revealed and notified to the supervision MicroBlaze (Figure 9.10). Similarly to the procedure followed by the BIST notification, the transient notification crosses the dual NoC and it is stored into the control bus memory. The supervision MicroBlaze periodically polls the memory also during run-time operations. Thus it reads the transient notification and updates its register with distribution and frequency of transient faults over time. Only when the number of transient notifications from the same link reaches a threshold is recovery action taken. For the sake of the demonstration, the MicroBlaze's software is set to run its reconfiguration procedure after three notifications (i.e., after the button has been pushed three times). Note that the 4x4 mesh at this stage is irregular since a link has been already disabled due to a previously detected permanent failure. Thus, the algorithm computes the reconfiguration bits for this irregular network and delivers them to the dual NoC (Figure 9.11). The new reconfiguration bits coming from the dual NoC cannot directly update the existing routing strategy, as during off-line operations, since applications are now running. Thus, the network implements the OSR-Lite reconfiguration mechanism which avoids stopping or draining network traffic during the transition from one network configuration to another. The switches of the FPGA platform start to inject tokens into the network. The tokens follow the channel dependency graph of the old routing function and progressively drain the network from old packets, as represented in Figure 9.12.

| Resource Utilization | Supervisor Subsystem only | Full Platform (fJ/bit) |
|---|---|---|
| Slice Registers | 5% | 22% |
| Slice LUTs | 16% | 88% |
| IOs | 20% | 20% |
| 36-bit BRAMs | 61% | 61% |

Table 9.1: Resource utilization of the Virtex 7 chip.

## 12 Conclusion

This chapter reports on the prototyping of a 16-core homogeneous multicore processor with a fault-tolerant, runtime reconfigurable and dynamically virtualizable on-chip network. The prototyped system has been successfully validated in its capability of boot-time testing and configuration, transient or intermittent fault detection, runtime reconfiguration of the routing function, and dynamic partitioning and isolation. The validated NoC prototype is a key enabler for the future evolution of embedded systems. First, it enables the integration of multiple software functions on a single multi- and many-core processor (multifunction integration). This is the most efficient way of utilizing the available computing power. Finally, a comprehensive reliability framework has been set into place, from switch- level to network-level, while covering all design aspects (e.g., reliable control signaling) and effectively co-optimizing different architectural features together (fault-tolerance, testing,

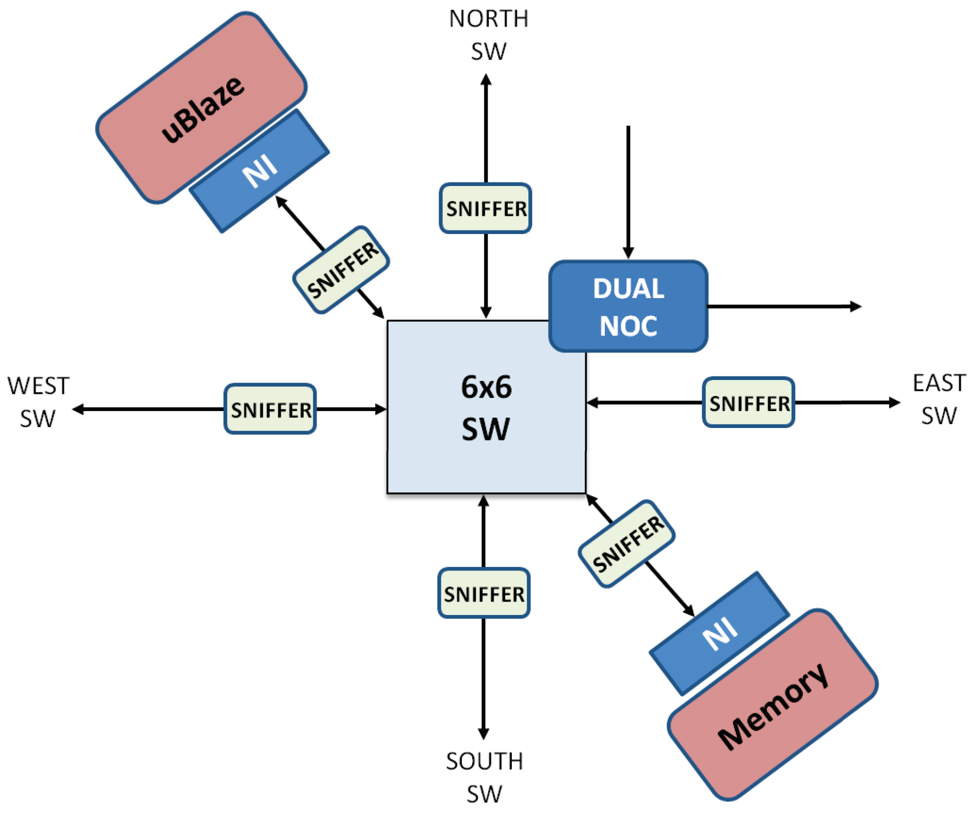


Figure 9.3: Basic components of the on-chip network.

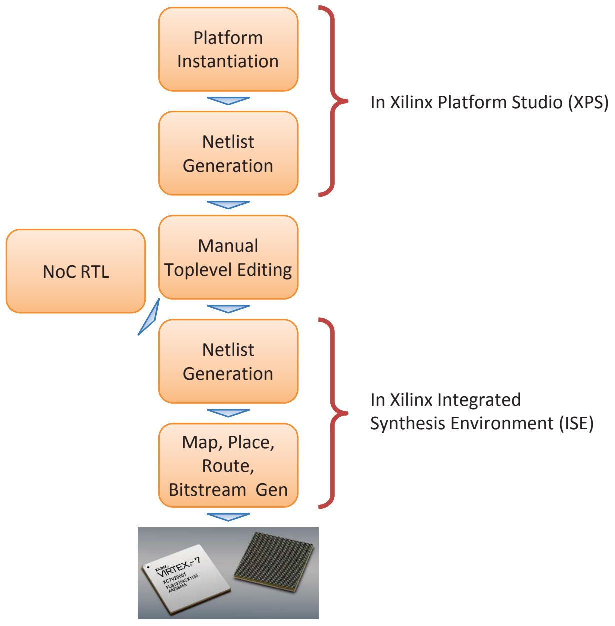

Platform Instantiation
Netlist Generation
In Xilinx Platform Studio (XPS)
NoC RTL
Manual Toplevel Editing
Netlist Generation
Map, Place, Route, Bitstream Gen
In Xilinx Integrated Synthesis Environment (ISE)
VIRTEX-7

Figure 9.4: Design flow for platform implementation.

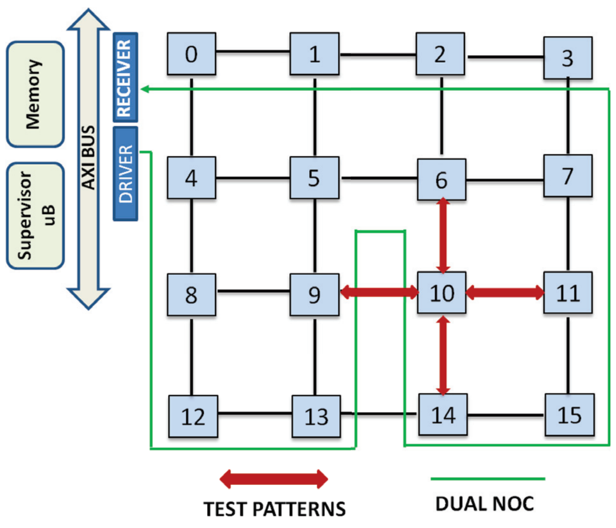


Figure 9.5: Built-In-Self-Testing at work (a).

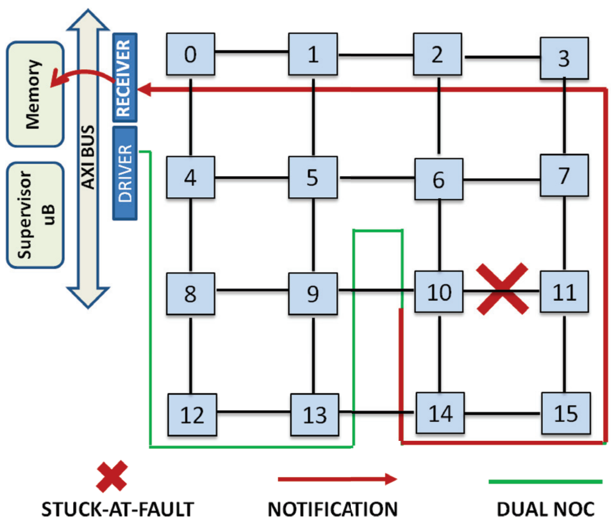


Figure 9.6: Built-In-Self-Testing at work (b).

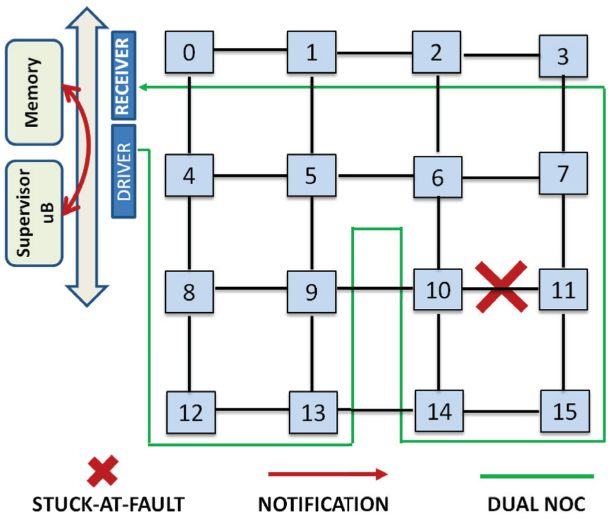


Figure 9.7: Built-In-Self-Testing at work (c).

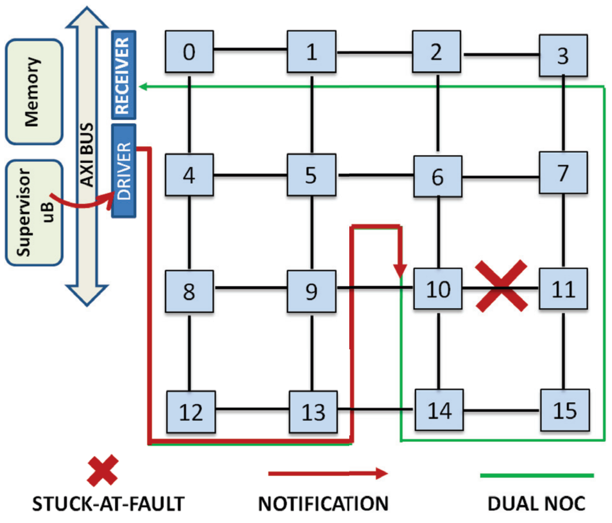


Figure 9.8: Built-In-Self-Testing at work (d).

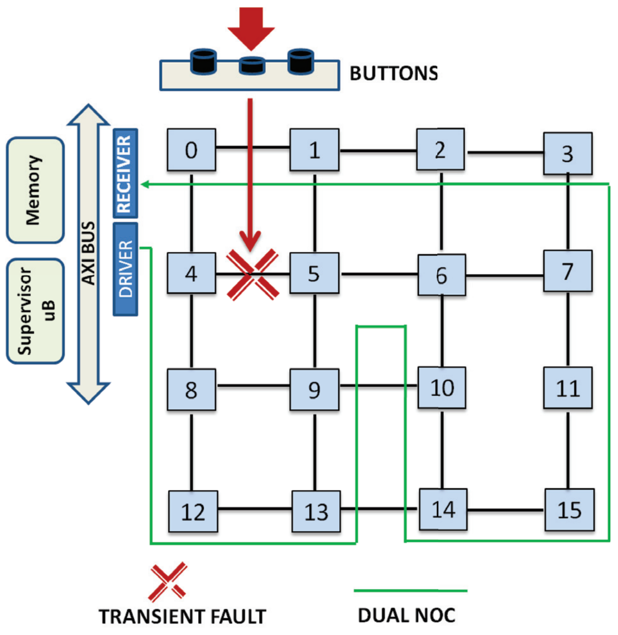


Figure 9.9: Transient fault detection and reconfiguration (a).

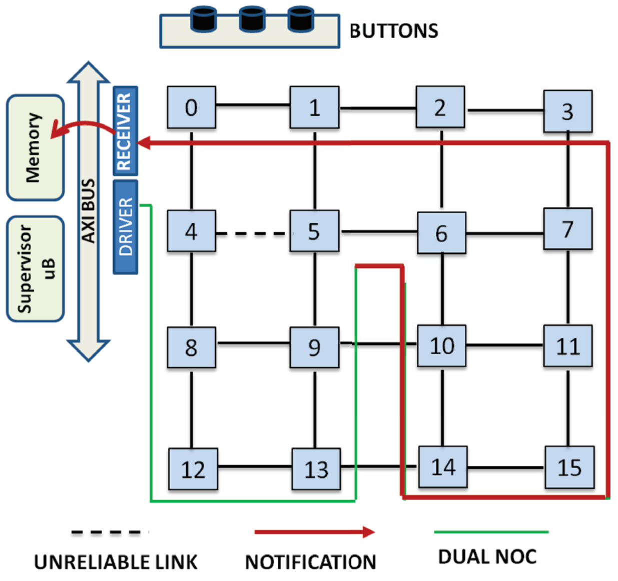


Figure 9.10: Transient fault detection and reconfiguration (b).

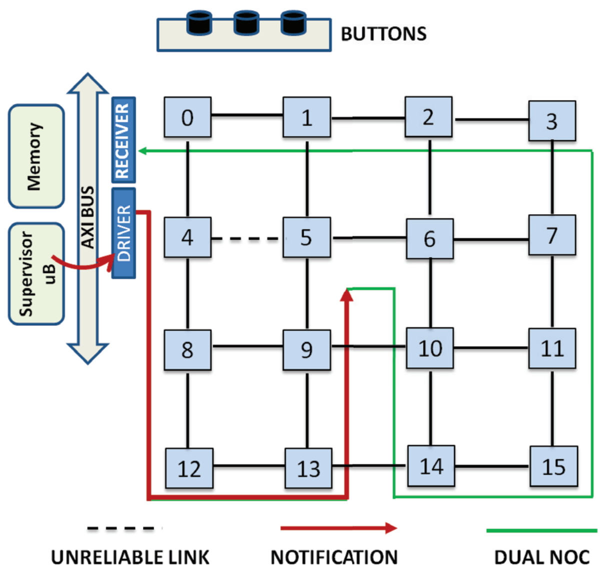


Figure 9.11: Transient fault detection and reconfiguration (c).

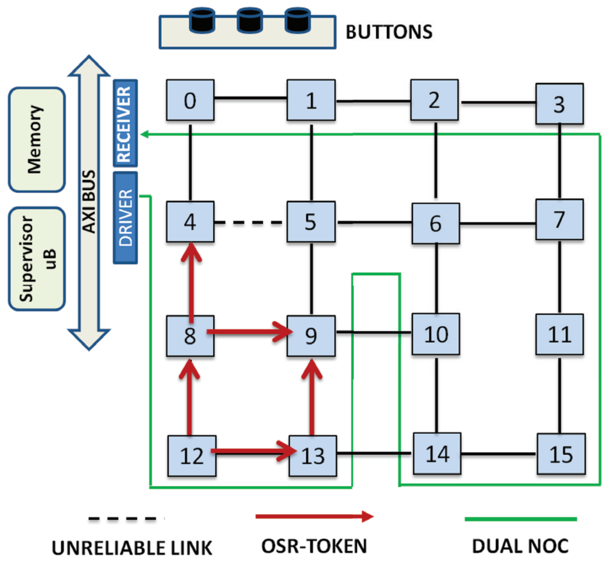


Figure 9.12: Transient fault detection and reconfiguration (d).

runtime reconfiguration, control signaling) [1].

[1]This chapter has included contents that are referred to a cooperative and interdisciplinary work where furher details are in[74].

# Chapter 10

# Power Characterization of Optical NoC Interfaces

## 1 Introduction

The objective of this chapter is to characterize the power consumptions of an optical network interface with respect to the electronic one. Every electronic component has been synthesized, placed and routed using a Low-Power 40nm industrial technology library, in order to provide realistic power measurements (not derived from optimistic or ideal estimations). Power metrics have been calculated by backannotating the switching activity of block internal nets, and then importing waveforms in the PrimeTime Tool. It is worth observing that we have applied clock gating for the sake of realistic measurement of static power. Energy per Bit has been computed by removing the Static Power by the Total power on a component-basis, under 50% switching activity assumption.

## 2 Optical Network Interface Architecture

This section describes, to the best of our knowledge, the first complete network interface architecture for optical networks as depicted in figure 10.1. As a consequence, the objective is not to present the best possible design point, but rather to start considering the basic components, and indicating which one deserves the most intensive optimization effort for prime time of optical interconnect technology. To avoid message-dependent deadlock, every network interface needs separate buffering resources for each one of the three message classes of the MOESI protocol. This should be combined with the requirements of wavelength routing: each initiator needs an output for each possible target, and each target needs an input for each possible source. As a result, in the baseline version of the NI, each initiator comes with 3 FIFOs for each potential target, and each target with 3 FIFOs for each potential initiator. In a more energy-efficient version of the NI (the one in Figure 10.1), all destinations share the same 3 FIFOs and the flits are sent to different paths afterwards (all logic components after 1x15 demuxes are replicated for each destination). All the FIFOs at both the transmission and the reception side must be dual-clock FIFOs to move data between the processor frequency domain (1.2GHz) and the one used inside the NI. The serializers are responsible for translating the flit into a 10 GHz bit stream. The reception side is specular: flits must follow the deserialization process and another set of dual-clock FIFOs. Clearly, ONoCs move most of their complexity to the NIs, which should therefore not be overlooked by means of overly abstract models. Another key issue to be considered in NIs concerns the resynchronization of received optical pulses with the clock signal of the electronic receiver. In this chapter we assume source-synchronous communication, which implies that each point-to-point communication requires a strobe signal to be transmitted along with the data on a separate wavelength, and used to correctly sample received data. Optical transmission of clock signals is an active research field: see for instance [114]. This strobe signal is generated starting from the electrical clock of the transmitter, and removes the need for phase-locked loops (PLLs) or delay-locked loops (DLLs). In this work, we assume that a form of clock gating is implemented, therefore when no data is transmitted, the clock signal is gated. Another typically overlooked issue consists of the backpressure mechanism. We opt for credit-based flow control because it does not rely on timing assumptions, and credit tokens can reuse the existing communication paths, thus avoiding any waveguide, and resulting in a milder impact over static power.

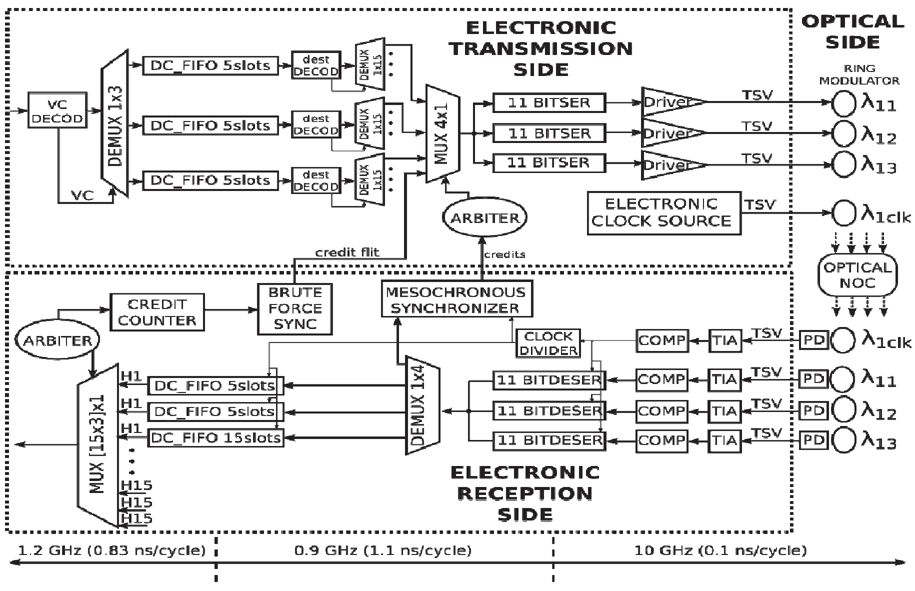


Figure 10.1: Optical Network Interface Architecture.

## 3 Power Characterization

Electronic NI buffering and frequency converters (dual-clock FIFOs) contribute around 11.5mW (see Fig.10.2 and Fig.10.3). The static power dissipated (Idle power) by the entire network (16 switches), is around 286 mW (see 10.4 and 10.5, only the top-level clock tree is omitted). Similarly, the power dissipation of Optical Network Interfaces is computed by composing the power consumption of each of its sub-blocks (DC FIFOs at the transmission sides, Demultiplexers, SERs, Synchronizers, DESERs, DC FIFOs at the reception sides, Multiplexers, and Credit counters).

The static power contribution of all optical components is given by: Laser sources, Thermal tuning, Transmitter (i.e.the driver-ring modulator couple), Receiver (i.e., Photodetector, Trans-Impedence Amplifier, and Comparator) and the source-synchronous clock. The latter addendum is internally composed by further laser sources, Transmitters, Receivers, and MRRs as well. For these parameters we assume values derived from the literature[113], [112]. These resources must be replicated as many times as the target bit parallelism, and also for the optical clock support. Power metrics of all basic blocks of our architectures are summarized in Table10.1. The derived static power values for electronic and optical components are illustrated in figures 10.4

and 10.5.

| Electronic Devices | Static Power (mW) | Dynamic Energy (fJ/bit) |
|---|---|---|
| DC_FIFO_TX_5 //3 | 0.12 | 10.65 |
| DC_FIFO_RX_5 //3 | 0.12 | 8.54 |
| DC_FIFO_TX_22 //3 | 0.12 | 39 |
| DC_FIFO_RX_15 //3 | 0.12 | 26.50 |
| MUX4x1_ARB //3 | 0.08 | 0.36 |
| MUX45x1_ARB //3 | 0.9 | 5.09 |
| SERIALIZER //3 | 0.0475 | 9.41 |
| DESERIALIZER //3 | 0.0289 | 7.74 |
| MESO_SYNCH //3 | 0.082 | 8 |
| BRUTE_FORCE //3 | 0.004234 | 1.4 |
| DC_FIFO_TX_5 //4 | 0.12 | 12.72 |
| DC_FIFO_RX_5 //4 | 0.12 | 10.2 |
| DC_FIFO_TX_22 //4 | 0.12 | 46.41 |
| DC_FIFO_RX_15 //4 | 0.12 | 31.65 |
| MUX4x1_ARB //4 | 0.11 | 0.49 |
| MUX45x1_ARB //4 | 0.9 | 5.09 |
| SERIALIZER //4 | 0.0417 | 2.63 |
| DESERIALIZER //4 | 0.0281 | 6.12 |
| MESO_SYNCH //4 | 0.113 | 11.1 |
| BRUTE_FORCE //4 | 0.00503 | 1.66 |
| DEMUX1x3 //4 | 0.000725 | 0.92 |
| DEMUX1x15 //4 | 0.0021 | 25.21 |
| DEMUX1x4 //4 | 0.00056 | 6.72 |
| COUNTER@4bits //4 | 0.02964 | 1.014 |
| TSV | / | 2.5 |
| TRANSMITTER (aggr) | 0.025 | 20 |
| TRANSMITTER (real) | 0.100 | 50 |
| RECEIVER (aggr) | 0.050 | 10 |
| RECEIVER (real) | 0.150 | 25 |
| THERMAL. T @20K | 0.020 | / |
| E-SWITCH (3VC) | 5.844 | 193 |

Table 10.1: Static and Dynamic Power Of Electronic Devices.

## 4 Analysis and Discussion

Network interfaces are typically oversimplified, and end up being abstracted by simple input/output FIFOs of infinite length. Similarly, the blocking effect of the backpressure mechanism is overlooked. As a consequence, the ONoC easily proves much more performance-efficient than the electronic counterpart. Moreover, the lack of a layout analysis in addition to a physical-layer analysis in ONoC design is another important source of optimism in previous evaluations. In contrast, the key strength of this research (AMF methodology) consists of a careful exploration of E/O and O/E interfaces, accounting for the contributions and effects of every building block: routing, buffering, serialization and deserialization processes, as well as optical transmitters and receivers, clock domain synchronizer, backpressure cost.

## 5 Conclusion

This chapter aims at a high level of practical relevance in the power characterization of an optical NoC vs. its electronic counterpart. The key novelty consists of the use of an electronic baseline aggressively optimized for low-power. With conservative projections for optical component parameters, the major role played by static power is apparent. This calls for new power gating techniques. With more aggressive projections, the network interface turns out to be the clear bottleneck to achieve the break-even point with low-power ENoCs, hence it should be thoroughly analyzed for optimization. In future work, we will investigate more communication-dominated scenarios, in an attempt to capitalize on the far lower dynamic power consumption of ONoCs. [1]

[1]This chapter has included contents that are referred to a cooperative and interdisciplinary work where furher details are in[115].

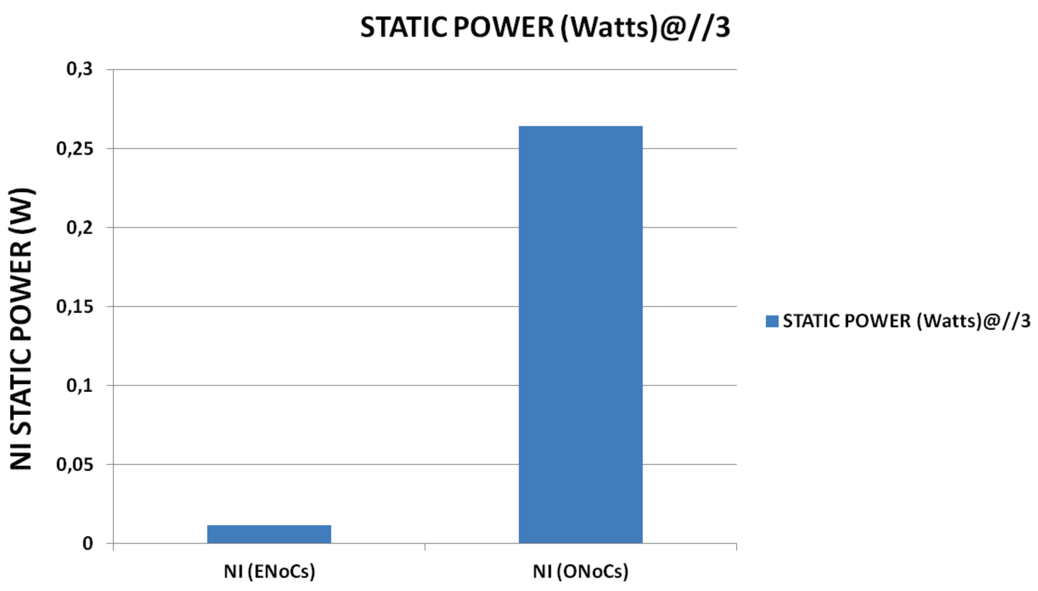


Figure 10.2: Static Power of Electronic Network Interface vs. Optical Network Interface@//3.

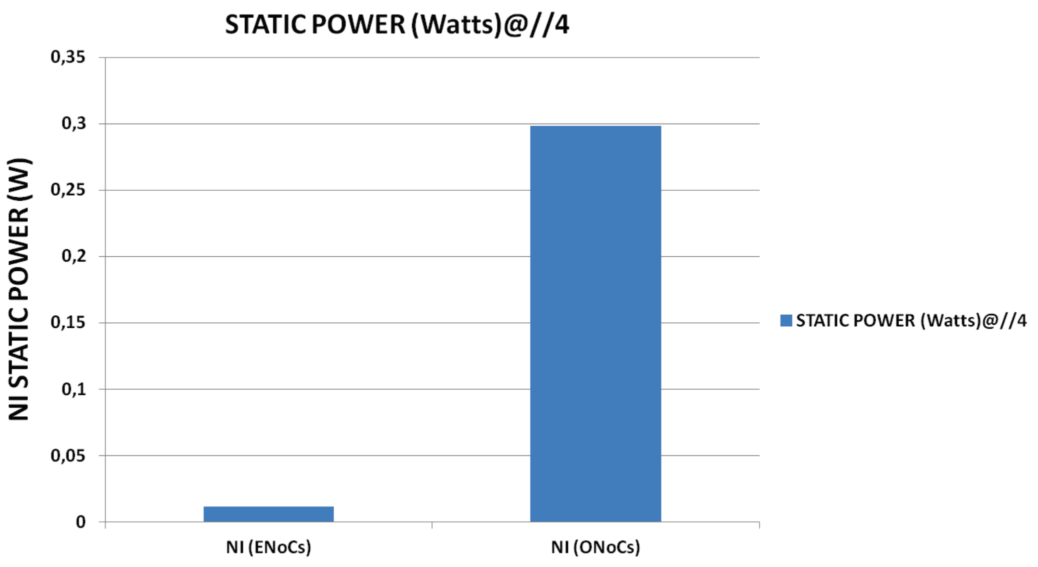


Figure 10.3: Static Power of Electronic Network Interface vs. Optical Network Interface@//4.

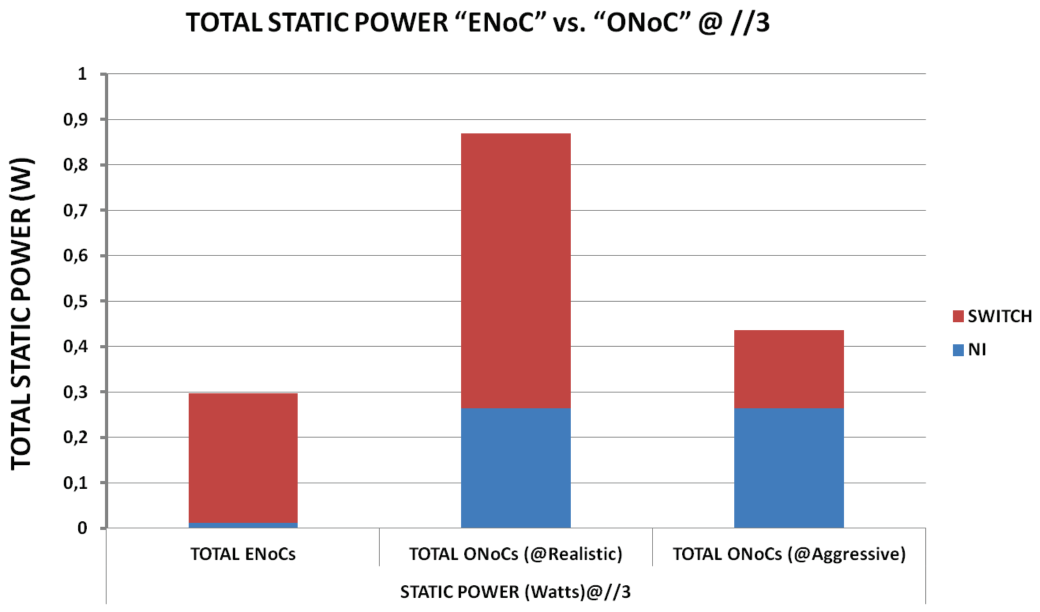


Figure 10.4: Total Static Power of Electronic Network vs. Optical Network @//3.

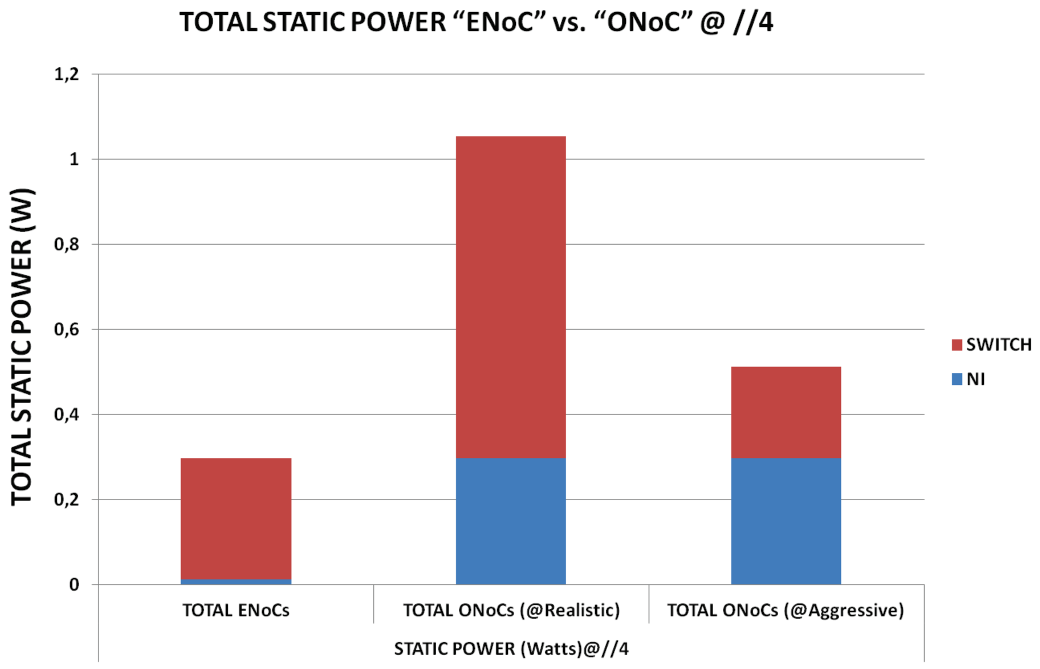


Figure 10.5: Total Static Power of Electronic Network vs. Optical Network @//4.

# Conclusions

This study is focused on the next generation of homogeneous many-core systems. In particularly it deals with cross-layer design, optimization and prototyping of interconnection on-chips. The main goals of this thesis were to design, optimize, test and prototype an on-chip interconnection network. To achieve these objectives, we first analyzed two architectural variants of a mesh and we made an architectural study of three types of interconnection on chip to better understand the various trade-offs (Power consumption, area and performance) between synchronous, mesochronous and multi-synchronous NoCs. Subsequently, we designed various testing infrastructures and we assessed the coverage, the area overhead and the testing cycles. Moreover, we designed an ultra-low latency network-on chip testing infrastructure, suitable for online testing. In addition, we design a new congestion avoidance mechanism named HACS to reduce congestion in the network with interesting results. The most important of all was the validation of some of the ideas of this thesis on a FPGA prototyping. Finally, our goals were achieved and we started to pave the way for emerging technologies such as optical interconnect technology by providing a key enabler for the characterization of power consumption of optical network interfaces. Overall, the thesis is a comprehensive contribution to the advance in the field of manycore NoC-based system design.

# Bibliography

[1] *Simone Terenzi, Alessandro Strano, Davide Bertozzi.*
*"Optimizing Built-In Pseudo-Random Self-Testing for Network-on-Chip Switches" - INA-OCMC 2012*

[2] *S.Y.Lin, C.C.Hsu, A.Y.Wu.*
*"A Scalable Built-In Self-Test/Self-Diagnosis Architecture for 2D-mesh Based Chip Multiprocessor Systems"*
*IEEE Int. Symp. on Circuits and Systems - 2009*

[3] *A. Strano, C. Gómez, D. Ludovici, M. Favalli, M.E. Gómez, D. Bertozzi.*
*"Exploiting Network-on-Chip Structural Redundancy for A Cooperative and Scalable Built-In Self-Test Architecture" - DATE -2011*

[4] *Markus, A.; Raik, J.; Ubar, R.*
*"Fast and Efficient Static Compaction of Test Sequences Using Bipartite Graph Representation"*
*Proc. of the Second Electronic Circuits and Systems Conference (ECS'99)*

[5] *Sheng Zhang, Sharad C seth, Bhargab B, Bhattacharya.*
*"Efficient Test Compaction for Pseudo-Random Testing"*
*Proc. of the 14th Asian Test Symposium (ATS '05)*

[6] *S.Stergiou et al.*
*"Xpipes Lite: a Synthesis Oriented Design Library for Networks on Chips" - DAC - 2005*

[7] *D.Wentzlaff et al.*
*"On-Chip Interconnection Architecture of the Tile Processor"*
*-IEEE Micro 2005*

[8] *J.Raik, V.Govind, R.Ubar.*
*"An External Test Approach for Network-on-a-Chip Switches"*
*Proc. of the IEEE Asian Test Symposium - 2006"*

[9] *J.Raik, V.Govind, R.Ubar.*
*"Test Configurations for Diagnosing Faulty Links in NoC Switches"*
*Proc. ETS - 2007*

[10] *D. A. IIitzky, J. D. Hoffman, A. Chun and B. P. Esparza*
*"Architecture of the Scalable Communications Core's Network on Chip"*
*IEEE MICRO - 2007*

[11] *J.Raik, V.Govind, R.Ubar*
*"DfT-based External Test and Diagnosis of Mesh-like NoCs"*
*IET Computers and Digital Techniques - 2009*

[12] *V.Bertacco, D.Fick, A.DeOrio, J.Hu, D.Blaauw, D.Sylvester*
*"VICIS: A Reliable Network for Unreliable Silicon"*
*DAC - 2009*

[13] *K.Peterson, J.Oberg*
*"Toward a Scalable Test Methodology for 2D-mesh Network-on-Chip"*
*DATE - 2007*

[14] *A.M. Amory, E.Briao, E.Cota, M.Lubaszewski, F.G.Moraes*
*"A Scalable Test Strategy for Network-on-Chip Routers"*
*Proc. of ITC-2005*

[15] *K.Arabi*
*"Logic BIST and Scan Test Techniques for Multiple Identical Blocks"*
*IEEE VLSI Test Symnposium 2002*

[16] *C.Grecu, P.Pande, B.Wang, A.Ivanov, R.Saleh.*
*"Logic BIST and Scan Test Techniques for Multiple Identical Blocks"*
*IEEE DFT - 2005*

[17] *R.Ubar, J.Raik*
*"Testing Strategies for Network on Chip" "- book edited by A.Jantsch and H.Tenhunen, Kluwer Academic Publisher" IEEE DFT - 2003*

[18] *C.Aktouf*
*"Testing Strategies for Network on Chip"*
*IEEE Design and Test of Computers - 2002*

[19] *Y.Lin, C.C.Hsu, A.Y.Wu "A Scalable Built-In Self-Test/Self-Diagnosis Architecture for 2D-mesh Based Chip Multiprocessor Systems"*
*IEEE Int. Symp. on Circuits and Systems - 2009*

[20] *M.Hosseinabady, A.Banaiyan, M.N.Bojnordi, Z.Navabi*
*"A Concurrent Testing Method for NoC Switches"*
*DATE - 2006*

[21] *C.Grecu, P.Pande, A.Ivanov, R.Saleh*
*"BIST for Network-on-Chip Interconnect Infrastructures"*
*VLSI Test Symposium-2006*

[22] *S.Rodrigo, J.Flich, A.Roca, S.Medardoni, D.Bertozzi, J.Camacho, F.Silla, J.Duato*
*"Addressing Manufacturing Challenges with Cost-Effective Fault Tolerant Routing"*
*NOCS-2010*

[23] *Antti Markus, Jaan Raik, Raimund Ubar*
*"Fast and Efficient Static Compaction of Test Sequences Using Bipartite Graph Representation"*
*Proc. of the Second Electronic Circuits and Systems Conference,(ECS)-1999*

[24] *F.Clermidy, R.Lemaire, X.Popon, D.Ktenas, Y.Thonnart*
*Euromicro Conference on Digital System Design-2009*
*"An Open and Reconfigurable Platform for 4G Telecommunication: Concepts and Application"*

[25] *F.Clermidy, C.Bernard, R.Lemaire, J.Martin, I.Miro-Panades, Y.Thonnart, P.Vivet, N.Wehn*
*"A 477mW NoC-based Digital Baseband for MIMO 4G SDR"*
*ISSCC-2010*

[26] *Y.Thonnart, P.Vivet, F.Clermidy*
*"A Fully-Asynchronous Low-Power Framework for GALS NoC Integration"*
*DATE-2010*

[27] *R.Dobkin, V.Vishnyakov, E.Friedman, R.Ginosar*
*"An Asynchronous Router for Multiple Service Levels Networks on Chip"*
*Proc. of ASYNC -2005*

[28] *T.Bjerregaard, J.Sparso*
*"A Router Architecture for Connection-Oriented Service Guarantees in the MANGO Clockless Network-on-Chip"*
*DATE-2005*

[29] *W.Kim, M.S.Gupta, G.Y.Wei and D.Brooks*
*"System Level Analysis of Fast, Per-Core DVFS using On-Chip Switching Regulators"*
*Int. Symp. on High-Performance Computer Architecture-2008*

[30] *W.Kim, M.S.Gupta, G.Y.Wei and D.Brooks*
*"System Level Analysis of Fast, Per-Core DVFS using On-Chip Switching Regulators"*
*Int. Symp. on High-Performance Computer Architecture-2008*

[31] *C.Cummings, P.Alfke*
*"Simulation and Synthesis Techniques for Asynchronous FIFO Design with Asynchronous Pointer Comparison"*
*SNUG-2002, San Jose-2002*

[32] *I.M.Panades, A.Greiner*
*"Bi-Synchronous FIFO for Synchronous Circuit Communication Well Suited for Network-on-Chip in GALS Architectures"*
*Int. Symp. on Networks-on-Chip-2007*

[33] *T.Ono, M.Greenstreet*
*"A Modular Synchronizing FIFO for NOCs"*
*International Network-on-Chip Symposium-2009*

[34] *D.Verbitsky, R.R.Dobkin, R.Ginosar*
*"A Four-Stage Mesochronous Synchronizer with Back-Pressure and Buffering for Short and Long Range Communications"*
*International Network-on-Chip Symposium*
*http://webee.technion.ac.il/ ran/papers*

[35] *M.Alshaikh,D.Kinniment, A.Yakovlev*
*"Robust Synchronization using the Wagging Technique"*
*Technical Report. TR NCL EECE-MSD-TR*

[36] *F.Mu, C.Svensson*
*"Self-Tested Self-Synchronization Circuit for Mesochronous Clocking"*
*"IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing-2001"*

[37] *A.Sheibanyrad, I.M.Panades, A.Greiner*
*"Multisynchronous and Fully Asynchronous NoCs for GALS Architectures"*
*IEEE Design and Test of Computers-2008*

[38] *M.N.Horak, S.M.Nowick, M.Carlberg, U.Vishkin*
*"A Low-Overhead Asynchronous Interconnection Network for GALS Chip Multiprocessors"*
*ACM/IEEE Int. Symp. on Networks-on-Chip-2010*

[39] *J.Bainbridge, S.Furber*
*"CHAIN: a Delay-Insensitive Chip Area Interconnect"*
*IEEE Micro-2002*

[40] *L.A.Plana, S.B.Furber, S.Temple, M.Khan, Y.Shi, J.Wu, S.Yang*
*"A GALS Infrastructure for a Massively Parallel Multiprocessor"*
*IEEE Design and Test of Computers-2007*

[41] *E.Beigne, F.Clermidy, P.Vivet, A.Clouard, M.Renaudin*
*"An Asynchronous NoC Architecture Providing Low Latency Service and Its Multilevel Design Framework"*
*IEEE Asynch. Symp.-2005*

[42] *D. Lattard, E. Beigne, F. Clermidy, Y. Durand, R. Lemaire, P. Vivet, F. Berens*
*"A Reconfigurable Baseband Platform Based on an Asynchronous Network-on-Chip"*
*IEEE Journal Of Solid State Circuits-2008*

[43] *B.Quinton, M.Greenstreet, S.Wilton*
*"Practical Asynchronous Interconnect Network Design"*
*IEEE Trans. on VLSI-2008*

[44] *S.Hollis, S.W.Moore*
*"Rasp: an Area-Efficient, on-Chip Network"*
*IEEE Int. Conf. on Computer Design-2006*

[45] *W.J. Dally and J.W. Poulton*
*"Digital Systems Engineering"*
*Cambridge University Press-1998*

[46] *F.Vitullo, N.E.L'Insalata et al.*
*"Low-Complexity Link Microarchitecture for Mesochronous Communication in Networks-on-Chip"*
*"IEEE Trans. on Computers-2008*

[47] *I.M.Panades, F.Clermidy, P.Vivet, A.Greiner*
*"Physical Implementation of the DSPIN Network-on-Chip in the FAUST Architecture"*
*ACM/IEEE Int. Symp. on Networks-on-Chip-2008*

[48] *D.Wentzlaff et al.*
*book chapter from "Designing Network On-Chip Architectures in the Nanoscale Era, edited by J.Flich and D.Bertozzi, CRC Press"*
*Networks of the Tilera Multicore Processor-2011*

[49] *S.Murali et al.*
*"NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems-on-Chip"*
*IEEE Trans. on Parallel and Distributed Systems-2005*

[50] *D. Ludovici, A. Strano, G. N. Gaydadjiev and D. Bertozzi*
*"Mesochronous NoC Technology for Power-Efficient GALS MPSoCs"*
*INAOCMC-2011*

[51] *S.Beer, R.Ginosar, M.Priel, R.R.Dobkin, A.Kolodny*
*"The Devolution of Synchronizers"*
*ASYNCH-2010*

[52] *A.Strano, D.Ludovici, D.Bertozzi*
*"A Library of Dual-Clock FIFOs for Cost-Effective and Flexible MP-SoCs"*
*Proc. of SAMOS-2010*

[53] *D. Ludovici and A. Strano and G. N. Gaydadjiev and L. Benini and D. Bertozzi*
*"Design Space Exploration of a Mesochronous Link for Cost-Effective and Flexible GALS NOCs"*
*Proceedings of Design, Automation and Test in Europe-2010*

[54] *D. Ludovici, A. Strano, D. Bertozzi*
*"Architecture design principles for the integration of synchronization interfaces into Network-on-Chip switches"*
*NoCArc: Proceedings of the 2nd International Workshop on Network on Chip Architectures-2009*

[55] *Kakoee, M.R. and Loi, I. and Benini, L.*
*"A New Physical Routing Approach for Robust Bundled Signaling on NoC Links"*
*Proceedings of the 20th Great Lakes Symposium on VLSI-2010*

[56] *M. Krstic, X. Fan, C. Wolf, A. Strano, D. Bertozzi*
*"A New Physical Routing Approach for Robust Bundled Signaling on NoC Links"*
*Deliverable D29 - Test and Measurement Report of Moonrake Chip, Galaxy Project-2010*
*www.galaxy-project.org*

[57] *Abts, D., Enright Jerger, N.D., Kim, J., Gibson, D., Lipasti, M.H.
"Achieving predictable performance through better memory controller placement in many-core CMPs"
ACM SIGARCH Computer Architecture News 37(3), 451 (Jun 2009)
http://portal.acm.org/citation.cfmdoid=1555815.1555810*

[58] *Das, R., Mutlu, O., Kumar, A., Azimi, M.
"Application-to-core mapping policies to reduce interference in on-chip networks"
Tech. rep., SAFARI Technical Report No. 2011 (2011)
http://www.ece.cmu.edu/omutlu/pub/interference-aware-noc-mapping-TR-SAFARI-2011-001.pdf*

[59] *Trivino, F., Sanchez, J.L., Alfaro, F.J., Flich, J.
" Virtualizing network-on-chip resources in chip-multiprocessors"
Microprocessors and Microsystems 35(2), 230245 (Mar 2011)
http://linkinghub.elsevier.com/retrieve/pii/S0141933110000712*

[60] *Gratz, P., Grot, B., Keckler, S.W.
"Regional congestion awareness for load balance in networks-on-chip"
HPCA. pp. 203214. IEEE Computer Society (2008)*

[61] *Li, M., Zeng, Q.A., Jone, W.B.
"DyXY: a proximity congestion-aware deadlock-free dynamic routing method for network on chip."
"Proceedings of the 43rd annual Design Automation Conference. pp. 849852. DAC 06, ACM, New York,2006
http://doi.acm.org/10.1145/1146909.1147125*

[62] *Marescaux, T., Rangevall, A., Nollet, V., Bartic, A., Corporaal, H.
"Distributed congestion control for packet switched networks on chip"
Proceedings of the International Conference of Parallel Computing: Current Future Issues of High-End Computing. vol. 33, pp. 761768-2005
http://citeseerx.ist.psu.edu*

[63] *Wu, D., Al-Hashimi, B.M., Schmitz, M.T.*
*"Improving routing efficiency for network-on-chip through contention-aware input selection"*
*Proceedings of the 2006 Asia and South Pacific Design Automation Conference. pp. 3641:*
*ASP- DAC 06, IEEE Press, Piscataway, NJ, USA (2006), http://dx.doi.org*

[64] *Sanchez, D., Michelogiannakis, G., Kozyrakis, C.*
*"An analysis of on-chip interconnection networks for large-scale chip multiprocessors"*
*ACM Transactions on Architecture and Code Optimization (TACO) 7(1), 4 (2010)*
*http://portal.acm.org/citation.cfmid=1736069*

[65] *Das, R., Mutlu, O., Moscibroda, T., Das, C.R.*
*"Application-aware prioritization mechanisms for on-chip networks"*
*Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture - Micro-42 p. 280 (2009)*
*http://portal.acm.org/citation.cfmdoid=1669112.1669150*

[66] *Grot, B., Keckler, S., Mutlu, O.*
*"Preemptive virtual clock: a flexible, efficient, and cost-effective QOS scheme for networks-on-chip"*
*Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture. pp. 268279. ACM (2009)*
*http://portal.acm.org/citation.cfmid=1669149*

[67] *Iyer, R., Zhao, L., Guo, F., Illikkal, R., Makineni, S., Newell, D., Solihin, Y., Hsu, L., Reinhardt, S.*
*"QoS policies and architecture for cache/memory in CMP platforms."*
*ACM SIGMETRICS Performance Evaluation Review 35(1), 25 (Jun 2007)*
*http://portal.acm.org/citation.cfmdoid=1269899.1254886*

[68] *Flich, J., Bertozzi, D.*
*"Designing Network On-Chip Architectures in the Nanoscale Era."*
*Chapman & Hall/CRC (2010)*

[69] *Chen, G., Li, F., Son, S.W., Kandemir, M.*
*"Application mapping for chip multiprocessors." Proceedings of the 45th annual conference on Design automation - DAC 08 p. 620 (2008)*
*http://portal.acm.org/citation.cfmdoid=1391469.1391628*

[70] *Das, R., Mutlu, O., Kumar, A., Azimi, M.*
*"Application-to-core mapping policies to reduce interference in on-chip networks"*
*Tech. rep., SAFARI Technical Report No. 2011 (2011)*
*http://www.ece.cmu.edu/omutlu/pub/interference-aware-noc-mapping-TR-SAFARI-2011-001.pdf*

[71] *Rodrigo, S., Flich, J., Roca, A., Medardoni, S., Bertozzi, D., Camacho, J., Silla, F., Duato, J.*
*"Addressing manufacturing challenges with cost-efficient fault tolerant routing."*
*NOCS 10: Proceedings of the 4th ACM/IEEE International Symposium on Networks-on-Chip. pp. 2532 (2010)*

[72] *Gilabert, F., Gomez, M.E., Medardoni, S., Bertozzi, D.*
*"Improved utilization of noc channel bandwidth by switch replication for cost-effective multi-processor systems-on-chip"*
*"Proceedings of the 2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip. pp. 165172."*
*IEEE Computer Society, Washington, DC, USA (2010)*
*http://dx.doi.org/10.1109/NOCS.2010.25*

[73] *Ubal, R., Sahuquillo, J., Petit, S., Lopez, P.*
*"Multi2Sim: A Simulation Framework to Evaluate Multicore-Multithreaded Processors"*
*Proc. of the 19th Intl Symposium on Computer Architecture and High Performance Computing (2007)*

[74] *"NaNoC: NaNoC design platform."*
*http://www.nanoc-project.eu*

[75] *"Roca, S., Flich, J., Silla, F., Duato, J."*
*"VCTlite: Towards an efficient implementation of virtual cut-through switching in on-chip networks"*
*International Conference on High Performance Computing (HiPC). pp. 112 (2010)*

[76] *Mejia A., Flich, J., Duato, J., Reinemo, S.A., Skeie, T.*
*"Segment-based routing: An efficient fault-tolerant routing algorithm for meshes and tori"*
*International Parallel and Distributed Processing Symposium 0, 84 (2006)*

[77] *"Multi2sim Wiki: SPLASH2 execution commands."*
*http://www.multi2sim.org/wiki/index.php5/SPLASH2_Execution_Commands*

[78] *Rijpkema, E.; Goossens, K.; Radulescu, A.*
*"Trade Offs in the Design of a Router with Both Guaranteed and Best-Effort Services for Networks on Chip."*
*DATE03, Mar. 2003, pp. 350-355.*

[79] *Jose Flich, Davide Bertozzi.*
*"Designing Network On-Chip Architectures in the Nanoscale Era"*
*by Chapman and Hall/CRC (2010).*

[80] *Diego Melpignano, Luca Benini, Eric Flamand, L. Benini, Bruno Jego, Thierry Lepley, Germain Haugou, Fabien Clermidy , Denis Dutoit.*
*"Platform 2012, a Many-Core Computing Accelerator for Embedded SoCs: Performance Evaluation of Visual Analytics Applications"*
*DAC 2012, June 3-7, 2012, San Francisco, California, USA.*

[81] *Peter Mandl, Udeepta Bordoloi*
*"General-purpose Graphics Processing Units Deliver New Capabilities to the Embedded Market"*
*http://www.amd.com/tw/Documents/GPGPU-Embedded.pdf*

[82] *S.Stergiou et al.*
*"Xpipes Lite: a Synthesis Oriented Design Library for Networks on Chips"*
*DAC, pp.559-564, 2005.*

[83] *A.Strano, F.Trivino, Jose L. Sanchez, Jose Flich, D.Bertozzi*
*"OSR-Lite: Fast and Deadlock-Free NoC Reconguration Framework*
*SAMOS 2012.*

[84] *"S.Rodrigo et Al."*
*"Addressing Manufacturing Challenges with Cost-Effective Fault Tolerant Routing"*
*NOCS 2010, pp.35-32, 2010.*

[85] *S.Terenzi, A.Strano, D.Bertozzi*
*"Optimizing Built-In Pseudo-Random Self-Testing for Network-on-Chip Switches"*
*INA-OCMC 2012.*

[86] *M.Radetzki, C.Feng, X.Zhao, and A.Jantsch.*
*"Methods for fault tolerance in networks on chip.*
*ACM Computing Surveys - 2012.*

[87] *A.Ghiribaldi, A.Strano, M.Favalli, D.Bertozzi.*
*"Power Efficiency of Switch Architecture Extensions for Fault Tolerant NoC Design."*
*IGCC12, 2012, California, USA.*

[88] *O. Lysne, et Al.*
*"An efficient and deadlock-free network reconfiguration protocol"*
*IEEE Transactions of Computers, pp.762779, 2008.*

[89] *A. Ghiribaldi, D. Ludovici, M. Favalli, D. Bertozzi.*
*"System-Level Infrastructure for Boot-time Testing and Configuration of Networks-on-Chip with Programmable Routing Logic".*
*VLSI-SoC, 2011*

[90] *Jared C. Smolens, Brian T. Gold, James C. Hoe, Babak Falsafi, and Ken Mai*
*"Detecting Emerging Wearout Faults" SELSE, 2007.*

[91] *D.Wentzlaff et al.*
*IEEE Micro, vol.27, no.5, pp.15-31, 2007.*

[92] *D. A. IIitzky, J. D. Hoffman, A. Chun and B. P. Esparza*
*"Architecture of the Scalable Communications Cores Network on Chip".*
*IEEE Micro, vol.27, Issue 5, pp.62 - 74,2007.*

[93] *S.Vangal et al.,*
*"An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS".*
*IEEE Journal of Solid-State Circuits, Vol.43, Issue 1, pp.29-41,2008.*

[94] *E. Flamand*
*"Strategic Directions Towards Multicore Application Specific Computing".*
*Proc. of DATE, pp.1266, 2009.*

[95] *M. Krstic et al;*
*"Globally Asynchronous, Locally Synchronous Circuits: Overview and Outlook".*
*IEEE Design and Test of Computers, vol. 24, no. 5, pp. 430-441, 2007.*

[96] *S.Borkar*
*"Design Perspectives on 22nm CMOS and Beyond"*
*Proc. of DAC 2009.*

[97] *R.Dobkin, V.Vishnyakov, E.Friedman, R.Ginosar*
*"An Asynchronous Router for Multiple Service Levels Networks on Chip"*
*Proc. of ASYNC, pp.44-53, 2005.*

[98] *T.Bjerregaard, J.Sparso*
*"A Router Architecture for Connection-Oriented Service Guarantees in the MANGO Clockless Network-on-Chip"*
*Proc. of DATE, pp.1226-1231, 2005.*

[99] *S. Stergiou, F. Angiolini, S. Carta, L. Raffo, D. Bertozzi, G. De Micheli "xpipes Lite: a Synthesis Oriented Design Library for Networks on Chips" Proc. of DATE, pp.11881193, 2005.*

[100] *Kakoee, M.R. and Loi, I. and Benini, L.*
*"A New Physical Routing Approach for Robust Bundled Signaling on NoC Links"*
*Proc. of GLSVLSI, pp.3-8, 2010.*

[101] *D. Ludovici and A. Strano and G. N. Gaydadjiev and L. Benini and D. Bertozzi*
*"Design Space Exploration of a Mesochronous Link for Cost-Effective and Flexible GALS NOCs"*
*Proc. of DATE, pp.679-684, 2010.*

[102] *D. Ludovici, A. Strano, D. Bertozzi*
*"Architecture design principles for the integration of synchronization interfaces into Network-on-Chip switches"*
*Proc. of NoCArc, pp.31-36, 2009.*

[103] *D. Ludovici, A. Strano, D. Bertozzi, L. Benini, G.N. Gaydadjiev*
*"Comparing tightly and loosely coupled mesochronous synchronizers in a NoC switch architecture"*
*Proc. of NOCS, pp.244-249, 2009.*

[104] *A. Strano and D. Ludovici and D. Bertozzi*
*"A Library of Dual-Clock FIFOs for Cost-Effective and Flexible MPSoCs Design"*
*Proc. of SAMOS, 2010.*

[105] *T.N.K.Jain*
*"Asynchronous Bypass Channels: Improving Performance for Multi-Synchronous NoCs"*
*Proc. of NOCS, pp.51-58, 2010.*

[106] *F.Vitullo et al.*
*"Low-Complexity Link Microarchitecture for Mesochronous Communica-*

*tion in Networks-on-Chip"*
*IEEE Trans. on Computers, Vol.57, issue 9, pp.1196-1201, 2008.*

[107] *D. Mangano, R. Locatelli, A. Scandurra, C. Pistritto, M. Coppola, L. Fanucci, F. Vitullo, D. Zandri*
*"Skew Insensitive Physical Links for Network on Chip"*
*Proc of NANO-NET, 2006.*

[108] *I.M.Panades, F.Clermidy, P.Vivet, A.Greiner*
*"Physical Implementation of the DSPIN Network-on-Chip in the FAUST Architecture"*
*Proc. of NOCS, pp.139-148, 2008.*

[109] *F.Clermidy, R.Lemaire, X.Popon, D.Ktenas, Y.Thonnart*
*"An Open and Reconfigurable Platform for 4G Telecommunication: Concepts and Application"*
*Proc of DSD, pp.62-74,2009.*

[110] *F.Clermidy, C.Bernard, R.Lemaire, J.Martin, I.Miro-Panades, Y.Thonnart, P.Vivet, N.Wehn*
*"A 477mW NoC-based Digital Baseband for MIMO 4G SDR"*
*ISSCC2010, pp.278-279, 2010.*

[111] *Y.Thonnart, P.Vivet, F.Clermidy*
*"A Fully-Asynchronous Low-Power Framework for GALS NoC Integration"*
*Proc. of DATE, pp.33-38, 2010.*

[112] *C. Batten, A. Joshi, V. Stojanovic, K. Asanovic.*
*"Designing chiplevel nanophotonic interconnection networks."*
*Emerging and Selected Topics in Circuits and Systems, IEEE Journal. vol. 2, no. 2, pp. 137-153, 2012.*

[113] *S. Beamer, C. Sun, Y. Kwon, A. Joshi, C. Batten, V. Stojanovic, K. Asanovic.*
*"Re-architecting DRAM memory systems with monolithically integrated silicon photonics."*
*ISCA 2010.*

[114] *J. Leu, V. Stojanovic.*
*" Injection-locked clock receiver for monolithic optical link in 45nm SOI."*
*(A-SSCC), 2011 IEEE Asian, 2011, pp. 149-152.*

[115] *Luca Ramini Paolo Grani, Herve Tatenguem Fankem, A.Ghiribaldi, S.Bartolini, D.Bertozzi*
*Assessing the Energy Break-Even Point between an Optical NoC Architecture and an Aggressive Electronic Baseline*
*DATE-2014, Dresden-Germany*