UNIVERSITY OF FERRARA

Engineering Department of the University of Ferrara

Doctorate Degree in Science of Engineering

Coordinator: Prof. Stefano Trillo

Cycle: XXVI

Methodologies and Toolflows for the Predictable Design of Reliable and Low-Power NoCs

ING-INF/01

Candidate:
Alberto Ghiribaldi

Advisor:
Prof. Davide Bertozzi

Academic Year 2013

To Paola, because no matter what it has always been the two of us.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my advisor, Professor Davide Bertozzi, for his continuous support, motivation and guidance during my research and study. His limitless energy and enthusiasm in this journey together motivated all his Ph.D. Students, including me.

In addition, I would like to thank Professor Steven Nowick for taking me as an intern at Columbia University. This experience made me understood the importance of transmitting your knowledge and how to relate with your co-workers.

A special thank goes to all my lab friends here at the MPSoC research group in Ferrara. What started here is much more than just work together, but are experiences and moments I will never forget.

I am also grateful to my parents, for supporting me and directing me through all these years of University.

My deepest and genuine gratitude goes to my girlfriend Paola, for staying by my side during these years of Ph.D., no matter how tough it was.

Chapter 1

Introduction

In recent years, in order to overcome speed and power limitations of monolithic solutions, Chip Multi-Core and Many-Core have become a common practice in modern industrial landscape. Network-on-Chip (NoC) have been a clear key enabler for effective Multi-Core system integration, because they offer a regular and distributed structure on top of which different processing elements (cores, memories of accelerators) can be integrated, offering in this way the scalability and modularity that traditional shared buses could not achieve.

In the embedded system domain, a first possible solution to create these Multi-Core systems is by replicating in a regular way a basic computation tile, usually composed by a CPU and a memory bank, and its associated NoC switch element. These architectures lend themselves to become general purpose programmable accelerators or processing units, since the associated computational power is strictly related to the number of cores active at a time. At the opposite, if systems Multi-Core are designed for a specific target application domain, they usually feature custom heterogeneous devices, not only CPUs and memory banks but also other integrated blocks, such as peripherals controllers or graphic accelerators and so on, all interconnected by a custom NoC architecture.

However, there is today the unmistakable need to evolve design methodologies and toolflows for Network-on-Chip based embedded systems. In particular, the quest for low-power requirements is nowadays a more-than-ever urgent dilemma. Modern circuits feature billion of transistors, and neither power management techniques nor batteries capacity are able to endure the increasingly higher integration capability of digital devices. Besides, power concerns come together with modern nanoscale silicon technology design issues.

On one hand, system failure rates are expected to increase exponentially at every technology node when integrated circuit wear-out failure mechanisms are not compensated for. As a result, all systems – not only high-availability or mission-critical systems – must be designed for resilience. Being the communication backbone of the entire system, NoCs are very susceptible to permanent, intermittent and transient faults, since a single error in the network can put the whole system integrity at risk, jeopardizing its connectivity. However, error detection and/or correction mechanisms have a non-negligible impact on the network power, and if power-reliability trade-off is not properly accounted for upfront, the design outcome can be no longer viable for its excessive energy requirements.

On the other hand, to meet the stringent time-to-market deadlines, the design cycle of such a distributed and heterogeneous architecture must not be prolonged by unnecessary design iterations. In particular, it is mandatory that modern and future toolflows take into account back-end physical constraints, in order to find from the first design stages the correct placement of various blocks, in this way minimizing hotspots, enhancing communication traffic locality and paving the way for final timing convergence. This holds especially when considering Application-Specific Embedded Systems. In fact, being these systems built from many different IPs, system construction requires a custom-tailored Network-on-Chip topology synthesis. For application specific NoCs, the synthesis flow is key in order to reach the most predictable and power-efficient solution for the system at hand.

Overall, there is a clear need to better discriminate reliability strategies and interconnect topology solutions upfront, by ranking designs based on power metric. In this thesis, we tackle this challenge by proposing power-aware design technologies. First of all, we start analyzing design methodologies for cost-effective fault-tolerant NoC design, deriving the most suitable application for both General-Purpose and Application-Specific embedded systems. In the second place, we exploit these design methodologies for fault tolerance as a case study on top of which we develop a comprehensive and interoper-

able synthesis flow for Application-Specific Network-on-Chip, spanning from application requirements down to layout generation, dealing with Network-on-Chip design on nanoscale technologies with less design iterations.
Finally, we take into account the most aggressive and disruptive methodology for embedded systems with ultra-low power constraints. Since clock distribution itself represents a considerable amount of the whole power consumption of the entire system, our final goal is to migrate NoC basic building blocks to asynchronous (or clockless) design style, proposing a switch architecture that outperforms its synchronous counterpart in terms of energy-per-bit, power consumption and area footprint, while maintaining comparable performance with its synchronous counterpart. We deal with this challenge delivering a standard cell design methodology and mainstream CAD tool flows, in this way partially relaxing the requirement of using asynchronous blocks only as hard macros.

Chapter 2

Design Methodologies for Fault-Tolerant NoC Design

1 Motivation

Modern integrated systems are increasingly multi-core, since congruent multiples in processing power are pursued by replicating processing engines on the same silicon die rather than by evolving the microarchitecture of monolithic processing cores. This latter approach has in fact historically resulted in diminishing returns. In this context, systems become susceptible to errors: even though replicated cores are available in such chip multiprocessors, they are not sufficient for providing system level fault protection due to the lack of fault tolerance and/or fault isolation in their shared components [1].

The system integration and communication infrastructure is certainly the most important of such shared components. For large scale systems, network-on-chip (NoC) architectures are today mainstream for global intra-chip communication: they facilitate the modular construction of many-core architectures, they provide communication abstractions and services across component boundaries and they enable the top-down design of highly power-manageable architectures.

As faults will appear with increasing probability due to the susceptibility of shrinking feature sizes to process variability, age-related degradation, crosstalk, and single-event upsets, designing efficient fault tolerant NoCs becomes a key requirement.

Transient faults cannot be handled by off-line strategies as they appear and disappear unpredictably. Fault-tolerant systems must be therefore employed to satisfy the high reliability constraints imposed by modern systems. In this direction, there are three major approaches.

First, Modular Redundancy can be adopted. For instance, Constantinides et al. demonstrated the BulletProof router, which efficiently uses N-modular redundancy (NMR) techniques for router level reliability [16]. However, NMR approaches are expensive, as they require at least N times the silicon area to implement. Additionally, network level reliability needs to be considered since some logic is impossible or expensive to duplicate (e.g., clock tree) or spares may run out, resulting in the loss of a router. Similarly, Time Redundancy (TR) [51] can be adopted to protect the NoC against faulty components. However, Time Redundancy decreases the performance of the NoC, since all information needs to be retransmitted.

Alternatively, error detecting codes can be used. The detection phase is followed by a recovery one, for instance based on the retry of the unsuccessful operation. Simple retransmission schemes are described in [18, 19]. In terms of implementation, [17,19] use a single transmission buffer that contains both sent and unsent flits together. [18] uses link-level retransmission together with the Unique Token Protocol (UTP) to ensure reliability. However, it requires at least two copies of a packet in the network, increasing buffer occupancy and flow control complexity. In contrast, [54] minimizes control logic by using a barrel shifter as retransmission buffer whose size is matched to the round trip notification latency of a NACK. The work in [37] targets virtual channel NoC implementations and uses dynamic packet fragmentation in tandem with a credit-based fault-tolerant flow control to recover from corrupted virtual channel states. In general, state-of-the-art in fault-tolerant flow control can be reviewed in [57], where the power inefficient ACK/NACK or the high-impact T-Error protocols are compared. The key take-away is that more research is needed in this domain, a challenge that for instance [37] takes on. Unfortunately, the solution in [37] comes with heavy throughput limitations.

Finally, error correcting codes (ECC) can be employed. They typically allow the correction of a limited amount of errors per codeword in order to contain complexity. Nonetheless, they are commonly reported to introduce a high timing penalty, because of the delay of the encoder/decoder and correction blocks. In [83] the router selects on the fly the most effective ECC scheme to send the data trough the link. The work in [23] proposes to use the Hamming Code on the input buffers to protect FIFO data. Similarly [21] protects the data-path via an ECC strategy. However, by using the fault tolerance techniques proposed in [21, 23, 83], but also in [63] and [62], only the links are protected, and incur large area and performance overhead. A retransmission scheme that enables graceful degradation of NoC communication performance under high failure rates is proposed in [40], but again the control path is not protected. Interestingly, erroneous behaviour in the functionality of the routing process or in output port arbitration may cause flit/packet misrouting. In the worst case, this results into loss of information or even into a deadlock condition. Clearly, robust protection against such upsets should be provided. Finally, [15] adopts error correcting coding to perform on-line testing but achieves a quite low coverage (63%). [54] proposes a mechanism able to exploit an ECC strategy for single error correction and a retransmission procedure once a double error is revealed. Anyway, the area overhead to support such mechanism is really severe, in addition to the high switching activity suffered from the retransmission buffer.

In general, the use of ECC in the above approaches suffers from two main limitations. First, the corrector is used at each clock cycle and ends up in the critical path. Second, the proposed architectures are not robust to many transient faults affecting the corrector itself. Moreover, there is consensus on the fact that error detection followed by retransmission typically has a milder impact on network power than error correction. This is the assumption of the work in [40] and the result of an ad-hoc experimental framework in [10] and [50]. However, none of these works relies on accurate microarchitectural studies and on physical implementation efforts, which are instead the key contributions of this thesis, in combination with the switch-level integration of design methods for fault-tolerance.

On the other hand, the capability of surviving permanent faults in the network closely depends on the flexibility of the routing framework, including both a reprogrammable routing mechanism [45], a fault-tolerant routing algorithm [29] and a suitable combination thereof [60]. Most of the proposals for fault-tolerant routing algorithms proposed so far for NoCs are based on programmable routing tables, storing the target output port to be taken by a packet for a given destination [53]. Unfortunately, table-based routing suffers from scalability limitations [61]. Above all, most fault tolerant routing frameworks do not address the testing problem: they just assume that faults are detected somehow on the fly and notified to the reconfiguration infrastructure [68].

The open literature offers a plethora of works tackling the problem of fault tolerance in the NoC domain. Unfortunately, most of them utilize distributed approaches which are inherently weak since they are deadlock prone or require flooding operations as well as the need to track several network status information thus calling for additional buffering resources. Zhang et al in [85] propose a topology reconfiguration strategy which is based on the concept of virtual topology. However, no discussion is provided regarding the reconfiguration mechanism of the entire system. In [33], a heuristic search algorithm for re-routing in on-chip network is presented. Unfortunately, the algorithm can not guarantee that every generated configuration is valid, therefore, a manual check is required. Authors in [2] present another rerouting algorithm which unfortunately does not guarantee 100% deadlock freedom, requires flooding strategies and increases the transmission time due to ping-pong re-routing. Furthermore, such an approach demands for considerable buffering resources in order to keep a history of all the packets traversing a switch. Configurable routing has been proposed in several forms, e.g., in [5], the authors propose a custom methodology, based on packet rerouting, to handle data transfers upon power management events or system faults. However, their approach is not general and is actually working around faults in the attached cores, not in the NoC itself. Deterministic routing is characterized by its simplicity and minimal overhead; it can be easily configured to avoid deadlocks [70] and natively guarantees in-order delivery. Unfortunately, deterministic routing does not adjust to the system evolution over time; on the contrary, dynamic routing has been proposed to achieve goals such as bypassing faulty nodes and minimizing congestion [3]. Dynamic routing needs decentralized decision processes and is therefore often achieved with dedicated logic in every router [9].

This thesis considers the common fault-tolerance strategies for SEUs (error correction, error detection and retransmission, triple modular redundancy) and provides the needed support for them to a baseline NoC switch for use in the embedded computing domain. The microarchitecture-level approach taken by this thesis brought two key novelties. On one hand, we could clearly identify missing gaps in the current landscape of fault-tolerant switches, namely fault-tolerant flow control and on-demand correction, and propose new solutions for them. On the other hand, we were able to contrast retransmission oriented vs. correction oriented techniques by capturing their global switch level implications (both data path and control path) in an architecture homogeneous experimental setting. Also, physical implementation analysis enabled to consider key second-order effects from logic synthesis, placement and routing. Overall, this thesis delivers practical insights for the provision of SEU tolerance to NoCs in a power-constrained environment.

In addition, this thesis moves from the perspective that fault detection and system reconfiguration can not be considered two separate tasks but they need to be co-designed with each-other and along with the underlying system from the ground up, and not as an afterthought. In fact, relevant interdependencies exist between the two frameworks:
(i) Testing and diagnosis granularity are dictated by the routing mechanism flexibility and by the hardware support for reliability in the network.
(ii) Similarly, the amount of control information that needs to be reprogrammed upon fault detection depends on the routing mechanism and on the way it exposes flexibility.
(iii) The place where responses to test patterns are analyzed and diagnosis information are computed has to be matched to the place where the course of action for network reconfiguration is taken. As an example, given a distributed built-in self-test (BIST) approach to NoC testing, failure patterns could be either notified to a global controller or directly used by a local mechanism for distributed reconfiguration. The system-level architectural implications in the two cases are radically different.

This thesis aims at overcoming the limitations stemming from the isolated development of testing and configuration strategies by **co-designing the two strategies together for better optimization** and by addressing the

system-level implications of implementing the entire fault-diagnosis and configuration procedure.

In particular, the contribution of this chapter can be detailed as follows:

- the **complete hardware infrastructure** for NoC fault-tolerance and configuration. We start from a baseline switch architecture featuring reconfigurable routing logic and built-in self-testing procedure, and we provide additional hardware support, spanning from three alternative solution for tackling SEUs up to the global infrastructure for control signaling. Above all, this thesis shows how all these components can be co-designed and specialized in the presence of a **logic-based distributed routing (LBDR) mechanism**, providing much better scalability to nanoscale technologies than forwarding tables from an area, power and delay viewpoint.
- the **global configuration strategy** orchestrating the operation of the hardware components together while at the same time implementing a mix of fault-tolerance and online testing strategies ensuring **reliable communication** between testing/diagnosis logic with the reconfiguration manager and viceversa. Primary objective of the design choices is to avoid flagging a corrupted NoC switch as operational because of faults in the configuration infrastructure. We opt for a centralized configuration mechanism where a global controller with full visibility of the network state is in charge of computing reconfiguration information for the routing mechanism in the switches.
- we provide **accurate characterization** of quality metrics of the proposed system infrastructure, which are typically overlooked in previous work or provided with lower accuracy (e.g., incomplete analysis, timing constraints for synthesis omitted) or under oversimplifying assumptions (e.g., single stuck-at faults). In particular, we assess incremental area overhead for BIST, diagnosis and for reconfiguration in an architecture-homogeneous experimental setting. Also, we perform fault simulation with a fault injector acting upon the gate level netlist of the configuration infrastructure and categorize the severity and occurrence probability of fault implications on the global configuration process.

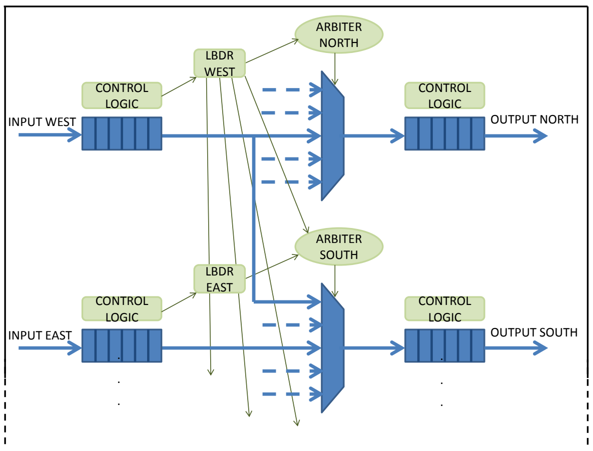


Figure 2.1: Baseline switch architecture. Not all connections are showed.

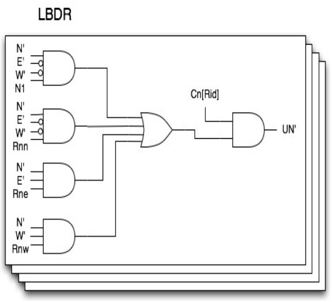
2 Background

The switch architecture proposed in this work is a major extension of the baseline ×pipesLite switch [71], illustrated in Figure 2.1, which targets the embedded computing domain with a very lightweight architecture. It implements both input and output buffering and relies on wormhole switching. The crossing latency is 1 cycle in the link and 1 cycle inside the switch, a design point suitable for the embedded computing domain. The switch relies on a STALL/GO flow control protocol (see section 4).
The switch architecture is divided into a control path and a data path. Input buffers, crossbar multiplexers and output buffers belong to the data path, while port-arbiters, routing modules and buffer control logic belong to the control path. The switch implements logic-based distributed routing (LBDR [59]): instead of relying on routing tables, each switch has simple combinational logic that computes target output ports from packet destinations. The support for different routing algorithms and topology shapes is achieved by means of 26 configuration bits for the routing mechanism of the switch.

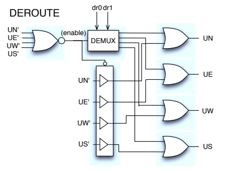
2.1 The LBDR routing mechanism

There are two main reasons leading us to select logic-based distributed routing (LBDR) as the routing mechanism of choice for our target NoC.

First, routing logic features better delay and area scalability with respect to forwarding tables, both with technology scaling and with increase of network size [61]. For small network instances, only register-based memory macro implementations are competitive, however they increase with the network size, while LBDR logic only grows with the switch radix [59].
Second, it reduces the amount of signaling between the NoC switches and the configuration manager, since the routing logic partially retains its flexibility by means of few configuration bits (namely routing R_{xy}, connectivity C_x and deroute bits dr). The number of these bits (26 in the LBDR variant of this thesis) is orders of magnitude smaller than the size of a forwarding table, yet makes the routing mechanism reconfigurable.



(a) Route computation logic.



(b) Deroute logic.

Figure 2.2: LBDR logic.

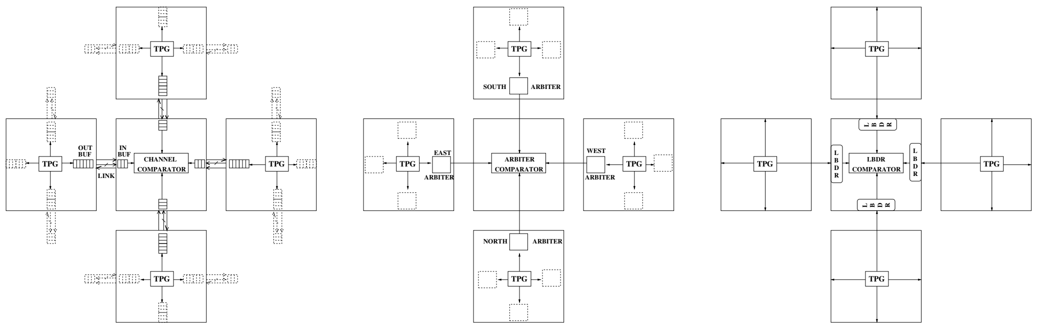
The core of LBDR logic is illustrated in Fig.2.2(a), showing the conditions that select the output port north UN' for packet routing. The routing decision is taken based on:

- the quadrant $N'/S'/W'/E'$ of packet destination (whose ID is embedded into the incoming packet);
- the routing restrictions posed by a deadlock-free routing algorithm (coded by the *routing bits*);
- the switch connections with the rest of the network (coded by the *connectivity bits*).

For some failure patterns, LBDR may not be able to find a route for the packet. In that case, a couple of additional *deroute bits* feed a logic which computes a valid output port to reach the destination. This extends the fault coverage of LBDR, which in [59] was proved able to work in roughly 60% of the irregular topologies derived from a 2D mesh. A larger coverage would require the NoC to revert to virtual cut-through switching and is therefore left for future work.
It is worth recalling that LBDR is a routing mechanism that supports the most widely used routing algorithms for irregular topologies, including segment-based SR [46] routing. As proved in [59], whenever a faulty 2D mesh topology can be handled by LBDR (including its deroute capability), it is always possible to find a suitable SR instance that can be used in combination with LBDR to route that topology.
Simply, a fault detection is equivalent to a change in the topology, and the routing, connectivity and deroute bits of all the switches have to be programmed from scratch or incrementally updated with respect to the original fault-free scenario. In [59], this is on burden of a configuration algorithm, which *needs the list of failed links to recompute the configuration bits for correct routing with the available communication resources.* Failure of a switch input or output port (and associated internal logic) can be viewed as the failure of the connected link.
Next section will illustrate the Built-In Self-Testing and Self-diagnosis (BIST/BISD) strategy which meets this requirement and *provides an indication of whether input and output ports of a switch are operational.*
Also, later on in the thesis it will be presented how to establish a reliable bidirectional path from the testing logic to the configuration manager and finally to the routing logic to be reprogrammed.

2.2 Built-In Self-Test/Diagnosis Framework

The key idea of our "Built-in Self-Test/Built-in Self-Diagnosis" (BIST/BISD) framework consists of exploiting the inherent structural redundancy of an on-chip network. We opt for testing the NoC switches in parallel, thus making test application time independent of the network size. Communication channels between switches are tested as a part of the switch testing framework.



(a) Testing communication channels.

(b) Testing output port arbiters.

(c) Testing LBDR routing logic.

Figure 2.3: The cooperative testing framework saving TPG instances and covering their faults.

Each switch can in turn test its several internal instances of the same sub-blocks (crossbar muxes, communication channels, port arbiters, routing modules) concurrently. In fact, all the instances are assumed to be identical, therefore they should output the same results if there is no fault. As a consequence, the test responses from these instances are fed to a comparator tree. This makes the successive diagnosis much easier. There is a unique test pattern generator (TPG) for all the instances of the same block, thus cutting down on the number of TPGs. Although the principle is similar to what has been proposed in [4, 6, 31], there is a fundamental difference. If the TPG of a set of block instances is affected by a fault, then the comparison logic will not be able to capture this since all instances provide the same wrong response. To avoid this, a cooperative framework is devised, such that each switch tests the block instances of its neighboring switches [73].

As an example, a switch tests the incoming communication channels from its north/south/west/east neighbors (i.e., it feeds their test responses to its local comparator tree), thus checking the responses to distinct instances of the same TPG. This way, a non-null coverage of TPG faults becomes feasible. Fig.2.3(a) clearly illustrates the cooperative testing framework for communication channels and the need for a single TPG instance per switch to feed test patterns to all of its output ports. Faults in the TPG, in the output buffer, in the link and in the input buffer will be revealed in the downstream switch. Each switch ends up testing its input links, while its output links will be tested by their respective downstream switches.

The same principle can be applied for the testing of switch internal block instances associated with each output port: crossbar muxes and output port arbiters. Fig.2.3(b) shows the case of port arbiters. The main requirement for testing these instances is that the communication channels bringing test responses to the comparators in the downstream switches are working correctly. Clearly, testing these modules can only occur after communication channels have been tested. Therefore, the procedures in Fig.2.3(a) and Fig.2.3(b) occur sequentially in time. Should one communication channel result defective, this would not be a problem, since it would not make any sense to test and use a port arbiter when the corresponding port is not operational. Crossbar multiplexers associated with each output port are tested in the same way and are hereafter not illustrated in Fig.2.3 for lack of space.

Inspired by such cooperative approach applied to the testing of channels and arbiters, we now extend such principle also to test block instances associated with each switch input port with some modifications. This is the case of the LBDR routing block. The key idea to preserve the benefits of cooperative and concurrent testing is to carry test patterns rather than test responses over the communication channels to neighboring switches, where the LBDR instances are stimulated and their responses compared (see Fig.2.3(c)). If the channel is not working, testing and using the downstream routing block is useless, since it is associated with an input port that will not be used.

On a cycle by cycle basis, comparator outputs are fed to a diagnosis logic which identifies where exactly the fault occurred. In our diagnosis framework, each switch checks whether test responses from its input ports are correct or not. As a consequence, the outcome of the diagnosis is coded in only 5 bits, one for each input port of the current switch (they would be of course doubled if a two-rail code is implemented to protect them against stuck-at faults). A '1' indicates that the port is faulty. In practice, the fault may be located either in the input buffer or in the LBDR module, in the connected communication link or even in the output buffer and associated port arbiter and crossbar multiplexer of the upstream switch. This further level of detail is not needed, since in any case the key take-away is that the link is unusable, and this is enough for a global controller to recompute the configuration bits for the LBDR mechanism. In the final implementation, other 5 bits will be needed to code the diagnosis outcome because of practical implementation issues associated with network flow control. Further details on this can be found in [73].

After the testing procedure has been completed, 10 bits are produced by each switch and have to be fed to the actual configuration infrastructure. Next section motivates the choice of our centralized approach and details the architecture of the configuration infrastructure, which is centered around a dual network for signaling of control information between switches and the controller and vice versa (see Fig 2.9).

3 Fault-Tolerant Architectures Under Test

This thesis considers and compares, at the microarchitecture and post place & route levels, three main approaches for SEU tolerance.

The three improved versions analyzed in this thesis are:

(1) Triple Modular Redundancy (TMR). In this architecture, links as well as switch data lanes are triplicated (Fig.2.4). As regards its control path, voting is performed for the registers of every finite state machine, to prevent a transient fault from misaligning their states. The TMR architecture can afford using the native STALL/GO flow control. This is the reference solution for fault-tolerance used for the sake of comparison.

(2) Error Correcting Switch Architecture. This solution is illustrated in Figure 2.5. Error correctors are used at every clock cycle both at the level of switch-to-switch communications and for intra-router data path operation. Concerning the control path, TMR is applied both to buffer FSMs and to routing logic/arbiters. Since correctors are enabled at every clock cycle and lie on the switch critical path, they introduce severe switching activity and delay penalty. In contrast, solutions based on ECCs are typically advocated by those designers who cannot stand the retransmission latency in spite of its low occurrence probability.

(3) NACK-GO Switch Architecture. In this retransmission-oriented solution, a fault-tolerant flow control protocol (NACK/GO) is used on the data path

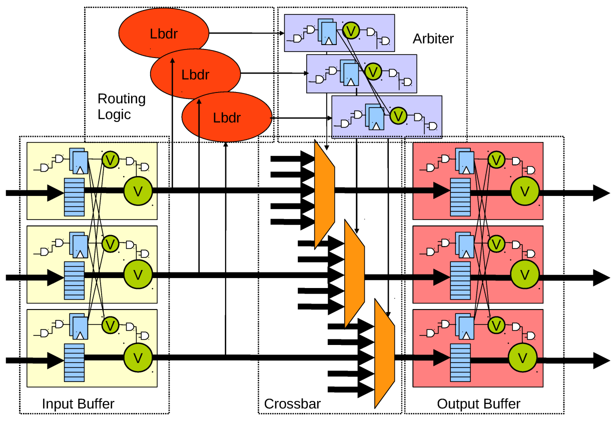


Figure 2.4: The TMR optimized switch.

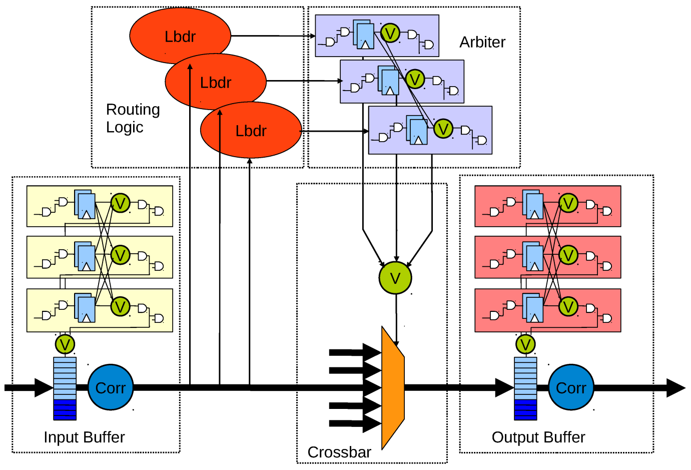


Figure 2.5: The optimized switch for error correction.

to notify error detection and trigger link-level data retransmissions, and also on the internal switch data path, to ask for data retransmissions from switch input to output buffers. The switch architecture is illustrated in Fig.2.6. Interestingly, on-demand correctors are used to repair corrupted values in the source buffers, which a retransmission would not fix. Thanks to the retransmission capability of the data path, the control path can implement a simpler dual-modular redundancy: in case of differences between the replicated paths, the current transfer is invalidated and a retransmission is required in the next cycle (see section 4.3). Next, the NACK/GO protocol is derived, and later on the control path is detailed accordingly.

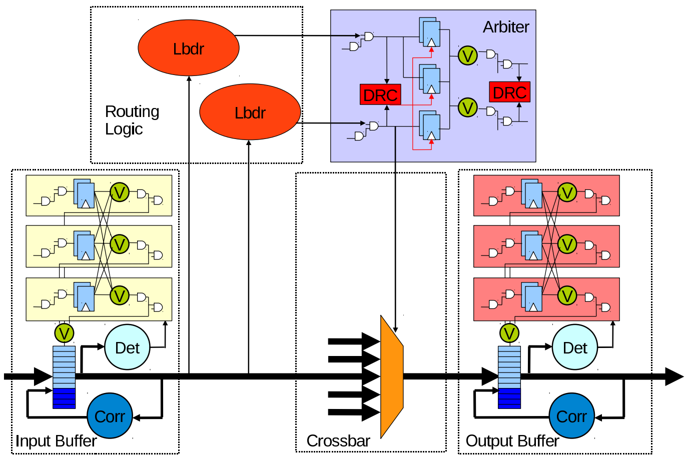


Figure 2.6: NACK/GO switch architecture.

4 The New Fault-Tolerant Flow Control

4.1 NACK/GO

STALL/GO is one of the simplest flow control protocols that can be found in the open literature. It leverages only one forward signal, that flags the availability of new valid data (Valid signal) and one backward signal, used to stop the communication flow when a new flit cannot be accepted due to congestion in the downstream node (Stall signal). Conversely, ACK/NACK is a flow control protocol with error detection/notification capabilities. It exploits a Go-back-N policy to manage and control correctness of the transmitted data. When an error is detected in a transmitted flit, the receiver signals this event to the sender, who will retransmit the flit with the corrupted information and all the (N) successive ones.
Unfortunately, the ACK/NACK protocol does not make a clear distinction between the backpressure phenomenon and the occurrence/detection of transient faults. As a consequence, a nack received by the upstream switch means that the flit should be retransmitted for some reason. In case of congestion of downstream paths, the protocol keeps retransmitting the same flit indefinitely regardless of the receiver state, thus proving power-inefficient. On the contrary when a STALL/GO protocol is considered then transmission freezes until a go arrives, in case a stall notification is received upstream. This protocol is much more power efficient but does not provide any kind of support

for fault tolerance and data retransmission.

Augmenting the protocol in this direction was one key objective of this thesis, thus coming up with the **NACK/GO flow control protocol**. NACK/GO is a new protocol, and associated switch implementation, that offers full error detection and notification capabilities of ACK/NACK while preserving the power efficiency of STALL/GO for error-free operation.

NACK/GO leverages four control signals to control transmission of data and achieve fault tolerance. There are two signals going in the same direction of the data stream, and two backward propagating signals.

Valid: this signal flags availability of new data, and it triggers the data transfer.

Trash: this control signal notifies that the data currently being transmitted is corrupted and should be discarded. The reason why the valid signal is not used for this is to optimize the internal critical path of the switch and avoid multi-cycle switch traversal.

Stall: it stalls the transmission in case of traffic congestion. The Stall signal stops the communication flow, freezing all the forwarding control signals and data to their current value.

Nack: Nack signal is de-asserted low when a valid flit has been received in the previous clock cycle (Acknowledgement). In contrast, Nack is asserted high whenever no valid flit is received, either because no transmission took place or because the accepted flit is detected as corrupted.

NACK/GO combines the best of STALL/GO and ACK/NACK.

Like STALL/GO, it exploits an efficient methodology to block the communication traffic in case of congestion, avoiding the unnecessary switching activity for flit retransmission as in ACK/NACK. It uses a signal to notify the availability of free buffer positions inside the receiver, so communication can be frozen when congestion occurs. Furthermore, a Stall signal avoids the roundtrip necessary to resume communication from the packet that was not accepted like in ACK/NACK, leading to a better average performance.

In addition, it features the error flagging capability not exposed by STALL/GO. In this way the system can re-establish a correct working point and resume its correct operating condition.

These advantages come at the cost of extra channel wiring, since a total

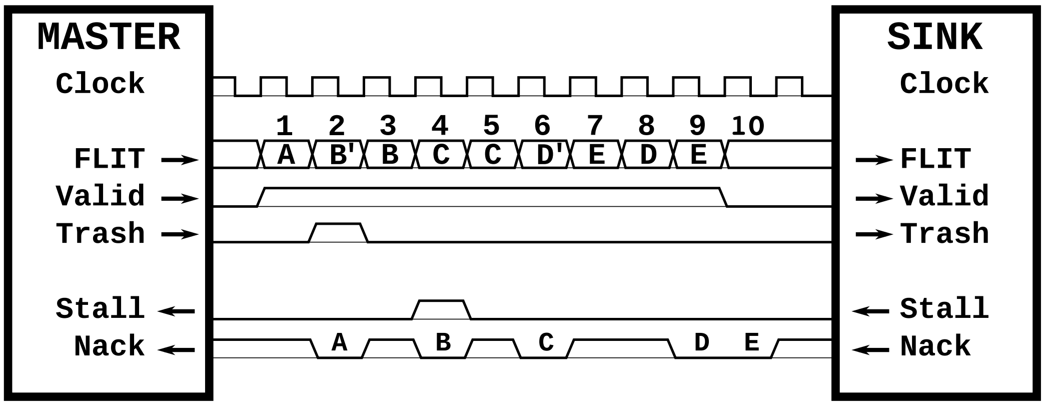


Figure 2.7: NACK/GO Flow control operation

of 4 control signals are needed, rather than the 2 of STALL/GO, and a more complex control logic inside the buffers and the switches. NACK/GO features worse minimum buffer slot requirements than STALL/GO, since it requires a minimum of 3 buffer slots. In addition, every pipeline stage inside the link should have not only flow control capabilities, but also error detection/correction capabilities. For this reason, every repeater must have at least three buffer positions, more than the two required for STALL/GO.

4.2 NACK/GO Operating Principle

In order to describe the details of the NACK/GO operating principle, this section will suppose a communication taking place between an upstream node M (master) and a destination node S (sink), as illustrated in Fig.2.7. The master node sends 5 data flits, named from A to E.

Clock cycle 1 The source node M has a new data available. It will begin the transfer by flagging the possibility to send a new flit, A, asserting the Valid control signal high. The destination node S will communicate its availability to receive the incoming transmission by asserting the Stall backward control signal low. In such a configuration, the flit A is successfully transferred from Master to Sink. It is however not possible to discard it in the Master node M yet, since it is still waiting for the acknowledgement from the destination that will take place in the next clock cycle.

Clock cycle 2 The source node tries to send the next flit available, B. Suppose that while transmitting, the sender detects an error inside B (i.e., it has received an incorrect flit from its upstream node, which is speculatively sent to S while performing error detection in parallel). The sender will notify that the ongoing flit is corrupted with the Trash signal in case of misspeculation. The receiver S will receive the B flit, but this will not be stored. The transmission of the flit is discarded, and the source will start the appropriate procedure to recover and re-establish the correctness of flit B before transmitting it again. In the meantime, destination S will check the A flit, received in the previous clock cycle. Since no error is detected, the Nack signal is de-asserted low.

Clock cycle 3 After an arbitrary number of clock cycles (not showed here), in which the Valid signal has been de-asserted low, Master has a new version of B, and can transmit it. It signals the transmission using Valid signal, and the Sink is able to accept it. Nack is asserted high, because during the previous clock cycle nothing has been received.

Clock cycle 4 After storing B, the destination's buffer is now full. The next coming flit cannot be stored before a new communication takes place between S and its downstream node. The Master will try to transfer the successive flit of the sequence, C, but will face the Stall signal controlled by the destination. In such a situation, M will keep the actual configuration until the Stall signal is unset. Concurrently, the received B flit is signaled as correct.

Clock cycle 5 When S has free space in its buffer memory, it will de-assert Stall signal and M will be able to complete the transfer of C.

Clock cycle 6 Master transfers the following flit, D. Let us suppose that something went wrong during the communication. Sink will store it, but the data received will be D', not a valid data word.

Clock cycle 7 After a redundancy check, the receiver signals to the transmitter that the previous flit transmission has been erroneous. In order to notify such event the Nack signal is asserted high. Since anyway any other flit following the corrupted one must be retransmitted, the current ongoing data flit, E, is discarded.

Clock cycle 8 The Sink module will be waiting for the transmitter to reply with the same flit signaled as corrupted. Nack signal will remain asserted high until a correct flit will be received. In the meanwhile, since a Nack signal has been received after a valid flit transmission, the transmitter will take an arbitrary number of clock cycles in order to recover the flit that failed to be correctly transferred. Such number of clock cycles can even be 0, if the adopted policy is not to correct the flit at all but just to retransmit it. This depends on the degree of protection against transient faults. In fact, if the fault occurs on the link, a simple retransmission is enough. Vice versa, if the stored value in the switch buffer has been changed by a SEU, such change is irreversible and only correction can restore the original value, while retransmissions will result only into the same incorrect receipt of data. These are choices that the architecture designer has to take while implementing NACK/GO communication protocol in an actual network. The next valid transmission will be a reply to the error condition previously flagged.

Clock cycle 9 Now, the data that had been transmitted while the Nack signal was asserted high is transmitted again. D flit is signaled as correctly received.

Clock cycle 10 Finally, the last flit transmitted is acknowledged by the receiver.

4.3 Novel Low-Power Fault-Tolerant Arbiter

By exploiting the retransmission capability provided by NACK/GO, we designed the fault-tolerant control path that supports NACK/GO flow control operation.

The novel fault-tolerant arbiter was designed following Figure 2.8. It represents an effective variant of a baseline TMR arbiter (like for the error correcting switch).

As showed in Figure 2.8, the *transition logic*, representing the arbiter combinational logic for computing the next FSM state, is doubled. Therefore, the outputs of the two *transition logic* instances are compared in a Two-Rail Checker (TRC) module (i.e. a redundant comparator block with fault tolerance capability) and feed the state memory register of the arbiter. The state memory register is triplicated as in a conventional TMR strategy although it is enabled by the TRC module output. When the two *transition logic* blocks generate the same result, then they are not affected by an error and the state memory register can sample the new state. On the other hand, the TRC block freezes the state of the arbiter registers when the *transition*

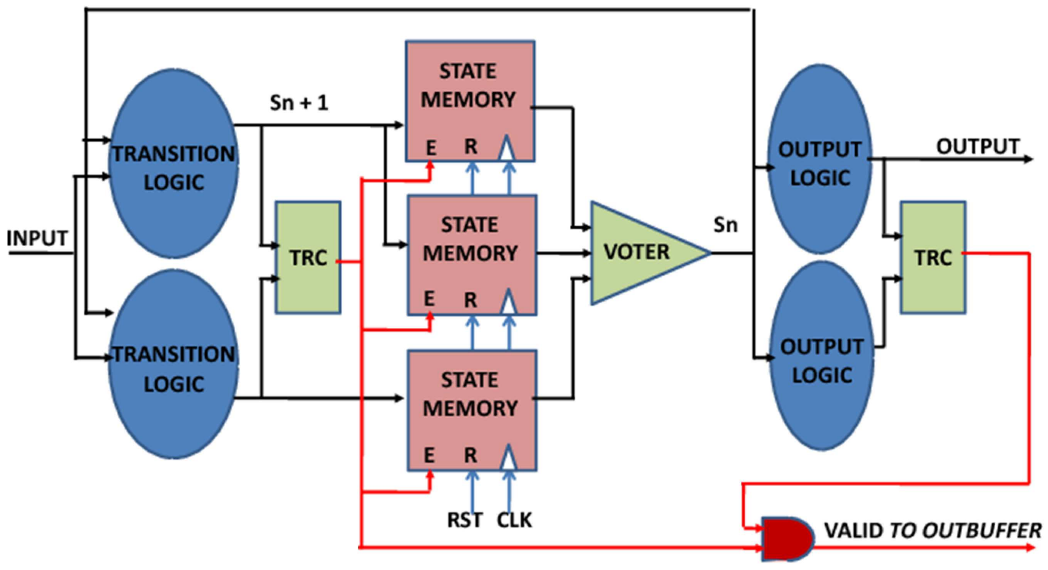


Figure 2.8: Fault tolerant arbiter implementation.

logic blocks provide different results. In this latter case, the *valid* signal to the output port is deasserted. As a consequence, the output port will not read the incoming information affected by errors, and since no valid data has been stored in the current clock cycle, the Nack signal will be asserted high in the following clock cycle. This will be interpreted by the input buffer as a request of retransmission.

Thus, the outputs of the three state memory registers are voted before feeding two instances of the output combinational logic with the arbiter current state. Then, the outputs of the combinational logic modules are compared in a TRC comparator following the previous implementation adopted for the *transition logic* modules. Finally, when the above mentioned comparator reveals an error, then the *valid* signal to the output port is deasserted and the information coming from the output combinational logic is discarded.

The baseline behavior of the arbiter remains the same: it selects between different inputs competing for the same output port, operating with a round robin arbitration policy in order to enforce fairness between all the requests. New conditions and events had to be managed in order to cope with error detection: whenever an error is detected in the output buffer, the Nack signal must be routed to the correct destination. If this takes place on the head flit of a packet, that stores the information about destination, in order to avoid misrouting the current source will lose the grant acquired by the arbiter. In the following clock cycle, the route will be re-computed and the arbitration process will take place again. On the other hand, when an error is signaled for the last flit of a packet (tail flit), the sender has already lost the grant given by the arbiter and has no longer exclusive access to the output port. For this reason, the arbiter does not consider any new request, and the sender that had the grant in the previous clock cycle acquires again access to the output port. In this way the correct status of the various actors is re-established, and the corrected flit can now be retransmitted. Lastly, when a flit is signaled as corrupted while being transmitted (Trash signal asserted high), the arbiter must propagate this condition to the output port, and ignore the current transaction taking place. The trash signal is forwarded to the output buffer, that will ignore the current incoming flit.

4.4 Fault-Tolerance of Routing logic and Buffer FSMs

In order to protect routing logic, two LBDR replicas directly feed the combinational logic cascaded to the TRC of the arbiter. A failure in the LBDR logic will be tackled by the arbiter's Two Rail Checkers, thus exploiting the cooperation with the arbiter to achieve fault-tolerance by means of an effective lightweight solution.

The status registers of buffer FSMs need additional protection. For this purpose, the simplest thing was to implement TMR in light of the low-complexity of replicated circuits. This solution is the same adopted for the error correction switch.

Figure 2.6 depicts the final NACK/GO switch architecture.

5 Integration with Network-Level Fault Tolerance

Although the above architectures have been primarily designed to tackle SEUs, they are both capable of handling intermittent faults.

Some physical effects such as wear-out end up in permanent faults, but 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 global controller (e.g., via the dual bus in [27]) which may monitor the distribution and frequency of transient faults over time and eventually take the proper course of recovery action. Exactly the same policy can be supported by the error correcting switch by notifying correction and/or voting events to the controller.

6 Global Strategy for Self-Configuration

A distributed configuration strategy avoids single points of failure at the cost of significant resource over-provisioning. In fact, complex communication extensions are typically needed both for baseline operation of the configuration logic (additional signaling mechanisms with neighboring switches, complex routing computation or table update logic which may potentially render the switch multi-cycle, rule checking mechanisms for deadlock avoidance) and to improve the percentage of supported fault patterns (e.g., broadcasting, virtual channels, hardware timeouts, large storage or combinations thereof). The main reason lies in the lack of visibility of global network state at a NoC switch. Therefore, this latter has to take the proper course of action to become aware of the state of neighboring nodes, to find a routing path to every network destination and to avoid deadlock and livelock. This is a lengthy process that partially offsets the benefits of distributed configuration. Even accepting this, final routing solutions cannot be always guaranteed to be completely deadlock-free and frequently incur mis-routing [2].
The suitability of these requirements for the embedded computing domain is questionable. Therefore, the investment of our work is on a centralized approach, which envisions a global controller with full visibility of the network state and that is able to reprogram the routing mechanism of the switches accordingly.
The previous section has already showed that minimizing signaling needs from the network to the controller and vice versa is an orthogonal concern

across all layers of the reconfiguration framework, starting from the routing mechanism up to the optimized design of the signaling architecture.

Above all, the main challenge we tackle in this thesis is to carry vital diagnosis and configuration information from the network to the controller and vice versa in a reliable way, since manufacturing yield is tightly related to the reliability of these communications. If we use the data network for such a critical information exchange, the intricacy is again associated with the reduced view each switch has of the network failure pattern, which tends to over-provision the NoC architecture (e.g., virtual channels, flooding capability) or to uncover many failure combinations. Alternatively, the controller might discover by itself the failure pattern by means of a lengthy and non-trivial process starting from neighboring nodes and through them reaching the most remote nodes. Finally, enforcing fault-tolerance of the whole data network for the sake of safe configuration is overly expensive area- and performance-wise, even when more advanced techniques than NMR (N-modular redundancy) are used [21].

The above reasons justify the use of a dedicated dual-network for control signaling. The network is composed by a set of replicated *routing primitives*, each one associated with, and connected to, a switch of the main data NoC. Unlike the full featured and richly connected main NoC, the dual NoC implements a straightforward ring topology where the routing primitives are simply cascaded. The global controller closes the ring, as illustrated in Fig.2.9.

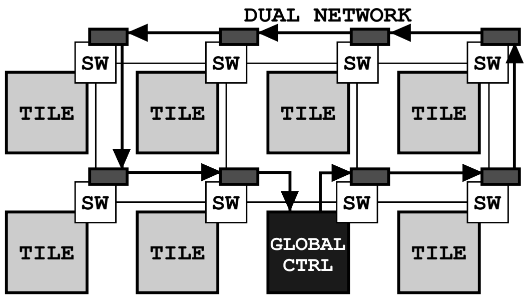


Figure 2.9: Topology of the dual network.

Overall, the dual network has to carry 10 diagnosis bits to the controller from every switch, and to get back 26 routing configuration bits to them. These relatively small requirements leave ample room for designing the dual

network aggressively for fault tolerance while marginally impacting overall NoC area footprint (see section 7).
The operating principle of our configuration strategy is as follows (see Fig.2.10): at system bootstrap, after performing the BIST/BISD phase, each switch has to communicate the result of its diagnosis to the global controller through the dual network. However, in order to make sure that no transfer errors occurred during this communication (unmasked by the fault-tolerance mechanism), a specific online testing protocol is implemented [1].

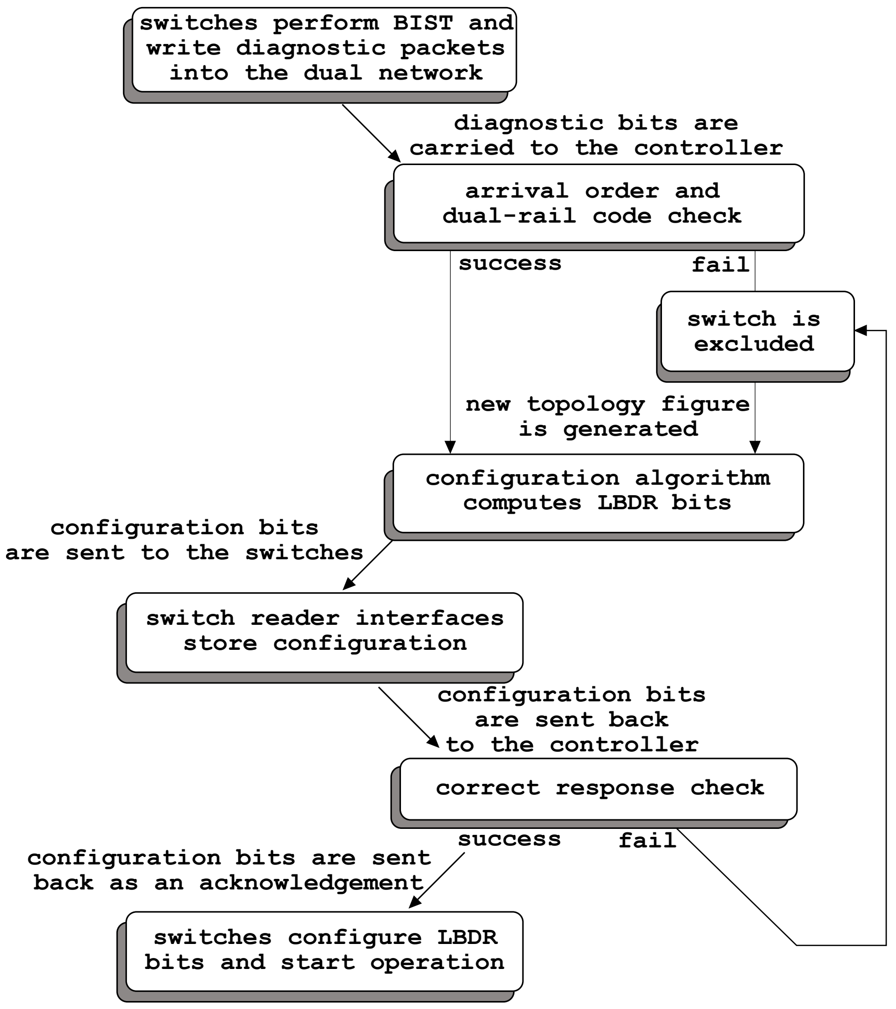

switches perform BIST and write diagnostic packets into the dual network
diagnostic bits are carried to the controller
arrival order and dual-rail code check
success
fail
switch is excluded
new topology figure is generated
configuration algorithm computes LBDR bits
configuration bits are sent to the switches
switch reader interfaces store configuration
configuration bits are sent back to the controller
correct response check
success
fail
configuration bits are sent back as an acknowledgement
switches configure LBDR bits and start operation

Figure 2.10: Configuration strategy at glance.

In particular, the *writer interface* of each switch (into its associated routing

[1]The word "online" here is relative to the configuration process.

primitive) sends the diagnostic bits by leveraging a two-rail encoding scheme in time. This way, the global controller can check whether an unmasked fault has corrupted the diagnosis information of a specific switch. In case, the switch is considered to be unusable and left out of the configuration process. Also, the arrival order of diagnostic bits is deterministic, therefore the controller can easily realize whether a switch is so corrupted not to be able to transmit. At the end of this initial signaling phase, the new topology connectivity pattern is derived, since diagnosis bits indicate which links/switch ports are faulty. Therefore, a configuration algorithm can be executed, thus computing configuration bits for the LBDR logic of each switch. These bits are then sent to all the switches via the dual network. Based on our testing protocol, switches send back the received bits to the controller, so that this latter can check whether the transmission was successful. If transmitted and received data match, then the controller sends them again to the switches as an acknowledgment. Upon the arrival of the same configuration bits twice, the switches then configure LBDR bits and trigger switch operation. This is done by deactivating the flow control stall signal to neighboring switches, which prevents to store incoming data until the configuration phase completes. This is the same signal used for normal flow control in the main NoC at regime.

The rationale behind the three-way handshake protocol is that, the *reader interface* of the switch from the routing primitive needs to be tested as well in order to make sure that it is able to correctly receive data and reconfigure the routing mechanism accordingly.

Now, let us assume that the controller detects that a switch is not able to correctly receive configuration bits. Then, the switch is excluded and the configuration algorithm recomputes routing, connectivity and deroute bits *selectively* for the neighboring switches, thus accounting for the new change of the actual topology. Other switches are not involved in this signaling round and are kept temporarily blocked. Affected switches in turn receive a different set of configuration bits than before, therefore they send them back again to the controller to check correct receipt. The procedure is iterated until no mismatch is detected by the controller: at that time, the acknowledgment is sent to all switches, so that they start operation concurrently.

Please note that unmasked stuck-at faults in the dual network are not necessarily incompatible with its correct operation. In fact, should they match by chance the configuration bit transmitted on that bit, the configuration works correctly. Every time the configuration bits are changed, they might reveal the stuck-at bit, but in this case the three-way handshake enables to detect this at the controller and to conservatively discard the affected switch.
Our configuration strategy aims at correctly configuring switches of the main NoC even in the presence of faults in the dual network and prefers to discard those switches that cannot either send diagnostic bits or receive configuration bits correctly because of unmasked faults by the fault-tolerance mechanisms. We want to avoid flagging a switch or a link as operational when it is actually not working, since this would make network operation not trustworthy. We rather prefer to conservatively discard well-behaving components when their ability to correctly interact with the controller is uncertain. Next section will detail the architecture of the routing primitive.

7 Routing Primitive Architecture

From an architecture viewpoint (see Fig.2.11), the routing primitive resembles an oversimplified version of an input buffered clocked switch. An input decoder discriminates between packets destined to the controller (collision between these latter and packets originating from the local switch is solved by an allocator) and those destined to the local switch. All packets on the dual network have a fixed 3 flit length. Flit width is 15 bits. The input buffer has 2 slots (for stall/go flow control management) while the allocator implements a fixed priority algorithm in light of the operating principle of the dual network. This latter is as follows. After the BIST phase completes in the local switch, diagnostic bits are generated and forwarded to the routing primitive through the writer interface. Packets are 3 flits long. The header flit carries sender switch ID. The second flit carries diagnostic bits while the third one carries negated diagnostic bits. The controller transmits configuration bits again in 3 flits. The header contains target switch ID and flit type. The second and third flits contain LBDR bits. Whenever the primitive receives a configuration packet, the *reader* interface eliminates the header information

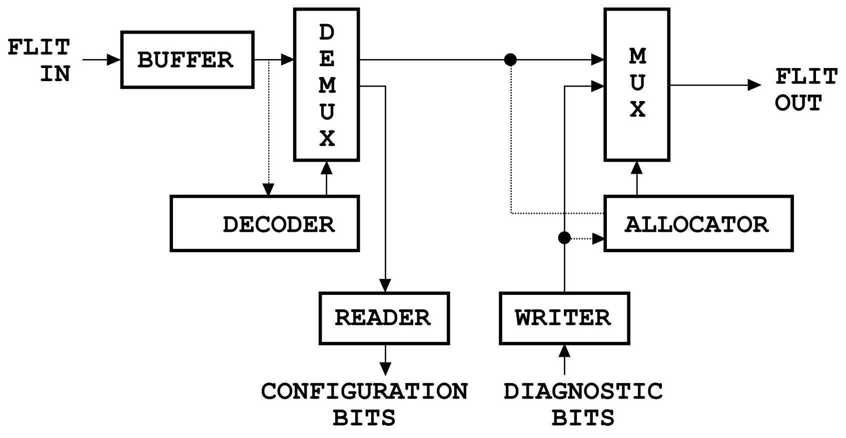


Figure 2.11: Dual network routing primitive.

and extracts the LBDR bits.

The dual network is a critical component for the correct configuration of the system. Therefore, we decided to provide fault-tolerance capability to it by means of triple modular redundancy (TMR). This latter has been preferred to information redundancy (e.g., error correcting codes, ECCs) for ease of design. In fact, TMR only requires the replication of the routing primitive in the dual network, whereas the use of ECCs (such as Hamming codes), although requiring less overhead, would imply the utilization of ad-hoc design techniques to avoid that signal fan-outs introduce errors that cannot be corrected.

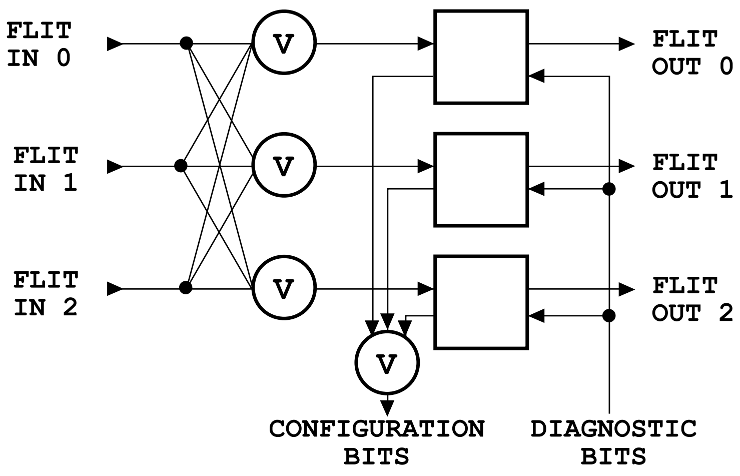


Figure 2.12: TMR approach with per primitive voting system.

The simplest approach to TMR implementation consists of replicating the dual-network three times and to vote it at the input of the controller interface (for switch-to-controller signaling). This approach, however, has a probability of failure that does not scale well with the size of the network. In this regards, let p_f be the probability that, because of faults, a single routing primitive is unable to deliver one input flit to the target output port uncorrupted. When TMR is applied with voting at the controller, the probability P_i that the i-th stage of the replicated dual network fails to correctly communicate with the controller is:

$$P_i = 1 - \text{ Prob}\{\text{at least two voter inputs are correct}\} = \\ 1 - \left(\binom{3}{2}\big((1-p_f)^i\big)^2\big(1-(1-p_f)^i\big) + \big((1-p_f)^i\big)^3\right) \quad (2.1)$$

where $(1 - p_f)^i$ denotes the probability that a voter input correctly receives the input of the i-th stage. In case $p_f \ll 1$, the above equation provides $P_i \simeq 6i^2p_f^2$. Of course, the presence of the term i^2 gives rise to problems in the presence of large values of i (the furthest away nodes from the controller). Also, since our dual network is not a one-way chain but a ring where configuration bits also flow, P_N (where N is the number of ring nodes) coincides with the failure probability of the ring as a whole. To solve this problem we decided to vote at the output of each stage of the dual network (see Fig. 2.12). In this scheme, in fact, P_i scales in a better way because it depends on i. In particular, let q_f be the probability that a routing primitive of the dual network or its output voter are faulty (therefore, $q_f > p_f$ because of the additional voter area). Therefore, the probability P_b that a triplicated primitive and associated output voters work is:

$$P_b = \text{Prob}\{\text{at least two voters outputs are correct}\} = \\ \left(\binom{3}{2}(1-q_f)^2(1-(1-q_f)) + (1-q_f)^3\right) \quad (2.2)$$

Finally, the probability that the i-th stage of the fault-tolerant dual network does not communicate with the controller is:

$$P_i = 1 - P_b^i \quad (2.3)$$

that if $q_f \ll 1$ becomes $P_i \simeq 3iq_f^2$, thus showing a linear dependency on i that scales better with the size of the dual network. Therefore, the scheme in Fig. 2.12 is adopted. It should be noted that also flow control signals (valid and stall/go) are voted. From the figure, voting of the incoming configuration bits is also apparent. Should a fault affect this voter, the online testing procedure is able to detect this, since transmitted configuration bits by the controller would not match the same bits that the switch sends back to the controller for a double check, and the switch would be discarded.

8 Experimental Results

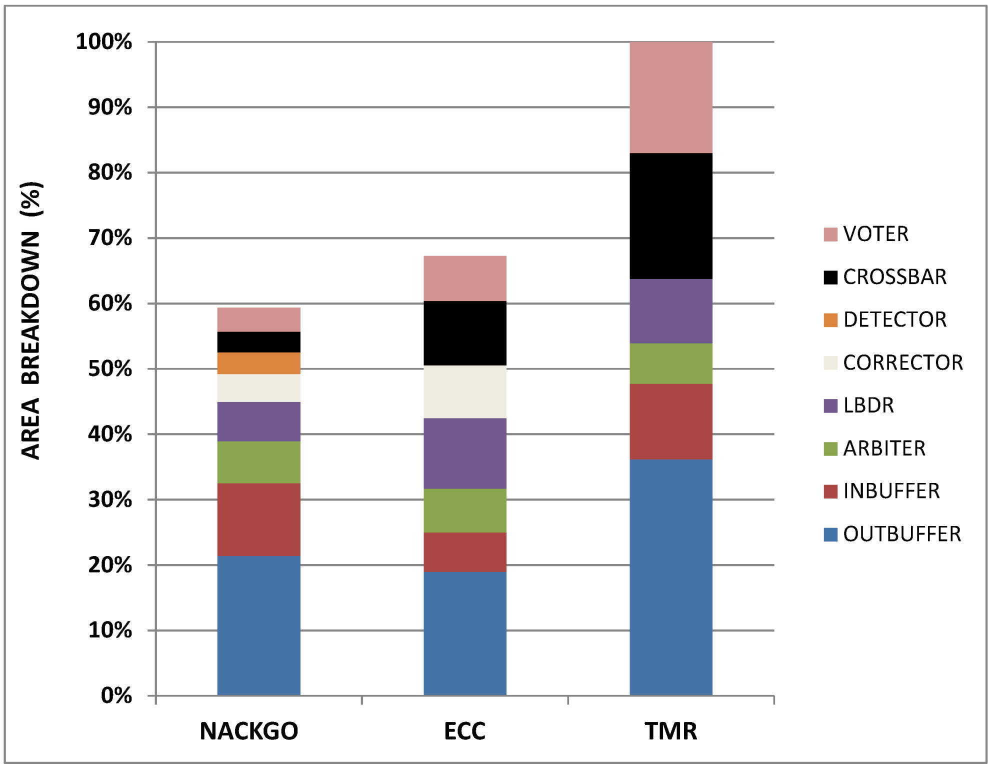


Figure 2.13: Area comparison between TMR, ECC and NACK/GO switch.

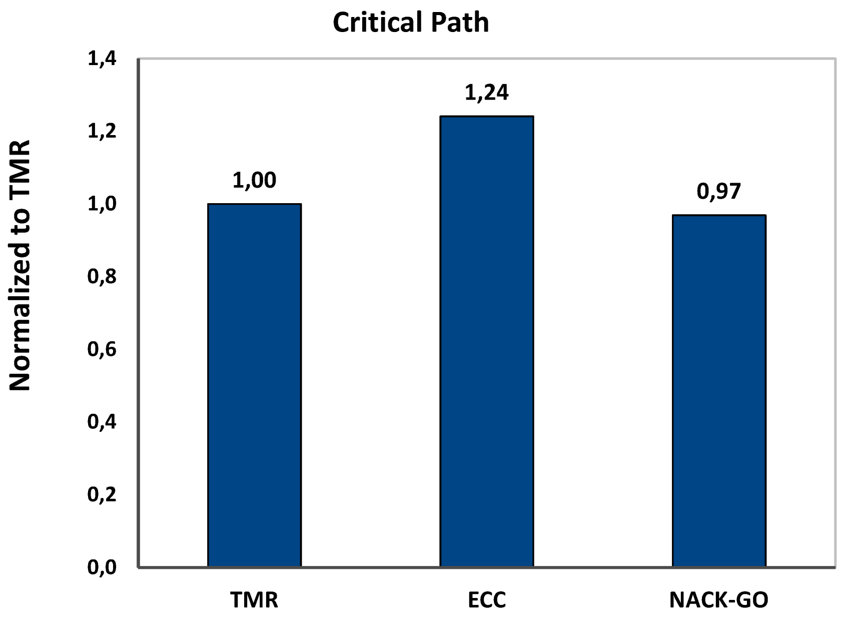


Figure 2.14: Critical path comparison between TMR, ECC and NACK/GO switch.

8.1 Area and Critical Path of the Fault-Tolerant Switch Architectures

In this section we illustrate the experimental results carried out for the three switches (TMR, NACK/GO, ECC) in terms of area, critical path and power consumption. All the analyzes discussed in this work have been carried out by means of a backend synthesis flow leveraging mainstream industrial tools. The technology library is a low-power low-Vth 65nm STMicroelectronics library.

The area results are showed in Figure 2.13, while critical path comparison is presented in Figure 2.14. Please note that all the results are normalized with respect to the plain TMR solution.

The area of the TMR switch is not only given by the various triplicated modules, but a non-negligible contribution comes from the numerous voters, instantiated not only in the control path inside the switch, but also for every bit of the data path of every input channel. In addition, due to the large area footprint, long wires affect performance of the component. Numerous electric buffers are instantiated during place&route to speed up transmission on internal paths, leading to additional area overhead. The result is a total area that is much more than triplicated (around 4.5x) with respect to the baseline solution, and a performance drop that exceeds the pure contribution of the voters on the critical path. These are very interesting second order effects that are typically overlooked in most literature surveys.

The version of the switch correcting at every clock cycle (ECC) greatly reduces the area footprint with respect to TMR, while almost balancing the area figures of the NACK-GO switch. Unfortunately, the corrector modules lie on the critical path and operate at every clock cycle, and not only upon fault detection as in the NACK-GO switch, so the performance is negatively affected. The critical path is 27% longer than the NACK-GO switch. The presented results refer to a single cycle switch. Please note that pipelining the correction phase would only speed up the throughput, but would negatively affect the latency making the switch multi-cycle. Furthermore it is important to note that this solution does not guarantee complete fault tolerance, because correctors are not immune against transient faults which may result in packet misrouting.

As far as the NACK/GO switch is concerned, a non-negligible area contribution comes from detector and corrector modules. These count for almost 13% of the total area. In many works in the literature targeting transient faults the problem of data corruption inside buffers is typically omitted, therefore the overhead of correctors in retransmission-oriented solutions is not accounted for.

When comparing NACK-GO with the ECC strategy, we can notice a similar area footprint for the output buffers while the ECC solution presents a lighter input buffer. This latter result is due to the oversizing of the NACK-GO input buffer. Indeed, a higher amount of slot buffers is required by NACK-GO with respect to STALL/GO. As regards the correctors, they bring a high area overhead to the ECC solution since they lie on the switch critical path. On the other hand, the NACK-GO switch has smaller correctors but it requires detectors not adopted by the ECC counterpart. As a consequence, the two switches require a similar total amount of area for correction and detection modules. Concerning the control path, the novel arbiter proposed in section 4.3 does not directly bring area benefit. In fact, the complexity

of the arbiter increases to support the NACK-GO protocol and offsets the area reduction achieved by avoiding the third arbiter replica. Anyway, the arbiter is the key enabler for further optimizations in both the LBDR, the crossbar and the voters. Indeed the NACK-GO LBDR is lighter than the ECC LBDR solution since the triplication of its logic is avoided by exploiting the retransmission mechanism introduced by the novel arbiter. Furthermore, voters are approximately cut by half into the NACK-GO switch because the novel arbiter is no longer requiring most of voters. Finally, the critical path between the arbiter and the crossbar is relaxed due to the voters removal, thus the crossbar results smaller. All together, the NACK/GO switch saves 12% of total area with respect to the ECC switch.
As regards the timing, despite the optimization done on the detector module as explained in section 4.2, the overall performance of the NACK/GO switch is limited by the additional complexity of buffer control logic and therefore matches TMR performance while clearly outperforming the ECC switch. In conclusion, the NACK/GO solution outperforms the two basic switch upgrades both from an area (59% area footprint with respect to the plain TMR solution) and performance (critical path lower by 22% with respect to the ECC switch) viewpoint.

8.2 Power Consumption

A number of experiments has been carried out to assess the power consumption of the NACK/GO switch and the two fault-tolerant counterparts (i.e. ECC and TMR).
The NACK/GO, TMR and ECC switches have been tested under different traffic patterns: idle, hotspot and parallel. Post-layout simulations have been carried out at 500MHz. All the results were grouped per traffic pattern. As we found out in previous experiments (in Section 8.1), the area overhead of the TMR solution comes along with a high power penalty. In fact, the TMR switch is the most power greedy under all possible traffic patterns, as showed in Fig. 2.15.
On the contrary, the NACK/GO and the ECC solution perform better than TMR although a small power penalty is paid by the NACK/GO switch with respect to the ECC counterpart. This penalty is mainly due to a larger

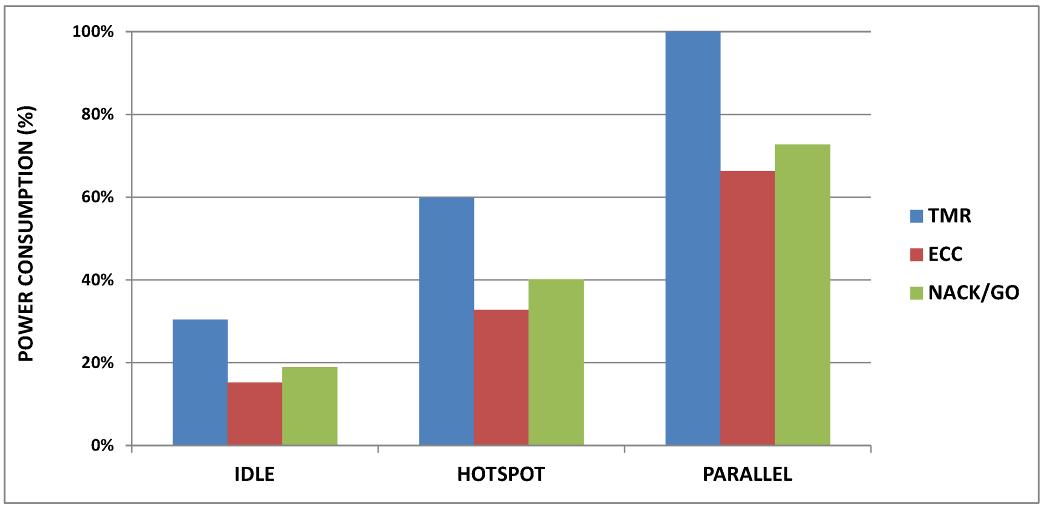


Figure 2.15: Power consumption for idle, parallel and hotspot traffic condition.

amount of buffering resources and the inherent complexity of the NACK/GO control logic which introduce power consumption overhead during the idle and the hotspot/parallel traffic conditions respectively.

Moreover, we estimated the peak power trend during detection/correction of failures. As a result, the ECC and the TMR solutions preserve the peak power results achieved in the failure-free scenario while the NACK/GO solution reduces peak power by 36%. This happens because, while the ECC and the TMR solutions enable correctors and voters each cycle, regardless of the actual failures, the NACK/GO switch requires a retransmission exclusively upon failure detection. In this latter case, the main part of the NACK/GO switch is stalled while only correctors are running.

8.3 Area Overhead of the Routing Primitive

A 5x5 NoC switch of the ×pipesLite architecture [71] has been synthesized with a 65nm industrial technology library in three variants: the baseline switch, the BIST-augmented switch and the BIST-augmented switch with a fault-tolerant routing primitive (like in Fig. 2.12). Two different target operating frequencies have been selected: 500 MHz and 700 MHz.

Area results are reported in Fig. 2.16. The area overhead of the routing primitive is 12.5% at 500 MHz and 12.8% at 700 MHz of the baseline switch, far better than replicating the entire primary network. Interestingly, total switch area with the fault-tolerant routing primitive is just +11.3% higher (at 500 MHz; +10.6% when operating at 700 MHz) than that of the BIST-augmented switch.

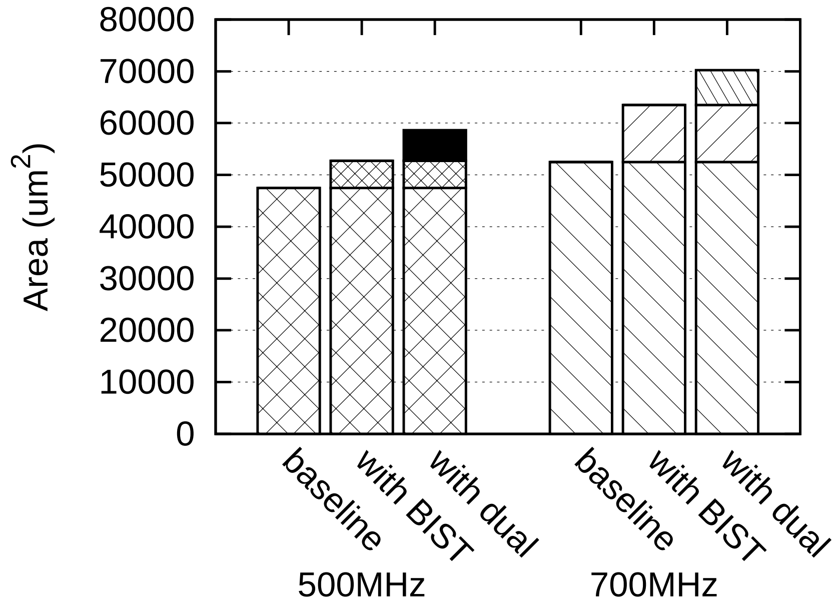


Figure 2.16: NoC configuration: area overhead.

8.4 Fault tolerance of the routing primitive

In what follows, the stuck-at fault model is assumed. Fault-tolerance of the TMR-augmented routing primitive was assessed by exhaustively injecting stuck-at faults in the gate level netlist. Then, test patterns generated in an exhaustive way were provided at the primary inputs to check response integrity. The stimulated input port depends on the working phase of the configuration strategy. Three working conditions were taken into account:
(i) transmission of diagnostic bits from the switch to the next element of the ring structure (communication toward the Global Controller);
(ii) generic transmission from the upstream node to the downstream node;
(iii) configuration transmission from the upstream node to the switch (programming of routing bits).

	redundant primitive			**TMR-less primitive**		
stuck-at	TOTAL	switch	system	TOTAL	switch	system
1	0.02	0.02	0	25.54	17.48	8.06
2	5.12	3.9	1.22	45.75	30.65	15.1
3	9.06	6.8	2.26	x	x	x
4	14.63	10.92	3.71	x	x	x

Table 2.1: Failure probability of a fault-tolerant routing primitive, as a percentage of the input test patterns.

The failure probability has been evaluated based on response correctness, i.e. even if a fault is present, if the response of the circuit is correct (i.e., 2 out of 3 output voters provide the correct input bit) the test passes successfully. This is also the case when a stuck-at fault has the same value of the corresponding transmitted bit (and other kinds of "lucky" situations). Tests with single stuck-at and with multiple stuck-at faults were run. Results are reported in Table 2.1 ("x" stands for not tested for lack of interest).

Column "total" represents the total failure probability of the routing primitive in the presence of n faults in the primitive itself. This means that TMR is not able to contain error spreading. However, in these unfortunate cases, our online testing strategies come into play (dual-rail encoding, three-way handshaking) and enable the global controller to take the proper course of action. Therefore, column "switch" indicates the percentage of cases where only the local switch is discarded from the configuration process by the controller. Only the "system" column indicates when the ring structure becomes unusable (the global controller can neither receive diagnostic bits from a group of switches nor correctly reconfigure the others).

Results show that, while redundancy allows for an almost 100% coverage of single stuck-at faults, without redundancy (simple primitive) the probability of transmission failure rises to 25.54%. When multiple stuck-at faults are considered, the probability of failure of the simple element skyrockets, while redundant primitive keeps working unaffected in most cases. Please notice that while single stuck-at faults were exhaustively injected, 10000 random injections for 2,3 and 4 faults were experimented.

8.5 Fault tolerance of the dual network

The data collected in the previous subsection can now be used in formulas (1) and (3) to provide the probability to use the dual network to configure at least one switch.
Assuming P_0 the probability of having a single fault, and consequently P_0^2 the probability of having two faults, x_1 the probability of having a system failure (see previous section) because of a single stuck-at fault and x_2 the probability of having a system failure because of two stuck-at faults, being $P_0 \ll 1$ we can calculate directly p_f of the first formula as

$$p_f = x_1' \cdot P_0 + x_2' \cdot P_0^2 = 8.06 \cdot P_0 + 15.1 \cdot P_0^2$$

and P_b of the third formula from

$$1 - P_b = x_1'' \cdot P_0 + x_2'' \cdot P_0^2 = 0 \cdot P_0 + 1.22 \cdot P_0^2$$

Comparing the two different instances of the dual network as from section 7, the improvement of the failure probability when voting at every stage rather than having a single voter before the Global Controller is approximately $40 \cdot n$, where n is the number of stages of the ring (maths omitted). This results in an architecture which is 630 times more robust when 16 network nodes are considered.

8.6 Performance analysis

In order to simulate the entire reconfiguration process, we set up a Verilog testbench materializing the following scenario. A 4x4 mesh is considered as reference topology; 16 routing primitives are instantiated and connected together thus creating a chain topology which is dual with respect to the main network (see Fig. 2.9). The dual network is completed with a zero-time behavioural model implementing the behaviour of the global controller. The fault injection takes place by automatically injecting a growing number of stuck-at faults among the primitives. The set of possible consequences of such fault injection are enumerated as follows:
Errors masqueraded: the fault injection is solved by the inherent fault tolerance capabilities of the dual network provided by the TMR and the voting

system.
Switch(es) exclusion: when fault injection is such that the TMR system of a primitive is fooled, the relative switch might not be able to correctly notify its status to the global controller (or conversely the global controller might not correctly reprogram the switch). Therefore, the configuration strategy implemented in the global controller would automatically exclude such switch element from the final topology. Please note that, even in such condition, the primitive might still properly serve as a gateway for the dual network thus correctly delivering diagnosis and reconfiguration information for other switches.
Unusable network: if the fault injection is such that the TMR system of the routing primitive is fooled and the primitive itself is no longer able to forward diagnosis and reconfiguration patterns from/to the other elements of the dual network, then the network becomes unusable (i.e., not configurable via the dual network).
Catastrophic programming errors: the global controller programs the switches erroneously but does not detect such event. This situation leads to an incoherent status of the routing mechanism in each switch and ends up with an overall wrongly programmed network. According to our simulation setup, such scenarios are very rare and take place only 0.3% of the times over 10k experiments (with a significant number of injected faults: from 6 to 9).

In order to characterize the aforementioned scenarios in reasonable time, each number of different faults has been injected 1000 times with a uniform random distribution in the routing primitive gate-level netlist.

As previously discussed when commenting the reconfiguration strategy of Figure 2.10, the time required by the configuration mechanism to correctly reprogram all the working switches of the system highly depends on the number of faulty primitives of the dual network. In fact, every time a switch has to be discarded by the global controller since not reachable through the dual network, the topology structure changes and part of the reconfiguration strategy (Fig.2.10) has to rerun. In the experiment reported in Fig. 2.17, an increasing number of faults were randomly injected in the dual network. The plot reports, for such number, the slow down factor experienced by the global configuration strategy when such number of stuck-at faults increases.

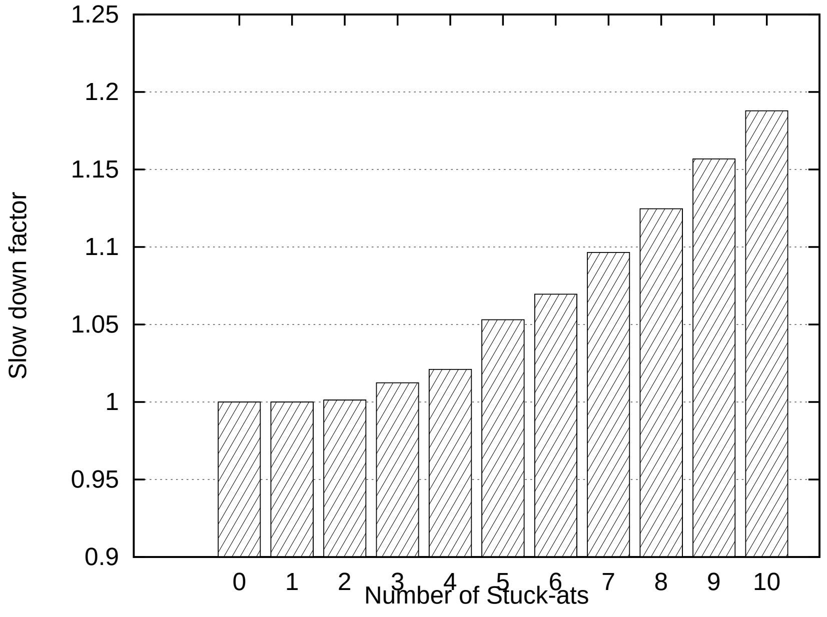


Figure 2.17: Average configuration time (excluding the configuration algorithm).

In fact, the higher the number of stuck-at faults, the higher the probability of communication failures through the dual network, therefore the reconfiguration strategy has to rerun several times before converging to a final topology picture. Please note that this analysis does not account for the number of cycles required by the configuration algorithm running on a microprocessor acting as the global controller. Therefore, the reported result is the execution time of the"bare" hardware of the configuration strategy [2].

Fig. 2.18 reports, for an increasing number of injected faults, the percentage of the *fully* working networks. We consider a fully working network, a network where no switches have been excluded because of a communication failure through the dual network. For those cases where the network features at least one non-working switch, the average number of discarded switches ranges between 3 and 4 thus pointing out the effectiveness of our combined BIST and fault tolerance techniques. The extreme case where all the switches are excluded is reported in Fig. 2.19. This figure accounts for the case where the wrong operation of one or more routing primitives prevents the global controller from correctly reprogramming all network switches thus ending

[2]For this experiment we assume the controller to generate configuration bits randomly

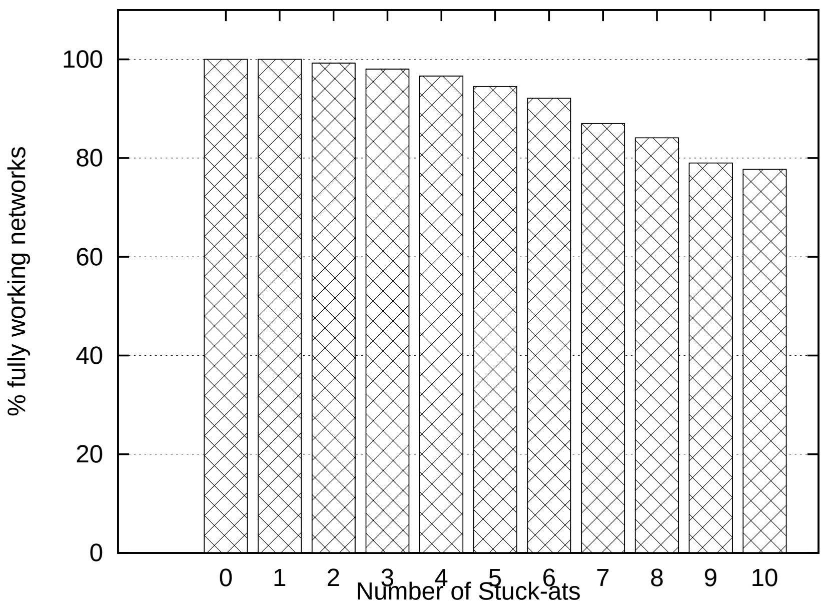


Figure 2.18: Percentage of fully working networks (no switch excluded) varying the number of injected stuck-at faults.

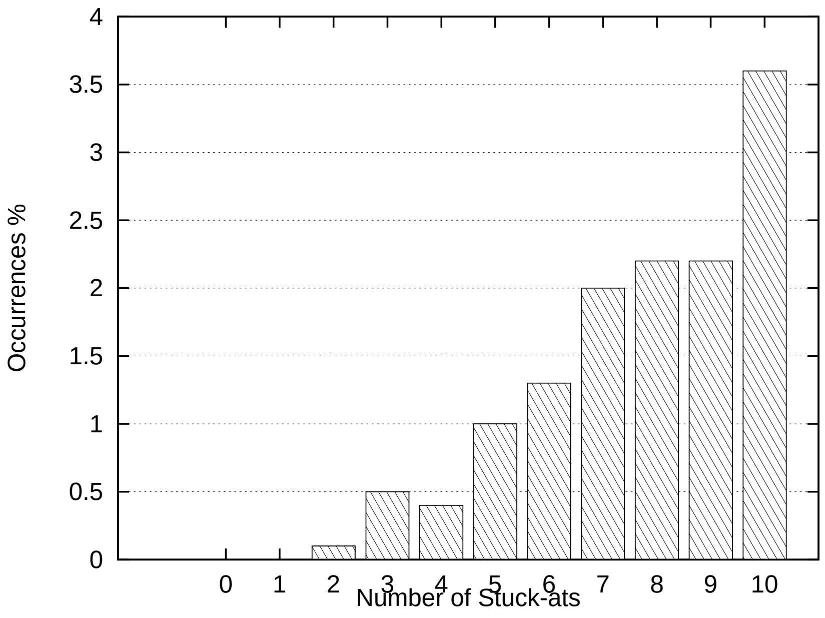


Figure 2.19: Unusable network.

up in a non-configurable system. Interestingly, even with a large number of injected faults, only a 3.5% of experiments report an unusable network.

9 Conclusions

This thesis takes on the challenge of guiding NoC designers toward the extension of their baseline architectures for fault-tolerance in a resource constrained environment. SEUs are targeted, in addition to intermittent faults. First, we proved that better solutions exist than traditional TMR approaches, which become increasingly prohibitive in nanoscale technologies. Second, error correction was proved to be readily available by careful choice of the correcting code and by blindly applying TMR only to the control path. As a result, error correction impacted the critical path by 22% and the total switch area by 13% with respect to link/switch-level retransmissions. In contrast, correction was surprisingly found to be the most power efficient approach for the same target speed, thus materializing the benefits of the lower complexity on the control logic and of the lower buffering requirements dictated by the native STALL/GO flow control. Moreover, we brought fault-tolerant flow control to the next stage by combining the benefits of STALL/GO and of ACK/NACK. The resulting solution pays the price of a complex control logic to manage retransmissions, although it enables to preserve the critical path and therefore suits the needs of high speed NoC realizations.

In addition, this thesis proposes the complete self-diagnosis and self-configuration NoC infrastructure, capable of readapting the network to work around single and double faults in less than 1 microsecond. As a relevant contribution, the thesis overcomes the limitations of an isolated development of the diagnosis infrastructure and of the configuration mechanism, thus capturing design inter-dependencies and coming up with a globally optimized hardware/software implementation.

The key idea consists of using a centralized controller thus exploiting its global visibility of network state for efficient routing reprogramming. Diagnosis bits from a distributed and scalable detection procedure are delivered to the controller through a fault-tolerant dual control network. Online testing strategies then enable to detect the cases where vital diagnosis and configuration bits are incorrectly signaled between the NoC switches and the controller (and vice versa), thus leading to the conservative discarding of those switches unable to interact with the controller. Please notice that in future work the same dual control network could be used also for other purposes such as debugging or congestion management.

Chapter 3

Non-Intrusive Trace & Debug NoC Architecture with Accurate Timestamping for GALS SoCs

1 Motivation

On-chip debug and trace solutions are essential to the modern *design* flow, where software has to be developed and optimized for a certain hardware platform. A suboptimal software routine can have a significant, diminishing impact on the overall performance of the system-on-chip (SoC). Decrease of performance can be caused by logical errors, such as bugs in a program's code. Detecting such errors could not always be a trivial task, e.g. when race conditions are present. Furthermore, there might be other deeper dependencies between the program flow and the rest of the activity occurring within the SoC. Such complex interactions might be hard to find. Therefore, it is advantageous to have a way of observing how a software program interacts

This chapter includes contents that are referred to a cooperative and interdisciplinary research. Furher details can be found in [79] with specific hardware components and how the flow of information caused by it spreads through the SoC. Thus, it is important that the actual hardware platform becomes, in a way, transparent to the software developer. Such a transparency is key to better *understanding how* a particular code behaves when executed and what its mutual impact with the system is. More precisely, the software development flow has to be enhanced by knowledge not only of the micro architecture, which is executing the software program, but also of the SoC as a whole. Information like the delay, contents and effects of different processing blocks on a particular transaction can help optimize and debug software and lead to shorter and more precise development cycles.

Additionally, observing the activity on of a particular SoC has to be done in a non-intrusive way. This will ensure that the produced observations do not interfere with the normal operation of the underlying hardware. In the converse case, it cannot be guaranteed that the behavior of the system being traced is the same as its behavior while not being observed. Thus, interference might render the whole observation trace useless.

Another requirement is that observations carry, besides the information of the observed activity, also information about the time at which they were produced in the form of *timestamps*. They are essential for ordering the observations into a complete trace, detecting parallel processes and estimating delays. Furthermore, as a lot of the modern SoCs comply to the globally asynchronous, locally synchronous (GALS) paradigm, it is also important that observations from different parts of a system can easily be combined into one single trace, with high temporal accuracy. SoCs designed for mobile communications and handheld devices rely heavily on power management strategies involving clock and power gating to preserve energy. This imposes a requirement on the way the system is observed to also be able to cope with partial and full system power downs. Finally, a minimal number of resources have to be spent on producing these observations.

This work presents a novel solution to the afore mentioned problem. The observation of the system is performed by reporting, by a dedicated output port of the chip, all the transitions taking place through the interfaces of each module with the main interconnect network. Here, the complete operation of the system can be reconstructed via software. The approach discussed

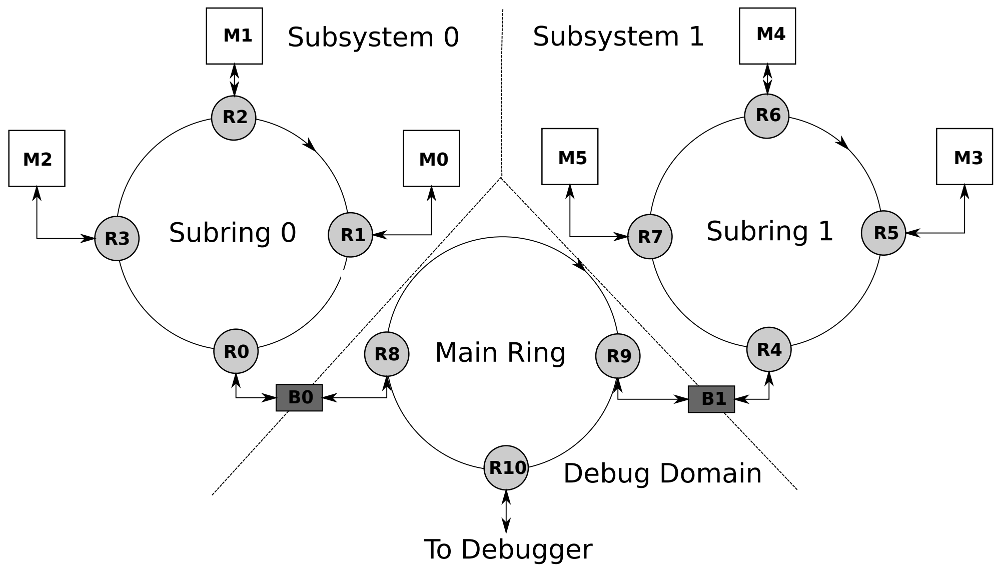


Figure 3.1: Hierarchical ring topology

in this paper utilizes a dedicated trace and debug network-on-chip (NoC) architecture for transporting observations between the monitoring devices, producing them, and the debugger module/interface. As the observations are transported on a separate interconnect, they become by definition non-intrusive to the normal operation of the SoC. The topology used to implement the trace and debug NoC is a *hierarchical ring, which eases its application into GALS SoCs and have a reduces hardware requirement for its implementation.* Furthermore, this work presents a novel timestamping strategy, which is differential in nature, and is able to give accurate time to observations independent of their subsystem of origin. The timestamping technique also allows the trace to be consistent even after multiple partial power ups and power downs of the individual subsystems. The final trace is composed by software at the debugger side from all the received observations. For completeness, the work also proposes a dummy abstract model of the monitoring devices, referred to as monitors.
The rest of the paper is structured as follows: Section 2 presents the related work on the problem. Section 3 presents the architecture and all the components of the debug and trace NoC. Section 4 presents the novel timestamping strategy. Sections 6 and 6 present, respectively, results and conclusions.

2 Related Work

Several approaches, which propose solutions to the problem of tracing a SoC, exist. For example, the work presented in [82] illustrates an AHB tracer, aimed at producing observations from an AHB bus. However, the authors concentrate only on one type of interconnect. The method lacks the notion of time and is also intrusive to the normal system operation. Other works, like [84] and [43], concentrate on directly debugging code executed within the processor by making use of scan-chains. Yet, their work primarily concentrates on inserting breakpoints and is limited to operating only within the scope of the processor, rather than the system as a whole. The works [41] and [30] both concentrate on transaction level debug, where the communication, rather than the state of the processor is observed. In [41] the regular NoC interconnect of the SoC is reused for passing the observations to their final destination. These observations, however, are generated only when the system cores are stopped. Thus, it is not particularly suitable for tracing live execution of software in a non-intrusive way. The work in [30] relies on scan-chains, which can severely limit the bandwidth for observations. The work also does not support accurate accounting of time. None of the works involve mechanisms for coping with power saving techniques such as partial system shutdowns.

As far as the hardware viewpoint is concerned, in [26] a dual network for control singaling is presented, but its purpose is for hardware test, so it is not suitable for software tracing and debugging.

3 Proposed Architecture and Topology

The topology of the architecture utilizes a hierarchical ring structure, where a single Main Ring is connected to multiple Subrings. *Each ring encompasses at most one subsystem of the SoC, where a subsystem is defined as all the modules residing in the same voltage and frequency domain. Routers have two input and two output ports, and are input-buffered. One output port of each router is connected to one input of the downstream one. This input-output pair of ports (ring side) creates the ring structure. On the contrary,* *the other pair (slave side) manages the bi-directional traffic between the ring and the attached structure. That is, in the Main Ring every router is attached to a Bridge module, that permits voltage and frequency crossing between the Main Ring and the Subring, while in Subrings every router is connected to the monitor module that is responsible for traffic observations.*

To reduce wiring overhead all the rings are implemented using unidirectional links. The only bidirectional connections exist between the main ring and the subrings, between the monitors and their corresponding routers and between the debugger and the corresponding router on the main ring. This implementation introduces greater latency upon the traffic on the rings, but, since traces have to be recollected and analyzed offline, high latency is not important as long as the trace can be correctly computed.

Fig. 3.1 illustrates this structure for an example SoC with two subsystems. Each sub ring in this model has 4 routers ($R0$-$R3$ for Subring 0 and $R4$-$R7$ for Subring 1). The main ring has only three routers ($R8$, $R9$ and $R10$). $B0$ and $B1$ are the bridges, which connect the two subrings to the main ring. The boxes, labeled $M0$-$M5$, represent the monitors that generate observations.

The routing of the packets in the architecture is simple. There are two directions, in which packets can flow. They can either propagate from a monitor to the debugger (*downstream*) or from the debugger to the monitors (*upstream*). The amount of traffic going upstream (configuration packets) is expected to be low as compared to the packets traveling downstream (trace packets). Packet headers are inspected by the routers and a packet is either left onto the particular ring or switched to the other port.

The monitors trace the activity on the bus interfaces, to which they are attached. The observations are encapsulated in the form of packets, which are injected into the subrings. These packets then, through the debug interconnect, will they reach the debugger attached to $R10$, where they get absorbed and included in the overall trace of the system.

The structure of the monitor module is not discussed in depth. It is an observation module, which can differ on the basis of the observed communication protocols. However, all different monitors share a set of common features. *The purpose of a monitor is to observe in a non-intrusive way the activity on a transmission channel, such as the input and ports of a core or a memory,*

encapsulate it in the form of NoC packets, and forwards them on the rings. All monitors use the same timestamping mechanism, as discussed in Section 4. A monitor can be programmed to be in non-operative mode, where no activity is reported, or operative mode, where it is sensing the transmission taking place on the observed channel. Upon monitor's activation or deactivation a special packet is emitted, as discussed below. As the monitor is able to receive as well as transmit packets, it can be constructed in such way as to interact with the module it is observing. This, for example, may allow the programming of breakpoints and, hence, allows the discussed solution to be used non only as as a trace, but as a debug architecture as well.
Because of the NoC's hierarchical ring structure, a special arbitration scheme has been designed, in order to provide fairness between the packets produced by different monitors. To explain the rationale of this, let's refer back in Fig. 3.1. In Subring 0, supposing all Monitors $M0$-$M2$ being operative, Router $R1$ attached to Monitor $M0$ will receive more traffic from the Ring port, where both $M2$ and $M1$ are sending packets, than from the one attached to $M0$. By using a simple round-robin arbitration, the Monitor nearest to the bridge would be privileged. Assuming that there is always a competition on the subring, the probability of winning all of the $K+1$ arbitrations is given in Eq. 3.1.

$$P_{pass}(\pi_K) = \left(\frac{1}{2}\right)^{K+1} = 2^{-K-1} \tag{3.1}$$

As it can be seen the probability varies with K and for $K >> 0$ it tends to 0. Thus, it is unfair for packets that have to traverse more hops. To resolve this issue a better, weighted arbitration scheme is used. Instead of having equal weights $w_{P0} = w_{P1}$ for the two ports, they are scaled in correspondence to the location of the router. This means that each of the two input ports have a different probability to have access to the ring output port. For the router attached to the K^{th} monitor, the weights are as defined by Eq. 3.2 - 3.3.

$$w_{K_{P0}} = N - K - 1 \tag{3.2}$$
$$w_{K_{P1}} = 1 \tag{3.3}$$

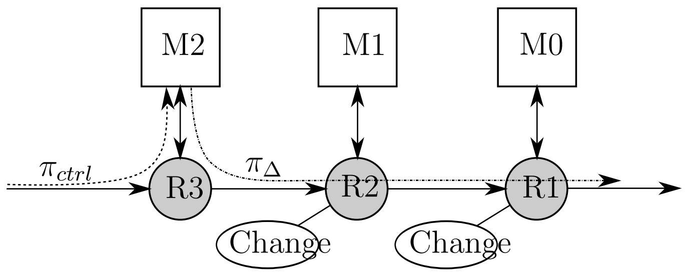


Figure 3.2: Transmission of a weight change packet π_Δ

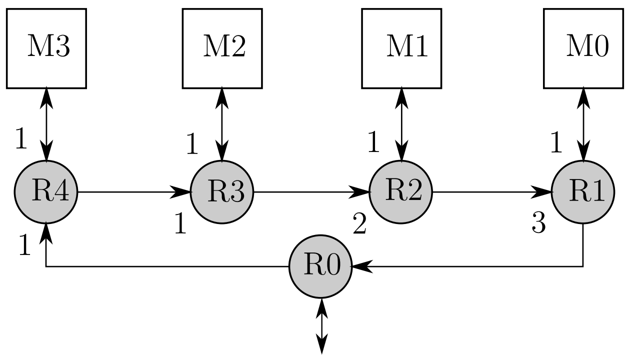
For a ring with N monitors the new weights allow for a new pass probability expression $P_{pass}^{fair}(\pi_K)$ to be computed (Eq. 3.4). It is independent of the relative position of the monitor and ensures fairness to all monitors within a subring.

$$P_{pass}^{fair}(\pi_K) = \frac{1}{N-K}\prod_{i=0}^{K-1}\frac{N-i-1}{N-i} = \frac{1}{N} \tag{3.4}$$

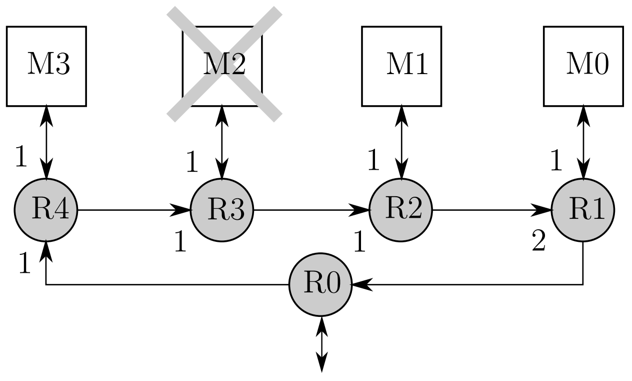
The fairness, however, can be lost because of monitor programming. If within a subring not all monitors are working, the weighting scheme may become *again* unfair. Thus, adjustable weights $w_{K_{P0}}$ are introduced (Eq. 3.5). The adjusting factor Δ signifies the change in the number of working monitors N. Hence, a ring can be seen as having variable size.

$$w_{K_{P0}} = N_{old} + \Delta - K - 1 = N_{new} - K - 1 \tag{3.5}$$

Whenever a monitor M is activated or deactivated, a special packet π_{Δ_M} is sent. When the routers, connected to the monitors, receive it on their ring port ($P0$), they adapt the weight according to the $\Delta \in \{-1, 1\}$ carried by the packet (Fig. 3.2). Fig. 3.3 shows an example of a sub-ring connected to four Modules. Fig. 3.3(a) illustrates the weights when all monitors are active for a particular subring. Fig. 3.3(b) shows how the weights look like when $M2$ is not working. The router attached to the last (K^{th}) monitor has weights equal to the ones of the successive downstream router, attached to the $(K-1)^{\text{th}}$ monitor. This an assumption based on the amount of traffic coming from the debugger. The control traffic (upstream) is expected to be negligible with respect to the one produced by the monitors. If the $w_{K-1_{P0}} > w_{K-1_{P0}}$, then the K^{th} monitor will get advantage over the $(K-1)^{\text{th}}$ one. Therefore, keeping $w_{K-1_{P0}} = w_{K-1_{P0}} = 1$ would be more fair.



(a)



(b)

Figure 3.3: Weight adjustment

4 Timestamping

Knowing exactly when a transmission is started or received by the actors of the communication is an important feature, required for an accurate inspection of the system under analysis. Every observation must carry such information, in order to produce a meaningful trace. This can be achieved by inserting information about time to each packet, and can be produced in several ways. As an example the system time can be exploited, or a dedicated counter calculating the clock cycles from the start of the system. Unfortunately, the first approach is intrusive to the SoC behavior as it requires access to a module, which is part of the system under observation. The second approach implies finite measure of time, since every counter will overflow sooner or later. In addition, the counter implementation requires high bitwidth counters and, thus, large timestamps, which consume bandwidth on the NoC. Furthermore, this approach fails under the assumption of partial system power downs and clock gating, because these procedures alter the behavior of the counter or modify its value. In addtion, in modern SoC there is not a common time reference, since the same architecture can be composed by multiple clock domains. For this reason, defining a reference on which to align every observation is even more difficult.

In order to overcome the previous limitations, and at the same time reduce the hardware utilized for this purpose, we designed a differential timestamping procedure. The main idea is that every ring of the topology shares the same clock domain, so shares the same time reference. For this reason, it is possible to use a small N-bit counter to keep track of time within the subsystem. Each subring S has its own counter C_S, or a number of unique and equivalent counters. The counters need not have the same size or partitioning among different subrings.

Each Monitor, when a transfer is observed, takes the counter value and includes it as a timestamp to the packet. Each router that bridges a subring S to the main ring ($R0$ and $R4$ in Fig. 3.1) and the router interfaced with the debugger ($R10$) periodically generate special packets, in order to inspect the progress of the counters inside every sub-system. Such a packet is generated and emitted every time the associated counter overflows, and it is referred to as end of period packet (π_{EOP_S}). The debugger keeps count of the absolute number of periods E_S for each subring by counting the number of received π_{EOP_S}. Thus, each time it receives a π_{EOP_S} it increases E_S by 1. When a normal trace packet π_S, originating from subring S, is received, it is ordered in the trace by the use of its differential timestamp δ_{π_S} and E_S. The absolute time estimate $\hat{t}(\pi_S)$ for an observation π_S with timestamp δ_{π_S} from subring

S with clock period T_{clk_S} can be computed by Eq. 3.6.

$$\hat{t}(\pi_S) = T_{clk_S}(E_S 2^{K_{C_S}} + (\delta_{\pi_S} \; mod \; 2^{K_{C_S}})) \tag{3.6}$$

However, this equation may lead to errors when, for example, back in Fig 3.1 monitor $M0$ generates an observation packet π_0 at time t, and shrtly after at time $t + \Delta t$, before π_0 has left the subring $R0$ generates the π_{EOP_0}. As π_{EOP_0} is generated in front of π_0, by the time the second reaches the debugger, the first would have already increased E_0 by 1. Thus, Eq. 3.6 will render false $\hat{t}(\pi_0)$, as the observation was obviously produced before the synchronization packet π_{EOP_0}.
To overcome this limitation, the N-bit counter used to construct the differential timestamps is further, virtually subdivided into two portions of K and M bits. Fig. 3.4 illustrates this for $N = 10$, $M = 2$ and $K = 8$. Thus, the higher M bits can be seen as a count on how many times the K-bit portion has overflowed. By associating the K bits to the time an observation is made within a period of $2^{K_{C_S}}$ clock cycles, the M bits are a counter mod $2^{M_{C_S}}$ of these periods.

$$\overbrace{\underbrace{0\;0}_{M}\;\underbrace{0\;0\;0\;0\;0\;0\;0\;0}_{K}}^{N}$$

Figure 3.4: 10-bit counter

By generating the π_{EOP_0} packet every time the K-bit portion of the associated counter overflows, the higher M bits serve as an indicator to the period to which a trace packet π relates. A relative period, indicating one of the last M periods of a subring S can be computed as $\hat{E}_S = E_S \; mod \; 2^{M_{C_S}}$. The equation for $\hat{t}$ is then adapted, such that when $\hat{E}_S < \lfloor \delta_{\pi_S} / 2^{K_{C_S}} \rfloor$ then Eq. 3.7 is used, otherwise Eq. 3.8.

$$\hat{t}(\pi_S) = T_{clk_S}\left(\left(E_S - (2^{M_{C_S}} + \hat{E}_S - \left\lfloor \frac{\delta_{\pi_S}}{2^{K_{C_S}}} \right\rfloor\right) 2^{K_{C_S}} + \right.$$
$$\left. +(\delta_{\pi_S}\ mod\ 2^{K_{C_S}})\right) \tag{3.7}$$
$$\hat{t}(\pi_S) = T_{clk_S}\left(\left(E_S - (\hat{E}_S - \left\lfloor \frac{\delta_{\pi_S}}{2^{K_{C_S}}} \right\rfloor\right) 2^{K_{C_S}} + \right.$$
$$\left. +(\delta_{\pi_S}\ mod\ 2^{K_{C_S}})\right) \tag{3.8}$$

This approach allows a packet π_S from S to experience a maximum delay of $2^{N_{C_S}}$ clock cycles before being falsely ordered in the trace. This makes the NoC observations traffic highly tolerant to latencies. As the trace generation is a computationally expensive task, it is assumed that the debugger carries it out in software.

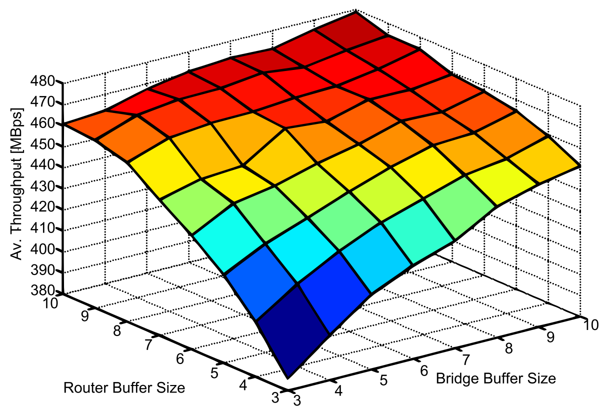
Finally, in order to comply with the partial power down and clock gating, another type of packet, a wake up packet π_{WUP_S}, is introduced. The π_{WUP_S} originates from the routers on the main ring, which connect it to the subrings (in Fig. 3.1 these are $R8$ and $R9$). A packet is created and sent to the debugger, whenever a router on the main ring senses the wake up of the attached subsystem S. It samples the value of C_{main} and uses it as a timestamp to π_{WUP_S}. Usually, bridges connecting different frequency islands make use of brute-force synchronizers, that introduce not only latency but also some uncertainity in the time a signal is sampled (usually at most one clock cycle). For simplicity, the time needed for synchronization is not included in the discussion, and it can be additionally included as a constant factor. By using either Eg. 3.7 or Eq. 3.8, an offset time $\hat{t}_{offset_S} = \hat{t}(\pi_{WUP_S})$ is computed. It indicates the time, at which the subsystem S has waken up according to the time of the main ring. This leads to the $\tilde{t}(\pi_S) = t_{offset_S} + \hat{t}(\pi_S)$ for the approximate absolute time estimate of the packets coming from S after wake up. Assuming that a subsystem S can wake at any point in time and no synchronization penalty, i.e. the router detects the wake up immediately, the introduced timing error ε is in $[0; T_{clk_M})$, where T_{clk_M} is the clock period on the main ring. The error lies below one clock of the main ring clock period, which is supposed to be the clock with higher frequency of the system, and makes the timestamping approach very robust. Compliance with clock gating is achieved in a similar way, by using the clock-gating signal of the subsystem as a reset to the associated subring counter.

5 Results

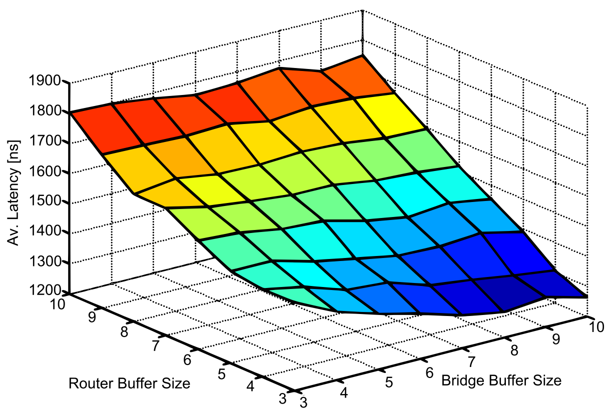
A SystemC transaction level model of the proposed solution has been implemented. The system used for the simulations has the following setup:

- 2 subsystems (2 subrings and 2 bridges)
- 5 monitors per subsystem
- Flit size of 16-bit
- $T_{clk_M} = 3\ ns$
- $T_{clk_0} = 7\ ns$
- $T_{clk_1} = 9\ ns$
- All counters have the same size, with $N = 10$, $K = 8$ and $M = 2$.
- All the monitors inject random observations of lengths between 4 and 40 flits

The higher the bandwidth provided, the more observations can be made simultaneously. If the monitors cannot inject the observed activity into the debug network due to congestion, information has to be trimmed to prevent incomplete or corrupted traces. Congestion on the debug network is usually avoided by having large buffer storage. On the other hand, the system is required to introduce as less hardware overhead as possible. To explore this tradeoff, the depths of the buffers in the routers and in the bridges are swept between the size of 3 and 10 flits. For each case a simulation has been performed. Fig. 3.5(a) shows how the throughput changes in accordance to the buffer resources for an average injection rate of 498MBps. It can be seen that increasing the buffers of all the routers has almost the same effect on the throughput (460MBps) as increasing the buffers only in the bridges (450MBps). Combining both does not produce a significant benefit (480MBps). The overhead of the router buffers, however, is much higher because there are a total of 24 buffers affecting the throughput in the routers vs. only 2 in the bridges (only downstream considered).



(a)



(b)

Figure 3.5: Throughput and latency vs. buffer size

Next, Fig 3.5(b) illustrates how the latency scales with the different buffer sizes. It can be seen that the latency grows with growth of buffer space in the routers and decreases with increase in buffer space in the bridges. The effect is caused by the increase of number of packets in transient, which are stored in the subrings. The buffers on the bridges do not contribute to the latency, but reduce it because they act as a storage space before the fast interconnect (the main ring). They allow a packet to be slowly stored while the router in main ring is transferring other packets, and then read rapidly in a bursty fashion once the access to the main ring has been granted. This observation also concludes that having increase in the bridge buffer space is better than increasing the buffer space of the routers.

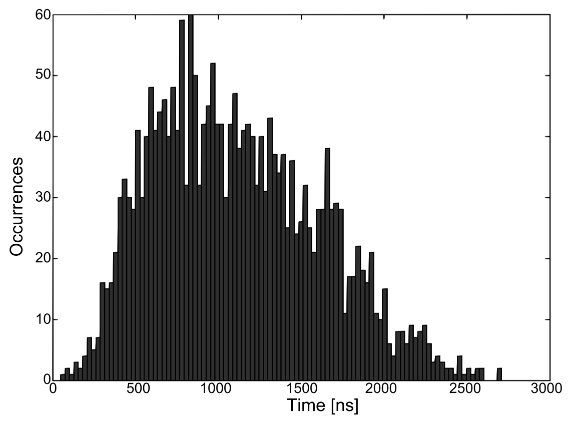


Figure 3.6: Latency histogram

Fig. 3.6 illustrates the latency histogram for a simulation run. The buffers of the bridges were set to depth of 10, while the routers were left with 3 flit buffers. It illustrates that packets can tolerate high latency. In the simulated conditions, even a latency of $2.7\mu s$ (900 clokc cycles) is acceptable and does not introduce an error in the trace.

Fig. 3.7 shows how the error of the time estimate $\tilde{t}$ looks like when partial shutdown is introduced in the simulation. The figure shows the plot of $\varepsilon = (\tilde{t} - t_{sim})/T_{clk_M}$ for the whole trace, where t_{sim} is the actual simulation time at which an observation was created. The figure shows that, even after multiple power down ($\textcircled{1}$, $\textcircled{2}$ and $\textcircled{5}$) and power up ($\textcircled{3}$ and $\textcircled{4}$

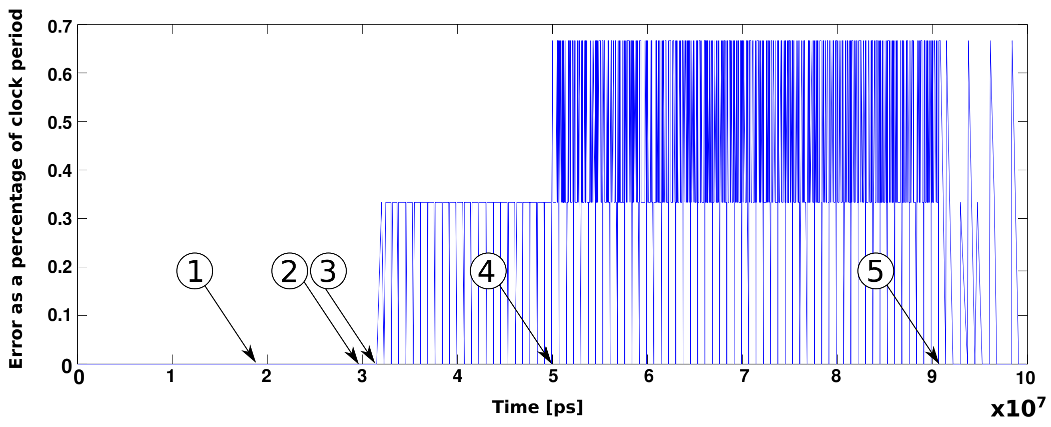


Figure 3.7: Power up error

) events, the total error of $\tilde{t}$ is always below one clock cycle of the Main Ring.

By requirement the debug and trace NoC has to come with as little overhead as possible. To demonstrate the small overhead of the presented solution, RTL models of the routers and the bridge were created and synthesis was performed. The results show that the area consumed for a subring with 6 routers (5 monitors) and a bridge is only 1002 FlipFlops and 2190 gates (3-flit buffers for the routers and 10-flit for the bridge). The maximum clock frequency achieved in $40nm$ is $800Mhz$, where the limiting factor is the bridge, which in its current implementation imposes a timing constraint of half a clock period.

6 Conclusion

This paper presented a novel NoC architecture for tracing and debugging GALS SoCs. The proposed solution provides capability for non-intrusive tracing, which is essential to observing the true interaction between software and hardware. Furthermore, the work presented a versatile and robust timestamping approach, which comes with minimum hardware overhead, is able to work across multiple domains and recovers with minimal error after partial system power downs and clock gatings. Finally, as shown, the solution requires low amount of hardware resources and at the same time provides unmatched capabilities.

Chapter 4

A Vertically Integrated and Interoperable Multi-Vendor Synthesis Flow for Predictable NoC Design in Nanoscale Technologies

1 Motivation

As of today, there are few (actually, not too many) design methodologies and CAD tool flows for application-specific networks-on-chip (NOCs) [7, 36, 69]. Especially in industry, interconnect vendors complement their interconnect IP offering with tooling to automate the creation of the system interconnect based on the communication requirements of the SoC at hand. State-of-the-art design methodologies and toolflows typically cover a limited range of the whole design process, especially topology synthesis [11, 39, 48, 49, 67, 80], and rely on a library-based approach to NoC design, wherein predesigned soft

This chapter includes contents that are referred to a cooperative and interdisciplinary research. Furher details can be found in [25] macros are composed at instantiation time to build arbitrary topologies.

In all cases, when we look at current toolflows with respect to future system requirements, we can identify the following criticalities:

A. EDA flow interoperability. From the system designer's viewpoint, the integration of various tools is a common requirement throughout the whole NoC synthesis flow. Also, in a design flow that requires largely interdisciplinary skills, it is very common that such tools come from different vendors, each focused and specialized on a specific design step. If tool interoperability is not properly addressed in a multi-vendor flow, the design cycle may largely prolong.

B. Beyond composition: integration. NoC synthesis flows (possibly multi-vendor) today rely on the juxtaposition of different design steps rather than on their full integration, thus missing global visibility and optimization opportunities. This causes, among the other things, poor inter-play between front-end and back-end design, which is prone to a lengthy and possibly inefficient convergence process. In the future, integration of tools into the whole NoC synthesis flow will have to be mastered by a backbone design methodology with global visibility, taking care of smart tool sequencing and of their interplay.

C. Evolving technology requirements. Today we are at a main transition point, where new features need to be there in design flows, all related to the intricacy of nanoscale designs. Issues such as process variations, power grid integrity, interconnect delay, etc. cannot be addressed as an afterthought any more, but from the ground up. At the same time, designs should be ranked with respect to these issues not late in the design flow, which may lead to lengthy iterations, but rather early, during a pruning step of the design space relying on technology awareness.

With respect to state-of-the-art, we develop a comprehensive and interoperable multi-vendor synthesis flow for application-specific nanoscale NoCs, and we present a structured validation framework of the novel features of this flow. In particular:

Claim A: *We deliver tool interoperability through the definition of a common and open specification format (named CEF), that provides a consistent representation of design data across all layers of the design process. Through*

the CEF format, design layers (and associated tools) exchange useful information, design intents, or directives. Therefore, CEF is the backbone of the proposed flow.
Validation Means: Tool Interoperability is explicitly proved by running the proposed multi-vendor flow for the design of a multimedia system, and by proving convergence (section 4.4.2). The CEF format is the actual means of interoperability between mainstream and prototype tools, building up the flow.

Claim B: *We deliver a vertically integrated design flow, meaning that (i) the entire flow is addressed, and (ii) global scope of the flow allows us to anticipate floorplanning, a typical back-end design step, in the front-end, thus biasing topology synthesis toward the most efficient physical implementations.* In addition, it is not just the length of wires that matters when synthesizing a topology: for the sake of low power, large amounts of traffic should be carried by short links overall; this is achieved by tightly coordinating floorplanning and topology synthesis, together with use-case specification, in the proposed flow.
Validation means: Claim B is validated in section 4.4.3 by proving the correlation between the communication cost metric used by the floorplanner tool to rank floorplans and the actual NoC dynamic power consumption of synthesized topologies, measured with post-layout analysis. This proves that topology synthesis has actually been driven to the most promising physical implementation.

Claim C: *The proposed flow coherently integrates mainstream industrial tools with new prototype tools addressing the most daunting challenges of NoC design in nanoscale technologies.*
C1. In existing mainstream flows, analysis of the power delivery network is not available until after physical implementation. At this stage, the turn-around-time is often too long to justify changes to the floorplan. As a solution, the proposed flow includes a fast, early-phase power delivery network analysis engine and integrates this into the prototype front-end floorplanner tool. The user can thus create floorplans that are optimized for system-level communication (see Claim B) as well as for power integrity.
C2. NoC links are the major cause of lengthy design iterations between

front-end and back-end designers. With the proposed flow it is possible to realistically project the actual length of NoC links, and even insert repeater stages upfront in order to meet predefined timing constraints, thus avoiding lengthy design iterations between front-end and back-end designers.
Validation means for claim C: The flow is put at work for the synthesis and physical convergence of an on-chip network for a media-rich mobile embedded system. The validation mean is the first-time-right design of this chip on a 40nm industrial technology library.

2 CEF File Format

CEF (Communication Exchange Format) is an open design format which specifies SoC-level architecture and communication infrastructure. It is designed to ease tool interoperability and reduce the iterative exploitation of tools at different stages and abstraction levels of the design flow. CEF allows for the expression of both *design intent* (objectives and constraints) and *design implementation*, describing for example:
(i) system cores, including relevant interface parameters;
(ii) communication requirements across cores, with support for multiple use cases;
(iii) clock and power domains, including frequency and voltage scaling;
(iv) interconnect implementation, including key architectural parameters and routes;
(v) rough floorplan of the design, including wire length annotations.
The CEF format allows for iterative and incremental design steps. For example, communication requirements can be provided in a coarse way or modeled accurately. The floorplan of the design can be omitted, then added when available, and subsequently modified. New usage scenarios, updated NoC revisions, or even entirely new system cores can be added at any time, either reflecting engineering changes or the better understanding of the system as development proceeds.
CEF was conceived to ease the interoperability of tools involved in the development of on-chip interconnects. To this end, it has been designed to be expressive enough to capture the key design steps and abstraction layers in-

volved in the design process. CEF does not enforce a strict order in which these tools must be used, instead promoting rich interaction between tools, back-annotation of parameters and successive refinements.
CEF has no aim to replace existing formats where established, useful standards exist, nor to model out-of-the-box every nuance of the system. For example, at the architectural level, CEF does not replace existing NoC models written either in C++, SystemC, Verilog, or other languages, nor it models in detail the architectural parameters of a specific vendor's NoC library. Instead, CEF provides generic, broadly applicable, syntax to specify router connectivity, with only some select properties (e.g. virtual channels, flit widths, buffer depths) explicitly included. The set of built-in properties was carefully chosen so as to remain generic and vendor-independent, yet able to express the key parameters that impact in a major way the performance/area/power trade-offs.
Rather than as an intermediate representation format, the CEF format is devised as a data exchange format. The need to maximize the interoperability of different tools has motivated the choice of the XML to make the format easily consumable by machines while remaining readable by humans.
Figure 4.1 shows an example of the different interconnect design tools that can interoperate on a single target system specification thanks to CEF. The tools operate at different stages of a design process, but can now be seamlessly mixed and matched instead of being forced in a sequential flow, enriching the backannotation and optimization opportunities. For instance:

- A NoC architectural simulator can now take into account wire propagation delays and variability effects identified by back-end tools;
- High-level exploration tools, optimizers and synthesizers can have a full view of constraints (e.g. communication requirements, block distances) and opportunities (e.g. architectural knobs such as buffering);
- Floorplanners can now automatically perform or optimize the system placement with full interconnect visibility, and can update it at the architectural level (e.g. by instantiating wire pipelining) if needed.

CEF was developed by the NaNoC consortium, including both academic and industrial partners with key expertise in system interconnect design. Since

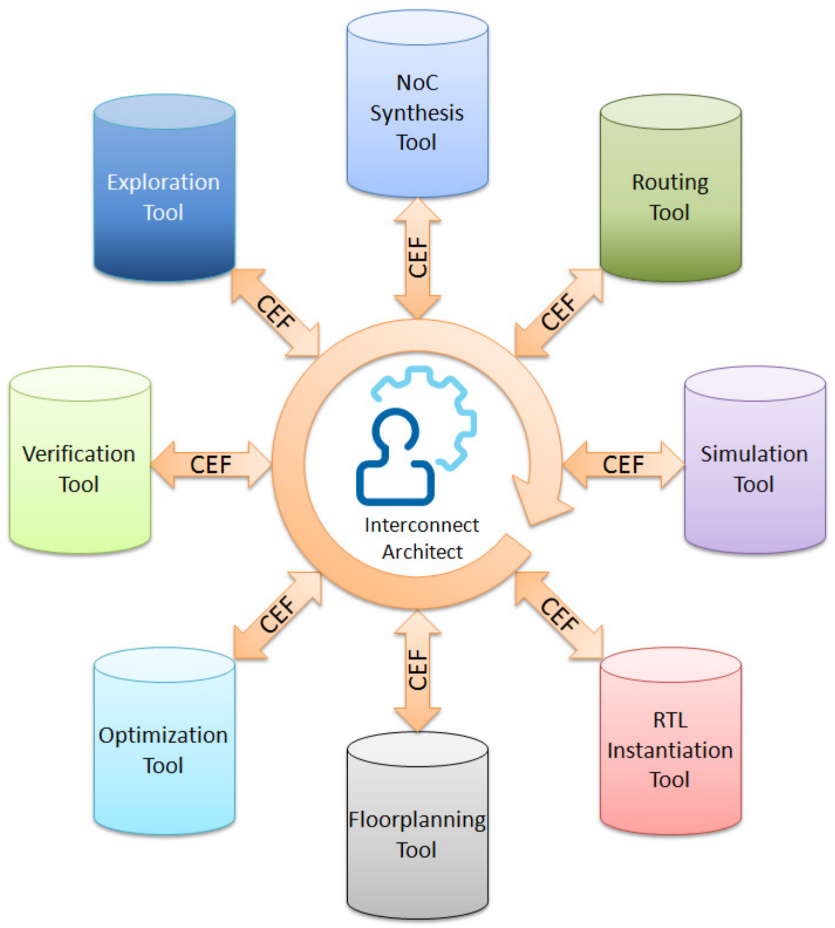


Figure 4.1: CEF-enabled interoperability between NoC design tools.

this thesis is focused on the NoC synthesis flow as a whole, the interested reader is referred to [22] for CEF details.

3 The Flow at a Glance

The design flow described in this thesis revolves around physical design convergence. The main concepts are to (i) automate the floorplanning of such heterogeneous systems according to their communication and power integrity requirements, (ii) perform NoC topology synthesis based on the resulting floorplan, (iii) a predictable physical synthesis process due to the early modeling and consideration of key technology-level convergence threats in the flow (namely interconnect delay and power grid integrity). A comprehensive

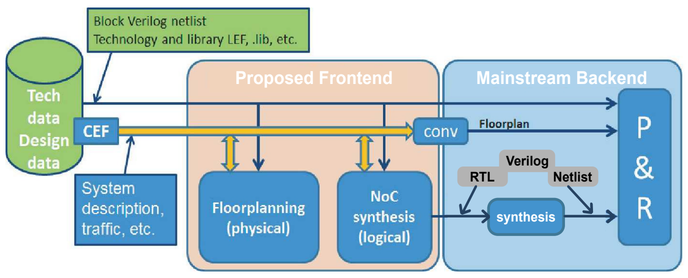


Figure 4.2: Proposed flow for the implementation of application-specific NoCs targeting heterogeneous systems.

overview of the flow is reported in Figure 4.2.

3.1 Front-End

The CEF file initially includes high-level specifications of the system, including communication bandwidth annotations between pairs of IP cores, specified on a use-case basis. A Liberty file characterizing the dynamic power usage of each IP core is also fed as input to the flow.

System floorplanning is performed *before* the synthesis of the NoC topology. Knowledge of the communication streams is used to direct the floorplanner tool to place closer together system blocks that communicate the most. Thus the total power of the NoC, once implemented, is lower, because the routing paths carrying high bandwidth streams are short. To enforce this behaviour, an *abstract communication cost metric* is defined by multiplying the bandwidth of communication streams by the distance between the associated communicating blocks in the floorplan. This metric is included in the objective function of the floorplanning tool.

The IR-drop of a design is greatly influenced by its coarse-grained placement, i.e., its floorplanning. Hence, in order to ensure power integrity closure, analysis of the power delivery network is warranted as early in the flow as possible [38]. Therefore, in the proposed flow the floorplanner not only reduces the communication cost function, but is also augmented to include an early-phase, dynamic IR-drop analysis directly in the optimization loop. This analysis leverages a prototype high-level power delivery network specification, and is fast enough (less than 50ms) for inclusion into the automated

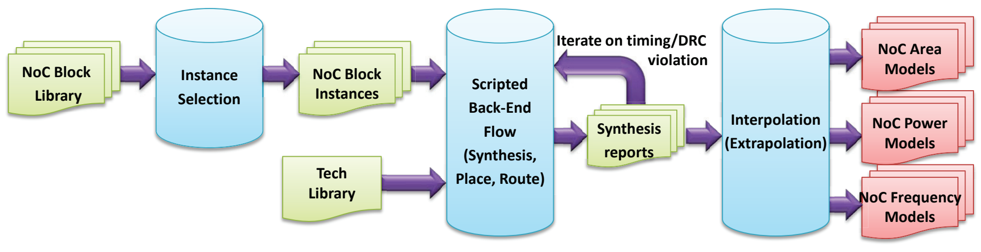


Figure 4.3: NoC Topology Synthesizer: block characterization flow.

floorplanning process in the design front-end stage. As a result, the user can create floorplans that are optimized for system-level communication as well as for power integrity.

With respect to commercially available tool flows, floorplanning frameworks typically miss awareness of system-level design properties. Moreover, IR-drop analysis is not available until after physical implementation.

The above novel features of the proposed flow have been incorporated, for the sake of experimentation, into the existing Teklatech FloorDirector tool [76]. Plug-and-play of this tool into the flow was straightforward by making it support CEF as its input and output format.

After the floorplanning stage the CEF file is refined to include the IP core placement. After this, it is handed over to the topology synthesis stage, where the Network-on-Chip is built according to the communication requirements of the system, described in the CEF too. The most relevant requirements for topology synthesis in the proposed flow are back-end modeling (parametric NoC component area and speed models, abstract component power models, interconnect delay models on the target technology), floorplan awareness for topology synthesis, and RTL automatic generation.

These features were to a large extent exposed by the commercial iNoCs NoC synthesis toolchain [36], which was then plugged into the whole flow by making it support the CEF format. The key novelty consisted of extending iNoCs framework to synthesize application-specific topologies on top of pre-assigned floorplans, delivered by a tool from a different vendor.

The final topology synthesis flow relies on NoC components characterization to compare different topology design points and block parameter settings. The characterization (Figure 4.3) relies on the complete synthesis, placement and routing of some instances of NoC RTL blocks, including Network Interfaces (NI), switches and links. This process yields power, area and maximum frequency values for those instances. Subsequently, by means of interpolation and extrapolation, predictions are made for all other possible instances of the library.

The baseline iNoCs' topology synthesizer was extended to accept a system floorplan from a CEF description as an input. The increasing impact of wiring, in terms of power and propagation delays, strongly suggests that it is essential to take into account the placement of the system cores in order to better estimate and optimize the interconnect [35,57]. If such a flow is followed, it is possible to better estimate the power consumption of the interconnect, and to design the topology (including, where needed, automatically instantiating link repeaters to break critical timing paths) to meet timing constraints from the ground up. In particular, the topology synthesis tool:

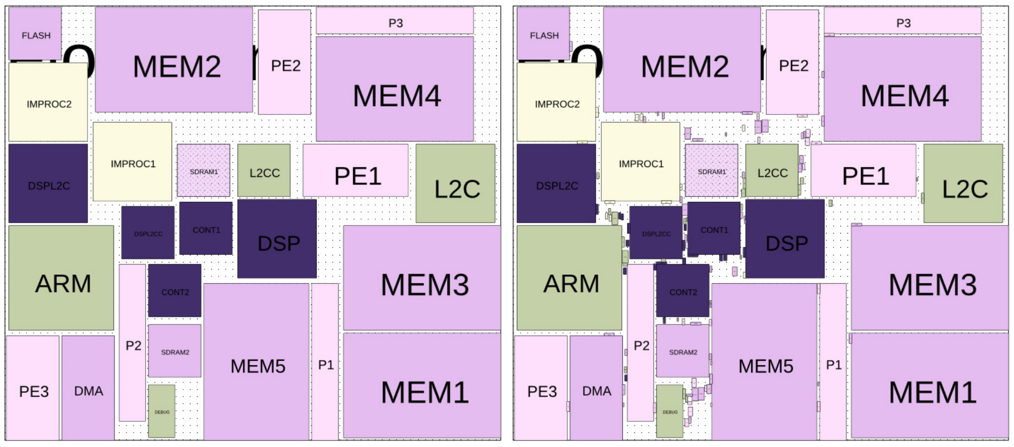
- estimates wire length with Manhattan distance, or more complex routing when hard macros are in the way;
- takes decisions based on a pre-characterization of wire performance in the target technology library;
- meets timing constraints on links through smart switch floorplanning and link segmentation, thus making convergence of P&R more predictable.

In order to guarantee smooth integration of this step into the whole flow, the insertion of NoC components into the design does not lead to expanding the floorplan boundaries (Figure 4.4). In fact, power grid is designed before NoC generation, hence expanding the floorplan would impair the assumptions of the previous stage. To minimize the concern, while trying to avoid lengthy iterations, optimizations were performed in the iNoCs floorplanning engine to converge towards more compact floorplans.

RTL of a complete NoC topology, together with its succinct CEF description, are then handed over to the physical synthesis flow for P&R.

3.2 Back-End

The RTL design is translated into a gate-level netlist mapped to the target technology library (tool used: Synopsys Design Compiler), according to the enforced constraints (mainly frequency and operating conditions). Boundary



(a) Input floorplan for a SoC design

(b) Output floorplan with NoC components instantiated

Figure 4.4: The NoC synthesizer inserts the NoC blocks into input floorplan by minimally perturbing block positions.

timing constraints have been conservatively set to 25% of the fastest clock in the design in order to take into account the delay of inter-switch links, 100× capacity of biggest inverter in the library has been used as output pin capacitance, and 0.4ns as transition time of signals at input pins.

The generated netlist is the starting point of the layout generation process, performed with Synopsys IC Compiler. We implemented a concurrent hierarchical Layout Generation methodology, exploiting a top-down approach. This flow is presented in Figure 4.5, and reflects mature industrial layout flows [47]. After the design has been imported, the floorplan definition (in particular die area and architectural blocks' position) is derived according to specifications extracted from CEF file. Then, power and ground grids are generated considering specifications from floorplanner tool. Next step is the partitioning of the Network-on-Chip into its fundamental components, thereby making it possible to perform place&route on each block concurrently and independently. This step is necessary to reduce layout generation complexity, machine runtime and to have tighter control on the layout generation process. Each block is then saved as a soft macro, so that only global routing between blocks is required in the top level. Once layout is completed, soft macros are "uncom-

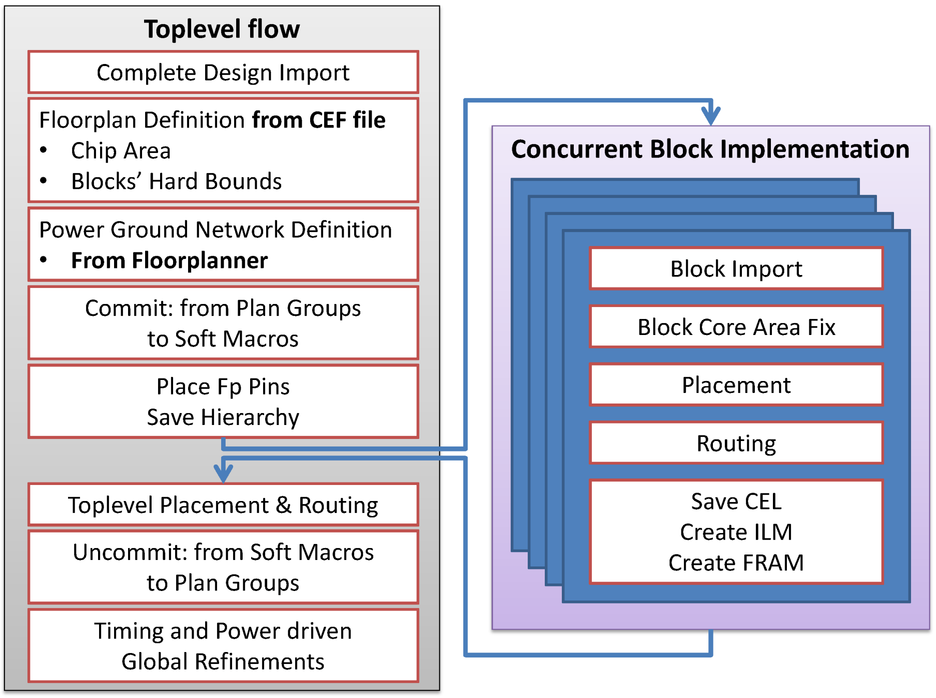

Toplevel flow
Complete Design Import
Floorplan Definition **from CEF file**
• Chip Area
• Blocks' Hard Bounds
Power Ground Network Definition
• **From Floorplanner**
Commit: from Plan Groups to Soft Macros
Place Fp Pins
Save Hierarchy
Toplevel Placement & Routing
Uncommit: from Soft Macros to Plan Groups
Timing and Power driven Global Refinements
Concurrent Block Implementation
Block Import
Block Core Area Fix
Placement
Routing
Save CEL
Create ILM
Create FRAM

Figure 4.5: Concurrent Hierarchical Layout Generation

mitted", i.e. reverted back to their composing standard cells, and final global refinements (power and/or timing driven optimizations) are performed.
The outcome of this step is a completely placed and routed netlist, together with the associated parasitic effects. At this point, accurate power analysis can be performed, by annotating switching activity on the resulting design. This is done by injecting traffic patterns that reflect the predefined use cases using for instance the Modelsim tool, and performing timing and power analysis with proper tools (e.g., the Synopsys PrimeTime suite).

4 Experimental Results

The target experimental setting is a multimedia chips for mobile phones (or similar devices), which runs different use cases. Its technical specifications reflect projected industrial trends of multimedia chip for the near future. This design under test features 25 IP Cores, which include CPU, Graphic Accelerator, various memory banks and peripherals. Further details in [74]. Extrapolating the usage scenarios of existing smart phones, communication

Floorplan	CommCost	IR-Drop (mV)
Best/Worst	15.83	386
Best/Best	18.87	166
Worst/Worst	43.91	383
Worst/Best	44.52	155

Table 4.1: Four selected reference floorplans in extreme corners with regard to CommCost and IR-drop.

requirements of CPU, hardware accelerators and memory for video playback have been scaled up to the high-end HD-TV resolution. The target technology library is an industrial 40nm 1.20V Low-Power CMOS Standard Cell library. *The goal of this section is to demonstrate that abstract metrics (the communication cost) and modeling frameworks (IR drops) used in the early steps of the proposed toolflow maintain a strong correlation with the physical properties of the design. In other words, we demonstrate that we are not pruning efficient points from the design space as an effect of abstract modeling of design properties or inaccurate/misleading objective functions in early-stage optimization tools. We thus selected 4 floorplan design points (Table 4.1), representative of the design space, and fed them to the next steps of the flow to prove correlation: promising design points will actually outperform the others as the design is refined.*

4.1 Floorplanning & Topology Synthesis, claims A and B

At this level, a correlation study was performed by synthesizing a large number of topology design points for the floorplans. The outcome can be seen in Figure 4.6, where dots of different colors represent topologies built on different reference floorplans. Recall that the communication cost computed by the floorplanning tool is expected to correlate with power consumption. The "red" and "maroon" dots represent solutions for the two floorplans with the lowest communication cost, and are very similar in actual power; the "blue" and "dark blue" dots represent solutions for the two floorplans with the

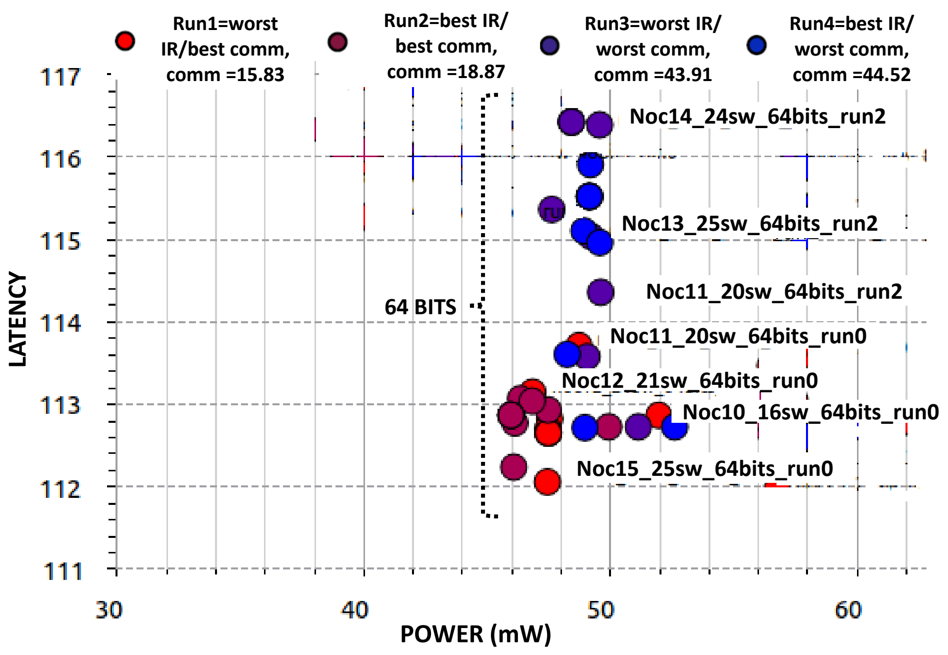


Figure 4.6: Topologies generated by the Synthesizer engine for the various floorplans for different flit widths.

highest communication cost, and are also clustered together. Indeed, there is a measurable difference (typically around 5 mW, or almost 10%) between "good" and "bad" floorplans. This separation is an average across use cases, and since this includes e.g. the Idle scenario in which the NoC is in fact inactive, it is clearly a conservative estimate; a separation of up to 15 mW or 25% was in fact noticed in traffic-intensive use cases. Moreover, the topologies built on a worse floorplan exhibit average flow latencies that are slightly higher, due to the need to insert, on average, a few more pipeline stages along links.

For the rest of the study, four specific design points among those produced by the Noc Synthesizer chosen, one per input floorplan. Without lack of generality, the flit width was fixed to 64 bits. Out of all the possible solutions at this width, we selected representative instances with low latency and typical power for the cloud of solutions based on the same floorplans. These four solutions are summarized in Table 4.2. Please note that reserved area takes into account a target cell occupancy of 50%.

The design points at this stage consist of floorplans and of application-specific

Corner Comm/IR	Switch Count	Flit Width (bits)	Reserved Area (mm^2)	NoC Power (mW)
Best /Worst	25	64	1.73	47.5
Best /Best	25	64	1.72	46.1
Worst /Worst	16	64	1.54	51.2
Worst /Best	16	64	1.54	52.7

Table 4.2: Main metrics of the four selected design points, one per input floorplan, as estimated by the NoC Synthesizer.

NoC topologies at the RTL level, custom-tailored for the floorplans and for the communication requirements of the application at hand. They have been derived by using tools from different vendors (Teklatech, iNoCs) coherently integrated into the same flow, and they have been showed to maintain correlation of power values across the hierarchy so far. With this respect, claim A is validated, and partly also claim B about correlation. The design points now undergo physical synthesis, involving the interplay with mainstream EDA tools and requiring to preserve correlation across the back-end design steps.

4.2 Physical Convergence, claims A and C2

The target design includes 10 clock domains. Since synchronization is taken care by Dual-Clock Fifos, clock signals have been defined as *asynchronous* to each other so the tool will not balance the sinks among the clock nets. In addition, a clock uncertainty of 10% has been enforced on every domain, to account for the possible clock skew.

During **Place&Route**, a hierarchical design flow is a must for an interconnect structure which is highly irregular, beyond being distributed. In order to have a tighter control of the results, Hard Bounds are created to define the areas where NoC building blocks will be placed.

Results: Total NoC area after layout turns out to be $946.152\mu m^2$ for Best Comm./Best IR drop topology and $898.744\mu m^2$ for Worst Comm./Worst IR drop topology, out of a chip area of $6855\mu m \times 6855\mu m$. Aggressive block

NoC Interconnect Timing Convergence			
Clock Domain	Target Frequency	Slack Pre-Opt	Slack Post-Opt
clk_Audio	100MHz	5,75ns	3,45ns
clk_CPU	500MHz	0,20ns	0,09ns
clk_DDR	250MHz	0,38ns	0,17ns
clk_DMA	200MHz	0,65ns	0,39ns
clk_DSP	300MHz	0,34ns	0,19ns
clk_Radio	150MHz	1,12ns	0,82ns
clk_USB	200MHz	0,53ns	0,23ns
clk_SPI	128MHz	6,33ns	2,27ns
clk_SRAM	500MHz	-0,19ns	0,14ns
clk_Video	300MHz	0,38ns	0,20ns

Table 4.3: Timing convergence for all clock domains with proposed methodology, before and after Global Optimization

boundary constraints applied to NoC switches led to timing convergence after the top level integration and global routing of the architectural blocks (3rd column of Table 4.3). Indeed, every clock domain has a positive slack, and only one has a small violation due to the unmatched sizing of a few gates between intra-switch links and switch output cells. This confirms that the link delay prediction and design engine of the topology synthesis tool have done a good job. Given this excellent starting point, the tool does not struggle during the global optimization phase to meet the timing requirements. Therefore, it can afford relaxing the faster paths to reduce area and power consumption (4th column of Table 4.3).

Overall, the described methodology actually delivers the fast convergence claimed by our flow (i.e., claim C2, or first-time right physical design), and completes the validation of claim A by proving effective integration of front-end and back-end multi-vendor tools.

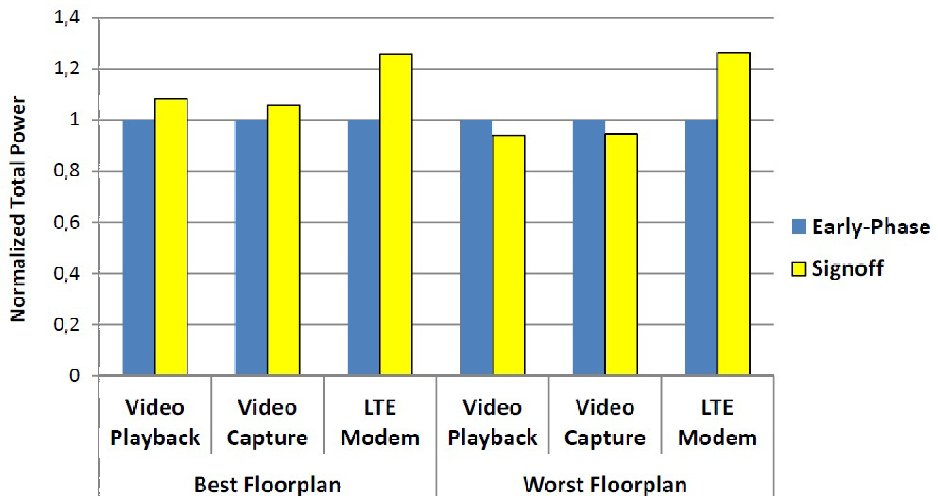


Figure 4.7: Correlation of NoC power between early-phase and post-layout analysis.

4.3 Correlation of Dynamic Power, claim B

It is indeed possible to prove correlation between Sign-off and Early-phase total NoC power numbers in communication-intensive use cases, such as in “Video Playback”, and “Video Capture” (see Figure 4.7). Deviations never exceed 8% for both “Best Comm” and “Worst Comm” floorplans.

Only in those use cases where the network is heavely underutilized (“LTE modem”), and approaches the idle use case, the power gap can be as large as 25%. This is due to the different choice of flip flops the synthesis tool made during the power model extraction methodology (Figure 4.3) and the actual physical synthesis of the design at hand. This difference is at such a low level of abstraction that it becomes extremely difficult to account for it upfront.

Given the large amount of specific timing and Design Rule optimizations each topology undergoes, all requiring different effort, and the necessarily more abstract power modeling performed during topology synthesis, these results ultimately validate claim B.

4.4 Correlation of IR Drop Maps, claim C1

This section needs to demonstrate that the IR drop maps derived by the front-end floorplanner (during the early phase analysis) and the same maps derived by the signoff tool indicate the same trends and critical hotspots. The In-Design Rail Analysis extension of Synopsys ICC was used as the reference sign-off tool.

It is worth observing that during early-phase analysis, only the current consumption of the IP cores is considered for IR drop analysis, in that the on-chip network has not been synthesized yet. To match (and later validate) this assumption, in the first place a special layout was inferred with the IP cores only.

In Figure 4.8 we report the IR drop maps displayed by Teklatech's early-phase floorplanning tool and the one displayed by Synopsys ICC for this design, with reference to the "Best Comm/Best IR drop" floorplan.

The correlation between the two maps is evident: they both agree on indicating the top-left quadrant of the design as the most critical in terms of IR drops. A similar correlation was observed for the "Worst Comm/Worst IR drop" floorplan.

When we compared the two floorplans (and their IR drop maps) with each other, we noticed that relative rankings between early-phase and signoff are in good agreement (degradations in the order of 2.3x and 1.5x respectively), thus confirming that the front-end floorplanner would have made the right choice when selecting the "Best Comm/Best IR drop" one.

Finally, we derived that the on-chip network has a maximum IR drop which is from 80 to 90% lower than that of IP cores, depending on the baseline floorplan. If we also look at the IR drop maps for the NoCs in isolation, we can see that they peak in almost the same physical spot of the IP core maps. This means that they do not even change the trend of the IP core maps. Therefore, it was hereby possible to validate the floorplan tool's assumption to perform early-phase power integrity check by neglecting the on-chip network.

5 Conclusion

This thesis reports a vertically integrated, interoperable and multi-vendor synthesis flow for NoCs. It ranges from application traffic specification to layout generation and physical convergence, and integrates prototype tools with mainstream industrial tools. Its main innovations include full vertical

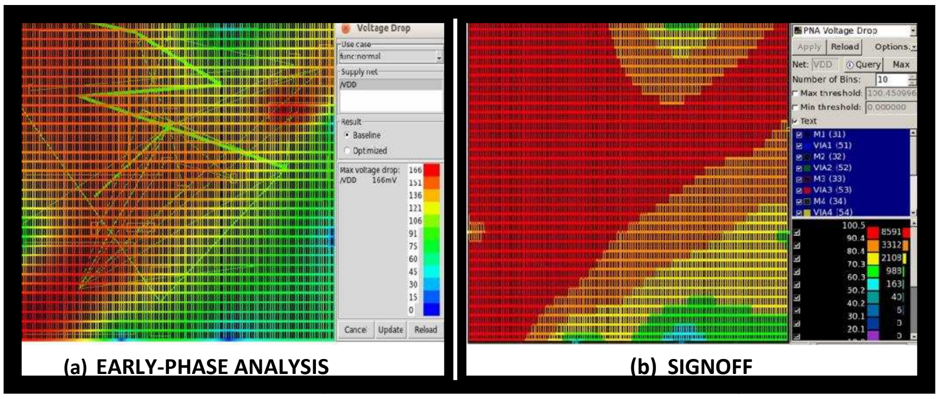


Figure 4.8: Floorplan with best communication and best IR drop.

integration, global optimization of design steps for technology-aware design, flow extensions to deal with nanoscale designs. The validation strategy of the flow revolved around first-time-right design and the proof of correlation of early-phase analysis and design choices with signoff.

Chapter 5

A Transition-Signaling Bundled Data NoC Switch Architecture for Cost-Effective GALS Multicore Systems

1 Motivation

There is today little doubt on the fact that a high-performance and cost-effective network-on-chip (NoC) can only be designed in nanoscale technologies under relaxed synchronization assumptions. On the other hand, such relaxation is even desirable for the upper design layers, since modern systems are typically structured into multiple, highly power-manageable voltage and frequency domains. The Globally Asynchronous Locally Synchronous (GALS) design paradigm can effectively support this architectural trend [13, 75]. Absorbing the heterogeneity of timing assumptions in such systems is ultimately a burden of the system interconnect, which serves as the global integration framework.

Asynchronous NoCs are the best candidates to take on this role, since they rely on clockless handshaking for inter-domain communication. These designs come with a number of potential advantages: average-case instead of worst-case performance, no switching power of a clock tree, easier convergence of hierarchical design flows, robustness to process/voltage/temperature variations, and efficient delivery of differentiated per-link performance. However, asynchronous NoCs are not yet at the stage of a mature interconnect technology for widespread industrial uptake, and their exploitation in real systems turns out to be slower than expected.

There are two fundamental barriers that prevent asynchronous NoCs from becoming a mainstream technology. First, they suffer from poor CAD tool support, in that design methods and tools for synchronous design cannot be directly applied. Many rely on a full-custom approach for the design of such circuits [32]. Some recent work holds the promise of bridging this gap to some extent [72, 77]. However, asynchronous NoC components are still typically delivered as rigid hard macrocells [78]. Second, the vast majority of previous work makes use of four-phase return-to-zero (RZ) protocols, involving two complete round-trip channel communications per transaction, as well as delay-insensitive (DI) data encodings (namely dual-rail, 1-of-4 or m-of-n) [52]. These design choices have typically resulted in an overly large area and energy-per-bit overhead with respect to their synchronous counterparts [42].

More recently, the above considerations have raised the interest in single-rail bundled data asynchronous protocols [52], which in principle provide designs with a lower timing robustness while significantly reducing area, wire-per-link and energy-per-bit overhead. In practice, with a bundled data approach, circuit timing must be carefully specified and controlled to ensure correct operation. At the same time, transition-signaling (i.e., two-phase communication protocol) is gaining momentum as a preferred match for high-performance asynchronous systems [34], especially for signaling across inter-router links [24]. Inside routers, four-phase protocols are still generally preferred since most existing two-phase asynchronous pipeline components are complex, with large latency, area and power overheads.

The objective of this thesis is to materialize a new design point for NoC switch architectures, bringing the benefits of asynchronous communication within reach of cost-constrained multicore systems. To the best of our knowledge, this is the first time a full 5-ported asynchronous switch is designed with a transition-signaling bundled data protocol. Our main target is twofold. On the one hand, we aim at a switch architecture that largely outperforms its synchronous counterpart with respect to area footprint, energy-per-bit and power consumption, while maintaining roughly comparable performance. In many previous quasi delay-insensitive (QDI) implementations, with delay-insensitive channels, while lower overall power is delivered, there are significant overheads in area and energy-per-bit. To validate our claim, we compare with one of the most cost-effective synchronous switch architectures for the multicore domain, called ×*pipesLite* [71]. On the other hand, we aim to be fully compatible with a standard cell design methodology and with a mainstream CAD tool flow for synchronous design. More specific contributions of this work are the following:

- *we take on the challenge of using two-phase bundled data* not only on inter-switch links but also *inside the switch microarchitecture,* to obtain high performance while not missing the low-complexity target;
- while a promising trade-off between cost and performance with transition-signaling bundled data has been previously been obtained with simple routing and arbitration primitives, this work is practically extended to a more intricate *full 5-ported switch architecture*;
- we aim at design convergence and high performance – *above 900 Mflit/sec* – in low-power standard-Vth 40nm technology;
- we introduce two novel, highly-concurrent and efficient asynchronous components, a transition-signaling *circular FIFO* and a *4-way arbiter,* each of which can be useful in other domains;
- we present a *semi-automated design flow* specific for transition-signaling bundled data design, which exploits commercial synchronous tools, and allows the creation of partially-reconfigurable macros;
- we *compare quality metrics of a post-layout design* of the new asynchronous switch *with a lightweight synchronous switch architecture (×pipesLite [71]),* in order to prove that through the selected asynchronous design style it is possible to provide an even more competitive design point, thus aiming to bring the benefits of asynchronous interconnect technology within reach of resource-constrained multicore systems;

- in the comparison framework with the synchronous switch, we consider *link parasitic effects,* which are of key importance in nanoscale technologies and whose implications on asynchronous switch performance are typically overlooked.

2 Previous Work

There has been a surge of interest in recent years in GALS and asynchronous design [75], [13]. Several GALS NoC solutions have been proposed to enable structured system design. Several of these approaches have been highly effective, especially for low- and moderate-performance distributed multicore systems [78], [8], thus targeting a different point in the design space than the proposed work. Some have low throughput (e.g., 200-250 MHz) [56], while those with moderate throughput (e.g., near 500 MHz [78], [12], [65], [64]) often have significant overheads in router node latency/area/energy-per-bit. Almost all use *four-phase return-to-zero protocols,* involving two complete roundtrip channel communications per transaction (rather than the single roundtrip communication targeted in our work), and delay-insensitive data encoding, resulting in lower coding efficiency than the single-rail bundled encoding used in this thesis [12], [65], [70], [3], [44].
Closer to our work is a promising recent approach targeting a two-phase protocol using a commercial computer-aided design (CAD) flow [58]. However, it has overheads due to a delay-insensitive (LEDR) data encoding and flipflop-based registers, and is not currently even suitable as a NoC. The GALS neural network system of [56] also includes two-phase channels between chips, with four-phase channels on chip, but uses delay-insensitive encoding.
The proposed NoC is based on MOUSETRAP pipelines [52, 66], which use a low-overhead single-latch-based architecture. This thesis delivers a previously unexplored design point for asynchronous NoC architectures, relying on two-phase bundled data encoding. We propose a more aggressive approach than [24], who limits the two-phase protocol to inter-switch links. The proposed solution builds on the work of [34] and [28], which demonstrate that transition-signaling single-rail bundled data can be efficiently employed in basic routing and arbitration functions. However, [34] and [28] target only

simple tree-based switch architectures, while this work addresses the intricacy of a full 5-ported switch design.

3 Switch Architecture

The proposed switch architecture is highly modular and can support the connection of an arbitrary number of input ports (Input Port Modules, IPMs) with output ports (Output Port Modules, OPMs). While the design space is potentially quite large, this thesis analyzes and characterizes a specific design point with the following features: 5 input and 5 output ports, 32-bit flit width, wormhole switching and algorithmic dimension-order routing. The ultimate goal is in fact to assess the quality metrics that transition-signaling bundled data can achieve on a specific design point of practical interest.
The switch architecture is inspired by the ×pipesLite architecture [71], which represents an ultra-low complexity design point in the space of fully-synchronous NoCs. Given this, coming up with an asynchronous switch consisting of the same building blocks while further cutting down on area and power is the challenge that this thesis takes on. ×pipesLite will be retained as reference design point to prove the claim of low implementation overhead and competitive design point.

3.1 Mousetrap Pipelines

The new asynchronous switch introduced in the following sections, is based on an existing asynchronous pipeline called MOUSETRAP [52, 66], which provides high-throughput operation with low hardware overhead. Each MOUSETRAP stage uses a single register based on level-sensitive latches (rather than edge-triggered flipflops) to store data, and simple stage control (replacing the clock) consisting of only a single combinational gate. These designs use single-rail bundled data encoding, where a synchronous-style data channel is augmented with an extra *req* wire, and a single transition on the *req* accompanying the data *bundle* indicates the data is valid. The *req* wire has a simple one-sided timing constraint that its delay is always slightly greater than the data channel. For further details, see [52, 66].

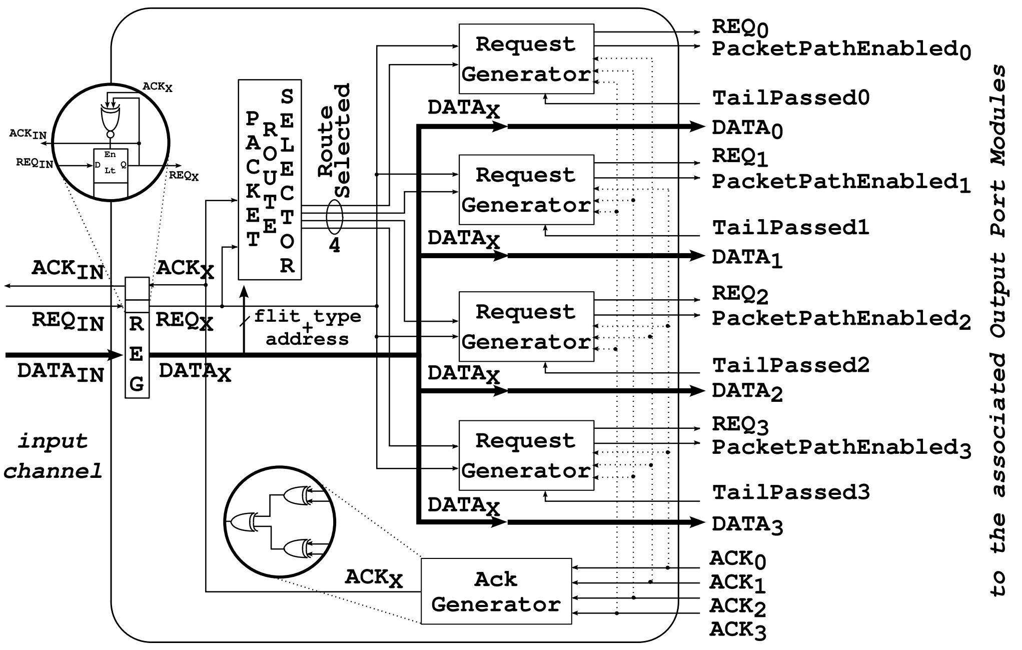


Figure 5.1: Input Port Module

3.2 Input Port Module Architecture

Input Port Modules route the packet to the correct Output Port Module, comparing the internal switch address with the destination address contained in the header flit. The microarchitecture of an IPM is presented in Fig. 5.1. The single-latch input register is normally transparent, as in MOUSETRAP pipelines, and the four Request Generator blocks are initially inactive. The basic operation of the module begins with a head flit arriving from the input channel, signalled by REQ_{IN} (soon after $DATA_{IN}$ arrives). The flit passes directly though the the input register, which then makes itself opaque, safely storing the data. It also sends the request (REQ_x) to all four Request Generator blocks, and an acknowledge (ACK_{IN}) on the input channel. The head flit also indicates to the Packet Route Selector to computes the single target output port, and to assert the corresponding one of four *RouteSelected* signals high. This signal sets the corresponding Request Generator to *packet processing mode*, which asserts its *Packet Path Enabled*$_i$ output high and sends it to the target Output Port Module. In tandem, the Req_x signal is broadcast to all four Request Generator modules, which result in transitions on *all four* output requests (REQ_0 to REQ_3, one to each of the four Output

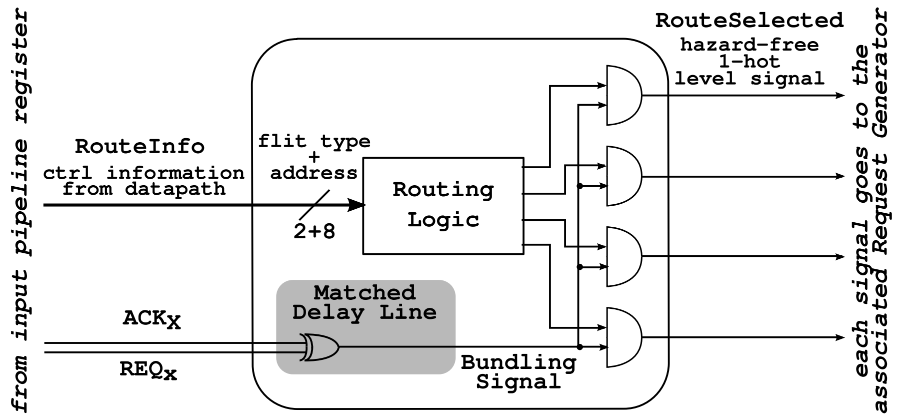


Figure 5.2: Packet Route Selector

Port Modules); however, only the one targeted Output Port Module will be activated, and the other requests are ignored (see details below).[1]

The target Output Port Module, after receiving both *Packet Path Enabled$_i$* and *Req$_i$*, sends acknowledgment ACK_i to the Input Port Module. The Ack Generator then makes a transition on output ACK_x, causing the input register to become transparent again. It is also sent to the Packet Route Selector, which deasserts the *Route Selected* output.

As long as the Request Generator Block still in *packet processing mode*, its *Packet Path Enabled$_i$* output remains high, and all the flits of the packet are directly transferred to the corresponding Output Port Module. Finally, when a tail flit is received, *Tail Passed$_i$* is asserted by the Output Port Module, which resets the Request Generator to inactive mode and deasserts the *Packet Path Enabled$_i$* signal.

Details of the *Packet Route Selector* are shown in Fig. 5.2. The XOR2 converts the two-phase signals Req and Ack to a level signal, and a matching delay line enforces hazard-free operation of the combinational routing logic.

The implementation of the *Request Generator* associated with Output Port 0 is shown in Fig. 5.3. The *Route Selected$_0$*, *Tail Passed$_0$* and *Packet Path Enabled$_0$* signals are all four-phase (i.e. level) and active-high; when the block is inac-

[1]Note that, since a two-phase signaling protocol is used, a REQ_i signal may at times have the *opposite polarity* of the incoming request REQ_x, depending on the number of flits that have been transmitted in previous transfers, and to which of the four Output Port Modules.

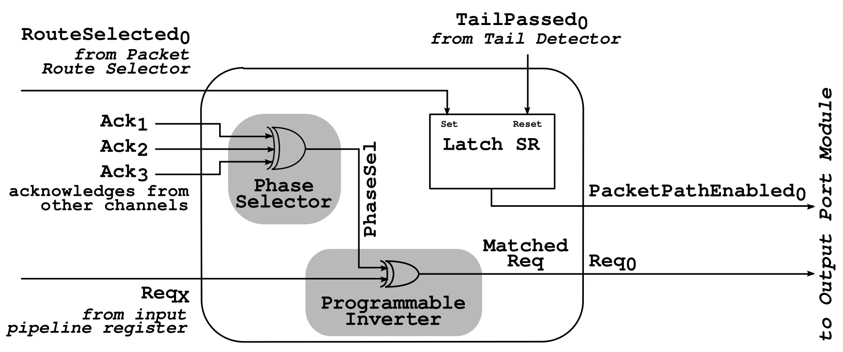


Figure 5.3: Request Generator for Output Port Module 0

tive, these signals are deasserted low. In contrast, the *Req* and *Ack* signals are two-phase. Whenever a request transition arrives on Req_x, it causes a transition on Req_0.

There are two cases of operation. If this is the Request Generator for the target output port, the unit receives $RouteSelected_0$ asserted high. It then enters *packet processing mode*, and asserts $Packet\ Path\ Enabled_0$ high. The XOR2 is used as a programmable inverter, where the correct polarity of the Req_0 output is selected by the XOR3 gate (i.e. phase converter logic). Eventually, when the tail flit arrives, the Output Port Module asserts $Tail\ Passed_0$ high, which resets the Request Generator to an inactive state while deasserting $Packet\ Path\ Enabled_0$ low. Finally, the $Tail\ Passed_0$ signal is deasserted low.

Alternatively, if this is not the Request Generator for the target output port, i.e. $RouteSelected_0$ is not asserted, the unit is not activated and $Packet\ Path\ Enabled_0$ remains low. As each flit arrives, the Req_x transition causes a Req_0 transition, which is ignored by the corresponding Output Port Module. In fact, in this case, Req_0 makes two transitions for each flit: the XOR3 observes the flit acknowledgment from *another* Output Port Module (i.e. ACK_1, ACK_2 or ACK_3), thus always returning Req_0 to its original value.

3.3 Output Port Module Architecture

Output Port Modules arbitrate between multiple incoming requests trying to access the associated output channel. The microarchitecture of an OPM is presented in Fig. 5.4.

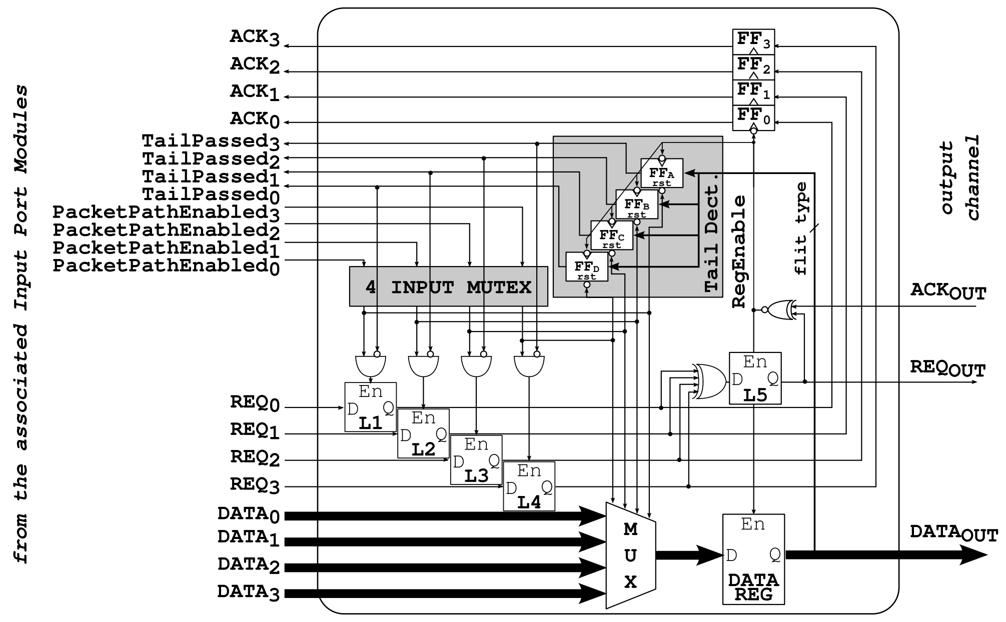


Figure 5.4: Output Port Module

Initially, all *Packet Path Enabled*$_i$ and *Tail Passed*$_i$ signals are low, and the wires of each transition-signaling REQ_i and ACK_i pair have the same values. Latches L1 to L4 are opaque, blocking new requests until they are arbitrated. Latch L5 and the Data Register are normally transparent, assuming no congestion, similar to a basic MOUSETRAP pipeline register.

A new transfer begins when a header flit arrives from one of the IPMs, concurrently with the associated *Packet Path Enabled*$_i$ signal asserted high. The 4-way mutex arbitrates requests from multiple IPM's trying to access to the same output channel, granting access to exactly one of them. Once the mutex is resolved, it performs two concurrent actions: it (i) selects the correct data input of the multiplexer, and (ii) forwards the winning request to the output register by making the corresponding latch (L1 to L4) transparent. The 4-input XOR gate functions as a *merge* element, joining four mutually-exclusive two-phase signals into a single request. This latch and the multiplexer are programmed once at the start of a packet transmission, and remain unchanged until after the tail flit arrives.

After the output channel request, REQ_{Out}, makes a transition, the data register and latch L5 are made opaque. They become transparent again when the acknowledge, ACK_{OUT}, is received, indicating that the flit has been re-

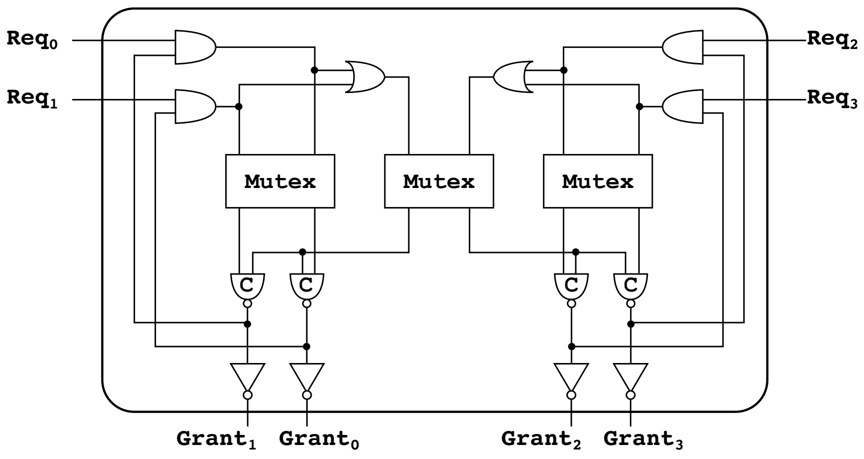


Figure 5.5: Microarchitecture of new 4-input arbiter

ceived downstream. When data and request are safely stored (*ReqEnable* goes low), the unit sends an acknowledge, ACK_i, to the appropriate IPM, completing the left handshaking communication. As each subsequent body flit of the packet arrives, as long an acknowledgment ACK_{OUT} has been received for the previous flit, its data $DATA_i$ propagates directly through the multiplexer and data register, and its request REQ_i propagates directly through the corresponding latch (L1 to L4), to the output channel.

Packet transmission ends after the tail flit arrives. When the flit is sent on the output channel, the $Tail\ Passed_i$ signal (asserted high) is sent to the source IPM, along with the transition-signaling acknowledge, ACK_i. Once asserted, $Tail\ Passed_i$ will also cause the corresponding request latch (L1 to L4) to become opaque. In turn, the corresponding IPM will deassert $Packet\ Path\ Enabled_i$, thereby releasing the mutex, and the Tail Detector then deasserts $Tail\ Passed_i$.

3.4 4-Input Mutex Design

The microarchitecture of the new 4-input mutex is presented in Fig. 5.5. While a previous widely-used 4-input mutex design [20, 64] uses 6 two-input mutex elements and has a serial critical path through 3 mutex elements, the proposed solution uses 3 two-input mutex elements and has a critical path through only 1 mutex element.

In this design, the left mutex element arbitrates between requests 0 and 1, the right mutex element arbitrates between requests 2 and 3, and the center mutex element arbitrates between requests from the right and left pairs. C-elements are used to synchronize the operation of the middle mutex with the side ones, both during the acquire and release phases. Whenever a grant is given, any other request coming from the channel on the same side of the winning one will be killed. The rationale is that, when the winning request will be deasserted, the middle mutex has to be released, so no other requests must be coming from the same side. This behavior provides fair arbitration between incoming requests: in fact, requests from the other side will now have an advantage in acquiring the middle mutex. In other words, the policy implemented is a round robin between left and right side, and round robin between requests within the same side.

3.5 Transition-Signaling Circular FIFO

FIFOs can be useful to provide additional storage capacity, so to improve system-level performance. Multiple MOUSETRAP registers can be placed one after the other, so to build a serial FIFO, but this introduces severe latency penalty. To overcome this issue, a new circular FIFO is here proposed. Unlike [14], which uses a bus-based interface, the proposed design can provide much lower latency and cycle time.
The microarchitecture of the transition-signaling circular FIFO is shown in Fig. 5.6. The FIFO uses a two-phase protocol with single-rail bundled data. Write and Read Pointers are 1-hot level signals, selecting the active Write or Read Control Blocks. A new transfer begins when new Data arrives, stabilizes and is then followed by Req_{IN} signal assertion (high in the case under analysis). A new data can be stored in a buffer slot only if: (a) the write pointer selects that slot, and (b) that slot is empty ($Full_i$ is at the same logic level of $Empty_i$). This condition is detected in the Write Control Block by an AND gate merging the two conditions, as can be seen in Fig. 5.7(a). The 2-input XOR in the Write Control block implements a phase conversion, to provide the correct polarity of the Request signal at the input of the Request Latch, similar to the strategy used in the $PacketRouteSelector$ in the Input Port Module. The new request causes the active Write Control block to assert $Full_0$, close the corresponding register to store the coming

Data (by deasserting En_0), and assert the acknowledge to the input. This acknowledge will be merged by the N-input XOR to generate the Ack_{OUT} signal upstream. Finally, when Req_{IN} and Ack_{OUT} are at the same logical value, the Write Counter selects the following buffer position for the next operation. This concludes data storage.

When $Full_0$ and $Empty_0$ are at different logic levels, it means that new data is available in the corresponding buffer position. If this is the case and the read pointer selects that position, the Read Control Block (see Fig. 5.7(b)) will assert a Request, which will be merged with all the other signals outgoing from the other Read Blocks to generate the output Request Req_{OUT}, together with data from the selected position. After this event (now Req_{OUT} and Ack_{IN} are at different logic values), the $ReadPointer_0$ is immediately deasserted, to safely freeze the current value of the output request. In more detail, request latches inside the Read Control Blocks are now closed, while the active Read Control Block makes its internal acknowledge latch transparent, to route the incoming signal to the correct buffer position. Next, an acknowledge signal comes from the downstream environment, causing $Empty_0$ to make a transition and $ReadPointer_1$ to be asserted high, thereby concluding data transmission.

This FIFO is an important element, not yet exploited in this thesis, but necessary when considering complete NoC topologies where larger queues might be required.

4 Virtual Channel Links

In this section the approach we exploit for implementing Asynchronous Virtual Channels is presented. The basic idea is to replicate the baseline switch N times, one for each VC, and share the same physical link between all the output ports associated with the same channel. Arbitration takes place on a per-flit basis, so that whenever a Virtual Channel is blocked, the other can proceed. Out target architecture exploits transition-signaling bundled data protocol, with two-phase control signaling. The reference architecture is shown in Figure 5.8.

The operation is the following: an output port of a switch request access to the

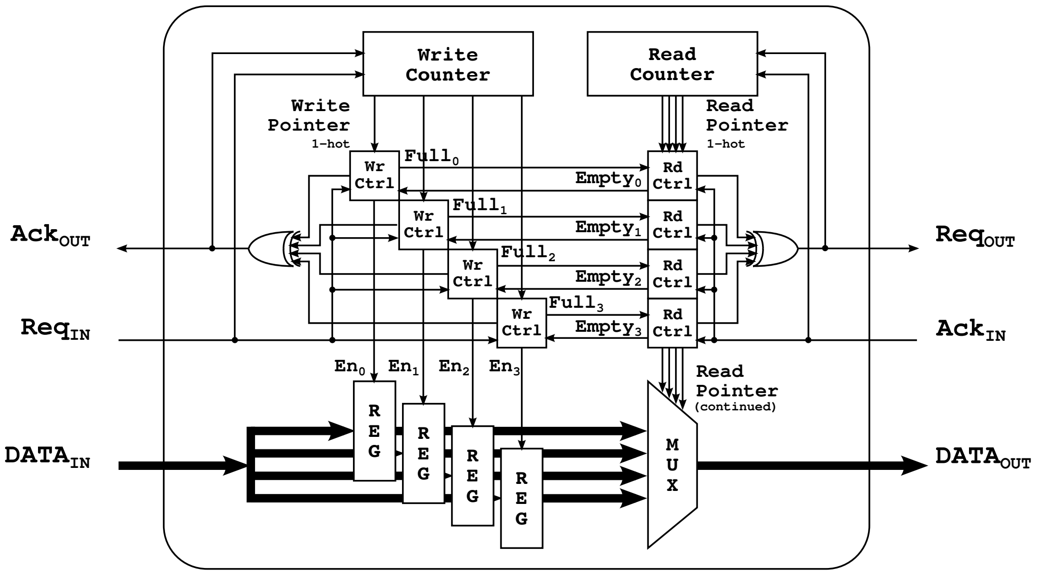
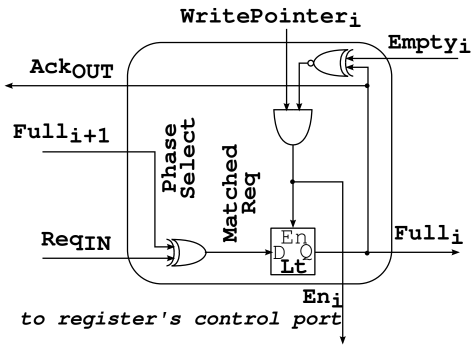
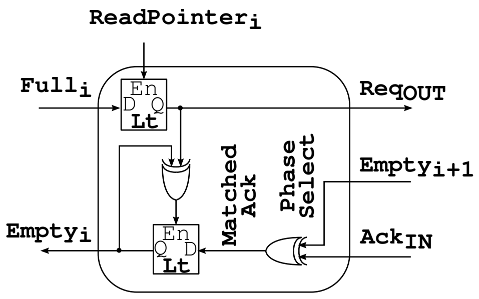


Figure 5.6: Circular FIFO: top-level view



(a) Write Control Block



(b) Read Control Block

Figure 5.7: Schematic of Write and Read Control Blocks

channel arbiter, which will allow transmission to the downstream connected switch. Once the flit is stored at the receiver (Ack_{link} makes a transition) the link is freed, but not the VC. The Sender Port is “fooled”, since it will not send the following flit until the associated Virtual Channel is freed, i.e. the previously transmitted flit has been accepted inside the switch (Ack_{VC} makes a transition).

While a similar approach has been widely used in the open literature [12,78], the main limitation of this solution is the huge roundtrip that is being created

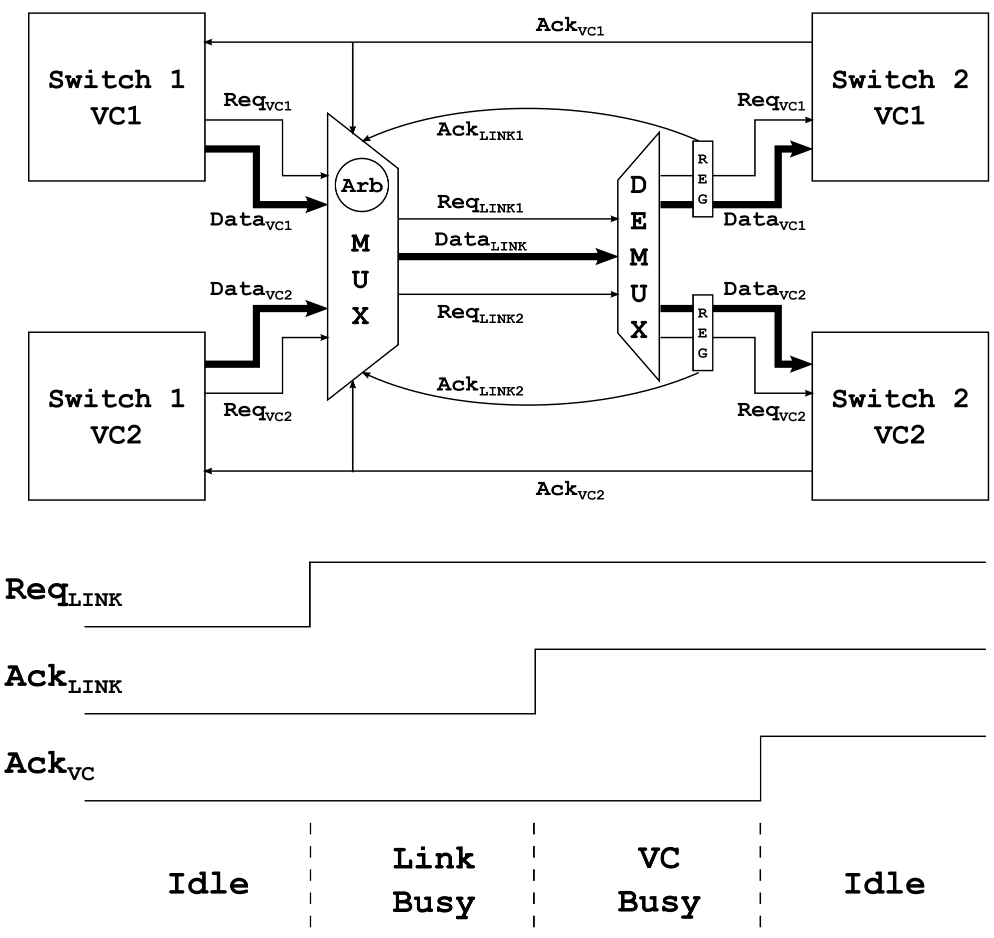


Figure 5.8: Virtual Channel, initial idea

on the link. Each request must traverse the entire link, reach the destination, and each acknowledge must make its way back. As we saw, asynchronous links require pipeline stages not to degrade performance. Pipelining is not straightforward in this case: it is useless to have pipelines only on the physical link, since the sender will not provide the following flit until the Virtual Channel will be freed (a transmission back from the receiver to the sender must take place). In addition, it is non-trivial to design pipelines considering the particular handshake mechanism implemented (each channel has one Request and two separate Acknowledges, and share the same physical Data wires). Furthermore, a channel must not block the other, so link pipeline stages architecture and communication protocol must enforce this behavior.

4.1 First Solution: Credit-based flow-control

One possible solution to tackle these problems is to implement an Asynchronous Credit-based Flow-Control. This solution is presented in Figure 5.9.

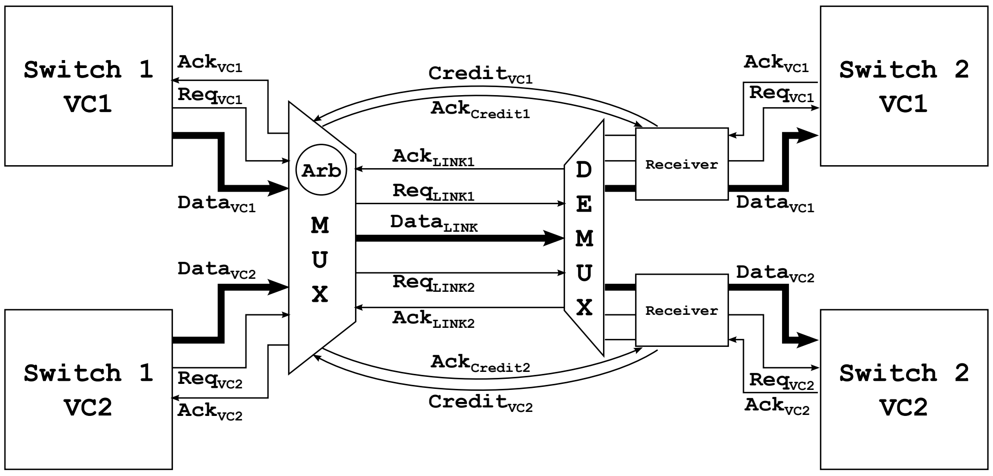


Figure 5.9: Virtual Channel, Credit-based

No end-to-end communication is present anymore. Each Virtual Channel requires 4 control lines, two (*Req* and *Ack*) are required for flit transmission downstream, and two (*Credit* and Ack_{Credit}) transmit the Credit upstream. The Receiver block has buffering resources. When a channel wants to transmit a flit, the arbitration block first checks if the receiver has space to accept it, to make sure that this flit will not be stalled in the pipeline stages of the physical link. If this condition is true, the flit has access to the link. Any number of pipeline stages on the link is tolerated, and at the end the flit will be stored inside the Receiver. When a flit is accepted inside the downstream switch, a slot position will be freed inside the Receiver and a Credit will be sent back to the sender. The credit propagation is performed just like req-ack flow control, but with no data associated to it.

Blocks detail

In this subsection the various building blocks of this solution are described. **Arbitration module** (Figure 5.10) is an extension of the arbitration primitive [34]. Virtual Channel counters have been added, in order to filter incoming requests: they increment when a flit is transmitted (i.e. a request makes a transition) and decremented when a credit is received. If they reach their maximum capacity, equal to the capacity of the Receiver, they will block any further incoming request. Also, the output latch stage is slightly different from your solution, in order to keep the two virtual channel requests separated.

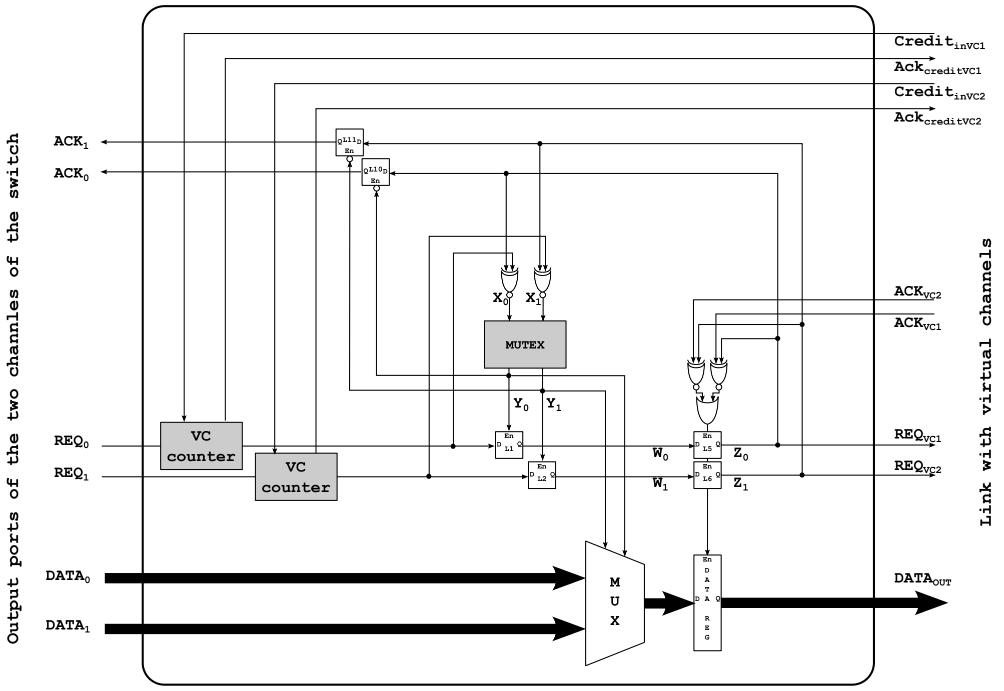


Figure 5.10: Virtual Channel Credit-based, Arbitration Module

In Figure 5.11 is shown the **Virtual Channel Counter**. It is implemented as a Muller Pipeline, where tokens injected from the left side do not flow directly to the right side, but they need an explicit “pull” to be removed from the pipeline (the input of the last C-element is not inverted). Tokens are injected by Request signal, while the received Credit is the “pull”.
To implement the **Receiver module** a simple fork-join of the control signals between the switch and the Fifo is sufficient (Figure 5.12).
The **Pipeline stage** that supports Virtual Channels is depicted in Figure 5.13. It is an extension of Mousetrap Pipeline, only any request can arrive with the associated data and make the register opaque. If a Req makes the register opaque, no other Virtual Channel can have access to the pipeline

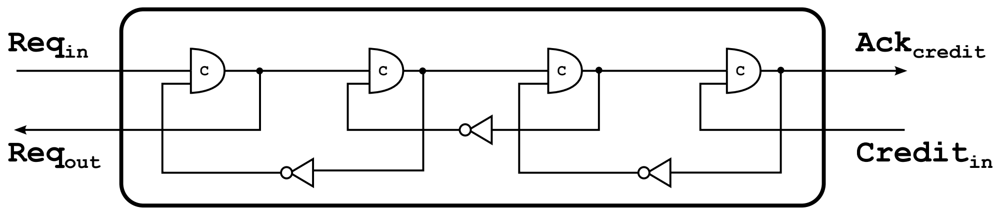


Figure 5.11: Virtual Channel Credit-based, Virtual Channel Counter

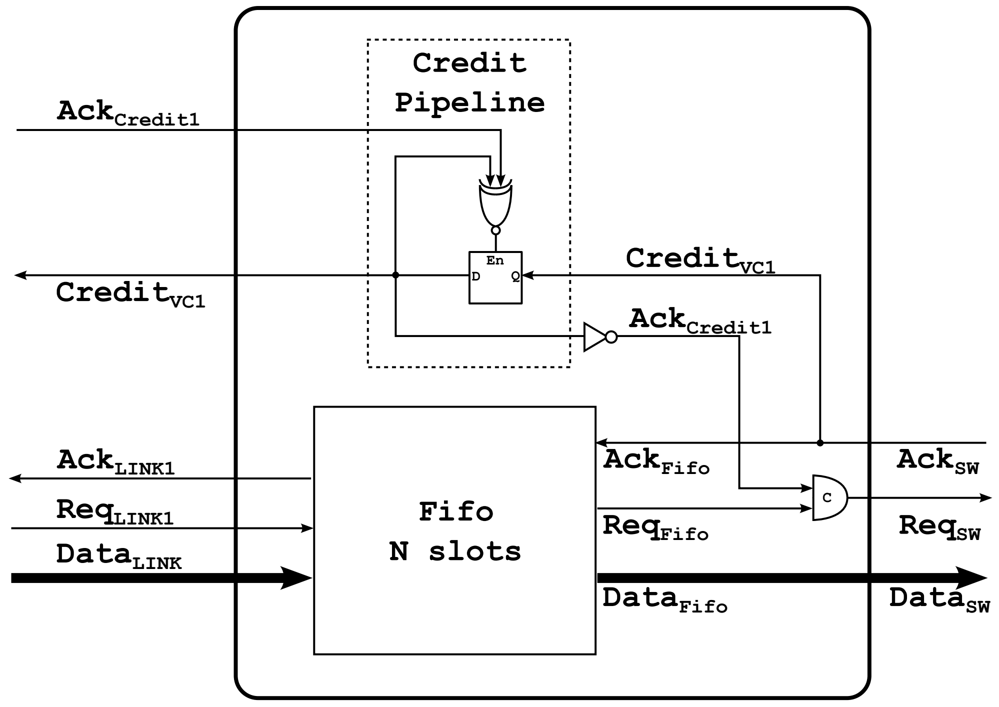


Figure 5.12: Virtual Channel Credit-based, Receiver Module

until the Acknowledge of the active transmission is being received.

4.2 Second Solution: Non-Blocking pipeline stages

A different solution can be represented by pipeline stages that do not block a virtual channel when the other is busy. This require that each pipeline stage can keep (at least) a flit belonging to every Virtual Channel. This solution avoids the need for Credit-based operation, but requires arbitration to be performed at every pipeline stage in order to decide which channel can have access to the physical link, in case of contention.

In order to allow a stage to be non-blocking, two different backward control

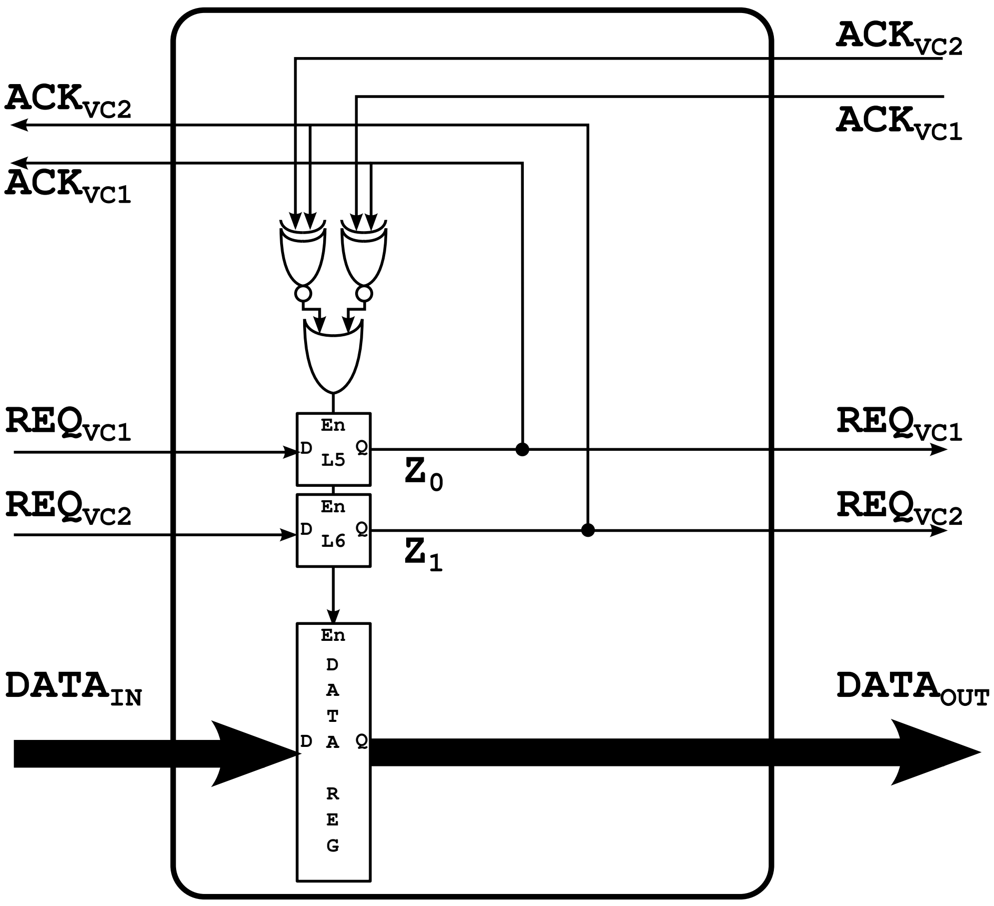


Figure 5.13: Virtual Channel Credit-based, Pipeline Stage

signals are used (exactly as shown in Figure 5.8). ACK_{LINK} instructs the sender that the transmitted information has been stored (frees the physical link), while ACK_{VC} informs the sender that the information is being sent to the downstream block (the buffer position associated to the virtual channel is again empty).

Blocks detail

Arbitration module: figure 5.14. The arbitration module is again similar to the arbitration primitive of your NoCS'10 paper. A flit of one of the two VCs can have access to the mutex if the virtual channel is free (the C-elements synchronize Req_{VC} and Ack_{VC}, preventing a new request to propagate to the mutex input if acknowledge from the virtual channel has not been received).

A flit that has been granted access by the mutex to the channel can access the physical link if Ack_{LINK} has been received from the downstream block (there are no valid transmission in the physical channel). Note that since a flit can have access to the physical link only if the downstream block has notified that the buffer position corresponding to the virtual channel has been freed, the flit will not be blocked between the two blocks (it will not bee stalled in the physical channel).

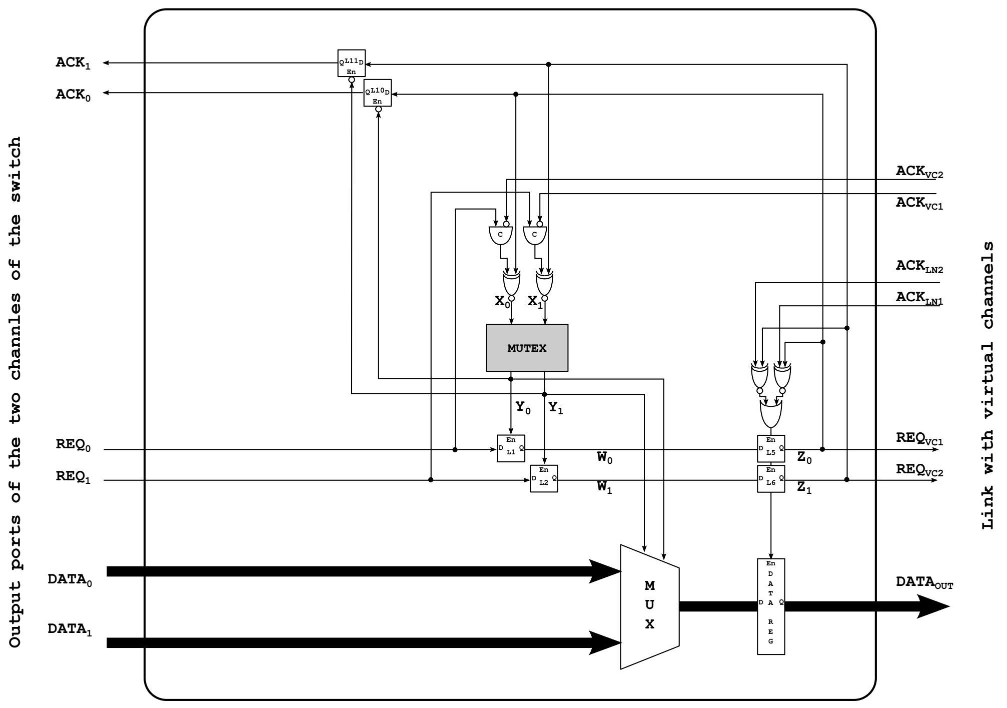


Figure 5.14: Virtual Channel Non-Blocking, Pipeline Stage

Pipeline stage: figure 5.15. The operation of this module is similar to the virtual channel arbiter. The main differences are the two mousetrap buffer stages at the input of the module that manage handshaking along the physical channel. Each buffer can store a flit from a different virtual channel.
Suppose that a flit from VC1 arrives at the input of the pipeline. Once the flit is stored, the physical channel is released, allowing transmission on the other virtual channel, while VC1 is still busy. Now, the newly stored flit can require access to the mutex to the output channel. Just like the arbitration module, the flit can have access to the output channel if the virtual channel is free. The implementation of the output register is slightly different, but the

functionality is exactly the same: we wanted to remove the cascade of two D-latches, that we found a bit redundant (see figures 5.14 and 5.15). when the flit is stored in the output register, the input pipeline becomes transparent again and the virtual channel is freed.

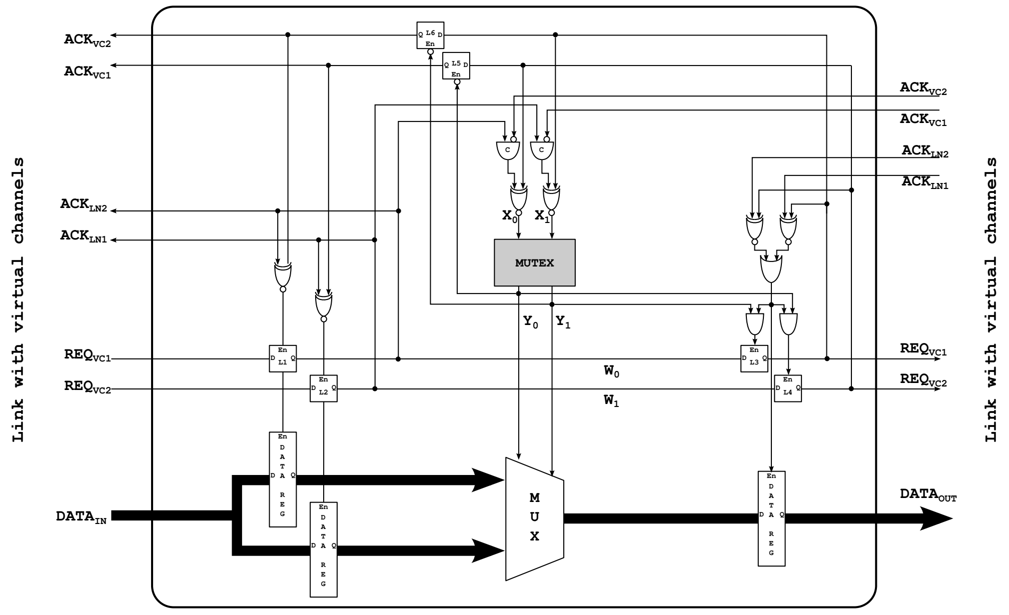


Figure 5.15: Virtual Channel Non-Blocking, Pipeline Stage

Receiver: figure 5.16. The implementation of the receiver is straightforward: these are just two separate mousetrap pipelines that accepts data from the associated virtual channel.

4.3 Timing Constraints

To work properly, the proposed architecture requires some one-sided timing constraints to be enforced.

One is the matching delay inside the *PacketRouteSelector* block, in order to provide glitch-free operation.

A second constraint is in the Output Port Module: data must be stable long enough before latches are closed inside the output register, i.e. to meet the latches' setup time. This constraint applies to both head flit path setup and body flit propagation, and requires constraining the control path (latches L1-4 and XOR gates) over the multiplexer delay.

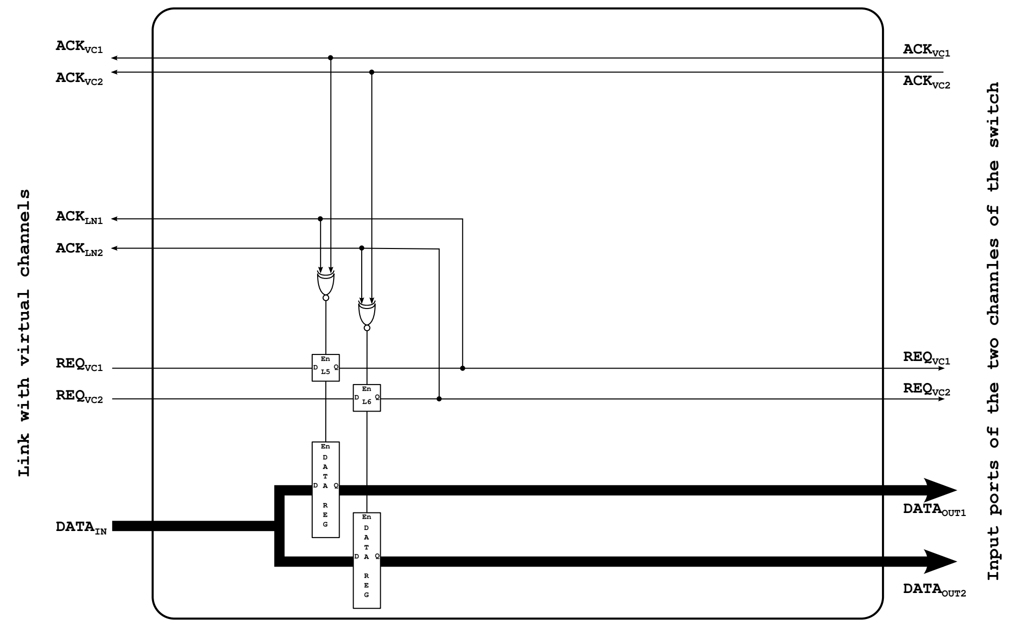


Figure 5.16: Virtual Channel Non-Blocking, Pipeline Stage

Finally, a subtle failure condition can occur during mutex release after a tail flit has passed. The first path starts in the Output Port Module, when acknowledge is generated. This path goes to the Input Port Module, through the acknowledge merge block, through the input register control gate, making the register transparent and allowing a request eventually pushing at its input to enter the module, to propagate through the *RequestControl* Block, and to reach the Output Port Module. The AND gates above latches L1-4 ensures that once *TailPassed* signal is asserted, these are closed when the new request comes, reducing the otherwise longer path through *PacketPathEnabled* deassertion and mutex release.

Other relative timing constraints exist inside the 4-input mutex, the Request Control Block and the circular FIFO, but they are typically satisfied in normal operation of the switch.

5 Semi-Automated Design Flow

All the previously mentioned constraints, plus other delay requirements needed to increase performance, have been enforced across all the steps from logical synthesis to layout design by means of mainstream CAD tools in a semi-automated design methodology. We use a Low-Power Standard-Vt 40nm Industrial Technology library, Normal Process, 1.2V Supply Voltage and 300K Nominal Temperature.

Entry level - The various blocks have been described with low-level RTL models: their functionality has been specified using logical operators, with only few exceptions when implementing specific asynchronous cells. Our technology library does not include C-elements nor mutexes, therefore we use their standard-cell equivalent implementations [55, 81].

Logic Synthesis - The design is synthesized and mapped to the target library using the Synopsys Design Compiler. When using asynchronous design style, not only functionality, but also dynamic behavior is important. In order to ensure glitch-free operation, the tool must be prevented from applying logic optimizations to the design. On the other hand, automatic buffer insertion is a useful optimization option that we would like to exploit. This behavior can be achieved by using the *set_compile_directives* and *set_structure* directives of the tool. While Design Compiler does not understand relative timing constraints, they can be enforced through multiple iterations: in a first run, only max delays are enforced, in order to meet a generic target (max performance, minimum area); then, in a second run, the delay of the paths that have to be matched can be extracted from the netlist (*get_timing_path*, *get_attribute* timing_path *arrival*) and assigned to the required path (*set_min_delay*). The same procedure can be used to check whether the given constraints have been fulfilled, and iterate again if necessary.

Physical Switch Design - For this purpose we used Synopsys IC Compiler. As in the technology mapping procedure, it is possible to enforce, extract and compare the delays between different paths, and automatically verify if constraints have been fulfilled. If not, it is possible to update constraints and iterate place and route again.

Top-Level Implementation - Again, the Synopsys IC Compiler functionality is leveraged. The placed and routed switch netlist is saved as a hard macro and then instantiated in the top level description of the system. This is a typical hierarchical design methodology, which achieves reduced runtimes and faster convergence. Since this hard macro is generated with the standard methodology described above, it still presents some degrees of freedom: data width is parameterizable, as well as its floorplan aspect ratio and the position of input-output ports. For this reason, this is different from a fixed hard macro designed with a full custom approach.

In order to route interconnection wires between any two switches, we choose to give maximum delay constraints over the link, while keeping the same max capacitance and max transition time constraints given inside the switch. After a first routing and buffer insertion, we freeze placement and driving strength of data wires along them (so that their delay will no longer change), and give minimum delay constraints over the request signal to satisfy the bundled data protocol requirement (i.e. a request must arrive always after the corresponding data has stabilized). Then, incremental physical optimization and routing are performed. A different approach has been adopted when implementing links with pipeline stages. A pipeline stage can help reduce cycle time over the link, at the cost of some additional forward latency. As before, switches were implemented as hard macros and placed a few millimeters apart. Architectural repeaters (i.e. MOUSETRAP FIFO stages) are described as soft macros, inferring their placement boundaries in order to have a regular floorplan and guide tool decisions. The tool is then allowed to size pipeline cells and insert buffers to reduce propagation delay.

Even though the previously described methodology could produce a valid outcome, the performance results were not satisfactory. This is because simply setting of max delays on wires does not take into account the internal timing arcs of the latches. This issue prevents the tool to correctly estimate the time spent on link traversal. In order to optimize our procedure, we exploited a similar method to [78], adapting it for the single-rail bundled data case. Through IC Compiler commands, we disabled the path through the feedback loop of register control logic, from input to output pin of latches inside repeaters, and defined the reset signal as a dummy clock. This approach makes latches similar to flip flops, and the clock period gives a constraint on the maximum forward delay. This leaves the acknowledge signal propagating back from each pipeline stage to the previous one unbounded (we disabled the timing path through register control logic), so a maximum delay constraint has been enforced on it in order to speed the path through it. Again, multiple iterations are required to enforce bundling constraint on a request signal. We found that this approach is more suitable when dealing with pipelined links, since it reduces tool run times and the number of iterations required to satisfy matching constraints with respect to the previous methodology.

6 Experimental Results

In order to evaluate our implementation choice, we compared the proposed 5x5 asynchronous switch with the baseline fully synchronous ×pipesLite switch.

Experimental Setup

While constraints required for correct functionality can be checked and fixed during synthesis and place and route procedures, performance evaluation must be assessed through simulation. In order to assess performance of the asynchronous switch, the following experimental setup has been exploited. The switch under test receives packets from injector modules and forwards them to absorber modules; injector and absorber are simply other instances of the same switch, with fast loops at their boundaries (i.e., maximum injection rate). This environment permits to assess the realistic handshaking mechanism between neighboring switches, thereby avoiding overly-optimistic results.

Latency metric is evaluated as the time interval from a request being asserted at the input port to a request asserted at the output port, assuming the switch is initially empty and there is no congestion. Latency varies depending on the flit position inside the packet. Head flits experience the highest latency since they have to set up the path from input to output port. Latency will be evaluated focusing on head flits, since they are responsible of packet propagation through the network. **Cycle time** is the interval between two successive acknowledgments received at the switch output port.

As far as the synchronous switch is concerned, the implementation allocates one clock cycle to traverse the switch and one clock cycle to traverse the link connecting two switches. For this reason, cycle time is one clock period, while latency amounts to two clock periods.

	Asynchronous	Synchronous
Area	4691 μm^2	16035 μm^2
Latency (Head Flit)	1195 ps	1960 ps
Cycle Time (Avg.)	903 ps	980 ps
Area Efficiency	236 $Mfps/mm^2$	63.63 $Mfps/mm^2$

Table 5.1: Asynchronous vs. Synchronous Switch

Area efficiency is a metric that evaluates how much area is necessary to provide a defined performance. It is calculated as:

$$AreaEfficiency = \frac{DeliveredThroughput}{AreaOccupancy} \left[\frac{Mfps}{mm^2}\right]$$

Comparative Analysis

Table 5.1 presents the measured metrics. The asynchronous switch requires 71% less area, and delivers 39% lower latency and comparable throughput. For this reason, the Area Efficiency metric is 3.7x higher for the asynchronous implementation. It should be observed that part of the area savings come from the different minimum buffering requirements of the two switches. Both of them are latched at inputs and outputs. However, the synchronous switch needs two slot buffers to properly support the stall/go flow control protocol (i.e., not to lose data upon stall assertion). Vice versa, flow control is inherently implemented in the asynchronous clockless handshaking, therefore only a single slot latching stage is sufficient for the asynchronous switch.

NoC Link Effect

A typically overlooked issue is the effect of the inter-switch links on performance, which are assumed ideal in Table 5.1. In order to account for this aspect, we implement a complete layout of two switches, placed a few millimeters apart, and route interconnections between them.
Fig. 5.17 shows how head latency and cycle time degrade for synchronous and asynchronous switches when varying the inter-switch distance. Having an entire clock cycle reserved for link traversal, the synchronous switch maintains a stable performance up to 4 mm link length. From there on, the critical path

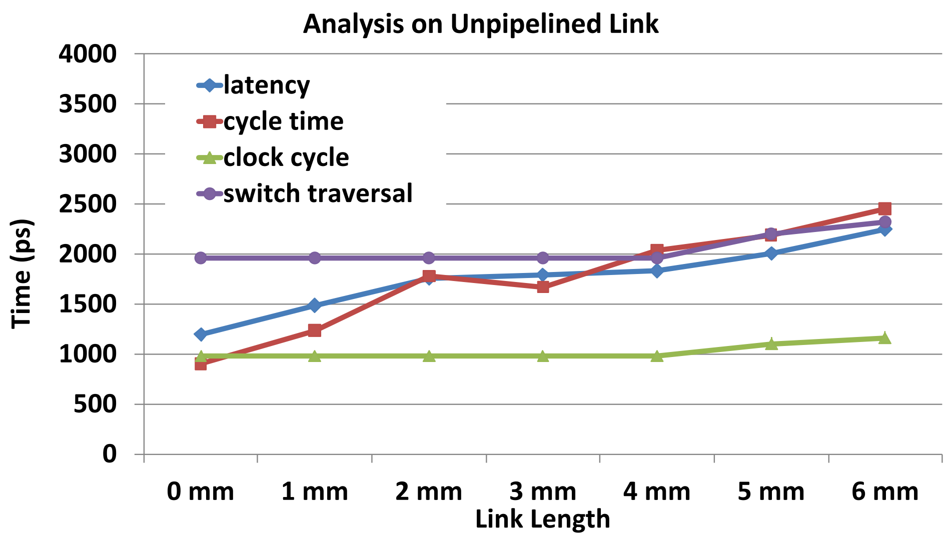


Figure 5.17: Performance results with unpipelined links

moves from inside the switch into the link, thus reducing the maximum clock frequency. On the other hand, performance of asynchronous switch gracefully degrades when increasing link length. Latency is always lower when compared with the synchronous counterpart, while cycle time has a steeper degradation, due to the signal roundtrip over the interconnect.

It is well-known that link pipelining is the way to reduce cycle time at the expense of additional latency. This holds for both synchronous and asynchronous design styles, but the implications are completely different. Adding a pipeline stage in synchronous logic always implies one additional clock cycle latency and full retiming and flow control stages, while in asynchronous logic the latency is affected by only a few gates delay since pipeline stages consist of simple latching stages.

Exploiting the procedure described in section 5, effects of link pipelining are analyzed. The number of pipeline stages is selected in order to obtain the same cycle time as the one measured for ideal link. As shown in Fig. 5.18, the additional pipeline stages negatively affect asynchronous latency, although for link lengths below 2.5 mm performance is always better or equal to that of the synchronous switch. For longer link lengths, the asynchronous solution suffers the most from link parasitics due to the roundtrip delay of its handshaking protocol.

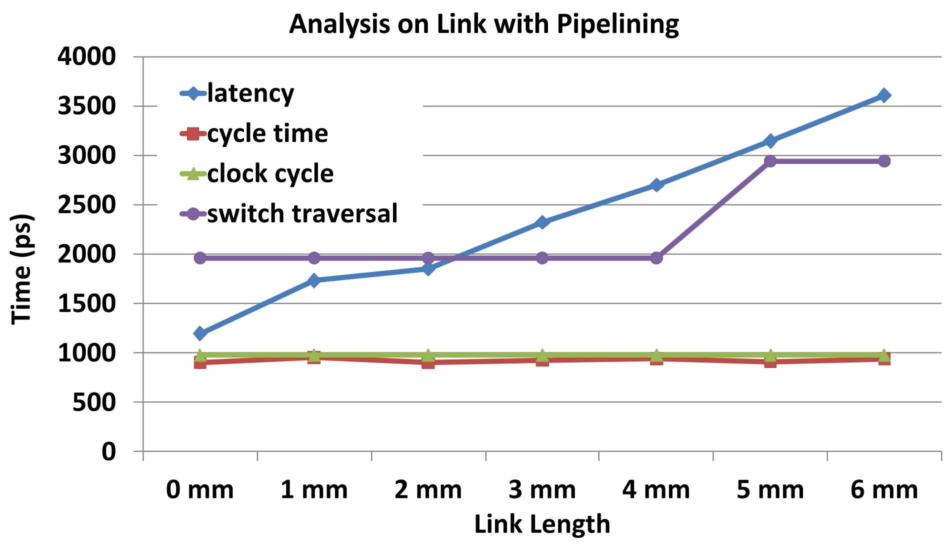


Figure 5.18: Performance results with link pipelining

Power Analysis

Power consumption of the switches is analyzed under different conditions, using Prime Time on post-layout implementations. For these experiments, the synchronous switch has been implemented in two versions, one without and one with clock gating.

Four different case analysis are presented in Fig. 5.19. *Leakage power* is smaller in asynchronous design, given the reduced area occupancy. *Idle power* is analyzed considering clock tree power consumption: even if clock gating technique can provide evident benefits with almost no performance penalty with respect to baseline synchronous switch, the asynchronous switch, being completely clockless, requires 90% and 97% less power than clock gated switch and baseline switch, respectively. The remaining two cases use two different traffic benchmarks, with packets composed by 3 and 8 flits. For *hotspot* (a single output channel is accessed by packets coming from every other input channel), the asynchronous switch has an average of 85% less power requirement than synchronous and and 73% less than synchronous with clock gating. For the *parallel* benchmark (all input ports competing for 5 different output ports, no contention), power consumption is reduced on average by 60% with respect to synchronous and 45% with respect to synchronous with clock gating. Overall the small difference between 3 and 8 flits

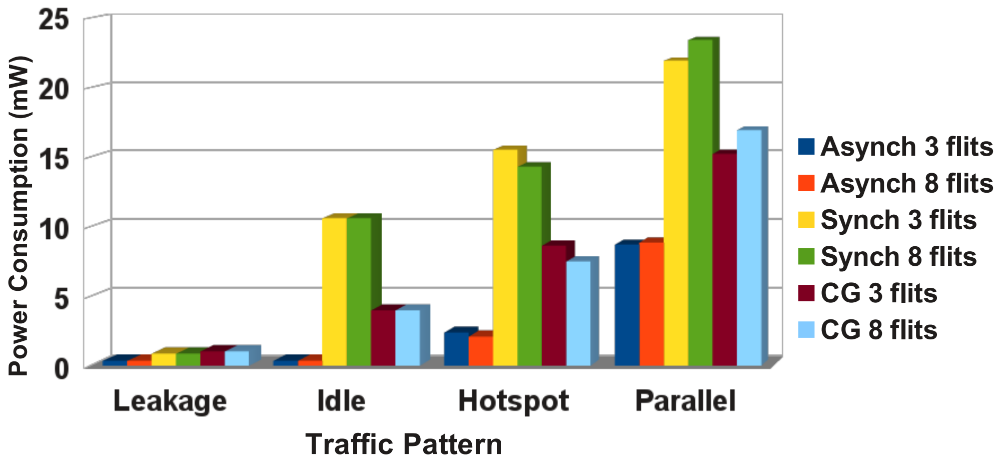


Figure 5.19: Power consumption of the different switch architectures, varying the traffic injected

packets is due to the extra power required during arbitration process. Power savings not only come from the clockless architecture of the asynchronous switch but also from its lower complexity and footprint. In fact, observing the energy-per-flit in Fig.5.20, there is a 44% average reduction with respect to the two synchronous switches.

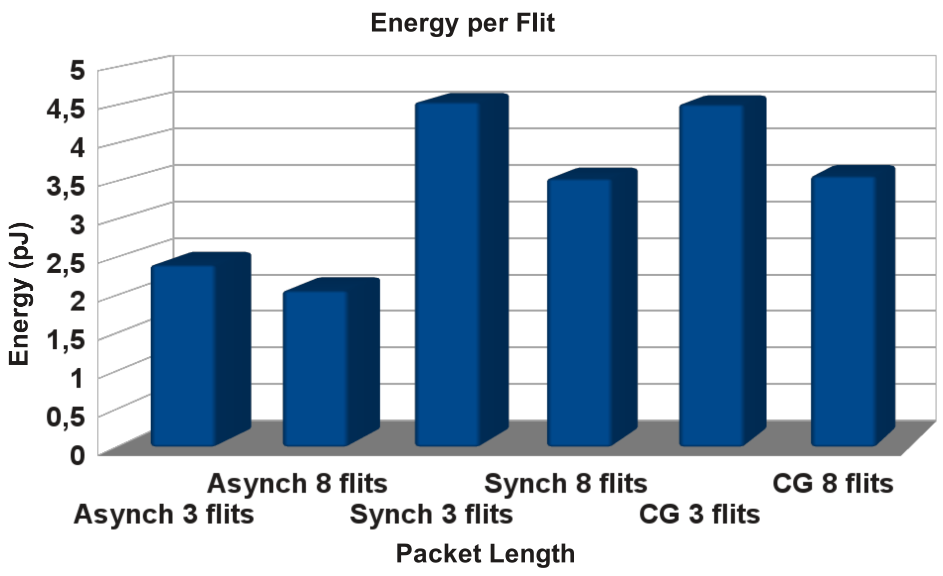


Figure 5.20: Average energy required to propagate a Flit form input to output

7 Conclusions

This thesis delivers a largely unexplored design point in asynchronous NoC switch architectures. It relies on transition-signaling bundled data to produce a low overhead design, which at the same time meets the performance of synchronous counterparts. Post-layout results indicate that area is reduced by 71%, idle power by 90%, and energy-per-flit by 45%. Throughput is roughly comparable with the synchronous switch, while latency is actually better up to link lengths of 2.5mm in 40nm technology. Overall area efficiency is superior (3.7x). Finally, the switch is delivered as a partially-reconfigurable hard macro for hierarchical design flows. Timing constraints are tightly controlled through a semi-automatic design flow relying on mainstream synchronous CAD tools.

Chapter 6

Conclusions

This work documents the effort of providing design technologies able to tackle the power requirements of Network-on-Chip infrastructures.

With this in mind, selective challenges of nanoscale technologies have been considered, in order to assess the energy cost of each solution.

As far as fault tolerance is concerned, we did not just engineered design methodologies for switch architecture, but we matched system-level infrastructure with local features of the switches. In this way, we found out that when targeting single-event-upsets, error correction techniques are often less power-hungry than error detection and recovery solutions. In addition, we proposed an effective and low-complexity design technique for addressing system-level diagnosis and reconfiguration.

At the same way, when dealing with the generation of application-specific SoCs, the global interconnect cannot be treated as any other monolithic block. The consequence would be a sub-optimal network infrastructure that would hardly meet communication requirements and whose implementation would not be able to reach the required performance and/or power consumption specifications. In this thesis, a tool flow for the generation of an efficient Network-on-Chip topology is described, exploiting the most suitable design methodology for fault-tolerance for irregular topologies as a test case. Thanks to early floorplan and topology exploration, we are able to rank designs upfront with respect to relevant metrics such as power consumption hot-spots and communication traffic locality. Fast convergence and predictable results of this methodology has been proven by demonstrating correlation of design

choices across the design hierarchy and accuracy of abstract models used in early phase analysis and synthesis.
Finally, the most power effective solution analyzed in this thesis results to be asynchronous design. In fact, results of this design methodology applied to switch architecture show up to 90% reduction in overall power consumption, and a 45% average reduction in energy-per-flit, thanks in particular to the complete removal of clock tree and associated switching activity, and intrinsic flow-control capability that simplifies control logic.
However, even though asynchronous design technology is becoming more and more appealing as clock distribution reaches its physical limits, this design style is still far from being a solid solution, since procedures for complete platform composition of clockless designs are not fully stable yet, even though we believe this thesis described the initial steps towards this direction.

Bibliography

[1] Nidhi Aggarwal, Parthasarathy Ranganathan, Norman P Jouppi, James E Smith, Kewal K Saluja, and George Krejci. Motivating commodity multi-core processor design for system-level error protection. In *Workshop on Silicon Errors in Logic-System Effects (SELSE)*, 2007.

[2] A. Alaghi, M. Sedghi, N. Karimi, M. Fathy, and Z. Navabi. Reliable noc architecture utilizing a robust rerouting algorithm. In *Design Test Symposium (EWDTS), 2008 East-West*, pages 200–203, 2008.

[3] M. Ali, Michael Welzl, and S. Hellebrand. A dynamic routing mechanism for network on chip. In *NORCHIP Conference, 2005. 23rd*, pages 70–73, 2005.

[4] A.M. Amory, E. Briao, E. Cota, M. Lubaszewski, and F.G. Moraes. A scalable test strategy for network-on-chip routers. In *Test Conference, 2005. Proceedings. ITC 2005. IEEE International*, pages 9 pp.–599, 2005.

[5] F. Angiolini, D. Atienza, S. Murali, L. Benini, and G. De Micheli. Reliability support for on-chip memories using networks-on-chip. In *Computer Design, 2006. ICCD 2006. International Conference on*, pages 389–396, 2006.

[6] K. Arabi. Logic bist and scan test techniques for multiple identical blocks. In *VLSI Test Symposium, 2002. (VTS 2002). Proceedings 20th IEEE*, pages 60–65, 2002.

[7] ARTERIS. http://www.arteris.com.

[8] John Bainbridge and Steve Furber. Chain: a delay-insensitive chip area interconnect. *IEEE Micro*, 22(5):16–23, 2002.

[9] E. Beigne, F. Clermidy, P. Vivet, A. Clouard, and M. Renaudin. An asynchronous noc architecture providing low latency service and its multi-level design framework. In *Asynchronous Circuits and Systems, 2005. ASYNC 2005. Proceedings. 11th IEEE International Symposium on*, pages 54–63, 2005.

[10] Davide Bertozzi, Luca Benini, and Giovanni De Micheli. Error control schemes for on-chip communication links: the energy-reliability tradeoff. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 24(6):818–831, 2005.

[11] Davide Bertozzi, Antoine Jalabert, Srinivasan Murali, Rutuparna Tamhankar, Stergios Stergiou, Luca Benini, and Giovanni De Micheli. Noc synthesis flow for customized domain specific multiprocessor systems-on-chip. *Parallel and Distributed Systems, IEEE Transactions on*, 16(2):113–129, 2005.

[12] Tobias Bjerregaard and Jens Sparso. A router architecture for connection-oriented service guarantees in the mango clockless network-on-chip. In *Design, Automation and Test in Europe, 2005. Proceedings*, pages 1226–1231. IEEE, 2005.

[13] Daniel M Chapiro. Globally-asynchronous locally-synchronous systems. *Ph. D. Thesis*, 1:50, 1984.

[14] Tiberiu Chelcea and Steven M Nowick. Low-latency asynchronous fifo's using token rings. In *Advanced Research in Asynchronous Circuits and Systems, 2000.(ASYNC 2000) Proceedings. Sixth International Symposium on*, pages 210–220. IEEE, 2000.

[15] Caroline Concatto, Debora Matos, Luigi Carro, Fernanda Kastensmidt, Altamiro Susin, Erika Cota, and Marcio Kreutz. Fault tolerant mechanism to improve yield in nocs using a reconfigurable router. In *Proceedings of the 22nd Annual Symposium on Integrated Circuits and System Design: Chip on the Dunes*, page 26. ACM, 2009.

[16] Kypros Constantinides, Stephen Plaza, Jason Blome, Bin Zhang, Valeria Bertacco, Scott Mahlke, Todd Austin, and Michael Orshansky. Bulletproof: A defect-tolerant cmp switch architecture. In *High-Performance Computer Architecture, 2006. The Twelfth International Symposium on*, pages 5–16. IEEE, 2006.

[17] Matteo Dall'Osso, Gianluca Biccari, Luca Giovannini, Davide Bertozzi, and Luca Benini. Xpipes: a latency insensitive parameterized network-on-chip architecture for multi-processor socs. In *Computer Design (ICCD), 2012 IEEE 30th International Conference on*, pages 45–48. IEEE, 2012.

[18] William J Dally, Larry R Dennison, David Harris, Kinhong Kan, and Thucydides Xanthopoulos. Architecture and implementation of the reliable router. In *Hot Interconnects II*, pages 197–208. Citeseer, 1994.

[19] William James Dally and Brian Patrick Towles. *Principles and practices of interconnection networks*. Elsevier, 2004.

[20] Rostislav Reuven Dobkin, Ran Ginosar, and Avinoam Kolodny. Qnoc asynchronous router. *Integration, the VLSI Journal*, 42(2):103–115, 2009.

[21] D. Fick, A. DeOrio, Jin Hu, V. Bertacco, D Blaauw, and D Sylvester. Vicis: A reliable network for unreliable silicon. In *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, pages 812–817, 2009.

[22] Communication Exchange Format (CEF) format. www.nanoc-project.eu.

[23] Arthur Pereira Frantz, Fernanda Lima Kastensmidt, Luigi Carro, and Erika Cota. Dependable network-on-chip router able to simultaneously tolerate soft errors and crosstalk. In *Test Conference, 2006. ITC'06. IEEE International*, pages 1–9. IEEE, 2006.

[24] Daniel Gebhardt, Junbok You, and Kenneth S Stevens. Comparing energy and latency of asynchronous and synchronous nocs for embedded

socs. In *Proceedings of the 2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip*, pages 115–122. IEEE Computer Society, 2010.

[25] A. Ghiribaldi, H.T. Fankem, F. Angiolini, M. Stensgaard, T. Bjerregaard, and D. Bertozzi. A vertically integrated and interoperable multi-vendor synthesis flow for predictable noc design in nanoscale technologies. In *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*, pages 337–342, Jan 2014.

[26] Alberto Ghiribaldi, Daniele Ludovici, Michele Favalli, and Davide Bertozzi. System-level infrastructure for boot-time testing and configuration of networks-on-chip with programmable routing logic. In *VLSI and System-on-Chip (VLSI-SoC), 2011 IEEE/IFIP 19th International Conference on*, pages 308–313. IEEE, 2011.

[27] Alberto Ghiribaldi, Daniele Ludovici, Francisco Triviño, Alessandro Strano, José Flich, José LUIS Sánchez, Francisco Alfaro, Michele Favalli, and Davide Bertozzi. A complete self-testing and self-configuring noc infrastructure for cost-effective mpsocs. *ACM Transactions on Embedded Computing Systems (TECS)*, 12(4):106, 2013.

[28] Gennette Gill, Sumedh S Attarde, Geoffray Lacourba, and Steven M Nowick. A low-latency adaptive asynchronous interconnection network using bi-modal router nodes. In *Proceedings of the Fifth ACM/IEEE International Symposium on Networks-on-Chip*, pages 193–200. ACM, 2011.

[29] M.E. Gomez, N.A. Nordbotten, J. Flich, P. Lopez, A. Robles, J. Duato, T. Skeie, and O. Lysne. A routing methodology for achieving fault tolerance in direct networks. *Computers, IEEE Transactions on*, 55(4):400–415, 2006.

[30] Kees Goossens, Bart Vermeulen, Remco van Steeden, and Martijn Bennebroek. Transaction-Based Communication-Centric Debug. In *Proceedings of the First International Symposium on Networks-on-Chip (NOCS'07)*, 2007.

[31] C. Grecu, P. Pande, A. Ivanov, and R. Saleh. Bist for network-on-chip interconnect infrastructures. In *VLSI Test Symposium, 2006. Proceedings. 24th IEEE*, pages 6 pp.–35, 2006.

[32] Simon Hollis and Simon W Moore. Rasp: an area-efficient, on-chip network. In *Computer Design, 2006. ICCD 2006. International Conference on*, pages 63–69. IEEE, 2007.

[33] Nima Honarmand, Ali Shahabi, and Zain Navabi. A heuristic search algorithm for re-routing of on-chip networks in the presence of faulty links and switches. *Proc. of IEEE EWDTS*, 2007.

[34] Michael N Horak, Steven M Nowick, Matthew Carlberg, and Uzi Vishkin. A low-overhead asynchronous interconnection network for gals chip multiprocessors. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 30(4):494–507, 2011.

[35] Bo Huang, Song Chen, Wei Zhong, and Takeshi Yoshimura. Application-specific network-on-chip synthesis with topology-aware floorplanning. In *Integrated Circuits and Systems Design (SBCCI), 2012 25th Symposium on*, pages 1–6. IEEE, 2012.

[36] iNoCs. http://www.inocs.com.

[37] Young Hoon Kang, Taek-Jun Kwon, and Jeffrey Draper. Fault-tolerant flow control in on-chip networks. In *Proceedings of the 2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip*, NOCS '10, pages 79–86, Washington, DC, USA, 2010. IEEE Computer Society.

[38] Nishit Kapadia and Sudeep Pasricha. A power delivery network aware framework for synthesis of 3d networks-on-chip with multiple voltage islands. In *VLSI Design (VLSID), 2012 25th International Conference on*, pages 262–267. IEEE, 2012.

[39] Gul N Khan and Anita Tino. Synthesis of noc interconnects for multi-core architectures. In *Complex, Intelligent and Software Intensive Systems (CISIS), 2012 Sixth International Conference on*, pages 432–437. IEEE, 2012.

[40] Adán Kohler, Gert Schley, and Martin Radetzki. Fault tolerant network on chip switching with graceful performance degradation. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 29(6):883–896, 2010.

[41] Cheng Kwang-Ting and A.S. Krishnakumar. A Design-for-Debug (DfD) for NoC-based SoC Debugging via NoC. In *Asian Test Symposium, 2008. ATS '08. 17th*, 2008.

[42] Didier Lattard, Edith Beigne, Fabien Clermidy, Yves Durand, Romain Lemaire, Pascal Vivet, and Friedbert Berens. A reconfigurable base-band platform based on an asynchronous network-on-chip. *Solid-State Circuits, IEEE Journal of*, 43(1):223–235, 2008.

[43] Kuen-Jong Lee, Si-Yuan Liang, and Alan Su. A Low-Cost SOC Debug Platform Based on On-Chip Test Architectures. In *IEEE International SOC Conference*, 2009.

[44] Andrew Lines. Asynchronous interconnect for synchronous soc design. *Micro, IEEE*, 24(1):32–41, 2004.

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

[46] Andres Mejia, Jose Flich, and Jose Duato. On the potentials of segment-based routing for nocs. In *Parallel Processing, 2008. ICPP'08. 37th International Conference on*, pages 594–603. IEEE, 2008.

[47] M.Igarashi. Concurrent hierarchical design with ic compiler. real life application on mobile multi-media processor.

[48] S Murali, D Atienza, G De Micheli, F Angiolini, L Benini, P Meloni, SM Carta, and L Raffo. Sunfloor: Application-specific design of networks-on-chip. In *Poster presentation at University Booth at the Design, Automation and Test in Europe Conference and Exhibition*, pages 6–10. Citeseer, 2006.

[49] Srinivasan Murali and Giovanni De Micheli. Sunmap: a tool for automatic topology selection and generation for nocs. In *Proceedings of the 41st annual Design Automation Conference*, pages 914–919. ACM, 2004.

[50] Srinivasan Murali, Theocharis Theocharides, Narayanan Vijaykrishnan, Mary Jane Irwin, Luca Benini, and Giovanni De Micheli. Analysis of error recovery schemes for networks on chips. *IEEE Design & Test of Computers*, 22(5):434–442, 2005.

[51] Michael Nicolaidis. Design for soft error mitigation. *Device and Materials Reliability, IEEE Transactions on*, 5(3):405–418, 2005.

[52] Steven M Nowick and Montek Singh. High-performance asynchronous pipelines: an overview. *Design & Test of Computers, IEEE*, 28(5):8–22, 2011.

[53] Maurizio Palesi, Shashi Kumar, and Rickard Holsmark. A method for router table compression for application specific routing in mesh topology noc architectures. In *Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 373–384. Springer, 2006.

[54] Dongkook Park, Chrysostomos Nicopoulos, Jongman Kim, Narayanan Vijaykrishnan, and Chita R Das. Exploring fault-tolerant network-on-chip architectures. In *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*, pages 93–104. IEEE, 2006.

[55] Luis A Plana, David Clark, Simon Davidson, Steve Furber, Jim Garside, Eustace Painkras, Jeffrey Pepper, Steve Temple, and John Bainbridge. Spinnaker: design and implementation of a gals multicore system-on-chip. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 7(4):17, 2011.

[56] Luis A Plana, Steve B Furber, Steve Temple, Mukaram Khan, Yebin Shi, Jian Wu, and Shufan Yang. A gals infrastructure for a massively parallel multiprocessor. *Design & Test of Computers, IEEE*, 24(5):454–463, 2007.

[57] Antonio Pullini, Federico Angiolini, Davide Bertozzi, and Luca Benini. Fault tolerance overhead in network-on-chip flow control schemes. In *Proceedings of the 18th annual symposium on Integrated circuits and system design*, pages 224–229. ACM, 2005.

[58] Bradley R Quinton, Mark R Greenstreet, and Steven JE Wilton. Practical asynchronous interconnect network design. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 16(5):579–588, 2008.

[59] S. Rodrigo, J. Flich, A. Roca, S. Medardoni, D. Bertozzi, J. Camacho, F. Silla, and J. Duato. Addressing manufacturing challenges with cost-efficient fault tolerant routing. In *Networks-on-Chip (NOCS), 2010 Fourth ACM/IEEE International Symposium on*, pages 25–32, 2010.

[60] S. Rodrigo, C. Hernandez, J. Flich, F. Silla, J. Duato, S. Medardoni, D. Bertozzi, A. Mejia, and D. Dai. Yield-oriented evaluation methodology of network-on-chip routing implementations. In *System-on-Chip, 2009. SOC 2009. International Symposium on*, pages 100–105, 2009.

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

[62] Daniele Rossi, Paolo Angelini, and Cecilia Metra. Configurable error control scheme for noc signal integrity. In *On-Line Testing Symposium, 2007. IOLTS 07. 13th IEEE International*, pages 43–48. IEEE, 2007.

[63] Daniele Rossi, Cecilia Metra, André K Nieuwland, and Atul Katoch. Exploiting ecc redundancy to minimize crosstalk impact. *Design & Test of Computers, IEEE*, 22(1):59–70, 2005.

[64] Dobkin Rostislav, Victoria Vishnyakov, Eyal Friedman, and Ran Ginosar. An asynchronous router for multiple service levels networks on chip. In *Asynchronous Circuits and Systems, 2005. ASYNC 2005. Proceedings. 11th IEEE International Symposium on*, pages 44–53. IEEE, 2005.

[65] Abbas Sheibanyrad, Alain Greiner, and Ivan Miro-Panades. Multisynchronous and fully asynchronous nocs for gals architectures. *IEEE Design & Test of Computers*, 25(6):0572–580, 2008.

[66] Montek Singh and Steven M Nowick. Mousetrap: High-speed transition-signaling asynchronous pipelines. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 15(6):684–698, 2007.

[67] Binjie Song, Shan Zeng, Yuchun Ma, Ning Xu, and Yu Wang. Tree-based partitioning approach for network-on-chip synthesis. In *Computer-Aided Design and Computer Graphics (CAD/Graphics), 2011 12th International Conference on*, pages 465–470. IEEE, 2011.

[68] Wei Song, Doug Edwards, Jose Luis Nunez-Yanez, and Sohini Dasgupta. Adaptive stochastic routing in fault-tolerant on-chip networks. In *Networks-on-Chip, 2009. NoCS 2009. 3rd ACM/IEEE International Symposium on*, pages 32–37. IEEE, 2009.

[69] sonics. http://sonicsinc.com.

[70] D. Starobinski, M. Karpovsky, and L.A. Zakrevski. Application of network calculus to general topologies using turn-prohibition. *Networking, IEEE/ACM Transactions on*, 11(3):411–421, 2003.

[71] S. Stergiou, F. Angiolini, Salvatore Carta, L. Raffo, D. Bertozzi, and G. De Micheli. times;pipes lite: a synthesis oriented design library for networks on chips. In *Design, Automation and Test in Europe, 2005. Proceedings*, pages 1188–1193 Vol. 2, 2005.

[72] Kenneth S Stevens, Yang Xu, and Vikas Vij. Characterization of asynchronous templates for integration into clocked cad flows. In *Asynchronous Circuits and Systems, 2009. ASYNC'09. 15th IEEE Symposium on*, pages 151–161. IEEE, 2009.

[73] A. Strano, C. Gómez, D. Ludovici, M. Favalli, M.E. Gomez, and D. Bertozzi. Exploiting network-on-chip structural redundancy for a cooperative and scalable built-in self-test architecture. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pages 1–6, 2011.

[74] HF Tatenguem, D Ludovici, A Strano, D Bertozzi, and H Reinig. Contrasting multi-synchronous mpsoc design styles for fine-grained clock domain partitioning: the full-hd video playback case study. In *Proceedings of the 4th International Workshop on Network on Chip Architectures*, pages 37–42. ACM, 2011.

[75] Paul Teehan, Mark Greenstreet, and Guy Lemieux. A survey and taxonomy of gals design styles. *Design & Test of Computers, IEEE*, 24(5):418–428, 2007.

[76] Teklatech. http://www.teklatech.com.

[77] Yvain Thonnart, Edith Beigné, and Pascal Vivet. A pseudo-synchronous implementation flow for wchb qdi asynchronous circuits. In *Asynchronous Circuits and Systems (ASYNC), 2012 18th IEEE International Symposium on*, pages 73–80. IEEE, 2012.

[78] Yvain Thonnart, Pascal Vivet, and Fabien Clermidy. A fully-asynchronous low-power framework for gals noc integration. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 33–38. European Design and Automation Association, 2010.

[79] Vladimir Todorov, Alberto Ghiribaldi, Helmut Reinig, Davide Bertozzi, and Ulf Schlichtmann. Non-intrusive trace &#38; debug noc architecture with accurate timestamping for gals socs. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS '12, pages 181–186, New York, NY, USA, 2012. ACM.

[80] Vladimir Todorov, Daniel Mueller-Gritschneder, Helmut Reinig, and Ulf Schlichtmann. A spectral clustering approach to application-specific network-on-chip synthesis. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1783–1788. EDA Consortium, 2013.

[81] Kees van Berkel, Ferry Huberts, and Ad Peeters. Stretching quasi delay insensitivity by means of extended isochronic forks. In *Asynchronous*

Design Methodologies, 1995. Proceedings., Second Working Conference on, pages 99–106. IEEE, 1995.

[82] Fu-Ching Yang, Yi-Ting Lin, Chung-Fu Kao, and Ing-Jer Huang. An On-Chip AHB Bus Tracer With Real-Time Compression and Dynamic Multiresolution Supports for SoC. In *IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS*, 2011.

[83] Qiaoyan Yu and Paul Ampadu. Adaptive error control for noc switch-to-switch links in a variable noise environment. In *Defect and Fault Tolerance of VLSI Systems, 2008. DFTVS'08. IEEE International Symposium on*, pages 352–360. IEEE, 2008.

[84] Jianmin Zhang, Ming Yan, and Sikun Li. Debug Support for Scalable System-on-Chip. In *Seventh International Workshop on Microprocessor Test and Verification (MTV'06)*, 2006.

[85] Lei Zhang, Yinhe Han, Qiang Xu, Xiao wei Li, and Huawei Li. On topology reconfiguration for defect-tolerant noc-based homogeneous many-core systems. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 17(9):1173–1186, 2009.

Author's Publications List

1. Ghiribaldi Alberto, Ludovici Daniele, Favalli Michele, Bertozzi Davide. "System-level infrastructure for boot-time testing and configuration of networks-on-chip with programmable routing logic."
   In VLSI and System-on-Chip (VLSI-SoC), 2011 IEEE/IFIP 19th International Conference on (pp. 308-313). IEEE. (2011, October).

2. Ghiribaldi Alberto, Strano Alessandro, Favalli Michele, Bertozzi Davide. "Power efficiency of switch architecture extensions for fault tolerant noc design."
   In Green Computing Conference (IGCC), 2012 International (pp. 1-6). IEEE. (2012, June).

3. Todorov Vladimir, Ghiribaldi Alberto, Reinig Helmut, Bertozzi Davide, Schlichtmann Ulf. "Non-intrusive trace & debug noc architecture with accurate timestamping for GALS SoCs."
   In Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis (pp. 181-186). ACM. (2012, October).

4. Ghiribaldi Alberto, Bertozzi Davide, Nowick Steven. "A transition-signaling bundled data NoC switch architecture for cost-effective GALS multicore systems."
   In Proceedings of the Conference on Design, Automation and Test in Europe (pp. 332-337). EDA Consortium. (2013, March).

5. Ghiribaldi Alberto, Fankem Herve Tatenguem, Angiolini Federico, Stensgaard Mikkel, Bjerregaard Tobias, Bertozzi Davide. "A vertically integrated and interoperable multi-vendor synthesis flow for predictable noc design in nanoscale technologies."

Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific.

6. Strano Alessandro, Ghiribaldi Alberto, Fankem Herve Tatenguem, Bertozzi Davide. "A Feature-Rich NoC Switch with Cross-Feature Optimizations for The Next Generation of Reliable and Reconfigurable Embedded Systems."
   In Proceedings of the 8th International Workshop on Interconnection Network Architecture: On-Chip, Multi-Chip (p. 2). ACM. (2014, January).
7. Ramini Luca, Grani Paolo, Fankem Herve Tatenguem, Ghiribaldi Alberto, Bartolini Sandro, Bertozzi Davide. "Assessing the Energy Break-Even Point Between an Optical NoC Architecture and an Aggressive Electronic Baseline."
   In Proceedings of the Conference on Design, Automation and Test in Europe. EDA Consortium. (2014, March)