NoCAlert: An On-Line and Real-Time Fault Detection Mechanism for Network-on-Chip Architectures

Andreas Prodromou[1], Andreas Panteli[1], Chrysostomos Nicopoulos[1], and Yiannakis Sazeides[2]

{prodromou.andreas, panteli.andreas, nicopoulos}@ucy.ac.cy, yanos@cs.ucy.ac.cy
[1]Department of Electrical and Computer Engineering, University of Cyprus
[2]Department of Computer Science, University of Cyprus

Abstract

The widespread proliferation of the Chip Multi-Processor (CMP) paradigm has cemented the criticality of the on-chip interconnection fabric. The Network-on-Chip (NoC) is becoming increasingly susceptible to emerging reliability threats. As technology feature sizes diminish into the nanoscale regime, reliability and process variability artifacts within the NoC start to become prominent. The need to detect the occurrence of faults at run-time is steadily becoming imperative. In this work, we propose ***NoCAlert****, a comprehensive on-line and real-time fault detection mechanism that demonstrates 0% false negatives within the interconnect, for the fault model and stimulus set used in this study. Based on the concept of invariance checking, NoCAlert employs a group of lightweight micro-checker modules that collectively implement real-time hardware assertions. The checkers operate seamlessly and concurrently with normal NoC operation, thus eliminating the need for periodic, or triggered-based, self-testing. More importantly, 97% of the faults are detected instantaneously. Extensive cycle-accurate simulations in a 64-node CMP demonstrate the efficacy of the proposed technique. Finally, hardware synthesis results using commercial 65 nm technology libraries indicate minimal area and power overhead of 3% and less than 1%, respectively, and negligible impact on the router's critical path.*

1. Introduction

Diminutive technology feature sizes have enabled microprocessors with billions of transistors on a single chip die [1]. This unprecedented abundance of on-chip resources, coupled with thinning Instruction-Level Parallelism (ILP), have urged designers to switch their attention to another computational archetype: the *Chip Multi-Processor* (CMP) [2]. The presence of multiple on-chip processing entities has precipitated a shift from computation-centric to communication-centric micro-architectures. As a result, the on-chip interconnection fabric is fast becoming a mission-critical component. Packet-based Networks-on-Chip (NoC) are widely viewed as the de facto communication medium of future multi-/many-core CPUs, primarily due to their inherent scalability attributes and modular nature [3].

However, the march towards CMPs with tens – or even hundreds – of processing cores has been marred by the emergence of an ominous threat: waning *reliability* [4]. The extreme downscaling trends of CMOS technology have rendered transistors more susceptible to both permanent and transient faults. Moreover, digital circuits are increasingly affected by growing process variability artifacts [5] and accelerated aging effects [6, 5], all of which are consequences of dwindling feature sizes. Just like any on-chip component, the interconnection backbone is also affected by decreasing reliability [7]. In fact, a single fault in the on-chip network may paralyze an otherwise healthy CMP. Faults within the NoC may result in such show-stopping predicaments as network disconnections, network-level deadlocks, protocol-level (cache coherence) deadlocks, lost packets, and severely degraded on-chip communication performance [8].

Architects and designers have proposed a multitude of techniques, mechanisms, and design modifications to increase the fault tolerance and reliability of the NoC. However, the vast majority of the related work found in the literature concentrates on fault *prevention* (improving durability/fault-tolerance, prolonging lifetime, etc.) [9] and/or *recovery* (redundancy, reconfiguration, adaptation, etc.) [10, 11]. The equally important aspect of ***fault detection*** has not been adequately addressed.

Traditionally, fault detection is undertaken by Built-In Self-Test (BIST) mechanisms that predominantly assume a disruption in the system's operation. The BIST process may be executed by the manufacturer prior to shipment, or it may constitute part of system boot-up [12, 13]. Runtime BIST is also possible, but system operation is (partially) halted while the module-under-test is examined [10, 14]. BIST usually entails the use of predefined test vectors, patterns, or routines, which tend to be pure overhead. Regardless, *detecting faults at run-time* is rapidly becoming a necessity, in light of the aforementioned decline in reliability. When BIST or BIST-like methodologies are employed within the context of on-line (run-time) testing, the process is usually triggered periodically [14]. Choosing the length of the period between two consecutive test sessions is certainly non-trivial: if testing is conducted too frequently, the impact on performance will be more pronounced, due to excessive interruptions; if testing is rarely performed, then faults may go unnoticed for a prolonged period of time [15]. Furthermore, periodic testing often implies the use of checkpointing, which adds further overhead (both in terms of performance and hardware/storage/power).

Near-instantaneous fault detection may be achieved in the datapath of the interconnect through the use of *error detecting codes*. Simple parity checks – or more elaborate coding – will detect (and may even correct) errors affecting the *contents* of in-flight packets [16]. While this methodology guarantees protection of the message contents, faults within the *control logic* of the NoC may still wreak havoc with the operation of the entire CMP. Hence, what is needed to guarantee functional correctness within the NoC – and, by extension, within the CMP – is to *protect the NoC's control logic* (assuming that the flit *contents* are protected by error-correcting codes). This thesis statement marks the central theme of our work.

Realizing the significance of accurate and timely run-time detection of faults within the NoC's control logic, we hereby propose **a comprehensive on-line fault detection mechanism**, aptly called ***NoCAlert***, which provides full fault coverage for all on-chip network control logic components and achieves *instantaneous* detection of any erroneous behavior. Depending on the application's criticality, instantaneous detection may be of paramount significance. The NoCAlert mechanism is based on the notion of ***invariance checking***, whereby the system is continuously checked for illegal outputs as a result of upsets (permanent, transient, or intermittent). An *illegal output* is defined here as an operational decision that violates the functional correctness rule(s) of a particular component. The underlying principle of this technique is inspired by prior efforts to protect the microprocessor by using invariances [17]. NoCAlert comprises several checker micro-modules distributed throughout the NoC router, which seamlessly and concurrently monitor all NoC modules for illegal activity. The checkers never interfere with – or interrupt – the operation of the NoC and they provide real-time on-line fault detection. In essence, NoCAlert implements an all-encompassing collection of extremely lightweight ***real-time hardware assertions*** that can detect illegal outputs within the NoC's control logic.

In particular, the main contributions of this work are:

1. The development of **a comprehensive on-line and real-time fault detection mechanism** for the control logic of the NoC of multi-core CMPs. The proposed NoCAlert checker modules operate *seamlessly and concurrently with normal NoC operation*, thus obviating the need for testing epochs and periodic triggering of testing sessions that may interrupt/impede normal system operation.
2. The NoCAlert protective blanket ensures **0% false negatives within the interconnect** for the fault model (single-bit transient) and stimulus set used in this study, with **97% of the faults detected instantaneously** (i.e., in the same cycle as the fault occurrence). This attribute allows for ultra-fast response by a potential fault recovery scheme and/or re-configuration mechanism. NoCAlert is intended to be used in conjunction with fault recovery techniques.
3. We demonstrate that by using checkers that solely detect *illegal* outputs (outputs that cannot be produced by any input) for all NoC control components, we observe 0% false negatives for the entire network using the fault model of this study. This empirical observation leads to an interesting hypothetical corollary about a NoC router's control components: if a unit produces a faulty but *legal* output, which does not lead to subsequent invariance violations, it is always benign as far as the overall NoC operation in concerned.
4. The entire NoCAlert scheme is **extremely lightweight in terms of all salient design metrics**. Hardware synthesis results using commercial 65 nm standard-cell libraries indicate minimal area and power overhead of 3% and less than 1%, respectively. More importantly, the critical path of the router is shown to be negligibly affected (around 1%), rendering the proposed mechanism transparent to normal operation. Our analysis clearly indicates that checkers used to detect only illegal outputs have significantly lower hardware cost, as compared to the cost of the unit they check; i.e., the complexity of determining whether an output is illegal – given an input – is much simpler than producing the output.
5. The NoCAlert framework is evaluated by injecting faults **in all possible locations** (according to the employed fault model) within the NoC of a 64-node CMP arranged in an 8×8 mesh. Extensive simulations were run in a cycle-accurate NoC evaluation framework. The results and ensuing analysis corroborate the efficacy of the NoCAlert mechanism.
6. Through a detailed experimental comparison, NoCAlert is shown to outperform ForEVeR [15], a recently proposed state-of-the-art fault detection and recovery framework. NoCAlert provides more than $100\times$ reduction in fault detection latency, with no loss in detection accuracy, and without the need to rely on a secondary, fault-free checker network for detection purposes.

To the best of our knowledge, this work constitutes the first attempt to utilize real-time hardware-based assertion checkers to ensure 0% false negatives within the on-chip interconnection network of CMPs.

The rest of the paper is organized as follows: Section 2 discusses related prior work in fault-tolerant NoCs. Section 3 introduces the idea of invariance checking within the on-chip network, while Section 4 delves into the description, implementation, and analysis of the proposed NoCAlert mechanism. Section 5 presents the employed evaluation framework, the various experiments, and accompanying analysis. Finally, Section 6 concludes the paper.

2. Related Work

In general, research in the field of fault tolerance revolves around two fundamental axes: (1) Fault *Detection*, and (2) Fault *Recovery/Protection/Isolation*. While the focus is often slanted more toward the latter, both axes are essential in delivering a robust system. Research in the field of *NoC* reliability naturally falls into two main categories: (a) Inter-router faults (i.e., faults within the links interconnecting the various switches), and (b) Intra-router faults (i.e., faults within the switches themselves). The following sub-sections will concentrate on these two categories.

2.1. Inter-Router Faults

Disabled inter-router links in the network reduce connectivity. Reduced connectivity may, in turn, lead to network deadlocks and – depending on the routing algorithm used – may lead to halted network operation. Broken network links imply reduced path diversity, creation of hotspots, and network delay due to back-pressure effects. Default Backup Paths (DBP) [18] were proposed as a means to maintain connectivity in the presence of faults. In [11], all the physical links are doubled in order to enhance NoC connectivity. Naturally, the presence of fully disabled links predominantly affects the routing algorithm within the network routers.

The assumption of *fully* disabling a parallel multi-bit inter-router link is overly pessimistic. In reality, each parallel link (ranging from 32 up to 256 wires) is individually driven. Whenever a wire fails, the rest can still function properly. Hence, in a real-world scenario, a fault within the links will give rise to *partially faulty links*. It is this realization that has led researchers to look into Error-Correcting Codes (ECC) [19, 20], utilizing redundant wires/bits. For online detection and diagnosis purposes, these codes are very effective. Retransmission mechanisms are typically required to co-operate with ECC schemes [19]. Researchers have devised methodologies to transfer flits through these partially faulty links through shifting and multi-cycle transmissions [21], and by using spare wires [20].

2.2. Intra-Router Faults

This sub-section presents prior work regarding fault-tolerant routing and architectural redundancy schemes.

Fault-Tolerant Routing in NoCs: Most NoC fault-tolerant routing algorithms are inspired from seminal work conducted in the domain of large-scale, multi-computer interconnection networks [22, 23]. NoCs are characterized by severely limited on-chip resources, scarce energy budgets, and the imperative need for ultra-high performance. As such, the fault-tolerant routing algorithms proposed for NoCs must account for these salient attributes. Universal Logic-Based Distributed Routing (uLBDR) [24] aims to eliminate fault-susceptible routing tables. Stochastic routing algorithms [25] have been employed to bypass faulty links in the network. Dynamic reconfigurable routing algorithms [26] determine forbidden turns at run-time to avoid deadlocks while bypassing faulty components. Deflection routing [27, 28] is another technique that favors routing resilience, and it has been employed as a fault-tolerant routing mechanism [28]. Distributed [29] and multi-path [30] routing strategies aim to evenly spread network traffic over a faulty network topology without deadlocks. Finally, the concept of exploration/scouting packets [31, 32] has also been used to identify faulty nodes ahead of regular data packets.

Architectural Techniques to Tackle Datapath and Control Logic Faults: Besides the multitude of routing algorithms designed to provide uninterrupted network functionality in the presence of faults, a lot of research has addressed fault-tolerance in the critical components comprising the datapath and control logic of NoC routers.

The Row-Column (RoCo) Decoupled router [33] provides extensive fault tolerance and graceful degradation by decomposing the router into two independent modules and by employing resource sharing. Bulletproof [9] proposes various online repair and recovery capabilities and investigates protection at various levels, ranging from system-level to arbitrary partitions of the design.

Fault-tolerant techniques provide effective *recovery* mechanisms that ensure correct functionality in the presence of faults. Of course, the basic assumption is that the fault must first be *detected*. While most of the techniques considered in this Section so far *assume* the presence of fault detection capability and concentrate on the recovery aspects, others have also tackled the non-trivial facet of detecting the faults in the first place.

The proposed mechanism in [13] broadcasts test vectors within the NoC *during boot-up only* and detects faults by examining the responses of the various router components. To accommodate run-time occurrence of faults, the work in [12] is also capable of generating *on-line* test vectors that are broadcast in the network. Test vector results are then evaluated by neighboring routers. However, in this scheme, the entire network's operation is halted for testing purposes. In order to mitigate the performance degradation caused by testing interruptions, the token-based mechanism in [14] interrupts only a small portion of the network at any given time. The Allocation Comparator of [19] performs on-line, real-time diagnosis by observing the occurrence of some invalid operations within the router arbiters as a result of transient faults. The Vicis router [10] employs ECC codes to detect *some* faults. Subsequently, specialized BIST testers located in each router are utilized for more extensive testing and fault localization [10]. Finally, the appropriate reconfiguration mechanism is triggered to combat the detected fault. Error-correcting codes have also been used in conjunction with a packet/flit counting technique to detect and diagnose permanent faults in the network [34].

The ***ForEVeR*** **framework** [15] was recently proposed, which complements the use of formal methods and runtime verification to ensure functional correctness in NoCs. While ForEVeR's goal is to protect against escaped design-time verification errors with a run-time technique, the scheme may also be used to provide robustness against run-time faults. Fault detection is achieved with the help of (a) an *additional* lightweight checker network that is *assumed to be 100% reliable*, (b) the Allocation Comparator from [19], and (c) an end-to-end checker. The checker network is used to alert destination nodes ahead of time about incoming flits. The destination node increases a flit counter upon a notification reception, and decreases the same counter upon flit reception. Time is separated in so called *epochs*, and at the end of each epoch the counter must have reached the value of zero at least once within the epoch interval. If not, a recovery mechanism is triggered, which delivers the in-flight data to the intended destination via the checker network. The use of timing intervals implies the non-trivial task of finding the optimal epoch duration to minimize false positives. In fact, if the epoch duration is not carefully chosen, the mechanism may give rise to false positives even in a fault-free environment. Moreover, the epoch duration is sensitive to the traffic injection rate, which hinders widespread applicability. More importantly, the use of an end-to-end, epoch-based scheme, such as ForEVeR, results in significantly delayed fault detection. Particular to ForEVeR, fault detection relies on ahead-of-time notifications sent through the checker network; hence, a *run-time* fault in the checker network would incapacitate fault detection. Finally, any faults that cause degradation in performance, but do not cause a functional error at the output (end-to-end delivery) will never be detected (i.e., only faults that cause functional errors are detected). A detailed quantitative comparison between NoCAlert and ForEVeR [15] will be presented in Section 5.

In summary, recovery and reconfiguration schemes rely on efficient, accurate, and quick-responding fault detection mechanisms. In the absence of such mechanisms, the efficacy of recovery is severely compromised. Inaccurate detection mechanisms can cause undue network/system performance degradation, while delayed detection will necessitate the presence and invocation of checkpointing mechanisms, which inevitably incur both hardware and performance overhead.

The NoCAlert mechanism proposed in this work ensures ***on-line and real-time fault detection*** within the NoC, and it guarantees ***0% false negatives*** under the employed fault model. Most importantly, the technique works concurrently with normal network operation (i.e., no testing interruptions) and is shown to be extremely lightweight. Moreover, **NoCAlert may be used to complement any other fault recovery scheme**, such as ForEVeR [15]. The recovery mechanism – aided by NoCAlert's instantaneous fault detection – may react much more rapidly (if deemed necessary), thus minimizing the effect on system-level performance.

3. Invariance Checking within the NoC

The NoCAlert mechanism is based on the concept of *invariance checking*. When checking for invariances, the system is continuously examined for illegal outputs as a result of some kind of perturbation (fault). As previously mentioned, the term *illegal output* is defined here as an operational output that is impossible to occur, based on the set of functional correctness rules of a given component. Thus, the term "invariance" describes a condition that cannot – by definition – vary. Consequently, an invariance violation is the breaking of fundamental rules within the context of a system component. Invariance is a general term that applies to every system governed by some rules within a specific context. Considering an adder circuit as an example, one derived invariance would be that the sum of two even numbers must always be even as well.

A well-known implementation of invariance checking is the use of *assertions* in software development [35]. Software assertions ensure that a forbidden state cannot be reached; if it is reached, a notification is issued.

In this work, we adopt the notion of invariance checking and apply it to all the modules of a NoC router's control logic to *detect* abnormalities in the network resulting from either transient, or permanent, faults. The assertions are implemented in *hardware* so as to provide near-instantaneous detection of anomalies.

The salient characteristic of invariance checking, in general, is the fact that only *functionally illegal* outputs are flagged as violations. In other words, a fault that causes the generation of an erroneous,

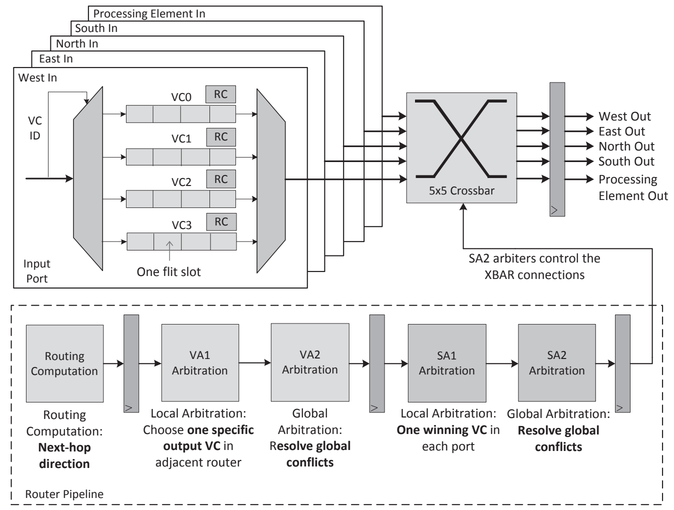
Processing Element In
South In
North In
East In
West In
VC ID
VC0
VC1
VC2
VC3
RC
One flit slot
Input Port
5x5 Crossbar
West Out
East Out
North Out
South Out
Processing Element Out
SA2 arbiters control the XBAR connections
Routing Computation
VA1 Arbitration
VA2 Arbitration
SA1 Arbitration
SA2 Arbitration
Routing Computation: Next-hop direction
Local Arbitration: Choose one specific output VC in adjacent router
Global Arbitration: Resolve global conflicts
Local Arbitration: One winning VC in each port
Global Arbitration: Resolve global conflicts
Router Pipeline

Figure 1: Overview of the router pipeline. The baseline router has five pipeline stages; namely, Routing Computation (RC), Virtual channel Allocation (VA), Switch Arbitration (SA), Crossbar (XBAR) traversal, and Link Traversal (LT).

yet functionally legal, output will *not* be identified as a breach of correctness.

Knowing this innate limitation of invariance checking, what we aim to explore in this work – among others – is how often, and under what conditions, such non-invariant faults could potentially lead to compromised network-level correctness. We will demonstrate empirically that non-invariant faults within the NoC routers, which do not cause any subsequent invariance violations (i.e., they are not caught by subsequent checkers), always prove to be innocuous at the system level, i.e., they do not cause network-level malfunction.

Before we proceed with the identification of invariant conditions (invariances) within the NoC, we first present – without loss of generality – a typical router micro-architecture [36], which forms the foundation of most router implementations discussed in the literature. It is important to note that this architecture is general enough to allow the proposed NoCAlert mechanism to be applicable to *any* router implementation. Later on in the paper, we will briefly show how NoCAlert can also be fitted to different router designs.

3.1. A Generic NoC Router Micro-architecture

Figure 1 presents a high-level, abstracted view of the baseline router micro-architecture assumed in this work. This generic input-buffered router design consists of five input/output ports. Four of them are used to communicate with the adjacent routers in the network (in the four cardinal directions of a 2D mesh) and the fifth port is used to communicate with the local processing element. Each input port has a number of Virtual Channels (VC) that support (potentially) the routing algorithm (e.g., adaptive) and, more importantly, the cache coherent protocol employed within the CMP. VCs are used to avoid protocol-level deadlocks in the network, as well as to enhance bandwidth utilization at the network level. A central crossbar (XBAR) facilitates the interconnection between the input and output ports, as shown in Figure 1. The main modules of the router's control logic are the Routing Computation (RC) unit, the Virtual channel Allocation (VA) unit, and the Switch Arbitration (SA) unit. The RC unit is responsible to compute the output direction that a packet must follow to get to the next hop, based on the destination information found in the header flit of each packet. The VA unit allocates a downstream VC to each packet. This is the VC that the packet will use in the adjacent router. Finally, the SA unit decides which flits traverse the crossbar in each cycle. The baseline router is assumed to be wormhole-switched (the predominant choice in *on-chip* networks) and to use credit-based flow control.

The employed router has a five-stage pipeline, with each stage corresponding to one of the major functional units within the router: RC, VA, SA, XBAR traversal, and Link Traversal (LT), as illustrated in Figure 1. The first two stages are executed only for the *header* flit of each packet (in order to set up the wormhole), while the remaining stages are executed for all flits. As can be seen in Figure 1, the VA and SA stages are further separated into *local* (intra-port) and *global* (inter-port) sub-stages. Local stages perform arbitration within a specific port, while the global stages resolve conflicts between the various ports. The data-path of the router comprises the input buffers and the XBAR switch. Each input port employs an input de-multiplexer and an output multiplexer to accommodate the sharing of one physical channel by multiple VCs. This organization implies that only one flit can arrive to, or leave from, an input port in each cycle. Furthermore, VCs may be atomic or non-atomic. Atomic VCs can only store the flits of a single packet at any given time. In other words, flits from two different packets cannot co-exist in the same VC.

3.2. Examples of On-Chip Network Invariances

This sub-section presents three representative examples of invariances found within the NoC. To aid understanding, the examples are depicted in Figure 2.

Assuming the 4$\times$4 mesh network in Figure 2(a), let us identify one important invariance pertaining to the widely used XY routing algorithm. Routing algorithms, in general, forbid some turns to avoid deadlocks and/or livelocks in the network. The XY routing algorithm, in particular, first routes a packet along the X dimension until the intended destination's X-coordinate has been reached, and then along the Y dimension until the destination node has been reached. Suppose the origin of the Cartesian system is the bottom left router, and assume that a packet is injected in router (1,1) with destination (1,3). Upon reaching router (1,2), a fault in the RC unit of said router forwards the packet to the East output port, toward router (2,2). This action constitutes an *invariance violation*, since a packet arriving from the Y dimension (North or South input ports) may not make a turn to the X dimension (East or West output ports) under XY routing. Such an invariance violation, if caught, indicates a malfunctioning RC unit.

As described in Section 3.1, there are five pipeline stages in the baseline router. Under normal operation, those pipeline stages should be executed in the correct order. To maintain the pipeline functionality, each VC keeps its own functional state. Figure 2(b) shows an example of a VC's status table. In this example, a header flit is present at the head of the queue and is waiting for the VA stage (VC allocation) to be executed. Since the "VA done" field in the status table is set to 0, an output VC has not yet been allocated to the specific packet. A malfunctioning SA arbiter, however, sends an active grant success signal to the VC, thus violating the correct pipeline order (SA success before VA is complete). This invariance violation identifies erroneous behavior within the SA module.

Finally, Figure 2(c) illustrates a router's input port with four VCs. Suppose that the flit at the head of the VC0 queue is ready to proceed to the XBAR stage, as indicated by the active "Read Signal" at VC0. However, a fault leads to the simultaneous assertion of the VC1 read signal in the same clock cycle. As can be seen in the figure, only one flit from each input port may depart in a single clock cycle, due to the

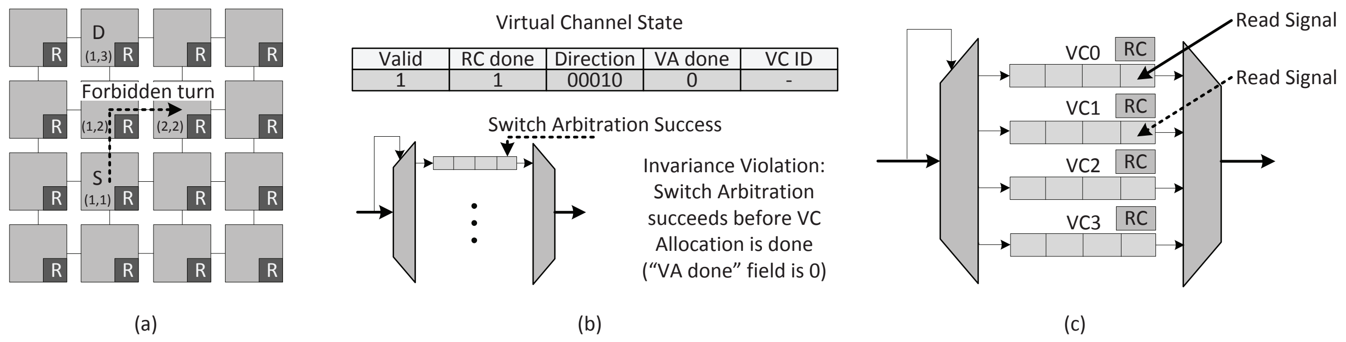
D
(1,3)
R
Forbidden turn
(1,2)
(2,2)
S
(1,1)
Virtual Channel State
Valid | RC done | Direction | VA done | VC ID
1 | 1 | 00010 | 0 | -
Switch Arbitration Success
Invariance Violation: Switch Arbitration succeeds before VC Allocation is done ("VA done" field is 0)
Read Signal
Read Signal
VC0
VC1
VC2
VC3
RC
(a)
(b)
(c)

Figure 2: Examples of NoC invariances. The first example (a) illustrates the case whereby a malfunctioning XY routing computation unit attempts to route a packet in a forbidden direction (S: Source node; D: Destination node). The second example (b) demonstrates an invariance violation that occurs upon receiving an SA success signal before the VA stage is complete. Finally, the third example (c) illustrates the erroneous case of having more than one active read signals in the same router port in the same cycle.

presence of the multiplexer. The presence of two concurrently active read signals in the same input port indicates an invariance violation.

3.3. Identifying Invariances within the NoC Router

In order to identify a component's invariances, one must carefully examine the operation of said component within the context of its governing functional rules. In general, it is not always obvious that a range of values will never appear under normal operation. In the case of NoC routers, invariance identification is possible, because of the inherent modularity of the constituent modules. Each router module is usually responsible for a very specific task. For example, the RC unit is only tasked with the determination of the output direction[1] of a particular incoming packet. An arbiter grants one out of a number of requests, and the crossbar module is responsible for interconnecting input and output ports.

In this work, invariances were constructed by observing the operation and behavior of each functional module. Specifically, the list of invariances is constructed using a bottom-up approach. The NoC router design is implemented in a modular and hierarchical manner; e.g., FIFO buffers → Arbiters → Input Port → Crossbar Switch → … → Entire Router. The *algorithm* responsible for the *functional* operation of each module (e.g., the routing algorithm) is then exhaustively inspected to identify all *functional rules*. This analysis allows us to identify the functional rules of all components (which are not prohibitively many in a NoC), and, by extension, all *functionally illegal* outputs. Hence, the assertions are derived from each functional rule in the algorithm that describes the operation(s) of each module. This methodology is repeated for higher levels in the design hierarchy until the whole router is covered. Finally, end-to-end invariances at the network level (considered to be the highest level in the hierarchy) are also identified. To be able to follow the same procedure, designers must keep the design *modular*, so as to enable the decomposition of each module's operation.

By viewing the design *hierarchically* (not just locally), invariances manifesting in a coupled/combined manner are also covered. By gradually moving up the hierarchy (from individual modules to groups of modules), new assertions are derived from functional rules governing the higher levels of the hierarchy (e.g., rules that apply to the input-port-level).

The completeness of invariances depends on the completeness of the functional analysis of the design itself: a NoC consists of a set of functional rules. These rules are defined by the modules (and their interactions) within the routers. If the invariance checkers cover all functional rules, NoCAlert will detect *any illegal* behavior.

[1]Some routing algorithms also provide the output VC, in addition to the output direction [36].

All identified invariances are listed in Table 1, which will be described in more detail in Section 4.

It should be stressed at this point that *our focus is on the control logic* of the on-chip network. The data-path is usually protected by the well-established and ubiquitous practice of augmenting the flit payload with *Error Detecting Codes (EDC)* [19, 10]. In fact, more elaborate codes can also be employed that can even correct some errors (bit flips) within the flit *contents*. Hence, our only assumption in this paper is that the *contents* of the flits/packets are protected by a simple error detecting code, which will alert the system of any undesired alteration in the message contents (the code usually provides coverage for both the payload and the network overhead bits). In its simplest guise, the EDC could be a single-bit parity check.

Protecting the message *contents*, however, is not enough to guarantee the functional correctness of the NoC. Erroneous behavior within the control logic can lead to catastrophic results, since the control logic is the coordinator of the network's operation. Control logic upsets may lead to flit drop, packet loss, network/protocol deadlocks, network livelocks, packet mixing (which is not detected by the EDC, since it involves the flits of different packets erroneously following the same wormhole), and degraded performance (at best).

It is precisely for this reason that we advocate the incorporation of the NoCAlert mechanism into the on-chip network. NoCAlert provides on-line and real-time detection of faults within the control logic of the entire NoC. The flow of packets – from injection to ejection – is seamlessly monitored for any digression from normalcy. The NoCAlert scheme acts as a guardian of NoC operation and casts a protective blanket over the entire interconnect.

4. NoCAlert: An On-Line, Hardware-Assertion-Based Fault Detection Mechanism for NoCs

The proposed NoCAlert utilizes the principle of invariance checking and implements it in the form of ***real-time hardware-based assertions***. The key idea is to have a simple *hardware* checker module for every NoC component. This checker module will take as inputs the inputs and outputs of the protected component and it will check whether any functional rule is broken during the component's operation. Checkers mostly perform simple comparisons and, hence, they comprise simple combinational circuits with very low complexity.

As previously mentioned, in order to deploy and integrate the checkers in the router design, all the invalid outputs of all the main modules of the router had to be identified through a comprehensive analysis of each module's operation. This detailed exploration of

	Routing Computation (RC) Unit	
1	Illegal turn	Routing algorithms forbid some turns to prevent deadlocks in the network.
2	Invalid RC output direction	There are some invalid RC output directions. For example, if the router has five ports (numbered 1 to 5), value 6 is invalid [19].
3	Non-minimal routing (if required)	The RC unit's output direction must take the flit one step closer to its destination.
	Arbiter Modules (VA and SA Stages)	
4	Grant w/o request	It is not possible for a flit to win a grant without making a request.
5	Grant to nobody	The arbiter must always provide a winner when there is at least one client request.
6	1-hot grant vector	The arbiter's output vector must have at most one bit set to logic high.
7	Grant to occupied or full VC	A grant to an occupied or full VC (based on the neighbor's credits) is forbidden.
8	One-to-One VC assignment	An input VC must not be assigned to multiple output VCs.
9	One-to-One port assignment	An input port must not gain simultaneous access to multiple output ports.
10	VA agrees with RC	The output VC assigned by the VA unit must be in agreement with the result of the RC stage, as originally proposed in [19].
11	SA agrees with RC	The SA result must be in agreement with the result of the RC stage, as originally proposed in [19].
12	Intra-VA stage order	If a VC wins the VA2 arbitration stage, it must have also won the VA1 stage.
13	Intra-SA stage order	If a VC wins the SA2 arbitration stage, it must have also won the SA1 stage.
	Crossbar (XBAR)	
14	1-hot column control vector	At most one connection must be active in each column of a matrix-style XBAR in each clock cycle (to avoid flit mixing).
15	1-hot row control vector	At most one connection must be active in each row of a matrix-style XBAR in each clock cycle (to avoid unwanted multicasting).
16	# of Incoming flits equals # of Outgoing flits	During each clock cycle, the number of flits exiting the XBAR must be equal to the number of flits entering the XBAR.
	***Buffer State* (Note: Each VC buffer maintains its own state)**	
17	Consistent VC buffer state	The NoC router pipeline stages must be executed in the correct order.
18	Only header flits in free VC buffers	A VC buffer is free when it is not allocated to an in-flight packet. During this state, only a header flit may enter the buffers (i.e., a new packet creating a wormhole).
19	Invalid output VC value	At the end of the VA stage, the computed output VC of the packet is saved in order to extend and maintain the wormhole. The output VC value cannot be out of range [19].
20	Complete RC stage on a non-header flit	Routing computation is performed only on header flits. Thus, to make a transition from the RC to the VA stage, a header flit must be present at the head of the buffer.
21	Complete RC stage on an empty VC	A transition from the RC to the VA stage is forbidden if the buffer of the respective VC is empty.
22	Complete VA stage on a non-header flit	Virtual channel allocation is performed only on header flits. Thus, to make a transition from the VA to the SA stage, a header flit must be present at the head of the buffer.
23	Complete VA stage on an empty VC	A transition from the VA to the SA stage is forbidden if the buffer of the respective VC is empty.
24	Read from an empty buffer	A "read" signal cannot be issued to an empty VC buffer.
25	Write to a full buffer	A "write" signal cannot be issued to a full VC buffer.
26	Buffer atomicity violation (if required)	If the buffers are atomic, only flits from a single packet may reside in the buffer at any given time. Thus, a header flit cannot arrive at a non-free VC buffer.
27	Packet mixing in non-atomic buffer	If the buffers are non-atomic, a tail flit may only be followed by a header flit.
28	Packet flit-count violation	Typically, packets belonging to the same message class have the same length, i.e., the same number of flits. Thus, the number of a packet's flits arriving at a VC belonging to a specific message class must always be the same (equal to a pre-defined constant) [34].
	Port-Level Invariances	
29	Concurrent read from multiple VCs	Only one flit may leave a single input port in each clock cycle (due to multiplexer).
30	Concurrent write to multiple VCs	Only one flit may arrive at a single input port in each clock cycle (due to de-multiplexer).
31	Concurrent RC stage completion of multiple VCs	Since only one flit can arrive at an input port in a single clock cycle, only one VC may complete its RC stage in a single clock cycle in each input port [assuming that (a) atomic buffers are used, and (b) all VCs in a single port use the same routing algorithm].
	Network-Level Invariance	
32	End-to-End delivery violation	The destination address of all packets ejected from a node should equal that node's address.

Table 1: Complete list of the invariances associated with the baseline NoC router design of Figure 1. Note that Invariance 5 (shaded in grey) is innocuous if the fault causing it is *transient/intermittent* (leading only to momentary performance degradation analogous to a NOP instruction in a microprocessor), while it may prove catastrophic if the fault causing it is *permanent* (packets stuck in NoC buffers).

the router's micro-architecture identified a total of **32 invariances**, which are listed in Table 1. The invariances are categorized based on the router module they are associated with: the Routing Computation unit, Arbiters, Crossbar, VC State, Port-Level, and End-to-End.

This list of invariances *completely* characterizes the operational behavior of the router: *any forbidden behavior* (as dictated by the functional rules that govern the router's operation) will be captured by *at least one* of these 32 assertion checkers.

4.1. Ensuring Network Correctness Using Invariances

As NoCs are increasingly becoming more complex, the task of ensuring their functional correctness as a whole is becoming more daunting. Prior research [37, 15] has identified four main conditions that *ensure* functional correctness within the network: (1) No packets are dropped, (2) Delivery time is bounded, (3) No data corruption occurs, and (4) No new packet is generated within the network. If satisfied, these four conditions guarantee functional correctness [37, 15].

Following this guideline, the 32 invariances of Table 1 are categorized according to the aforementioned four general requirements, as illustrated in Figure 3. Each number in the diagram refers to the corresponding entry of Table 1. Even though the original categorization of [37, 15] was made at the *packet* level, we choose to operate at the *flit* level, since the smallest unit of flow control is the flit. By

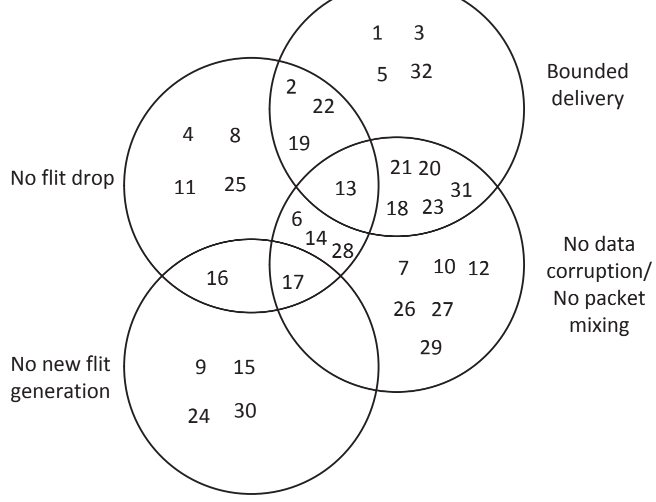

Figure 3: Using invariances to ensure functional correctness. The 32 invariances of Table 1 are categorized based on the 4 fundamental conditions that ensure functional correctness within the NoC [37, 15].

doing this transformation, we actually make the four requirements even stronger, because flits are sub-units of packets. For example, if an extra flit is generated in the network and becomes part of an existing packet, the flit-level rule will correctly identify this as an error, whereas the packet-level rule would not capture this anomaly. Note that operating at the flit level adds the additional requirement that *intra-packet* flit ordering is maintained by the network (a typical assumption in NoCs). Upsets causing such flit ordering violations also violate some of the fundamental invariances monitored by NoCAlert; thus, the proposed mechanism also safeguards against intra-packet flit order changes.

Bounded delivery implies the delivery of all flits to their intended destination within a finite amount of clock cycles. This rule specifies that no deadlock or livelock should occur in the network. *No flit drop* specifies that no flit should be lost during its traversal through the network. *No new flit generation* specifies that no new flits should be spontaneously generated within the NoC. Abnormal flit duplication is also included in this requirement. Finally, *no data corruption/packet mixing* specifies that there should be no collision of flits, and that no flit belonging to a packet should enter the wormhole of another packet (packet mixing). Even though the message contents are assumed to be protected by error-detecting codes, data corruption could still occur by packet mixing, which would escape the per-flit error-detecting codes.

Due to lack of space, only two of the 32 invariances of Table 1 will be described in detail. Specifically, two invariances will be analyzed, which can cause several types of errors. In particular, invariances 13 and 17 sit at the intersection of multiple categories in Figure 3 and may breach *three* out of the four functional correctness requirements.

As described in Section 3.1, the Switch Arbitration stage is further separated into the SA1 (local) and SA2 (global) arbitration stages. Invariance 13 (see Table 1) specifies that if a VC wins in the SA2 arbitration, it must also have won its SA1 arbitration. If a VC wins the SA2 stage without winning SA1, there is a possibility of being forwarded to a full VC in the adjacent router (since credits are evaluated in SA1), and, therefore, it will be dropped (*No flit drop* violation). Additionally, since the SA2 stage drives the crossbar switch,

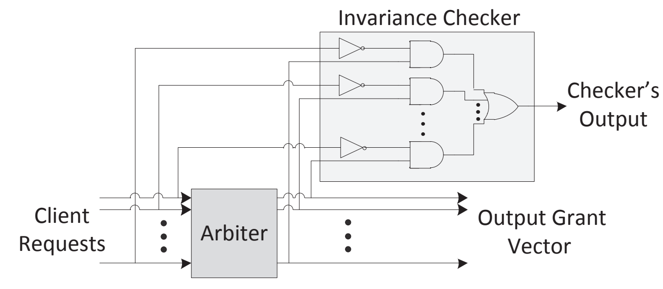

Figure 4: An example NoCAlert checker circuit. This checker constantly monitors an arbiter module at run-time to detect whether a grant signal has been issued at the arbiter's output without any requests at the arbiter's inputs. Note that the figure is not drawn to scale; the checker module is exaggerated for clarity. In reality, the invariance checkers are significantly smaller than the units they check.

a flit might be sent in a different direction than the one calculated by the RC unit. If the flit happens to be sent to an idle VC, this may lead to a deadlock due to "breaking up" of the packet (malfunctioning wormhole). Thus, a *Bounded delivery* rule violation will occur. Finally, if the flit is sent to an occupied VC, the *No packet mixing* rule will be breached.

Invariance 17 states that pipeline stages must be executed in the correct order. Suppose that the VA stage is executed before the RC stage. In this case, the flits of the packet might be forwarded to an occupied VC in the adjacent router (*No packet mixing* violation). Now suppose that the SA stage is executed before the VA stage. The flit will be forwarded to the adjacent router without correct VC ID information. Thus, the flit will be written to an arbitrary VC. If that VC is full, the flit will be dropped (*No flit drop* violation). Finally, suppose that the SA stage is executed before the RC stage, i.e., on an empty VC buffer. This will cause a flit to be forwarded to an adjacent router, but garbage information will be sent (since buffers employ pointers to maintain FIFO order, an "empty" buffer slot is not blank). Therefore, a new flit may be generated (*No new flit generation* violation).

4.2. Hardware Complexity of the NoCAlert Checkers

The NoCAlert fault detection mechanism consists of an array of distributed hardware checkers, which constantly and seamlessly monitor the modules comprising the control logic of the router. Each checker is a simple combinational circuit performing a specific check, according to the rules of the module being monitored.

An example checker circuit is shown in Figure 4. This checker is responsible to monitor an arbiter module and detect whether there is an active grant signal without any requests at the arbiter's inputs. As can be seen from the figure, only two logic gates are needed for each input/output of the arbiter, as well as an OR gate to combine all individual checks. Furthermore, the checker size grows *linearly* with the number of arbiter inputs, whereas the arbiter size grows in a polynomial fashion.

Invariance checking relies mostly on value comparison, which, in hardware terms, translates into simple combinational circuits consisting of inverters, AND, OR, and XOR gates. Therefore, the NoCAlert checkers provide a lightweight and holistic approach to run-time fault detection, as will be demonstrated through hardware synthesis results in Section 5.

4.3. Faults That Do Not Cause Invariance Violations

As previously mentioned, invariance checking only detects *illegal* outputs, not necessarily *incorrect* ones. Faults that give rise to *func-*

tionally legal outputs – based on the given input – will not be detected. A simple NoC example to illustrate this scenario is the RC unit's functionality. Suppose a packet enters a router from the East port and is destined to the West output port. Even under deterministic XY routing, a misdirection to the North output port will *not* constitute an invariance violation, since X-to-Y turns are allowed in XY routing. Moreover, *adaptive* routing algorithms – such as Duato's Protocol [38] – inherently allow more than one routing options to avoid congestion. It is clear that faults in the RC unit have a good chance of still returning a valid/legal output that does not violate any invariance.

The two elemental questions here are the following:

- If such non-invariant upsets cause some other functional/invariance violation *later on* in the network, will the fault be caught by one/some subsequent NoCAlert checkers?
- If these non-invariant upsets do *not* cause any other functional/invariance violation *later on* in the network (i.e., they are never caught by any NoCAlert checker), do they end up affecting the overall network correctness (as defined in Section 4.1 and [37, 15])?

An example relevant to the second question is when a packet requests VC1 of a specific output port, but a fault occurrence causes the grant of, say, VC2 of the same output port, which also happens to be available. If both of these VCs belong to the same protocol-level message class, then this fault does *not* cause an invariance violation and it is, in fact, *benign*, i.e., no functional error manifests itself at the network or system level later on.

The extensive simulations of Section 5 will answer these two important questions. It turns out (empirically) that all the non-invariant faults that end up causing a functional error later on are, indeed, successfully captured by subsequent NoCAlert checkers, whereas the non-invariant faults that do not cause any other invariance violation later on turn out to be benign, as far as overall network correctness is concerned.

4.4. Applicability of the NoCAlert Framework to Any Router Micro-architecture

Based on our exploration so far, it is clear that the invariance concept is closely related to the micro-architecture under test. Changes in the router's micro-architecture may result in subtle (or not so subtle) changes in the components' invariances. However, the underlying principles will still be the same: study each individual module and identify invariances, while gradually moving up to coarser granularities (e.g., port-level). There are many proposed router architectures [39, 33, 40, 41], with each one involving changes to the constituent modules, or the pipeline stages and associated flow. The inherent modularity of all router designs (a direct consequence of the router's parallel nature) allows the designer to fairly easily identify the new functional invariances.

This sub-section will briefly investigate the key changes to the invariances of the generic router model when some key router parameters are varied. The chosen variations are typical alterations observed in the literature. For example, the router design may forego the use of VCs, it may choose to employ non-atomic FIFO buffers, it may implement a speculative design (e.g., the VA and SA happening concurrently), and it may employ a more elaborate routing algorithm. By exploring how these changes will affect invariance checking, one may appreciate the flexibility and widespread applicability of the NoCAlert scheme.

In the absence of virtual channels in the design, the VA pipeline stage is eliminated. Hence, all the invariance checks pertaining to the VA stage in Table 1 may be removed. For example, invariances 29 and 30 in the table are no longer needed.

Non-atomic buffers allow the simultaneous storage of flits belonging to different packets (albeit without mixing), unlike the atomic buffers in the baseline architecture, which only allow the flits of a single packet to reside in the buffer at any given time. If non-atomic buffers are used, all invariances that forbid a new packet to arrive in an already-occupied VC buffer are discarded. At the same time, however, a new invariance is created (invariance 27 in Table 1). The mixing of flits from two different packets is still forbidden in non-atomic buffers. This means that an assertion should be raised if the flit following a tail flit is *not* a header flit of a new packet.

In speculative router designs [36], the VA and SA stages are executed in parallel. In this case, the SA may, in fact, finish before the VA stage. Thus, invariance 17 in Table 1 must be altered, so as not to raise an assertion if SA succeeds before VA is done.

The functional definition of a routing algorithm defines its invariances. Most routing algorithms have some turn restrictions in order to prevent network deadlocks and livelocks, as well as protocol deadlocks. Some routing algorithms also provide a specific output VC, in addition to the output direction. In all cases, the NoCAlert checkers are derived from these restrictions. For example, Duato's Protocol [38] dictates that "when making a turn from the East to the North, a packet must enter VC0." This statement immediately defines an assertion checker.

5. Experimental Evaluation

5.1. Evaluation Framework

The goal of the experimental evaluation is to thoroughly assess the *efficacy* and *efficiency* of the NoCAlert mechanism in a realistic environment. Our evaluation approach is double-faceted and consists of (a) extensive simulations in a cycle-accurate simulator, and (b) hardware evaluation based on a full Verilog implementation of NoCAlert and synthesis using 65 nm commercial standard-cell libraries.

For the former part, the cycle-accurate GARNET NoC simulator [42] is employed. GARNET models the packet-switched routers down to the micro-architectural level. The simulator was further extended with all the checker modules listed in Table 1 (see Section 4) and the fault injection framework to be described in Section 5.2.

Since the focus of this work is the *fault detection performance* of NoCAlert (and not the network/system performance), the use of synthetic traffic patterns in an 8×8 mesh suffices to accurately capture the salient characteristics of the design. Synthetic traffic patterns are typically more effective in stressing the router design to its limits and isolating the inherent attributes of the network itself. Hence, we employ synthetic (uniform random) traffic at various injection rates to ensure all router components are stressed over a range of traffic intensities.

The NoCAlert framework is also compared to ForEVeR [15], a recently proposed state-of-the-art fault detection and recovery framework (see Section 2). The ForEVeR mechanism was cycle-accurately implemented within GARNET with all three of its key fault-detecting techniques: the secondary checker network (including the counters and timers), the Allocation Comparator from [19], and the end-to-end checker.

Without loss of generality, the router architecture assumed in this evaluation is the baseline implementation described in Section 3.1. The router is five-stage pipelined (4 intra-router stages + 1 link traversal stage), with four 5-flit deep VCs per input port, and 128-bit inter-router links. Atomic VC buffers, wormhole switching, and credit-based flow control are also assumed. The routing algorithm used is deterministic XY.

For the hardware evaluation part, we implemented the baseline NoC router augmented with the NoCAlert mechanism (all 32 invariance checker modules) in Verilog Hardware Description Language (HDL). The resulting design was synthesized using Synopsys Design Compiler and 65 nm commercial TSMC libraries at 1 V operating voltage and 1 GHz clock frequency. The results were used to perform detailed area/power/timing analysis and evaluate the overhead footprint of NoCAlert.

5.2. Fault Model and Fault Injection Framework

Throughout the evaluation, we assume the occurrence of *single* faults in the NoC mesh. Specifically, the simulator injects *single-bit, single-event transient faults* at different locations and at different instances (network states). The above-mentioned fault model is widely used in the literature and it was chosen as a proof-of-concept for NoCAlert. More elaborate fault models are left for future work. Even though we employ *transient* fault injections for the purposes of our simulations, the mechanism works with permanent failures in an identical manner. Effectively, the fault model used evaluates fault behavior for single-event upsets and single permanent faults. The difference is that the NoCAlert checkers will raise permanent/prolonged (rather than momentary) assertions upon the occurrence of a permanent/intermittent fault. In other words, since the NoCAlert checkers raise an exception upon an invariance violation, NoCAlert's performance/accuracy is orthogonal to whether the invariance is temporary or permanent; as soon as the invariance violation *commences*, NoCAlert will detect it. The reasoning is that a permanent fault, or an intermittent fault, will trigger the same checker as a transient fault, but the checker's flag will remain raised for more than one cycle (indicating an intermittent, or permanent, fault). Note that even if the erroneous value disappears after one clock cycle, the effects of that short "malfunction" perturbation may propagate through the network with unpredictable results.

Our fault model looks at the router micro-architecture at the **fine granularity of individual sub-components**. These sub-components comprise all the modules responsible for the router's control logic: individual RC units, control status tables, VC buffer status, arbiters in both VA and SA, and the crossbar control logic. Our only assumption is that the packet/flit *contents* are already protected by error-detecting codes (see Section 3.3), so the datapath of the router is also covered. Our model has the capability of *injecting single-bit faults at the inputs and the outputs of each individual module*. The fault injection framework is illustrated in Figure 5. By looking at the router micro-architecture at this fine granularity, we are able to inject single-bit faults at 205 different locations within a single 5-port NoC router. Taking into account corner and edge routers (which have fewer ports), the *total number of fault locations* is 11,808 in an 8×8 mesh network.

5.3. Experimental Methodology

One simulation run at a single traffic injection rate and one network state consists of 11,808 different simulations (to exhaustively inject faults in all possible locations of an 8×8 mesh, assuming the specific single-fault injection model used in this work). The traffic injection rate was varied from low to high (0.1–0.4 flits/node/cycle) in steps of 0.05 flits/node/cycle. Moreover, three different scenarios of fault injection instances were studied (fault injection at cycle 0, 32K, and 64K). Hence, 21 different scenarios were investigated (7 injection rates × 3 injection times), for a total of $21 \times 11,808 \approx 248$K fault-injection simulations.

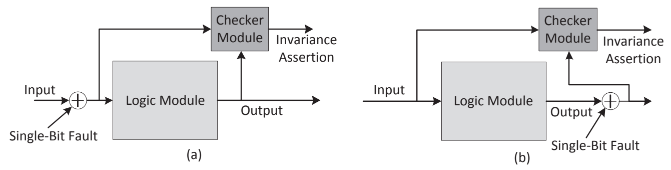

Figure 5: Abstract view of the employed fault injection framework. The evaluation framework used in this work has the capability of injecting single-bit faults at the inputs (a), or outputs (b), of each individual router module. The module granularity is very fine, which results in 11,808 possible fault locations in an 8×8 mesh network.

The exact same experiments were also run in a *fault-free* environment and detailed flit ejection logs were collected and compiled in a so called *Golden Reference* (GR) report. The GR is then used to ensure that no violations of the four network correctness rules of Section 4.1 and [37, 15] occur. Furthermore, the GR also detects any changes in the intra-packet flit order (as previously mentioned, such order violations constitute erroneous behavior). Since NoCAlert only captures faults that cause invariances, the GR is used to facilitate the investigation of the two key questions posed in Section 4.3. Moreover, by comparing the GR with the equivalent *under-fault* log report, we can study the effects of any fault occurrence on overall network correctness. This allows us to assess the *false positive* (assertions that prove benign) and *false negative* (undetected network correctness violations) performances of both NoCAlert and ForEVeR [15].

5.4. Simulation Results

As discussed in Section 5.1, simulation experiments were performed in an 8×8 2D mesh network using synthetic traffic patterns. In this sub-section, we present the results and an evaluation of NoCAlert's efficacy and efficiency in terms of several key metrics. Moreover, we conduct a quantitative comparison with the ForEVeR [15] framework.

We begin our exploration with NoCAlert's fault detection capabilities. It is important at this point to differentiate the *injected faults* from the *actual errors* manifesting themselves at the network-level (as defined in Section 4.1 and [37, 15]). NoCAlert's ultimate goal is to ensure that no *actual error at the network-level* escapes detection. Therefore, injected faults that do NOT cause a real functional error within the network are viewed as benign. Based on this crucial differentiation, we classify each of NoCAlert's detection outcomes into one of four main categories:

- **True Positive**: Event detected by NoCAlert when the injected fault causes an actual error at the network-level (network correctness violation).
- **False Positive**: Event detected by NoCAlert when the injected fault turns out to be benign.
- **True Negative**: Nothing detected by NoCAlert when the injected fault turns out to be benign.
- **False Negative**: Nothing detected by NoCAlert when the injected fault causes an actual error at the network-level (network correctness violation).

In order to identify which injected faults turned out to be malicious (i.e., they caused a network correctness violation), we used the Golden Reference (GR) log report described in Section 5.3.

Obviously, the most important metric when evaluating the performance of a *detection* mechanism is the occurrence of *False Negatives*, i.e., actual faults that evade the detection process.

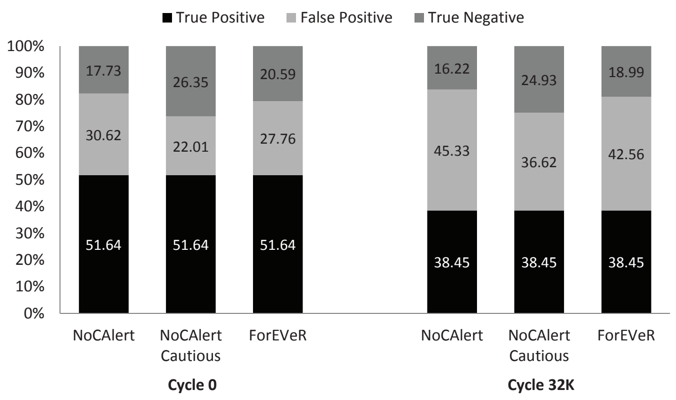

Figure 6: Fault coverage breakdown (over all injected faults) using synthetic (uniform random) traffic in an 8×8 mesh at two different fault injection instances (cycle 0 and cycle 32K). The "NoCAlert Cautious" bars refer to a system where Invariances 1 and 3 of Table 1 are considered low risk (see text for details).

Observation 1: Out of all the simulations we ran, NoCAlert registered *zero* false negatives. In other words, *all faults that violated network correctness were successfully captured* by NoCAlert. The same was true for ForEVeR [15]. Thus, both mechanisms exhibit the same fault detection *accuracy*.

Figure 6 presents a breakdown of the fault detection performance of both NoCAlert and ForEVeR at two different fault injection instances: cycle 0 and cycle 32K (the results for fault injection at cycles 32K and 64K are very similar; thus, only the 32K results are shown here for brevity). The results for cycle 0 are representative of an empty network, while the results for cycle 32K are representative of networks at steady-state (warmed up). Note that the true positive percentages are identical for NoCAlert and ForEVeR, since both mechanisms detected *all* network correctness violations (all of the injected faults that actually violated the network correctness). The notable difference is in the *False Positives*, where NoCAlert is slightly worse in both cases. This result is attributed to the real-time nature of NoCAlert, which raises assertions instantaneously. Instead, ForEVeR is epoch-based, which means that some benign faults simply "vanish" by the time the epoch expires. In general, the false positive percentages are higher for cycle 32K – as compared to cycle 0 – because violations are more likely to be masked by other traffic in a warmed up network. For example, in an empty network, an erroneous switch allocation request would propagate to the output uncontested (since there are no other packets competing for crossbar access). However, in a more congested environment, the erroneous request may lose the arbitration to another packet.

The false positive percentages may be markedly reduced if the *recovery* mechanism's reaction is guided by the checkers' *risk levels*. In other words, the NoCAlert checkers may be classified into different categories, based on their *risk levels*. Low-risk checkers would trigger a *delayed/deferred* response, in order to account for the high probability of a false positive. For instance, we made a very interesting observation regarding NoCAlert. In our conducted set of experiments, invariances 1 and 3 of Table 1 *never* led to network-level incorrectness when asserted *alone*, even though they might *theoretically* have led to a deadlock. These invariances are violated if the RC unit misdirects a header flit (possibly in a direction further away from the packet's destination). We noticed that many benign faults registered as false positives by NoCAlert were caused by those two invariances. In all those cases, the invariances were asserted by

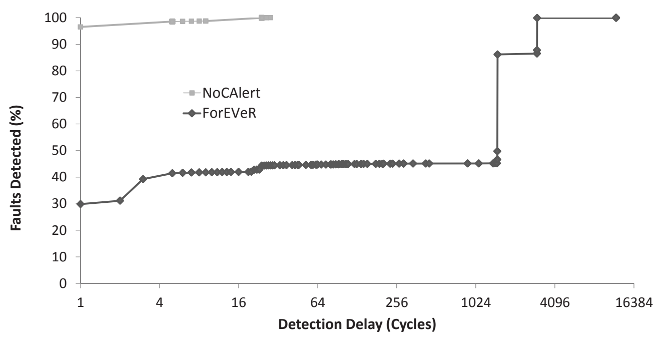

Figure 7: Cumulative fault-detection delay distribution for the true positive faults (The epoch duration in ForEVeR was set to 1,500 cycles; see text for details).

themselves (no other assertion was raised). Hence:

Observation 2: If the fault recovery mechanism connected to NoCAlert sees either Invariance 1 or Invariance 3 (Table 1) violated, without any other assertions raised, it could move into a "cautious" state, whereby the fault recovery mechanism is not triggered until there is further evidence later on that a deadlock actually occurred. If this strategy is followed, then NoCAlert's *False Positive* rates would drop to 22.01% and 36.62% for cycles 0 and 32K, respectively, as indicated by the "NoCAlert Cautious" bars in Figure 6.

Invariance 5 in Table 1 exhibits noteworthy behavior. Said invariance is violated whenever an arbiter produces an all-zero grant vector (i.e., no arbitration winner is declared), even though there was at least one active client request. If the fault is transient or intermittent, this fault would only result in brief performance degradation, similar to a NOP instruction in a microprocessor's pipeline. However, if the fault is permanent (i.e., the checker remains permanently asserted), the consequences could be quite dramatic. The fault may lead to network/protocol deadlocks.

Observation 3: Invariance 5 in Table 1 exhibits the unique characteristic of being benign (in terms of network correctness) under transient/intermittent faults, but malicious under permanent faults.

The real strength of NoCAlert is its fault detection *latency*. Figure 7 shows the cumulative fault detection delay distribution for both NoCAlert and ForEVeR. In this figure, only faults that resulted in true positives were evaluated. Notice that 97% of all faults are captured instantaneously (in the same cycle) by NoCAlert, 99% are captured within 9 clock cycles, and 100% are captured after 28 cycles. ForEVeR's epoch-based scheme takes significantly longer, with 99% of faults being captured after 3,000 cycles and 100% captured after 11,995 cycles. The epoch duration in ForEVeR was set to 1,500 cycles, which was the shortest period that did not yield excessive false positives *under the fault model employed in this work*. These results demonstrate that:

Observation 4: NoCAlert provides near-instantaneous fault detection with a staggering 97% of all true positive faults captured at the instance of injection (same cycle). The worst-case detection latency is only 28 cycles after fault injection. Moreover, NoCAlert achieves *several orders of magnitude* lower fault detection latency than ForEVeR [15].

In order to answer the critical questions posed in Section 4.3, we need to examine all injected faults that did not result in an invariance violation at the instance of fault injection. It turns out that 78% of those faults did not cause a subsequent invariance violation, and *all of them* turned out to be benign (no network correctness violation). The remaining 22% caused a subsequent invariance violation and

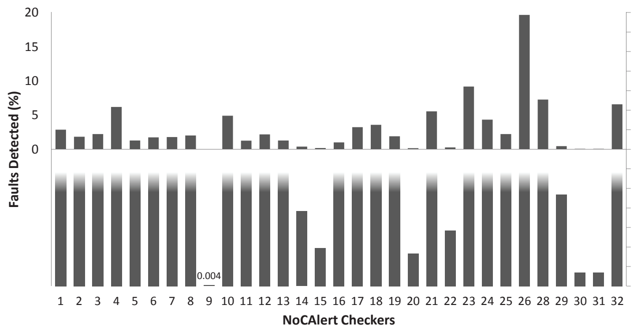

Figure 8: Percentage of invariance violations captured by each individual NoCAlert checker of Table 1 (over all experiments). The bottom part of the figure has a finer y-axis scale and focuses on the very low y-axis values of the top part. Invariance 27 is missing, because it is only applicable to non-atomic VC buffers.

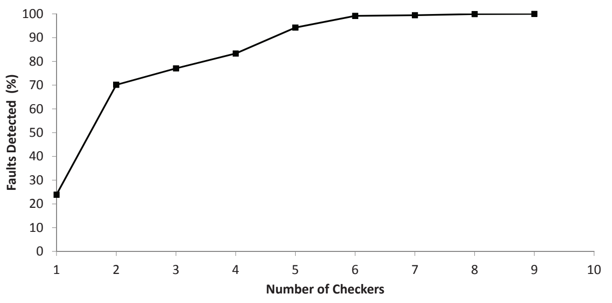

Figure 9: Cumulative distribution of invariance violations as a function of the number of simultaneously asserted checkers.

were successfully captured by NoCAlert.

Observation 5: Injected faults that do *not* cause any invariance violation in the network are *always* benign (i.e., they never cause any network correctness violation).

Figures 8 and 9 evaluate the behavior of the 32 invariance checkers. Specifically, Figure 8 shows the percentage of invariance violations caught by each individual checker over all experiments. Note that Invariance 27 is missing, because it refers to non-atomic buffers (we used atomic VC buffers in our simulations, as stated in Section 5.1). It should also be noted that all checkers detected invariances in the absence of any other checker assertions. This fact indicates that *no single checker is redundant*. Finally, Figure 9 shows the cumulative distribution of invariance violations as a function of the number of simultaneously asserted checkers. Most invariances were caught by two checkers, while the maximum number of checkers triggered due to a single invariance violation was 9.

5.5. Hardware Evaluation – Area/Power/Timing Overhead

As described in Section 5.1, a baseline NoC router augmented with the complete NoCAlert mechanism was implemented in Verilog HDL and synthesized using 65 nm commercial standard-cell libraries.

In order to assess the *scalability* of NoCAlert, we vary the number of VCs per port from two [41] to eight [39] and evaluate the NoCAlert percentage area and power overhead. The number of VCs per port is dictated by the employed routing algorithm (e.g., deterministic vs. adaptive) and/or the cache-coherence protocol (number of message classes). The **area** results are shown in Figure 10. To better appreciate the size of NoCAlert, we also implemented a design with Double Modular Redundancy (DMR) in the entire NoC control logic (designated as "DMR-CL" in the figure). DMR serves

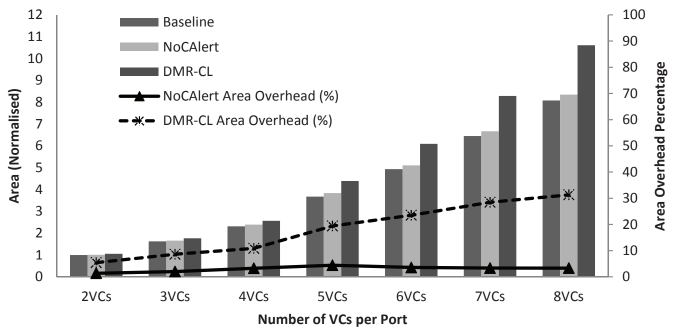

Figure 10: The NoCAlert area overhead as a function of the number of VCs per input port. A comparison with double modular redundancy in the control logic ("DMR-CL") is also presented.

as the most complete fault detection solution possible, albeit a very expensive one. Clearly, the NoCAlert area overhead is minimal and ranges from 1.38% to 4.42% (3%, on average) and the *percentage* overhead remains fairly constant as the number of VCs increase. On the other hand, the *percentage* area overhead of DMR increases linearly from 5.41% in the case of two VCs, up to 31.32% in the case of eight VCs per port.

The **power** results exhibit the same trends and are, thus, omitted for brevity. The absolute numbers, however, are much smaller for NoCAlert, since the checkers comprise purely combinational logic and have no power-hungry storage elements. Hence, the percentage power overhead ranges from 0.3% to 1.2% (0.7%, on average), i.e., it is negligible. The power numbers were extracted from the Synopsys Design Compiler power report, with switching activity set to 50% for all nets.

The final key design metric evaluated was the **critical path delay**, which sets the maximum possible operating frequency. Our synthesis results indicate minimal impact on the critical path of at most 3% and, on average, around 1%. This means that the proposed NoCAlert mechanism is, essentially, transparent to overall network operation.

These results corroborate the fact that NoC control logic checkers used to detect only illegal outputs have significantly lower hardware cost than the units they check.

6. Conclusions

This paper proposes NoCAlert, a comprehensive on-line and real-time fault detection mechanism that ensures 0% false negatives within the NoC, under the employed fault model. NoCAlert is based on the concept of *invariance checking*, whereby the outputs of the control logic modules of the on-chip network are constantly checked for *illegal* outputs, based on current inputs. By combining a collection of such micro-checker modules dispersed throughout the router's control logic modules, the proposed mechanism implements real-time hardware assertions. The checkers operate seamlessly and concurrently with normal NoC operation, thus obviating the need for periodic (epoch-based), or triggered-based, self-testing.

Extensive simulation results validate the efficacy of the NoCAlert mechanism and yield important insight as to the behavior of the network when non-invariant faults (that evade the checkers) occur. Specifically, non-invariant faults either cause some subsequent invariance violation (and are captured), or they prove benign at the network/system level. Hardware synthesis analysis using 65 nm commercial libraries demonstrates the extremely lightweight nature of NoCAlert in terms of area/power/timing overhead. Furthermore, a detailed comparison with a recently proposed framework [15] highlights higher than 100× improvements in detection latency, with no loss in detection accuracy and with much lower overall complexity.

In summary, this work demonstrates the potential for extremely accurate and near-instantaneous fault detection within the NoC using minimally intrusive hardware-based invariance checkers.

Acknowledgments

This work was supported by "EuroCloud, Project No 247779" of the European Commission 7th RTD Framework Programme – Information and Communication Technologies: Computing Systems, and by the Cyprus Research Promotion Foundation's Grant ΤΠΕ/ΠΛΗΡΟ/0609(ΒΙΕ)/09 (co-funded by the Republic of Cyprus and the European Regional Development Fund).

References

[1] S. Damaraju et al. A 22nm ia multi-cpu and gpu system-on-chip. In *Proc. of the International Solid-State Circuits Conference (ISSCC)*, 2012.

[2] K. Olukotun et al. The case for a single-chip multiprocessor. In *Proc. of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 1996.

[3] W.J. Dally and B. Towles. Route packets, not wires: on-chip interconnection networks. In *Proc. of the Design Automation Conference (DAC)*, 2001.

[4] S.R. Nassif, N. Mehta, and Yu Cao. A resilience roadmap. In *Proc. of the Design, Automation and Test in Europe Conference (DATE)*, 2010.

[5] S. Borkar. Microarchitecture and design challenges for gigascale integration. In *Proc. of the International Symposium on Microarchitecture (MICRO)*, 2004.

[6] E Wu et al. Interplay of voltage and temperature acceleration of oxide breakdown for ultra-thin gate oxides. In *International Journal of Solid-State Electronics*, November 2002.

[7] C. Nicopoulos et al. On the effects of process variation in network-on-chip architectures. *In IEEE Trans. on Dependable and Secure Computing*, July 2010.

[8] K. Aisopos, C.-H.O. Chen, and L.S. Peh. Enabling system-level modeling of variation-induced faults in networks-on-chips. In *Proc. of the Design Automation Conference (DAC)*, 2011.

[9] K. Constantinides et al. Bulletproof: a defect-tolerant cmp switch architecture. In *Proc. of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2006.

[10] D. Fick et al. Vicis: A reliable network for unreliable silicon. In *Proc. of the Design Automation Conference (DAC)*, 2009.

[11] M.R. Kakoee, V. Bertacco, and L. Benini. Relinoc: A reliable network for priority-based on-chip communication. In *Proc. of the Design, Automation and Test in Europe Conference (DATE)*, 2011.

[12] A. Strano et al. Exploiting network-on-chip structural redundancy for a cooperative and scalable built-in self-test architecture. In *Proc. of the Design, Automation and Test in Europe Conference (DATE)*, 2011.

[13] M. Hosseinabady, A. Dalirsani, and Z. Navabi. Using the inter- and intra-switch regularity in noc switch testing. In *Proc. of the Design, Automation and Test in Europe Conference (DATE)*, 2007.

[14] M.R. Kakoee, V. Bertacco, and L. Benini. A distributed and topology-agnostic approach for on-line noc testing. In *Proc. of the International Symposium on Networks-on-Chip (NOCS)*, 2011.

[15] R. Parikh and V. Bertacco. Formally enhanced runtime verification to ensure noc functional correctness. In *Proc. of the International Symposium on Microarchitecture (MICRO)*, 2011.

[16] A. Kohler, G. Schley, and M. Radetzki. Fault tolerant network on chip switching with graceful performance degradation. *In IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, June 2010.

[17] Albert Meixner, Michael E. Bauer, and Daniel Sorin. Argus: Low-cost, comprehensive error detection in simple cores. In *Proc. of the International Symposium on Microarchitecture (MICRO)*, 2007.

[18] M. Koibuchi et al. A lightweight fault-tolerant mechanism for network-on-chip. *in Proc. of the International Symposium of Networks-on-Chip (NOCS)*, 2008.

[19] D. Park et al. Exploring fault-tolerant network-on-chip architectures. In *Proc. of the International Conference on Dependable Systems and Networks (DSN)*, 2006.

[20] S. Shamshiri, A. Ghofrani, and Kwang-Ting Cheng. End-to-end error correction and online diagnosis for on-chip networks. In *Proc. of the International Test Conference (ITC)*, 2011.

[21] M. Palesi, S. Kumar, and V. Catania. Leveraging partially faulty links usage for enhancing yield and performance in networks-on-chip. *In IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, March 2010.

[22] J. Duato. A theory of fault-tolerant routing in wormhole networks. *In IEEE Trans. on Parallel and Distributed Systems (TPDS)*, August 1997.

[23] W.J. Dally et al. The reliable router: A reliable and high-performance communication substrate for parallel computers. In *Proc. of the International Workshop on Parallel Computer Routing and Communication (PRCW)*, 1994.

[24] S. Rodrigo et al. Addressing manufacturing challenges with cost-efficient fault tolerant routing. In *Proc. of the International Symposium on Networks-on-Chip (NOCS)*, 2010.

[25] T. Dumitraş, S. Kerner, and R. Mărculescu. Towards on-chip fault-tolerant communication. In *Proc. of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2003.

[26] B. Fu et al. An abacus turn model for time/space-efficient reconfigurable routing. In *Proc. of the International Symposium on Computer Architecture (ISCA)*, 2011.

[27] T. Moscibroda and O. Mutlu. A case for bufferless routing in on-chip networks. In *Proc. of the International Symposium on Computer Architecture (ISCA)*, 2009.

[28] A. Kohler and M. Radetzki. Fault-tolerant architecture and deflection routing for degradable noc switches. In *Proc. of the International Symposium on Networks-on-Chip (NOCS)*, 2009.

[29] D. Fick et al. A highly resilient routing algorithm for fault-tolerant nocs. In *Proc. of the Design, Automation and Test in Europe Conference (DATE)*, 2009.

[30] S. Murali et al. A multi-path routing strategy with guaranteed in-order packet delivery and fault-tolerance for networks on chip. In *Proc. of the Design Automation Conference (DAC)*, 2006.

[31] K. Aisopos et al. Ariadne: Agnostic reconfiguration in a disconnected network environment. In *Proc. of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2011.

[32] V. Puente et al. Immunet: Dependable routing for interconnection networks with arbitrary topology. *In IEEE Trans. on Computers*, December 2008.

[33] J. Kim et al. A gracefully degrading and energy-efficient modular router architecture for on-chip networks. In *Proc. of the International Symposium on Computer Architecture (ISCA)*, 2006.

[34] A. Ghofrani et al. Comprehensive online defect diagnosis in on-chip networks. In *Proc. of the VLSI Test Symposium (VTS)*, 2012.

[35] C. A. R. Hoare. An axiomatic basis for computer programming. *In Communications of the ACM*, October 1969.

[36] L.S. Peh and W.J. Dally. A delay model and speculative architecture for pipelined routers. In *Proc. of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2001.

[37] D. Borrione et al. A generic model for formally verifying noc communication architectures: A case study. In *Proc. of the International Symposium on Networks-on-Chip (NOCS)*, 2007.

[38] J. Duato, S. Yalamanchili, and L. Ni. *Interconnection Networks: An Engineering Approach*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1st edition, 1997.

[39] J. Howard et al. A 48-core ia-32 processor in 45 nm cmos using on-die message-passing and dvfs for performance and power scaling. *In IEEE Journal of Solid-State Circuits*, Jan. 2011.

[40] A. Kumary et al. A 4.6tbits/s 3.6ghz single-cycle noc router with a novel switch allocator in 65nm cmos. In *Proc. of the International Conference on Computer Design, 2007. (ICCD)*, 2007.

[41] S.R. Vangal et al. An 80-tile sub-100-w teraflops processor in 65-nm cmos. *In IEEE Journal of Solid-State Circuits*, Jan. 2008.

[42] N. Agarwal et al. Garnet: A detailed on-chip network model inside a full-system simulator. In *Proc. of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2009.