An Online and Real-Time Fault Detection and Localization Mechanism for Network-on-Chip Architectures

KYPROS CHRYSANTHOU, PANAYIOTIS ENGLEZAKIS, ANDREAS PRODROMOU, ANDREAS PANTELI, CHRYSOSTOMOS NICOPOULOS, and YIANNAKIS SAZEIDES, University of Cyprus
GIORGOS DIMITRAKOPOULOS, Democritus University of Thrace

Networks-on-Chip (NoC) are becoming increasingly susceptible to emerging reliability threats. The need to *detect* and *localize* the occurrence of faults at runtime is steadily becoming imperative. In this work, we propose ***NoCAlert***, a comprehensive online and real-time fault detection and localization mechanism that demonstrates 0% false negatives within the interconnect for the fault models and stimulus set used in this study. Based on the concept of invariance checking, NoCAlert employs a group of lightweight microchecker modules that collectively implement real-time hardware assertions. The checkers operate concurrently with normal NoC operation, thus eliminating the need for periodic, or triggered-based, self-testing. Based on the pattern/signature of asserted checkers, NoCAlert can pinpoint the location of the fault at various granularity levels. Most important, 97% of the transient and 90% of the permanent faults are detected instantaneously, within a single clock cycle upon fault manifestation. The fault localization accuracy ranges from 90% to 100%, depending on the desired localization granularity. Extensive cycle-accurate simulations in a 64-node CMP and analysis at the RTL netlist-level demonstrate the efficacy of the proposed technique.

CCS Concepts: • **Networks** → **Network on chip**; • **Hardware** → *Online test and diagnostics*; Fault tolerance

Additional Key Words and Phrases: Networks-on-chip, NoC, fault detection/diagnosis, fault localization

ACM Reference Format:
Kypros Chrysanthou, Panayiotis Englezakis, Andreas Prodromou, Andreas Panteli, Chrysostomos Nicopoulos, Yiannakis Sazeides, and Giorgos Dimitrakopoulos. 2016. An online and real-time fault detection and localization mechanism for network-on-chip architectures. ACM Trans. Archit. Code Optim. 13, 2, Article 22 (June 2016), 26 pages.
DOI: http://dx.doi.org/10.1145/2930670

1. INTRODUCTION

Incessant technology scaling has enabled immense transistor integration densities. Nevertheless, the march toward manycore microprocessors has been marred by the emergence of an ominous threat: waning *reliability* [Nassif et al. 2010]. Transistors are more susceptible to both permanent and transient faults. In addition to

This article is a journal extension over previously published work [Prodromou et al. 2012]. This work was supported by Harpa, Project No. 612069 of the European Commission 7th RTD Framework Program – Information and Communication Technologies: Computing Systems.
Authors' addresses: K. Chrysanthou, P. Englezakis, A. Prodromou, A. Panteli, and C. Nicopoulos, Department of Electrical and Computer Engineering, University of Cyprus, Nicosia, Cyprus; emails: {chrysanthou.kypros, englezakis.panayiotis, prodromou.andreas, panteli.andreas, nicopoulos}@ucy.ac.cy; Y. Sazeides, Department of Computer Science, University of Cyprus, Nicosia, Cyprus; email: yanos@cs.ucy.ac.cy; G. Dimitrakopoulos, Department of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece; email: dimitrak@ee.duth.gr.

DOI: http://dx.doi.org/10.1145/2930670

manufacturing and early-life failures, future designs are anticipated to be increasingly vulnerable to aging and wear-out artifacts. Just like any on-chip component, the interconnection backbone is also affected by decreasing reliability [Nicopoulos et al. 2010]. A multitude of approaches to increase the fault tolerance and reliability of the NoC have been proposed. However, the vast majority of these approaches concentrate on fault *prevention* [Constantinides et al. 2006] and/or *recovery* [Fick et al. 2009b; Kakoee et al. 2011b]. The equally important aspect of ***fault detection and localization*** has not been adequately addressed.

Traditionally, fault detection is undertaken by Built-In Self-Test (BIST) mechanisms. The BIST process may be executed by the manufacturer prior to shipment, or it may constitute part of system boot-up [Strano et al. 2011; Hosseinabady et al. 2007]. Runtime BIST is also possible, but system operation is (partially) halted while the module under test is examined [Fick et al. 2009b; Kakoee et al. 2011a].

Near-instantaneous fault detection may be achieved in the datapath of the interconnect through the use of *error -detecting codes*. 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 is to *protect the NoC's control logic* (assuming that the flit *contents* are protected by error-detecting/-correcting codes).

We hereby propose **a comprehensive online fault detection and localization 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 caused by single-bit, single-event faults. More specifically, NoCAlert can detect (a) single stuck-at (permanent) faults, and (b) single-event transient faults. Moreover, the proposed mechanism is also able to pinpoint (localize) the fault with extremely high accuracy. The NoCAlert mechanism is based on the notion of ***invariance checking***, in which the system is continuously checked for illegal outputs as a result of upsets. 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 [Meixner et al. 2007]. NoCAlert comprises several checker micro-modules distributed throughout the NoC router. The checkers never interfere with, or interrupt, the operation of the NoC and provide real-time online fault detection. In essence, NoCAlert implements extremely lightweight ***real-time hardware assertions*** that can detect illegal outputs within the NoC's control logic. Upon detection of a fault, NoCAlert can analyze the pattern of raised assertions to determine the fault location.

Overall, the main contributions of this work are:

(1) The development of a comprehensive *online and real-time* fault detection and localization mechanism for the control logic of the NoC. NoCAlert ensures **0% false negatives** within the interconnect for the fault models (single-bit, single-event transient and permanent) and stimulus set used in this study, with **97% of the transient and 90% of the permanent faults detected instantaneously**.
(2) The NoCAlert framework is extensively evaluated at two abstraction levels: (a) at the **RTL netlist-level**, in which faults are injected in all possible locations (approximately 22,000) of a Verilog-implemented and synthesized NoCAlert router; (b) within a high-level, **cycle-accurate simulation framework**, in which faults are injected in all possible locations within the NoC of a 64-node CMP.
(3) NoCAlert is augmented with a fast and lightweight *fault -localization mechanism*, which can rapidly analyze the patterns of asserted checker flags (upon fault detection) to pinpoint the location of the fault. The proposed technique is fully distributed, with no need for central aggregation/processing.

(4) Three different *fault -localization granularities* are supported: (a) router-level (i.e., the faulty router is identified), (b) pipeline-stage-level (i.e., the faulty pipeline stage within the faulty router is identified), and (c) input/output-port-level (the fault is localized down to a single input/output port). The first two granularity levels yield **100% accurate fault localization**. The accuracy under the third granularity level ranges from 90% to 100%. However, an accuracy of less than 100% simply means that NoCAlert provides more than one possible fault location, with the actual fault location always included in the reported candidate locations.
(5) Hardware synthesis results using commercial 65nm standard-cell libraries indicate minimal area and power overhead of 3% and less than 1%, respectively, for the detection mechanism. More important, the critical path of the router is shown to be negligibly affected (around 1%). For the fault localization mechanism, hardware synthesis results indicate similarly low area and power overhead, while fault localization is achieved within a single clock cycle. Additionally, NoCAlert is shown to outperform ForEVeR [Parikh and Bertacco 2011, 2014], a recently proposed state-of-the-art fault detection and recovery framework.

To the best of our knowledge, this work, together with our previous results in Prodromou et al. [2012], constitutes the first attempt to utilize real-time hardware-based assertion checkers to detect and localize faults within the NoC.

2. RELATED WORK

In general, research in the field of fault tolerance revolves around two fundamental axes: (1) Fault *Detection/Localization*, and (2) Fault *Recovery/Protection/Isolation*. The following sections will concentrate on these two elemental axes (detection and recovery) of fault-tolerant NoC systems.

2.1. Detection/Localization

The concept of exploration/scouting packets [Puente et al. 2008] has been employed to identify faulty nodes ahead of regular data packets. Similar to scouting packets, a common method used for fault detection is the broadcasting of test vectors, either during boot-up [Strano et al. 2011; Hosseinabady et al. 2007] or at runtime. In order to mitigate the performance degradation caused by testing interruptions, token-based mechanisms have been explored [Kakoee et al. 2011a, 2014] as well as hardware for monitoring specific modules, such as the router arbiters [Park et al. 2006] or datapath components [Seitanidis et al. 2014]. A well-established technique to the problem of compromised/corrupted transmission of packets between routers is the use of Error Detecting/Correcting Codes (EDC/ECC). The designs in Fick et al. [2009b] and DeOrio et al. [2012] employ EDC codes to detect faults, then rely on specialized BIST testers for more extensive diagnosis. Forward Error Correction (FEC) could be used to monitor link traversal, while the error syndromes could provide fault localization. In Pellegrini and Bertacco [2014], fault detection is implemented in hardware, whereas the supported reconfiguration is conducted through software. EDC/ECCs have also been used in conjunction with other techniques, such as packet/flit counting [Ghofrani et al. 2012]. The ForEVeR framework [Parikh and Bertacco 2011, 2014] 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 runtime technique, the scheme may also be used to provide robustness against runtime faults.

ForEVeR relies on time *epochs* and counters, which, if needed, trigger a recovery mechanism, which delivers the in-flight data to the intended destination via an additional lightweight checker network that is assumed to be 100% reliable. The use of

timing intervals implies the nontrivial task of finding the optimal epoch duration to minimize false positives. More important, the use of an end-to-end, epoch-based scheme results in significantly delayed fault detection. A comparison between NoCAlert and ForEVeR [Parikh and Bertacco 2011, 2014] is presented in Section 5.

NoCAlert will be empirically demonstrated to provide 100% fault-detection coverage. While 100% fault-detection coverage is also offered by some existing techniques, those primarily operate using *periodic* testing. Periodic testing is typically associated with system/service interruption, and long detection latencies. NoCAlert's *concurrent* approach does not suffer from these weaknesses.

2.2. Recovery/Protection/Isolation

Several fault-tolerant NoC systems focus on the links interconnecting the network's routers. Disabled interrouter links in the network reduce connectivity. One approach to maintaining connectivity is the use of additional links in the form of escape paths [Koibuchi et al. 2008; Iordanou et al. 2014; Kakoee et al. 2011b]. A more radical approach is to replace simple links with Quad-Function Channel buffers [DiTomaso et al. 2014]. The Parikh and Bertacco [2013] design considers the links as pairs of unidirectional links, in which only one of the two would be affected by a single fault.

A less pessimistic and more realistic approach to model the effects of faults within links is to allow for *partially faulty links*. It is this realization that has led researchers to look into ECC as a means to correct faults [Park et al. 2006; Shamshiri et al. 2011; Boraten and Kodi 2013]. Retransmission mechanisms are typically required to cooperate with ECC schemes [Park et al. 2006]. Researchers have devised methodologies to transfer flits using these partially faulty links through shifting and multicycle transmissions. Techniques have been proposed to partition each link into sections [Palesi et al. 2010; Chen et al. 2012] to reconstruct the flit at the receiver [Vitkovskiy et al. 2012] and to use spare wires [Shamshiri et al. 2011].

Several fault-tolerant routing algorithms have been proposed in the literature; we focus here on some recent developments. The algorithms in Rodrigo et al. [2010] and Yu et al. [2012] obviate the need for fault-susceptible routing tables (through logic-based routing) in regular and high-radix topologies. Stochastic routing algorithms [Dumitraş et al. 2003] have been employed to bypass faulty links in the network. Dynamically reconfigurable routing [Fu et al. 2011] determines forbidden turns at runtime to avoid deadlocks, while bypassing faulty components. Routing reconfiguration was also implemented in a distributed manner in Aisopos et al. [2011] by leveraging up*/down* routing. BLINC is also a distributed algorithm that uses routing metadata to compute alternative (emergency) routes locally [Lee et al. 2014]. Deflection-based [Moscibroda and Mutlu 2009; Kohler and Radetzki 2009], distributed [Fick et al. 2009a], and multipath [Murali et al. 2006] routing strategies aim to evenly spread network traffic over a faulty network topology without deadlocks. Adaptive routing has been used to address the emerging issue of hardware wearout [Ancajas et al. 2015; Wang et al. 2014]. Finally, the work in Ren et al. [2014] uses acyclic dependency graphs (calculated offline) to achieve deadlock freedom without the need for virtual channels.

A lot of research has targeted the datapath and control logic of NoC routers. The router in Kim et al. [2006] provides graceful degradation by decomposing the router into two independent modules and by employing resource sharing. Bulletproof [Constantinides et al. 2006] proposes various online repair and recovery techniques. Architectural modifications have also been proposed for each pipeline stage [Poluri and Louri 2013]. The TRNoC mechanism provides a low-cost solution to variation-induced timing errors [Panteloukas et al. 2015]. The issue of hardware wearout has been addressed by powering off stressed components [Zoni and Fornaciari 2013], or by "exercising" ones with low load [Kim et al. 2013, 2015].

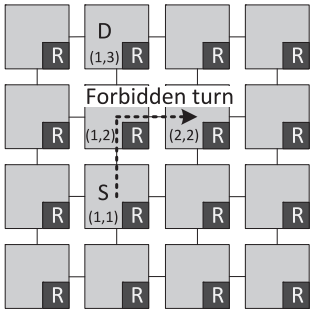

Fig. 1. Example of an NoC invariance. A malfunctioning XY routing computation unit attempts to route a packet in a forbidden direction (S: Source node; D: Destination node).

3. INVARIANCE CHECKING WITHIN THE NETWORK-ON-CHIP

The NoCAlert mechanism is based on the concept of *invariance checking*. The system is continuously examined for illegal outputs as a result of some kind of fault. 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. The assertions are implemented in *hardware* to provide fast detection of anomalies. The pattern of raised assertions is then used to pinpoint the location of the detected fault.

The salient characteristic of invariance checking is the fact that only *functionally illegal* outputs are flagged as violations. A fault that causes the generation of an erroneous, yet functionally legal, output will *not* be identified as a breach of correctness. What we aim to explore in this work, among other issues, is how often and under what conditions such noninvariant faults could potentially lead to compromised network-level correctness.

Note that this work assumes a typical router micro-architecture [Peh and Dally 2001]. The baseline router has five pipeline stages: Routing Computation (RC), Virtual channel Allocation (VA), Switch Arbitration (SA), Crossbar (XBAR) traversal, and Link Traversal (LT).

As an indicative example of an NoC invariance, let us investigate the XY routing algorithm, assuming the 4×4 mesh network in Figure 1. The XY routing algorithm first routes a packet along the X dimension until the intended destination's X-coordinate has been reached, then along the Y dimension until the destination node has been reached. Suppose that 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 may not make a turn to the X dimension under XY routing. Such an invariance violation indicates a malfunctioning RC unit.

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 tasked only with the determination of the output direction 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; for example, FIFO buffers $\rightarrow$ Arbiters $\rightarrow$ Input Port $\rightarrow$ Crossbar Switch $\rightarrow \cdots \rightarrow$ 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*. 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* to enable the decomposition of each module's operation.

The completeness of invariances depends on the completeness of the functional analysis of the design itself. If the invariance checkers cover all functional rules, NoCAlert will detect *any illegal* behavior. In the case of the baseline NoC router microarchitecture, a total of 32 invariance types (categories) were identified and documented in Prodromou et al. [2012]. A list of all identified invariance types is also presented in the Electronic Appendix of this article. Note that the original 32 invariance types have been extended with 3 additional invariance types, which are required to detect *permanent* faults. The purpose of the new invariances is to monitor for *legality of inaction*. For example, when an active VC is eligible for VA/SA arbitration, it should always issue a request to the respective arbiter. An absence of such a request may indicate a stuck-at-0 fault in the request signals. If the request signal referring to a VC gets stuck at 0, that particular VC will be starved. On the other hand, this is at most a performance issue in the case that the fault is transient, since normal operation would resume in the subsequent cycle.

In total, the 35 invariance categories translate to 35 extremely lightweight checker modules. The checker modules' outputs are not registered (i.e., stored); they are fed directly to the localization logic of NoCAlert, as will be explained later.

It should be stressed that *our focus is on the control logic* of the NoC. Our assumption in this article is that the *contents* of the flits/packets (i.e., payload and network overhead bits) are protected by a simple EDC scheme.

4. NOCALERT: AN ONLINE, HARDWARE-ASSERTION-BASED FAULT DETECTION AND LOCALIZATION MECHANISM FOR NETWORKS-ON-CHIP

The proposed NoCAlert framework 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 input the inputs and outputs of the protected component and will check whether any functional rule is broken during the component's operation.

Note that the two assumed fault models in this work are both *single-event* (see Section 5.2.1 for more details). Thus, if a fault occurs within the NoCAlert framework itself—e.g., within a checker, or one of the Localization Units—then the logic of the actual NoC must, by definition, be fault-free. Consequently, the impact of a fault in the NoCAlert logic is, at worst, equivalent to a *false-positive* scenario.

4.1. Ensuring Network Correctness Using Invariances

Prior research [Borrione et al. 2007; Parikh and Bertacco 2011] 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. These four conditions guarantee functional correctness [Borrione et al. 2007; Parikh and Bertacco 2011]. The 32 identified invariances were categorized according to the aforementioned four general requirements, as described in Prodromou et al. [2012]. The 3 new invariances target the second requirement, that is, *bounded delivery*, since they mainly monitor for packets/flits that may get stuck in some router due to a fault.

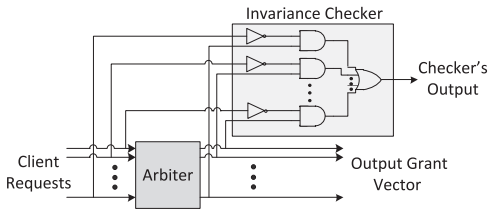

Fig. 2. An example of an NoCAlert checker circuit.

4.2. Hardware Complexity of the NoCAlert Checkers

The NoCAlert fault detection mechanism consists of an array of distributed hardware checkers, which constantly 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 of a checker circuit is shown in Figure 2. This checker is responsible for monitoring an arbiter module and detecting whether there is an active grant signal without any requests at the arbiter's inputs. 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 approach to runtime fault detection.

4.3. Faults That Do Not Cause Invariance Violations

As previously mentioned, invariance checking detects only *illegal* outputs, not necessarily *incorrect* ones. Faults that give rise to *functionally legal* outputs will not be detected. The two elemental questions here are the following:

—If such noninvariant 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 noninvariant 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 Borrione et al. [2007]; Parikh and Bertacco [2011])?

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

4.4. Applicability of the NoCAlert Framework to Any Router Microarchitecture

The invariance concept is closely related to the microarchitecture under test. However, the underlying principles will still be the same: study each individual module and identify invariances, while gradually moving up to coarser granularities. Any alterations to key router components or the addition of new modules within the router will simply generate new invariances based on the modified/new functional specifications. As long as the procedure outlined earlier is followed, a new set of invariances can be produced to describe any router microarchitecture. Examples of new invariances describing more elaborate NoC designs are described in Prodromou et al. [2012].

4.5. Enabling Fault Localization

To minimize the impact on performance and the disruption in the system's operation, two NoCAlert delay elements must be minimized: (a) the delay between the

manifestation of a fault and its detection, and (b) the delay between detection and localization. Most important, ambiguous (imprecise) fault localization leads to overly pessimistic/conservative solutions. For example, the inability to pinpoint a single permanently faulty component within an NoC router may lead to the disconnection of the entire router from the network. Given a quick and accurate fault localization unit, recovery mechanisms can be fine-tuned accordingly to yield a near-optimal fault-tolerant system with respect to performance degradation and service disruption.

To enable fault localization, NoCAlert utilizes the output of its detection checkers. The outputs of all checker modules within a router may be **grouped into a single *vector***, in which each bit corresponds to the output of a single invariance checker. Whenever a checker detects a fault, this bit vector is updated to reflect the disturbance. Essentially, the fault-detection vector can be viewed as a *fault-detection* ***trace***, which, according to the generated vector pattern, could potentially yield the location of the fault. It should be noted that fault localization is useful for both **transient** and **permanent** faults (the two fault models investigated in this work). While the utility of localization for transient faults may seem limited, the ability to identify the component that sustained a transient fault may still be exploited by higher levels in the computing stack, for diagnostic/logging purposes, and to complement any checkpointing/rollback schemes. In addition to isolating *faults*, the NoCAlert detection and localization processes could also be potentially used to detect and isolate design and runtime *bugs*, which may have escaped other verification/validation mechanisms.

4.5.1. The Localization Unit. Detecting the occurrence of a fault in a system is only the first step in providing fault tolerance. The reaction to a fault-detection event typically depends on the fault *type*, that is, whether it is permanent, transient, or intermittent. The ability to *localize* a fault would, in turn, help identify the fault type; for example, a repeating fault at the same location would indicate intermittent or permanent failure. Once the fault type is identified, an appropriate reaction could be triggered (if such functionality is supported). The ***Localization Unit*** introduced here—one per router—tries to tackle the issue of identifying (i.e., localizing) the fault-afflicted NoC router component by analyzing the assertions of the NoCAlert checkers that have detected the fault. The output bits of all NoCAlert checker modules within each router are fed directly into the router's Localization Unit, and are not latched at the end of each cycle. As soon as an assertion is raised by any one of the checker modules within a router, the Localization Unit of said router is triggered into action; that is, it generates a so-called ***Assertion Vector***, which includes the status (asserted/deasserted) of all checkers in the local router at the instant of the first raised assertion (i.e., the instant of first detection). The size of the Assertion Vector for the 5-port baseline router described in Section 5.1 is 300b. This Assertion Vector is then used by the Localization Unit to identify the fault's location within the router. Hence, the Localization Unit in each router is triggered into operation only when a fault is detected, i.e., as soon as at least one invariance assertion is raised. To minimize complexity, the Localization Unit operates only on the first-detection Assertion Vector; any subsequent repeated assertions (in following cycles) by permanent faults are ignored until the first event is localized.

Each Localization Unit operates using only the assertions of a single router. Thus, **the localization process is distributed across routers**, and each one of the localization units is *independent*, that is, does *not* need to communicate with other routers' units to identify the fault location. This valuable attribute is a direct consequence of the instantaneous fault detection capability of NoCAlert, which implies that the detection assertions are spatially close to the fault location. Experimental evaluations later on will validate this NoCAlert property. In general, fault localization is the process of arbitrarily dividing a module, for example, the NoC router, into multiple sectors and then pinpointing the fault location to within a particular sector. The size and granularity of

Table I. Abstract Examples of Pairs Matching Specific Fault Locations to Specific Assertion Vectors

Fault Location	Assertion Vector ID
RC Unit	1
In Port: South	2
VC ID: 0	3
VA1 Arbiter	4
In Port: North	...
VC ID: 2	10
...	...

these sectors will be discussed further in Section 4.5.2. The Localization Unit consists of a combinational circuit, which calculates whether one of the aforementioned sectors could be the faulty one. This circuit is empirically created, based on a large number of fault-injection simulations. For each simulation, a pair is created, which correlates the specific Assertion Vector created at the instant of detection (henceforth termed the *first-detection cycle*) with the location where the fault was injected. Each distinct assertion vector is given a unique Assertion Vector ID for identification purposes; that is, the ID is used to refer to a specific Assertion Vector, which is matched to a specific fault location. Abstract examples of such pairs are shown in Table I. Note that each fault location can be matched (paired) with multiple Assertion Vectors, as shown in Table I, because a fault at a specific location may violate any number of different invariances. For instance, in Table I, a fault in the RC unit of VC0 in the South input port of a router is matched to three different Assertion Vectors, indicated with Assertion Vector IDs 1, 2, and 3. All these pairs are then analyzed to deduce signatures that could identify whether a fault happened at a specific location. The process of building and optimizing these pairs/correlations is heuristic-based and makes use of the *Espresso* logic minimizer tool [Brayton et al. 1984]. Initially, all fault injection simulations are inspected. For every simulation that results in a fault detection (i.e., at least one checker module is asserted), a pair is generated. The pair consists of (1) the location of the fault (as injected in that specific simulation), and (2) the corresponding Assertion Vector created in the first-detection cycle of that simulation. For example, consider the case in which a fault in the SA1 arbiter of the East input port of router X forces its output to a state that is not one-hot. The resulting Assertion Vector (which consists of specific asserted checker flags) is given a unique Assertion Vector ID, and is "linked" to the specific fault location (i.e., SA1 arbiter of the East input port of router X). Next, the Assertion Vector's information is simplified (compacted), as will be described in Section 4.5.3. All simplified pairs are then fed into the *Espresso* logic minimizer tool [Brayton et al. 1984], which calculates the logic equations governing the Localization Unit's output.

Based on these generated pairs, when the Localization Unit receives one of these Assertion Vectors as input, it will output the respective fault location. The size (in bits) of the Localization Unit's output depends on the localization *granularity*, as will be explained in the next section. This output is registered, so that it can be subsequently used for diagnostic and/or recovery purposes by another mechanism (if it exists), or a higher level in the system stack.

Furthermore, since the Localization Unit uses the Assertion Vector created by the NoCAlert checkers at the instant of detection, the localization process is valid for the same fault models as the ones supported by the NoCAlert *detection* phase, that is, both single-bit transient and permanent faults.

4.5.2. Localization Granularity. The complexity of the localization unit is a trade-off between localization *ambiguity* (imprecision) and *granularity*. Ambiguous localization

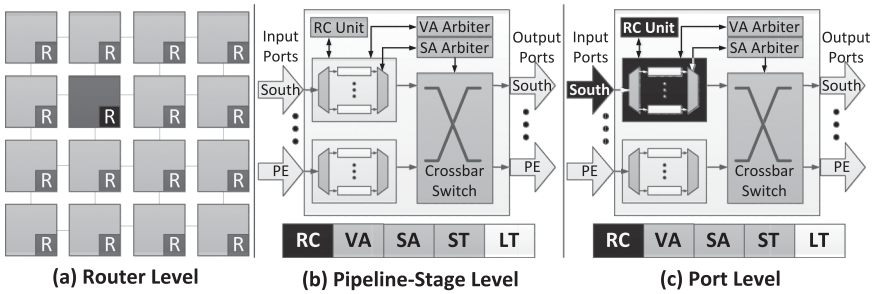

Fig. 3. The NoCAlert framework supports three different localization granularities: (a) identification of the faulty router; (b) identification of the faulty pipeline stage in the affected router; and (c) identification of not only the pipeline stage of the afflicted router, but also the affected input/output port of said router.

occurs when the localization unit yields more than one possible fault location, that is, the fault may have occurred in any one of two (or more) locations. The reason for such ambiguity in the localization unit's output is the nonuniqueness of the Assertion Vector itself. Localization granularity refers to the size of the reported fault location. This is related to the size of the "sectors" described in Section 4.5.1. As can be seen in Figure 3, the NoCAlert framework supports three different localization granularities. At the coarsest granularity (*router level*), the Localization Unit of each router tries to identify if the fault has occurred within the router or in any one of its 4 neighboring routers (this decision is based solely on the local Assertion Vector). The next granularity level (*pipeline-stage level*) is finer; the Localization Unit can pinpoint a location within a single pipeline stage of the affected router. In this work, the router pipeline consists of 4 stages: Routing Computation (RC), Virtual-channel Allocation (VA), Switch Arbitration (SA), and Switch Traversal (ST). The Link Traversal (LT) stage is assumed to be protected by ECC, since it is part of the datapath. Pipeline-stage-level localization granularity could, for instance, facilitate more targeted diagnosis, focusing only on the particular hardware units of the affected pipeline stage. Finally, the finest localization granularity supported is at the router's *port level*. The Localization Unit not only distinguishes the pipeline stage of the afflicted router, but also the affected input/output port of said router. Such fine granularity could be very beneficial if the fault is determined to be permanent (recall that the ability to localize a fault can help identify whether the fault is transient/intermittent or permanent). If the fault is permanent, then a recovery mechanism—assuming that it is supported by the system—could decide on an appropriate action, for example, some form of reconfiguration.

The size (in bits) of the output of each Localization Unit depends on the chosen localization granularity, and varies from 5 (coarsest) to 44 (finest). In the absence of localization ambiguity, the output is a one-hot bit vector, which uniquely identifies the fault location (at the supported granularity level). On the other hand, an ambiguous output is a bit vector that is *not* one-hot, with each "1" indicating a unique (and separate) possible fault location.

It will be demonstrated empirically later that the first two localization granularities always yield unambiguous and correct fault locations. However, the finest granularity level may occasionally suffer from some ambiguity, that is, more than one input/output port might be reported as the possible fault location. This means that the Localization Unit's output is not one-hot, indicating multiple possible fault locations. Nevertheless, the correct location is *always* included in these ambiguous results. Thus, the Localization Unit always identifies the fault location correctly. The ambiguity is due to the fact that fault injections in different locations may yield the same Assertion Vector. Since the Localization Unit cannot distinguish between these locations, it reports all

of them as possible fault sites. As will be demonstrated in Section 6.1.2, the degree of ambiguity is low (on average, 1.3 input/output ports are reported out of a total of 5 input and 5 output ports), and—as mentioned before—the correct fault location is always part of the reported set.

4.5.3. Simplifying the Localization Logic. The Localization Unit relies on the analysis of the Assertion Vector, which is generated from the outputs of the hardware checkers (i.e., the detectors). However, not all bits in the Assertion Vector are always necessary to yield a fault location. This fact allows us to perform several optimizations in order to minimize the information processed by the Localization Unit. This, in turn, simplifies the entire localization logic. Specifically, we employ two optimizations: (a) compacting the Assertion Vector, and (b) discarding portions (i.e., pruning) of the Assertion Vector for each localization granularity level.

When localization happens at the router- or pipeline-stage-level granularities, the specific instance of the asserted hardware checker is not significant. In many cases, the only required information is the specific *invariance type* (category) to which the asserted checker output bit belongs. Thus, the Assertion Vector can be reduced to include only one "summary" bit for some invariance types.

Moreover, since the Localization Unit is created based on a set of empirically generated pairs of Assertion Vectors and corresponding fault locations, portions of the Assertion Vector might, in fact, be redundant. For example, several checkers have the same value for all faults injected in a particular location (i.e., some checkers are either always asserted, or always deasserted). The employed pruning process carefully analyzed the contribution of every single entry in the Assertion Vector to identify the "utility" of each checker with respect to the localization process. Pruning also took care of situations in which two (or more) checkers were always simultaneously asserted. Thus, using only one of those checkers would be enough for localization. Eventually, the pruning process yielded a smaller Assertion Vector that included only the checkers that were responsible for correct and accurate fault localization. Specifically, the original 300b of the Assertion Vector were pruned to 170b (for the *router*-level localization granularity), or 225b (for the *pipeline-stage*-level and *input/output-port*-level localization granularities).

In order to understand the simplification process, let us consider the toy example illustrated in Figure 4. This example assumes (for simplicity) that there are only five invariance types (categories), and each invariance type has three instances, that is, three single-bit checkers. This means that the initial (before simplification) Assertion Vector consists of a total of 15b. Let us now assume that, after all conducted simulations, a specific fault location L within the router has been matched to 4 different Assertion Vectors (differentiated by their distinct Assertion Vector IDs), as indicated in the figure. First, these four Assertion Vectors are inspected to identify *essential* invariance types, if there is an Assertion Vector with asserted bits of only one invariance type. For instance, Invariance Type 3 is essential for Assertion Vector 2 in Figure 4. After establishing all essential invariance types, the Assertion Vectors are inspected for invariances that are redundant for the specific fault location. For example, Invariance Type 5 was never asserted; thus, its associated bits can be removed. Finally, the simplification process investigates whether any of the remaining invariance types (1, 2, and 4 in this example) can also be removed. The invariance types are investigated in an ascending order, based on the total number of asserted bits in all four Assertion Vectors. This is a greedy approach in trying to remove as many invariances as possible. Invariance Type 4 is inspected first, since it has the lowest number of total asserted bits (2). By removing Invariance Type 4, all four vectors are nonzero (detection is still possible); thus, Invariance Type 4 can be removed. Next, Invariance Type 2 is investi-

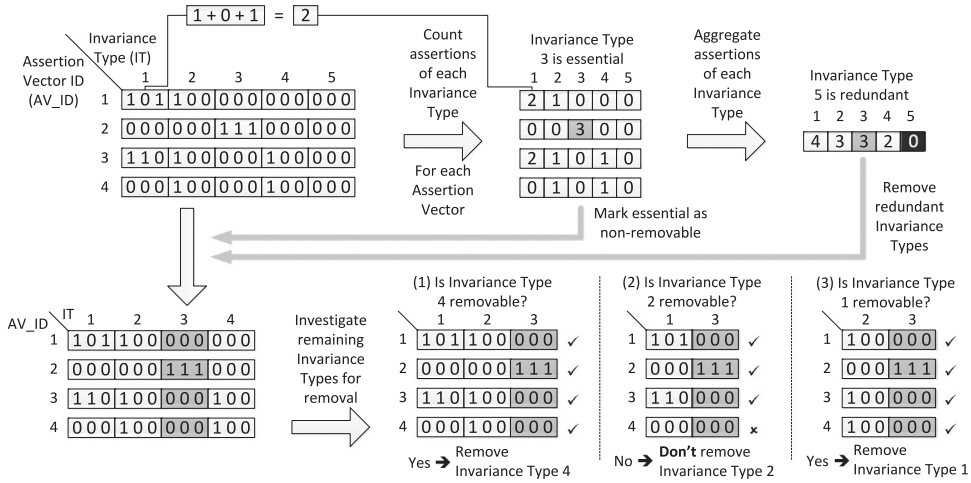

Fig. 4. A toy example to demonstrate the simplification process of the input to the Localization Unit. In this example, the process allows the compaction of the Assertion Vector from (initially) 15b to only 6b.

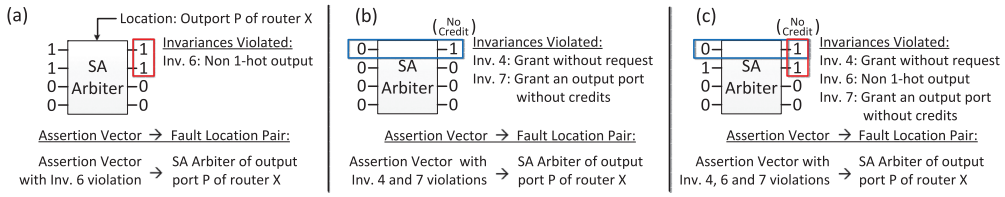

Fig. 5. An example of the fault localization process, which focuses only on a single router component: an SA arbiter. Assuming that a faulty arbiter may yield any one of three possible errors, three corresponding Assertion Vectors are "matched" to this particular router location (i.e., the SA arbiter).

gated. By removing Invariance Type 2 (after having removed Type 4), Assertion Vector 4 is left with all zeros; thus, Invariance Type 2 cannot be removed. Finally, using the same concept, Invariance Type 1 can be safely removed. After completing the simplification process, it is evident that only the checker bits of Invariance Types 2 and 3 must be provided to the Localization Unit. In this example, the simplification process has compacted the Assertion Vector (i.e., the input to the Localization Unit) from 15b to 6b.

4.5.4. An Example of the Fault Localization Process. To demonstrate how the fault localization process works, let us assume the use of the finest localization granularity supported by NoCAlert (i.e., port-level), and let us focus on the SA arbiter of output port P in router X. Recall that the SA arbiter is used for switch allocation in the SA pipeline stage. If a fault occurs in this particular SA arbiter, the Localization Unit should be able to pinpoint the fault, that is, identify the SA arbiter of output port P in router X.

A faulty SA arbiter may yield any one of these possible erroneous results: (a) grant more than one output port to a requesting input port; (b) grant an output port that has no credits, without a request being made for that port; and (c) grant multiple output ports to a single input port, with one/some of the granted output ports having no credits, and/or no corresponding requests (i.e., simultaneously having scenarios (a) and (b)). These three cases are illustrated in Figure 5. In reality, there may be more than these three erroneous possibilities, but let us assume in this example—for the sake of simplicity—that these three are the *only* possibilities. Erroneous scenario (a) will trigger a violation of Invariance Type 6 (see Table A in the Electronic Appendix),

erroneous scenario (b) will violate Invariance Types 4 and 7, while scenario (c) will violate Invariance Types 4, 6, and 7. Thus, the location of this particular arbiter will be "matched" to three distinct Assertion Vectors, with each corresponding to one of the three aforementioned scenarios. This matching process results in three distinct pairs, with each linking a specific fault location (the arbiter) to a specific Assertion Vector. If any of the three Assertion Vectors is sent as input to the Localization Unit, the latter will correctly and accurately identify the fault location.

Of course, the simplification process described in Section 4.5.3 can simplify the required localization logic by compacting the Assertion Vector. For the specific fault location, all invariance types other than 4, 6, and 7 are irrelevant. Additionally, Invariance Type 4 can be ignored without affecting the localization capability for the SA arbiter. Hence, the Localization Unit will identify the correct fault location (the specific SA arbiter in the specific output port), if the input Assertion Vector has asserted bits corresponding to Invariance Types 6 or 7.

Note that, even at the finest localization granularity supported by NoCAlert (i.e., input/output port-level), the number of possible fault locations within the router is by no means excessive.

5. EXPERIMENTAL METHODOLOGY

5.1. Experimental Setup

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, covering two abstraction levels: (a) a commercial, **cycle-accurate simulation framework** implemented in C++ is used to assess NoCAlert at the **network level**; faults are injected in all possible locations (according to the employed fault models) within the NoC of a 64-node CMP arranged in an 8×8 mesh. (b) NoCAlert's functionality is also verified at the **RTL netlist level**, whereby faults are injected in all possible locations (approximately 22,000) of a Verilog-implemented and synthesized NoCAlert router.

For the *network-level* evaluation, the cycle-accurate GARNET NoC simulator [Agarwal et al. 2009] is employed. GARNET models the packet-switched routers down to the microarchitectural level. The simulator was further extended with all the NoCAlert checker and localization modules and a fault injection framework, which will be described in Section 5.2. Since the focus of this work is the *fault detection and localization performance* of NoCAlert (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 that all router components are stressed over a range of traffic intensities.

The NoCAlert framework is also compared to ForEVeR [Parikh and Bertacco 2011, 2014], a 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 Park et al. [2006], and the end-to-end checker. Without loss of generality, the router architecture assumed in this evaluation is the baseline implementation described in Peh and Dally [2001]. The router is 5-stage pipelined (4 intrarouter stages + 1 link traversal stage), with four 5-flit deep VCs per input port, and 128b interrouter 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 and RTL simulation parts, we implemented the baseline NoC router augmented with the NoCAlert detection and localization mechanisms in Verilog Hardware Description Language (HDL). The resulting design was synthesized using Synopsys Design Compiler and 65nm commercial TSMC libraries at 1V operating voltage and 1GHz clock frequency. The results were used to perform detailed area/power/timing analysis and to evaluate the overhead footprint of NoCAlert. Furthermore, the resulting netlist (after synthesis) was used for the aforementioned *RTL netlist-level* evaluation of NoCAlert.

5.2. Evaluation Framework

5.2.1. Fault Models. Throughout the evaluation, we assume the occurrence of *single* faults in the NoC mesh. Specifically, **two single-bit, single-event fault models** were employed: (a) *transient*, and (b) *permanent* stuck-at faults (stuck-at-0 and stuck-at-1). The simulations involve injections of such faults at different locations and at different instances (network states). The implementation of NoCAlert is near identical for both transient and permanent stuck-at faults. However, there are some additional requirements for the detection of permanent faults, as described in Section 3.

Single-bit fault models were chosen over multibit ones, because of two important reasons: (1) modeling multibit faults would exponentially increase the required simulations, since all combinations/permutations of multiple bit flips would have to be conducted; (2) tests for single-stuck-at faults have been shown to cover a very high percentage (greater than 99.6%) of multiple stuck-at faults [Agarwal and Fung 1981]. Therefore, being able to detect single-bit stuck-at faults will also detect (with a very high probability) multiple stuck-at faults.

Nevertheless, it should be stressed that NoCAlert's overarching concept of invariance checking is not tied to a particular fault model. The underlying principles and ideas are the same, irrespective of the fault model: NoCAlert monitors against illegal outputs, irrespective of how the illegal output was generated.

5.2.2. Network-Level Evaluation Methodology. For the *network-level* evaluation, our fault models view the router microarchitecture at the **fine granularity of individual subcomponents**. These subcomponents 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 EDCs; thus, 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 6. By looking at the router microarchitecture 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.

One simulation run at a single traffic injection rate and one network state consists of 35,424 different simulations (to exhaustively inject faults in all 11,808 possible locations of an 8×8 mesh, assuming the specific single-fault injection models used in this work, i.e., transient bit-flips, and permanent stuck-at-0 and stuck-at-1 faults). 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, 32,000, and 64,000). Hence, 21 different scenarios were investigated (7 injection rates $\times 3$ injection times), for a total of $21 \times 35{,}424 \approx 744{,}000$ fault-injection simulations.

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)

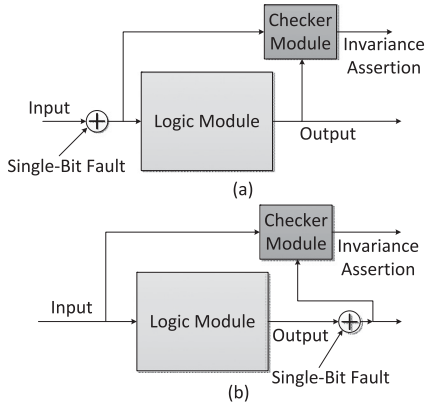

Fig. 6. Abstract view of the employed fault injection framework. Single-bit faults are injected at the inputs (a) or outputs (b) of each individual router module.

report. The GR is then used to ensure that no violations of the four network correctness rules of Section 4.1 and Borrione et al. [2007] and Parikh and Bertacco [2011] occur. Furthermore, the GR also detects any changes in the intrapacket flit order (such order violations constitute erroneous behavior). Since NoCAlert captures only faults that cause invariance violations, 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 [Parikh and Bertacco 2011, 2014].

5.2.3. RTL Netlist-Level Evaluation Methodology. While high-level (e.g., implemented in C++) simulators are useful in evaluating large designs, they are typically not as detailed as lower-level implementations, for example, RTL-based designs. The latter are much closer to the actual hardware implementation; thus, they capture all intricacies and salient features of the design. Such fine-grained modeling detail is very important when assessing fault-tolerant mechanisms; since faults can affect *any bit* in the design, the use of a detailed hardware model is insightful. Recall, for instance, that the high-level simulator allows only for the injection of faults at the inputs and outputs of modules, not inside the intramodule logic. Here, we present the methodology employed to inject faults at the netlist level, that is, at *any* location in the router's logic.

Note that the purpose of the RTL netlist-level analysis is the following: *any* internal fault (i.e., a fault injected within a module, not only at its inputs/outputs) should manifest as one of two cases: (1) masked fault (by internal logic), which would yield no error; or (2) error at a module's output, which would be captured by the checker (i.e., the case covered by the input/output approach of the network-level evaluation). Thus, the netlist-level analysis will primarily shed light on the percentage of the single-bit faults masked by internal logic, that is, faults that do *not* propagate to the module's output, due to intramodule masking. This analysis identifies the percentage of netlist-level injected faults that are actually relevant, that is, not internally masked.

A single fully functional NoC router employing the exact same microarchitecture and specifications as those used in the network-level evaluations (Section 5.1) was implemented in Verilog HDL. The router was also augmented with the NoCAlert detection framework, that is, all hardware checkers and all required logic. After carefully validating the functional correctness of the design with an appropriate test bench, a

Synopsis design compiler was used to obtain the netlist of the entire router. The generated netlist was subsequently analyzed to identify *all fault injection locations* within the router design. Clearly, this approach allows us to inject faults *in every net of the design*; this is precisely the benefit of performing fault injection experiments at such a low-level hardware implementation. Specifically, the number of fault injection locations at the RTL netlist level was about 22,000, as compared to the 205 locations available in our GARNET-based (network-level) simulator. It is important to note that—for this RTL-level analysis—we evaluated NoCAlert in a single NoC router only, since RTL-based simulations of an entire NoC would be prohibitively slow (especially given the large number of separate simulations required for complete assessment). Regardless, this analysis still yields significant insight into the effectiveness of NoCAlert.

To evaluate NoCAlert at the RTL netlist level, we injected faults at the output of *every standard cell* that resides in the router's control path. There is no need to inject faults at the inputs of a standard cell due to the fact that the input is merely another standard cell's output. Clearly, the first level of standard cells is treated differently; in those cases, we also inject faults at the inputs of the modules.

The goal was to verify whether an injected fault actually affected a module's output and/or the entire router's output. In such a fine-grained implementation, it is expected that the effects of some faults would be masked (e.g., logical masking) and not propagate towards a module's output. The same fault was injected (in each and every possible fault location) 30 distinct times. In every iteration, the fault injection cycle was altered to create a range of fault injection instants spanning 30 cycles. Thus, each fault was injected at cycle x in one simulation, at cycle $x + 1$ in the next simulation, and so on, until cycle $x + 29$. This increases the probability of a fault affecting the module during its operation; also, the state of the network is different during each of these cycles, thus simulating a variety of network states. Faults were injected within the cycle ranges of 10 to 39 (idle router), 100 to 129 (moderately warmed-up router), and cycles 200 to 229 (warmed-up router). This resulted in a total of 670,000 distinct simulations.

6. EVALUATION RESULTS

6.1. Network-Level Simulation Results

6.1.1. Fault-Detection Evaluation. As discussed in Section 5.1, simulation experiments were performed in an 8×8 2D mesh network using synthetic traffic patterns. In this section, we present a quantitative analysis of NoCAlert's efficacy and efficiency, as well as a comparison with the ForEVeR [Parikh and Bertacco 2011, 2014] framework.

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 [Borrione et al. 2007; Parikh and Bertacco 2011]). 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).

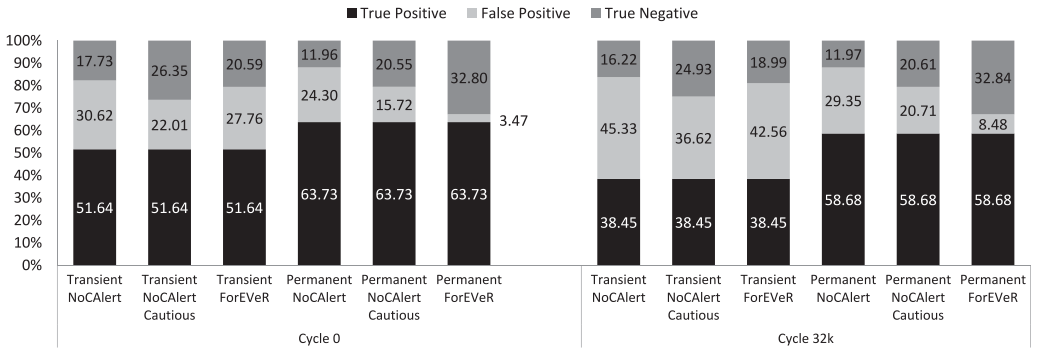

Fig. 7. Fault coverage breakdown (over all injected faults) using synthetic (uniform random) traffic in an 8×8 mesh at two different fault injection instances (cycle 0 and cycle 32K). The "NoCAlert Cautious" bars refer to a system for which low-risk invariances are not immediately flagged as errors. Instead, the error is flagged only if further evidence indicates a problem [Prodromou et al. 2012].

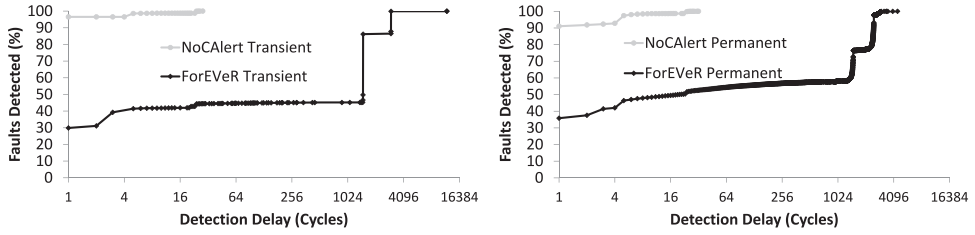

Fig. 8. Cumulative fault-detection delay distribution for True Positive faults under (a) transient and (b) permanent faults. The epoch duration in ForEVeR was set to 1,500 cycles.

Recall that if a fault (either transient or permanent) occurs within the NoCAlert logic itself, the impact would be equivalent to a *false-positive* scenario. The *single-event* fault models assumed in this work imply that a fault can either be in a checker/localization unit or in the remaining NoC logic, but not in two locations simultaneously.

In order to identify which injected faults turned out to be malicious, we used the GR log report described in Section 5.2.2. The simulation results under both examined fault models are summarized in Figures 7 and 8. Specifically, Figure 7 presents a breakdown of the fault detection performance at two different fault injection instances: cycle 0 (empty NoC) and cycle 32K (warmed-up NoC). The results at cycles 32K and 64K are very similar; thus, the 64K results are omitted for brevity. Figure 8 shows the cumulative fault-detection delay distribution for True Positive faults. Overall, the evaluation of the detection capabilities of the two architectures under comparison yields 5 key observations:

(1) Out of all simulations, NoCAlert registered *zero* false negatives under both fault models, that is, *all faults that violated network correctness were successfully captured* by NoCAlert. The same was true for ForEVeR [Parikh and Bertacco 2011, 2014].
(2) Comparing the results between the transient and permanent fault models in Figure 7, the trends are similar, but there is a significant difference: permanent

stuck-at faults give rise to a higher percentage of True Positives. Since permanent faults are persistent, they are much more likely to eventually cause an error.

(3) As can be seen in Figure 7, the true-positive percentages are identical for both NoCAlert and ForEVeR, since they both detected *all* network correctness violations. There is a notable difference in the *False Positives*. This is attributed to the real-time nature of NoCAlert, whereas ForEVeR's epoch-based approach allows for some benign faults to be masked. The False Positive events that did not cause ForEVeR to raise an assertion were faults in the RC and VA logic. In the former case, packets were just misrouted; in the latter case, packets would miss some arbitrations before eventually being assigned to a fault-free VC. This difference in False Positives can be markedly reduced by introducing "NoCAlert Cautious," whereby low-risk invariances are not immediately flagged as errors. Instead, the error is flagged only if further evidence indicates a problem [Prodromou et al. 2012].

(4) NoCAlert provides near instantaneous fault detection, with 97% of all true-positive transient faults and 90% of all true-positive permanent faults captured at the instant of manifestation (same cycle), as depicted in Figure 8. The worst-case detection latency—after fault manifestation—is only 28 cycles for transient faults and 32 cycles for permanent faults. NoCAlert achieves *several orders of magnitude* lower fault-detection latency than ForEVeR [Parikh and Bertacco 2011, 2014].

(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). Moreover, noninvariant upsets that cause a subsequent invariance violation are always successfully captured by NoCAlert. These fundamental results answer the two key questions posed in Section 4.3.

Our experiments indicate that at least one checker from every invariance type detected invariances in the absence of any other invariance type's checker assertions. This fact indicates that *no single invariance type is redundant*. More detailed results pertaining to NoCAlert's *detection* capabilities can be found in Prodromou et al. [2012].

6.1.2. Fault-Localization Evaluation. NoCAlert's Localization Units utilize the Assertion Vector generated only at the instant of detection, that is, at the cycle in which the first hardware checker(s) are asserted. While more checkers may raise assertions later, our experiments indicate that fault localization can still be effectively facilitated through the first generated Assertion Vector. Further, by looking only at the first-detection cycle, localization can exploit the fast detection times and ascertain that the fault has not had the time to cause widespread system malfunction. As illustrated in Figure 9, which combines the results of both transient and permanent fault models, over 99% of the assertions raised during the first-detection cycle are within the affected router, and *all* assertions are at most one hop away. This implies that, on detection, it is known—without any further action—that the faulty router is either the one with the assertion(s) or one of its four neighbors. Since the Localization Unit operates solely on the single Assertion Vector generated during the first-detection cycle, any repeated assertions in subsequent cycles (e.g., by permanent faults) are ignored, until the first event is localized. This simplifies the logic of the Localization Unit.

The following analysis is based on all the simulation results from both examined fault models (permanent and transient). The results were combined because the trends were very similar between the two fault types.

The average number of raised assertions (i.e., bits with value 1) in the Assertion Vector at the first-detection cycle tends to be small, thus expediting the localization process. As illustrated in Figure 10, in more than 65% of the simulations, there were up to three raised assertions in the Assertion Vector; the highest number of raised assertions was 17.

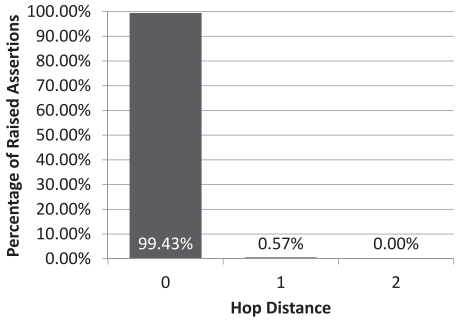

Fig. 9. Distance distribution of raised assertions from the faulty router at the moment of detection.

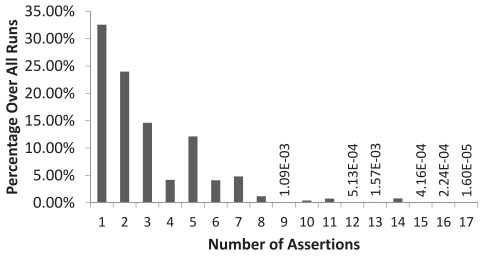

Fig. 10. Average number of raised assertions in the Assertion Vector at the first-detection cycle.

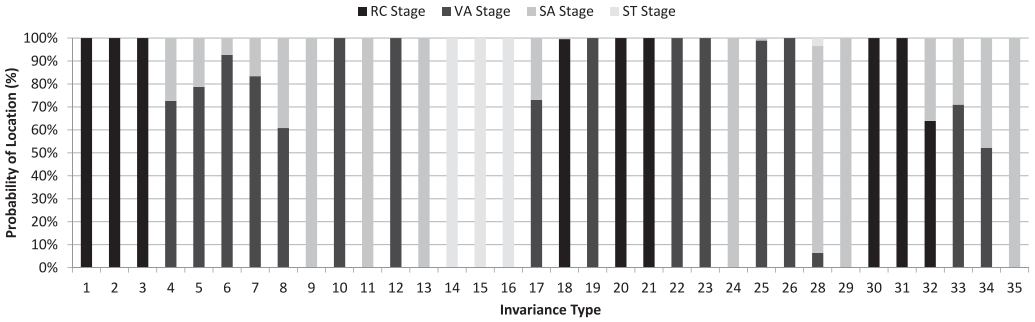

Fig. 11. Localization capabilities of each NoCAlert invariance checker module. The fault locations that each invariance type's checkers can help identify are indicated. There are invariance checker modules capable of localizing faults on their own. Invariance checker module 27 is missing, since it is only applicable to nonatomic VC buffers.

Figure 11 summarizes the localization capabilities of each NoCAlert hardware checker module. The x-axis lists the 35 NoCAlert invariance types; the y-axis indicates the fault locations that each checker can help identify, based on all conducted experiments. For instance, whenever a checker of invariance type 4 was asserted, the fault was located in a VA arbiter in 72.6% of the conducted experiments (dark-gray portion of the bar), while the fault was located in an SA arbiter in the remaining 27.4% of the times (gray portion). It is evident in Figure 11 that some checkers can, by themselves, determine the fault location, with no need to inspect other checker outputs. For example, invariance type 2 checkers—that is, the *Invalid RC Output Direction* invariance type—always pinpoint the fault location to the RC pipeline stage, as shown in Figure 11, since no other fault triggers this particular checker module.

NoCAlert's Assertion Vector allows for a fast and highly accurate localization solution. As discussed in Section 4.5, localization is achieved by identifying "signatures" (patterns) inside the Assertion Vectors that uniquely correspond to a specific fault location. When operating at the coarsest localization granularity (router-level), the output of the Localization Unit is unambiguous (i.e., only one possible faulty router is identified) and 100% accurate. The Localization Unit will either identify the local router as the affected one or a specific one of the 4 neighboring routers. Recall that there is one Localization Unit in each router, and each unit operates only on *local* information (the

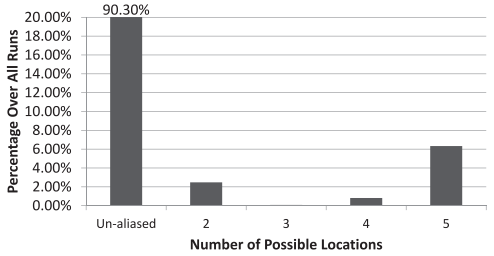

Fig. 12. Probability of ambiguity in the output of the Localization Unit, when faults are localized at the port-level granularity.

Assertion Vector generated within each router). The output is also unambiguous and 100% accurate when operating at the pipeline-stage-level localization granularity.

Ambiguity (aliasing) in the Localization Unit's output appears only at the finest localization granularity, the port level. Figure 12 presents the probability of ambiguity in the output of the Localization Unit, when faults are localized at the port-level granularity. Obviously, the output is ambiguous only 10% of the time, that is, the output lists only one possible faulty input/output port in around 90% of the examined cases. When the output is unambiguous, the reported fault location is 100% accurate. When the output is ambiguous (10%), the Localization Unit reports more than one (up to 5) possible faulty ports in the affected router, as shown in Figure 12. Recall that there are a total of 10 ports, 5 input and 5 output, in the router. However, even in those ambiguous cases, the actual faulty port is *always* included in the list of reported suspects. Thus, if the Localization Unit reports two possible ports, the fault is guaranteed to be in one of those two ports.

As expected, the three supported localization granularity levels have different hardware implementation complexities. Finer-grained granularity incurs additional complexity and cost. The entire Localization Unit is implemented in combinational logic. The hardware cost of the Localization Unit (for all three supported fault localization granularities) is presented in Section 6.3. It will be shown that increasingly finer localization granularity results in higher (hardware) implementation cost. Thus, the localization granularity is a design trade-off between desired localization accuracy and incurred hardware cost. Often, the localization accuracy is closely intertwined with the employed recovery mechanism, that is, the recovery scheme may dictate the required localization accuracy. A coarse localization granularity could serve as a fast first-order solution that can swiftly isolate the faulty component for further in-depth diagnosis.

While the fault localization process is typically expected to merely identify the *location* of a fault, it is interesting to investigate whether a similar process could be utilized to identify the fault *type*, that is, permanent or transient. Note that the process of identifying the fault *type* is orthogonal to the process of identifying the fault *location* (at any granularity level) since both process types operate on the same input data, that is, the Assertion Vector in the first-detection cycle, but they output a different result. Nevertheless, it turns out that the single Assertion Vector generated in the first-detection cycle is not sufficient to accurately identify the fault type. Our experiments using the same simulations as previously described indicate that the fault type could be correctly identified in only 67.33% of all examined cases. Obviously, such accuracy is not sufficiently high for any practical purposes. In order to increase the accuracy and successfully support fault-*type* identification, more than one Assertion Vector would be needed (i.e., more than one time "snapshot," captured at different time instants after the first-detection cycle).

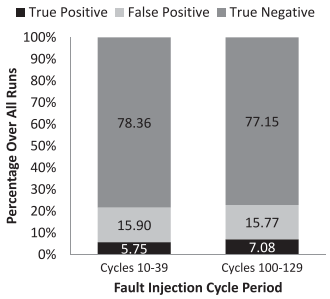

Fig. 13. Breakdown of the RTL netlist-level fault detection performance over two different fault-injection time periods.

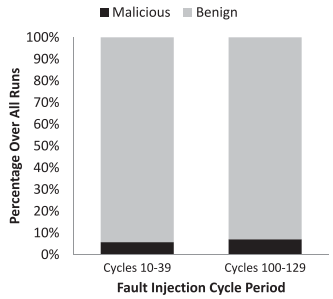

Fig. 14. Breakdown of the effect on network correctness of all injected faults at the RTL netlist level.

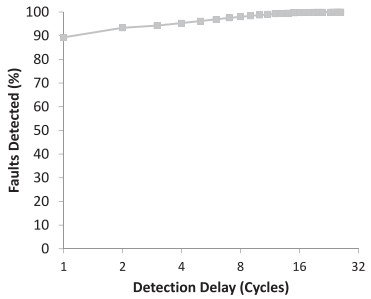

Fig. 15. Cumulative fault-detection delay distribution for True Positive faults, when simulating at the RTL netlist level.

6.2. RTL Netlist-Level Evaluation

As described in Section 5.1, a baseline NoC router augmented with the NoCAlert detection mechanism was implemented in Verilog HDL and synthesized using 65nm commercial standard-cell libraries. The obtained netlist was used to evaluate NoCAlert's *detection* performance at the RTL netlist level. The evaluation process at this level was kept the same as the corresponding high-level (network-level) methodology. Experiments were performed under a fault-free scenario to generate a GR, which was later compared with the output logs of the fault-injected simulation runs. The goal was to verify whether the router was able to route all the flits to the required output ports. The main difference with the high-level evaluation is that we are now considering a single router as the unit under test instead of the entire network.

Similar to Figure 7, Figure 13 presents a breakdown of the RTL netlist-level fault detection performance over two different fault-injection time periods: cycles 10 to 39 (empty NoC), and cycles 100 to 129 (warmed-up NoC). Note that the RTL netlist-level evaluation was conducted using only the transient fault model. Thus, we compare the RTL results of Figure 13 with the corresponding "Transient" bars in Figure 7. The first major observation is that, **even under this extremely detailed evaluation, NoCAlert yields 0% False Negatives**, thereby demonstrating full fault coverage. One obvious difference between the network-level and RTL netlist-level results is the large decrease in the percentage of True Positives in the RTL case. Figure 14 illustrates the reason behind this observation: a vast percentage of the injected faults at the netlist level turn out to be benign due to masking. On the other hand, the high-level evaluation process injects faults only at the inputs/outputs of modules. Such fault injections have a much higher probability of affecting the module's output, while the probability of intramodule masking is significantly reduced. Regarding the decreased number of False Positives, the reason remains the same. Faults are now also injected within a module (not only at its inputs/outputs), which results in a higher probability of masking before the fault propagates to the module's output. If the fault "disappears" (through masking) before reaching the module's output, it will not cause any checker assertions.

Finally, Figure 15 shows the cumulative fault-detection delay distribution for True Positive faults; it is the RTL netlist-level equivalent of Figure 8. The RTL delay trend is very similar to the network-level behavior, while the worst-case fault-detection latency is almost identical to the one reported at the network level.

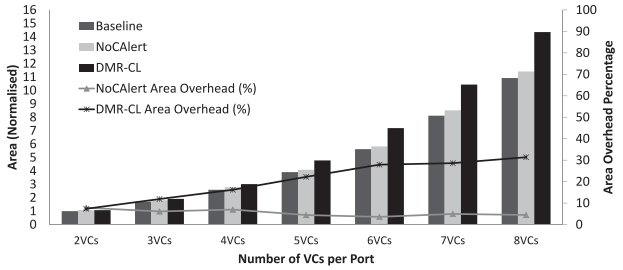

Fig. 16. The NoCAlert fault detection mechanism's area overhead as a function of the number of VCs per input port. A comparison with double modular redundancy in the control logic (DMR-CL) is also presented.

6.3. Hardware Overhead Analysis

The Verilog-implemented and synthesized NoC router of Section 6.2 was also used to evaluate NoCAlert's hardware overhead (in terms of area, power, and timing) for both the fault-*detection* and -*localization* mechanisms.

We start with the analysis of the fault-detection mechanism of NoCAlert. In order to assess the *scalability* of NoCAlert, we vary the number of VCs per port from 2 [Vangal et al. 2008] to 8 [Howard et al. 2011] and evaluate the detection mechanism's percentage area and power overhead. To better appreciate the size of NoCAlert, Figure 16 shows a comparison in area overhead between NoCAlert's detection mechanism, the baseline router described in Section 5.1, and a design applying Double Modular Redundancy (DMR) in the entire NoC control logic (denoted as DMR-CL). Recall that the baseline router is five-stage pipelined (4 intrarouter stages + 1 link traversal stage), with 5-flit-deep atomic VC buffers per input port, and 128b interrouter links. The router uses wormhole switching, credit-based flow control, and deterministic XY routing. In this particular experiment, the number of VCs per port is varied from 2 to 8, and the results are normalized to the 2-VC baseline design. In general, DMR serves as the most complete (albeit expensive) fault-detection solution. NoCAlert's detection mechanism incurs a minimal area overhead, ranging from 1.38% to 4.42%, while the *percentage* overhead remains fairly constant. In contrast, the *percentage* area overhead of DMR increases linearly from 5.41% to 31.32%, when using 2 and 8 VCs, respectively. The detection component of NoCAlert was further evaluated with respect to the power and timing (i.e., critical path delay) overheads. With the switching activity set to 50% for all nets, the power overhead ranges from 0.3% to 1.2%. Finally, the critical path delay increases by at most 3% (around 1%, on average) over all examined VC configurations.

Since the finest supported fault-localization granularity is at the input/output *port* level (not at the VC level), the hardware of NoCAlert's Localization Unit is minimally affected by the number of VCs per port. Hence, for the analysis of the Localization Unit, we focus on a router design with 4 VCs per input port, that is, same as in the baseline router of Section 5.1. Table II reports the area/power overhead and timing attributes of NoCAlert's Localization Unit, *with* and *without* the compaction/pruning of the Assertion Vector. The area/power overhead reported in the table is relative to the baseline router of Section 5.1.

Note that, in the case of the nonsimplified localizer (i.e., the one using the full, non-compacted Assertion Vector), the trend in area/power/timing is, in fact, the *opposite* of what one would expect: the *router*-level granularity has the biggest overhead and is the slowest of the three supported fault-localization granularity levels. This is due to the following reason: since the Assertion Vector is not pruned in the nonsimplified localizer variants, the presence of redundancy within the Assertion Vector is not exploited for simplification. Hence, the circuits for all three localization granularities are

Table II. Hardware Synthesis Evaluation of NoCAlert's Localization Unit

Fault Localization Granularity	Nonsimplified (No AV compaction)			Simplified (After AV compaction)		
	% Overhead at 1GHz		Max. freq.	% Overhead at 1GHz		Max. freq.
	Area	Power	(GHz)	Area	Power	(GHz)
Router	6.7	1	1.16	1.4	0.48	1.66
Pipeline stage	6.5	0.95	1.20	2.1	0.52	1.64
Input/output port	6.2	0.92	1.37	4.4	0.66	1.42

Note: Results are presented for all three supported fault-localization granularity levels, and for both "nonsimplified" and "simplified" variants. The "simplified" variants are obtained after applying the Assertion Vector (AV) compaction process described in Section 4.5.3.

very similar; essentially, all three circuits include the logic of the finest granularity—*input/output-port*-level—plus additional logic to aggregate/summarize the larger output into a smaller output (to yield the *router*-level and *pipeline-stage*-level fault localization granularities). On the contrary, when the simplification/compaction process of Section 4.5.3 is followed, redundancies are appropriately removed *beforehand* for each granularity level, and the resulting hardware overhead trends are as expected, i.e., as the localization granularity becomes finer, the hardware cost increases.

As indicated in Table II, the biggest benefit of compaction is in the consumed area: the overhead when using compaction is significantly lower (especially in the cases of *router*-level and *pipeline-stage*-level fault localization granularities). There is a benefit in the consumed power as well, but the power overhead of the nonsimplified localizer is still relatively low. In terms of timing, both the nonsimplified and simplified Localization Units can operate within a single cycle at 1GHz (the setup used in our experiments), but the simplified/compacted variants have a smaller critical path delay, which means that they could also achieve single-cycle operation at higher operating frequencies, as indicated in the table.

The results of this section corroborate the fact that the checkers used to detect only illegal outputs have significantly lower hardware cost than the units that they check. Moreover, fault localization incurs minimal cost and only single-cycle latency.

7. CONCLUSIONS

This article proposes NoCAlert, a comprehensive **online and real-time fault *detection* and *localization*** mechanism that ensures 0% false negatives within the NoC, under both transient and permanent faults. 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. A collection of such microchecker modules is used to implement real-time hardware assertions. The checkers operate concurrently with normal NoC operation, obviating the need for periodic, or triggered-based, self-testing. Upon fault detection, the status of the checkers within each router is analyzed by a Localization Unit, which can accurately pinpoint the fault location at three different supported granularity levels.

Extensive simulation results, both at the **network level** and the **RTL netlist level**, validate the efficacy of the NoCAlert mechanism and yield important insight as to the behavior of the network when noninvariant faults (that evade the checkers) occur. Specifically, noninvariant faults either cause some subsequent invariance violation (and are captured) or they prove benign at the network/system level. Hardware synthesis analysis using 65nm commercial libraries demonstrates the lightweight nature of NoCAlert in terms of area/power/timing overhead. Furthermore, a detailed comparison with a state-of-the-art framework [Parikh and Bertacco 2011] highlights more 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 near-instantaneous fault detection and swift, single-cycle fault localization within the NoC. This feat is achieved using minimally intrusive hardware-based invariance checkers, and fully distributed fault localization units.

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library. The appendix contains a list of all 35 identified invariance types.

REFERENCES

N. Agarwal, T. Krishna, Li-Shiuan Peh, and N. K. Jha. 2009. GARNET: A detailed on-chip network model inside a full-system simulator. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'09)*. 33–42.

V. K. Agarwal and A. S. F. Fung. 1981. Multiple fault testing of large circuits by single fault test sets. *IEEE Transactions on Circuits and Systems* 28, 11, 1059–1069.

Konstantinos Aisopos, Andrew DeOrio, Li-Shiuan Peh, and Valeria Bertacco. 2011. ARIADNE: Agnostic reconfiguration in a disconnected network environment. In *Proceedings of the 2011 International Conference on Parallel Architectures and Compilation Techniques (PACT'11)*. IEEE Computer Society, Washington, DC, 298–309.

D. M. Ancajas, K. Bhardwaj, K. Chakraborty, and S. Roy. 2015. Wearout resilience in NoCs through an aging aware adaptive routing algorithm. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 23, 2, 369–373.

T. Boraten and A. Kodi. 2013. Energy-efficient runtime adaptive scrubbing in fault-tolerant network-on-chips (NoCs) architectures. In *IEEE 31st International Conference on Computer Design (ICCD'13)*. 264–271.

D. Borrione, A. Helmy, L. Pierre, and J. Schmaltz. 2007. A generic model for formally verifying NoC communication architectures: A case study. In *1st International Symposium on Networks-on-Chip (NOCS'07)*. 127–136.

Robert King Brayton, Alberto L. Sangiovanni-Vincentelli, Curtis T. McMullen, and Gary D. Hachtel. 1984. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, Norwell, MA.

Changlin Chen, Ye Lu, and S. D. Cotofana. 2012. A novel flit serialization strategy to utilize partially faulty links in networks-on-chip. In *6th IEEE/ACM International Symposium on Networks on Chip (NoCS'12)*. 124–131.

K. Constantinides, S. Plaza, J. Blome, B. Zhang, V. Bertacco, S. Mahlke, T. Austin, and M. Orshansky. 2006. BulletProof: A defect-tolerant CMP switch architecture. In *12th International Symposium on High-Performance Computer Architecture*. 5–16.

A. DeOrio, D. Fick, V. Bertacco, D. Sylvester, D. Blaauw, Jin Hu, and G. Chen. 2012. A reliable routing architecture and algorithm for NoCs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31, 5, 726–739.

D. DiTomaso, A. Kodi, and A. Louri. 2014. QORE: A fault tolerant network-on-chip architecture with power-efficient quad-function channel (QFC) buffers. In *IEEE 20th International Symposium on High Performance Computer Architecture (HPCA'14)*. 320–331.

Tudor Dumitraş, Sam Kerner, and Radu Mărculescu. 2003. Towards on-chip fault-tolerant communication. In *Proceedings of the 2003 Asia and South Pacific Design Automation Conference (ASP-DAC'03)*. ACM, New York, NY.

David Fick, Andrew DeOrio, Gregory Chen, Valeria Bertacco, Dennis Sylvester, and David Blaauw. 2009a. A highly resilient routing algorithm for fault-tolerant NoCs. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'09)*. European Design and Automation Association, 3001 Leuven, Belgium, 21–26.

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

Binzhang Fu, Yinhe Han, Jun Ma, Huawei Li, and Xiaowei Li. 2011. An abacus turn model for time/space-efficient reconfigurable routing. In *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA'11)*. ACM, New York, NY, 259–270.

A. Ghofrani, R. Parikh, S. Shamshiri, A. DeOrio, Kwang-Ting Cheng, and V. Bertacco. 2012. Comprehensive online defect diagnosis in on-chip networks. In *IEEE 30th VLSI Test Symposium (VTS'12)*. 44–49.

M. Hosseinabady, A. Dalirsani, and Z. Navabi. 2007. Using the inter- and intra-switch regularity in NoC switch testing. In *Design, Automation Test in Europe Conference Exhibition (DATE'07)*. 1–6.

J. Howard, S. Dighe, S. R. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V. K. De, and R. Van Der Wijngaart. 2011. A 48-core IA-32 processor in 45 nm CMOS using on-die message-passing and DVFS for performance and power scaling. *IEEE Journal of Solid-State Circuits* 46, 1, 173–183.

C. Iordanou, V. Soteriou, and K. Aisopos. 2014. Hermes: Architecting a top-performing fault-tolerant routing algorithm for networks-on-chips. In *32nd IEEE International Conference on Computer Design (ICCD'14)*. 424–431.

M. R. Kakoee, V. Bertacco, and L. Benini. 2011a. A distributed and topology-agnostic approach for on-line NoC testing. In *5th IEEE/ACM International Symposium on Networks on Chip (NoCS'11)*. 113–120.

M. R. Kakoee, V. Bertacco, and L. Benini. 2011b. ReliNoC: A reliable network for priority-based on-chip communication. In *Design, Automation Test in Europe Conference Exhibition (DATE'11)*. 1–6.

M. R. Kakoee, V. Bertacco, and L. Benini. 2014. At-speed distributed functional testing to detect logic and delay faults in NoCs. *IEEE Transactions on Computers* 63, 3, 703–717.

Hyungjun Kim, Siva Bhanu Krishna Boga, Arseniy Vitkovskiy, Stavros Hadjitheophanous, Paul V. Gratz, Vassos Soteriou, and Maria K. Michael. 2015. Use it or lose it: Proactive, deterministic longevity in future chip multiprocessors. *ACM Transactions on Design Automation of Electronic Systems* 20, 4, Article 65, 26 pages.

Hyungjun Kim, Arseniy Vitkovskiy, Paul V. Gratz, and Vassos Soteriou. 2013. Use it or lose it: Wear-out and lifetime in future chip multiprocessors. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'13)*. ACM, New York, NY, 136–147.

Jongman Kim, Chrysostomos Nicopoulos, and Dongkook Park. 2006. A gracefully degrading and energy-efficient modular router architecture for on-chip networks. *SIGARCH Computer Architecture News* 34, 2, 4–15.

Adan Kohler and Martin Radetzki. 2009. Fault-tolerant architecture and deflection routing for degradable NoC switches. In *Proceedings of the 3rd ACM/IEEE International Symposium on Networks-on-Chip (NOCS'09)*. IEEE Computer Society, Washington, DC, 22–31.

Michihiro Koibuchi, Hiroki Matsutani, Hideharu Amano, and Timothy Mark Pinkston. 2008. A lightweight fault-tolerant mechanism for network-on-chip. *International Symposium on Networks-on-Chip,* 0, 13–22.

Doowon Lee, R. Parikh, and V. Bertacco. 2014. Brisk and limited-impact NoC routing reconfiguration. In *Design, Automation and Test in Europe Conference and Exhibition (DATE'14)*. 1–6.

Albert Meixner, Michael E. Bauer, and Daniel Sorin. 2007. Argus: Low-cost, comprehensive error detection in simple cores. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'07)*. IEEE Computer Society, Washington, DC, 210–222.

Thomas Moscibroda and Onur Mutlu. 2009. A case for bufferless routing in on-chip networks. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA'09)*. ACM, New York, NY, 196–207.

Srinivasan Murali, David Atienza, Luca Benini, and Giovanni De Michel. 2006. A multi-path routing strategy with guaranteed in-order packet delivery and fault-tolerance for networks on chip. In *Proceedings of the 43rd Annual Design Automation Conference (DAC'06)*. ACM, New York, NY, 845–848.

S. R. Nassif, N. Mehta, and Yu Cao. 2010. A resilience roadmap. In *Design, Automation Test in Europe Conference Exhibition (DATE'10)*. 1011–1016.

C. Nicopoulos, S. Srinivasan, A. Yanamandra, Dongkook Park, V. Narayanan, C. R. Das, and M. J. Irwin. 2010. On the effects of process variation in network-on-chip architectures. *IEEE Transactions on Dependable and Secure Computing* 7, 3, 240–254.

Maurizio Palesi, Shashi Kumar, and Vincenzo Catania. 2010. Leveraging partially faulty links usage for enhancing yield and performance in networks-on-chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 29, 3, 426–440.

A. Panteloukas, A. Psarras, C. Nicopoulos, and G. Dimitrakopoulos. 2015. Timing-resilient network-on-chip architectures. In *IEEE 21st International On-Line Testing Symposium (IOLTS'15)*. 77–82.

Ritesh Parikh and Valeria Bertacco. 2011. Formally enhanced runtime verification to ensure NoC functional correctness. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'11)*. ACM, New York, NY, 410–419.

Ritesh Parikh and Valeria Bertacco. 2013. uDIREC: Unified diagnosis and reconfiguration for frugal bypass of NoC faults. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'11)*. ACM, New York, NY, 148–159.

Ritesh Parikh and Valeria Bertacco. 2014. ForEVeR: A complementary formal and runtime verification approach to correct NoC functionality. *ACM Transactions on Embedded Computing Systems* 13, 3s, Article 104, 30 pages.

D. Park, C. Nicopoulos, J. Kim, N. Vijaykrishnan, and C. R. Das. 2006. Exploring fault-tolerant network-on-chip architectures. In *International Conference on Dependable Systems and Networks (DSN'06)*. 93–104.

L.-S. Peh and W. J. Dally. 2001. A delay model and speculative architecture for pipelined routers. In *7th International Symposium on High-Performance Computer Architecture (HPCA'01)*. 255–266.

A. Pellegrini and V. Bertacco. 2014. Cardio: CMP adaptation for reliability through dynamic introspective operation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33, 2, 265–278.

P. Poluri and A. Louri. 2013. Tackling permanent faults in the network-on-chip router pipeline. In *25th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'13)*. 49–56.

A. Prodromou, A. Panteli, C. Nicopoulos, and Y. Sazeides. 2012. NoCAlert: An on-line and real-time fault detection mechanism for network-on-chip architectures. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'12)*. 60–71.

V. Puente, J. A. Gregorio, F. Vallejo, and R. Beivide. 2008. Immunet: Dependable routing for interconnection networks with arbitrary topology. *IEEE Transactions on Computers* 57, 12, 1676–1689.

Pengju Ren, Qingxin Meng, Xiaowei Ren, and Nanning Zheng. 2014. Fault-tolerant routing for on-chip network without using virtual channels. In *51st ACM/EDAC/IEEE Design Automation Conference (DAC'14)*. 1–6.

S. Rodrigo, J. Flich, A. Roca, S. Medardoni, D. Bertozzi, J. Camacho, F. Silla, and J. Duato. 2010. Addressing manufacturing challenges with cost-efficient fault tolerant routing. In *Proceedings of the 2010 4th ACM/IEEE International Symposium on Networks-on-Chip (NOCS'10)*. IEEE Computer Society, Washington, DC, 25–32.

I. Seitanidis, A. Psarras, E. Kalligeros, C. Nicopoulos, and G. Dimitrakopoulos. 2014. ElastiNoC: A self-testable distributed VC-based network-on-chip architecture. In *8th IEEE/ACM International Symposium on Networks-on-Chip (NoCS)*. 135–142.

S. Shamshiri, A. Ghofrani, and Kwang-Ting Cheng. 2011. End-to-end error correction and online diagnosis for on-chip networks. In *2011 IEEE International Test Conference (ITC)*. 1–10.

A. Strano, C. Goandmez, D. Ludovici, M. Favalli, M. E. Goandmez, and D. Bertozzi. 2011. 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'11)*. 1–6.

S. R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar. 2008. An 80-tile sub-100-W TeraFLOPS processor in 65-nm CMOS. *IEEE Journal of Solid-State Circuits* 43, 1, 29–41.

A. Vitkovskiy, V. Soteriou, and C. Nicopoulos. 2012. A dynamically adjusting gracefully degrading link-level fault-tolerant mechanism for NoCs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31, 8, 1235–1248.

Liang Wang, Xiaohang Wang, and T. Mak. 2014. Dynamic programming-based lifetime aware adaptive routing algorithm for network-on-chip. In *22nd International Conference on Very Large Scale Integration (VLSI-SoC'14)*. 1–6.

Qiaoyan Yu, J. Cano, J. Flich, and P. Ampadu. 2012. Transient and permanent error control for high-end multiprocessor systems-on-chip. In *6th IEEE/ACM International Symposium on Networks on Chip (NoCS)*. 169–176.

Davide Zoni and William Fornaciari. 2013. Sensor-wise methodology to face NBTI stress of NoC buffers. In *Design, Automation Test in Europe Conference Exhibition (DATE'13)*. 1038–1043.

Received December 2015; revised March 2016; accepted April 2016