TALLINN UNIVERSITY OF TECHNOLOGY
DOCTORAL THESIS
43/2019

Cost-Effective Concurrent Hardware Checkers for Network on Chip based System on Chip

RANGANATHAN HARIHARAN

TAL TECH PRESS

TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Department of Computer Systems
The dissertation was accepted for the defence of the degree of Doctor of Philosophy in Computer and Systems Engineering on 27 June 2019

Supervisor: Prof. Dr. Jaan Raik
Department of Computer Systems, School of Information Technologies
Tallinn University of Technology
Tallinn, Estonia

Co-supervisor: Dr. Tara Ghasempouri
Department of Computer Systems, School of Information Technologies
Tallinn University of Technology
Tallinn, Estonia

Opponents: Professor Dipl.-Ing. Dr.techn. Andreas Steininger
Vienna University of Technology
Vienna, Austria

Dr. Johnny Öberg
KTH Royal Institute of Technology
Stockholm, Sweden

Defence of the thesis: 21 August 2019, Tallinn

Declaration:
Hereby I declare that this doctoral thesis, my original investigation and achievement, submitted for the doctoral degree at Tallinn University of Technology, has not been submitted for any academic degree elsewhere.

Ranganathan Hariharan _____
signature

ISSN 2585-6898 (publication)
ISBN 978-9949-83-467-9 (publication)
ISSN 2585-6901 (PDF)
ISBN 978-9949-83-468-6 (PDF)

TALLINNA TEHNIKAÜLIKOOL
DOKTORITÖÖ
43/2019

Kulutõhusad süsteemiga paralleelsed rikkemonitorid kiipvõrkudel põhinevatele kiipsüsteemidele

RANGANATHAN HARIHARAN

TAL TECH
KIRJASTUS

To my Family …

Table of contents

LIST OF PUBLICATIONS

The work of this thesis is based on the following publications:

Publication I: Behrad Niazmand, Ranganathan Hariharan, Vineeth Govind, Gert Jervan, Thomas Hollstein, and Jaan Raik. Extended checkers for logic-based distributed routing in network-on-chips. In *2014 14th Biennial Baltic Electronic Conference (BEC)*, pages 77–80. IEEE, 2014

Publication II: Pietro Saltarelli, Behrad Niazmand, Jaan Raik, Ranganathan Hariharan, Gert Jervan, and Thomas Hollstein. A framework for comprehensive automated evaluation of concurrent online checkers. In *2015 Euromicro Conference on Digital System Design*, pages 288–292. IEEE, 2015

Publication III: Pietro Saltarelli, Behrad Niazmand, Ranganathan Hariharan, Jaan Raik, Gert Jervan, and Thomas Hollstein. Automated minimization of concurrent online checkers for network-on-chips. In *2015 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, pages 1–8. IEEE, 2015

Publication IV: Pietro Saltarelli, Behrad Niazmand, Jaan Raik, Vineeth Govind, Thomas Hollstein, Gert Jervan, and Ranganathan Hariharan. A framework for combining concurrent checking and on-line embedded test for low-latency fault detection in noc routers. In *Proceedings of the 9th International Symposium on Networks-on-Chip*, page 6. ACM, 2015

Publication V: Ranganathan Hariharan, Behrad Niazmand, and Jaan Raik. On fault detection efficiency of reliability checkers obtained by verification assertion qualification. In *RESCUE 2017 Workshop on Reliability, Security and Quality European Test Symposium (ETS) Fringe Workshop, May 25-26*. IEEE, 2017

Publication VI: Ranganathan Hariharan, Tara Ghasempouri, Behrad Niazmand, and Jaan Raik. From rtl liveness assertions to cost-effective hardware checkers. In *2018 Conference on Design of Circuits and Integrated Systems (DCIS)*, pages 1–6. IEEE, 2018

OTHER RELATED PUBLICATIONS

Publication VII: Pietro Saltarelli, Behrad Niazmand, Ranganathan Hariharan, Jaan Raik, Gert Jervan, and Thomas Hollstein. A framework for area-efficient concurrent online checkers design. In *MEDIAN 2015 Workshop on Manufacturable and Dependable Multicore Architectures at Nanoscale*, *November 10-11*, 2015

Author's contributions to the publications

I In Publication I, the author was involved in developing the initial idea of evaluating and minimizing concurrent checkers and in implementing the additional checkers to the existing set of checkers. The author also implemented the design used for experimental case study. The author prepared the part of the paper for publication.

II Publication II is an extension of Publication I where the author contributed to the core idea of devising checkers. The author was involved in implementing the checkers module. The author implemented the design used for experimental study. The author prepared the part of the paper for publication.

III Publication III is an extension of Publication I where the author contributed to the core idea of devising checkers. The author implemented the design used for experimental study. The author prepared the part of the paper for publication.

IV Publication IV is an extension of Publication I where the author contributed to the core idea of devising checkers. The author implemented the design used for experimental study. The author prepared the part of the paper for publication.

V The goal of Publication V was to analyze the correlation between register-transfer level assertions and gate-level checkers synthesized from the former. The author developed the checker module and conducted checkers qualification procedure. The author also implemented the design used for experimental study. The author prepared the part of the paper for publication.

VI In Publication VI, the author has proposed the framework flow and conducted the assertion fault analysis and minimization procedure. The author was also involved in translating the liveness assertions to safety assertions. The author implemented the designs used for experimental case study. The author has prepared the part of the paper for publication and presented it at the conference.

VII In , the author was involved in implementing the checker module. The author implemented the design used for experimental study.

Abbreviations

AC	Allocation Comparator
BICST	Built-In Concurrent Self-Test
BIST	Built-In Self-Test
CEI	Checkers Efficiency Index
CMP	Chip Multi Processors
DMR	Double Modular Redundancy
DUV	Design Under verification
DwC	Duplication with Comparison
ECC	Error Correcting Codes
EDC	Error Detecting Codes
FC	Fault coverage
FIFO	First-In-First-Out
FPR	False Positive Ratio
FSM	Finite State Machine
HBH	Hop By Hop
hr	high-radix
IC	Integrated Circuit
IIR	Inherent Information Redundancy
LBDR	Logic-Based Distributed Routing
NI	Network Interface
NoC	Network on Chip
PE	Processing Element
ROWR	Reduced Observation Width Replication
RR	Round-Robin
RTL	Register Transfer Level
SA	Switch Allocation
SAF	Stuck-At Fault
SEE	Single Event Effect
SET	Single Event Transient
SEU	Single Event Upset
SoC	System on Chip
SSA	Single Stuck-At
SSBDD	Structurally Synthesized Binary Decision Diagram
TMR	Triple Modular Redundancy
TT	Turbo Tester
VC	Virtual Channel

1 INTRODUCTION

This thesis addresses a set of timely issues in reliability by proposing a methodology for generating cost-effective concurrent hardware checkers. The main emphasis is to reuse the verification assertions to generate hardware checkers for online, real-time fault detection within fault tolerant systems.

This introductory chapter presents the motivation leading to this research, followed by a more detailed problem formulation. Finally, a summary of the main contributions and an overview of the thesis structure are provided.

1.1 Motivation

As the technology scale is shifting steadily from micro- to the nano-scale for today's design and manufacturing, the advancements in reliability have not kept up the pace [8]. Reliability is the probability that the system functions without failure in the specified environmental conditions in the defined time interval. Reliable functioning of electronic systems is of paramount concern as the millions of users depend on these systems every day. Unfortunately, most of the systems still fall short of users' expectation of reliability. The lifetime failure rate of the system can be illustrated by the use of the bathtub curve [9]. Figure 1 shows the bathtub curve changes for different technology nodes over time. The failure rate is characterized by the phases of infant mortality (beginning of the bathtub curve), random constant failures (the smooth and straight horizontal middle section) and system wear-out (the rising end of the curve), which produce a bathtub looking curve over the lifetime of the system. As it can be seen from the figure, the probability of lifetime errors has been increasing rapidly with the transition towards nano-scale Integrated Circuits (ICs) and thus, there is a need to handle these lifetime issues.

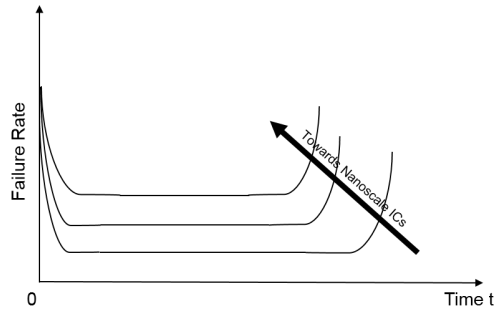


Figure 1 – Failure rate over a life-time of a hardware system with shrinking technology

Figure 1 shows that due to extreme down-scaling trend in semi-conductor technologies, the digital circuits are becoming more susceptible to both permanent and transient faults. The adverse effect of process variations, aging and wear-out due to the nano-scale regime causes the reliability to dwindle. Even though most of the faults are identified during the manufacturing test, detecting faults at run-time is becoming more and more imperative. An online fault detection mechanism aims to monitor the digital circuits at run-time and detects the undesired behavior while the device is in operation. The online fault detection can be achieved with the help of hardware checker infrastructure. However, designing checkers by hand can be tedious and error-prone task. Checkers can also be automatically generated, but care needs to be taken so that the checkers do not have unacceptable negative impact on performance, power or area overhead. Therefore, there is a need for qualification and minimization of checkers. Moreover, due to scalability

issues at gate-level, there is a need to move to higher abstraction levels. The higher the abstraction level, the lower the simulation time, due to the smaller size and complexity of the design. Verification assertions written in a high-level language can be reused in gate-level verification by converting them to checkers to reduce the simulation time.

Regarding architectures applied in nano-scale ICs, System on Chip (SoC) is a paradigm for designing integrated circuit that integrates several cores on a single chip to accomplish the system task. With the number of cores getting increased, the on-chip communication efficiency has become one of the factors in determining the overall system performance and cost. A packet based, on-chip intercommunication network known as Network on Chip (NoC) [10] is emerging as an alternative solution to address the increasing interconnect complexity. However, NoC based interconnects, because of advanced router architectures, complex operation and concurrent communication are highly susceptible to faults during the runtime of the system. Without taking an appropriate run-time solution to ensure that such faults do not affect the operation of NoC based interconnects, there could be possibility of data getting misrouted, dropped, corrupted, deadlocked or several types of on-chip communication performance degradation.

This thesis is addressing the challenges mentioned above.

1.2 Problem formulation

The general objective of this thesis is developing a methodology for generating a set of cost-effective concurrent checkers from verification assertions. Currently, designing such checker infrastructure is a manual and error-prone work. A possible solution to automate the synthesis of concurrent checkers is to derive them from verification assertions. However, the number of assertions is generally far too high to allow for area-efficient checking infrastructure. Therefore, there is a need for qualification and minimization of assertions with a prospect of reusing them as hardware checkers. To derive low-area, high fault coverage hardware checkers from many assertions, there is a need for a framework for selecting a set of high-quality and minimized assertions by combining a data mining technique with the fault analysis approach along with an assertion conversion methodology that converts liveness assertions into safety assertions. The framework should be capable of synthesizing these safety assertions to hardware checkers to be evaluated at the gate level to provide a cost-effective checking infrastructure.

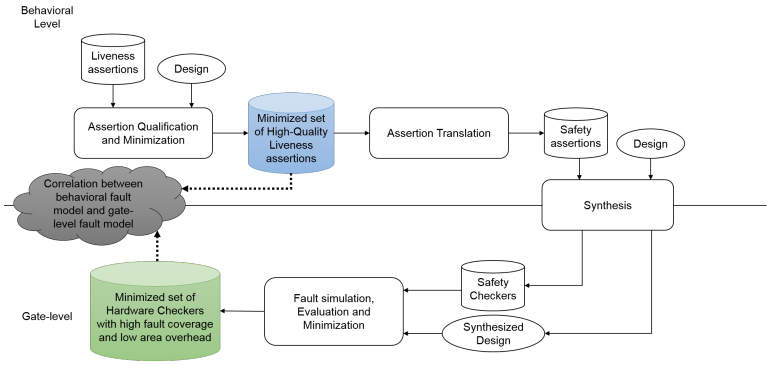


Figure 2 – Overview of the Thesis flow

Figure 2 depicts the overall flow of the framework for generating a set of cost-effective concurrent checkers from verification assertions. An assertion is a precise description of what behavior is expected when an input is fed into the design. Assertion ascends the

level of abstraction closer to design specification. A verification environment consists of a set of liveness assertions that collectively can detect a range of bugs in the considered design. However, not all the assertions are essential to detect this range: some assertions are dominated by others, or by a set of other assertions, some assertions are equivalent in terms of bug detection capabilities, etc. Discarding such assertions which do not detect any unique bugs leads to obtaining a set of minimized assertions. These assertion quality estimation and minimization tasks should be carried out to derive a minimized set of high-quality liveness assertions.

A checker is a hardware module whose output is a Boolean signal which assumes the value false when the sequence of values applied to the inputs do not satisfy the checking condition. For a liveness assertion, the output of the checker stays false until the condition becomes satisfied, whereas for a safety assertion, the output of the checker stays true when the condition is violated. And so, a translation procedure that translates the liveness assertions to safety assertions should be executed. The framework proceeds with the synthesis of safety assertions along with the considered design. The output of the synthesis is the hardware checkers and design implementation in terms of logic gates. The gate level fault simulation is carried out for the synthesized hardware checkers along with the considered design to evaluate the fault detection capability of the checkers. A trade-off between the fault coverage and area overhead of the hardware checkers are eventually outlined which in turns leads to derive a set of minimized hardware checkers. The optimized set of concurrent hardware checkers should be matching the target values of fault coverage and area overhead constraints.

The correlation between the assertion quality and the fault detection capability of the checkers needs to be studied and validated such that when assertion qualification and minimization task is carried out at a higher level of abstraction, it takes consideration of the gate-level fault coverage and area overhead of the hardware checker generated from these assertions.

1.3 Contributions of the thesis

This thesis proposes a framework for evaluating the fault detection capabilities of concurrent checkers for NoC routers. The goal is to achieve low-latency, low area overhead and high fault coverage checkers outlining the trade-off between the area constraint and fault coverage constraint. Also, a framework is proposed for selecting a set of high-quality liveness assertions by combining a data mining technique with the fault-analysis approach which allows reusing the verification assertions in hardware checkers synthesis. The quality of assertions is validated by studying the correlation between the fault detection capabilities of the checkers and assertions.

The main contributions of this thesis are:

- Providing evaluation of the fault detection capability of the concurrent checkers by formally proving the absence or presence of true misses over all possible valid inputs for a checker and targeting the minimum fault detection latency of a single clock-cycle. Pseudo-combinational extraction guarantees the possibility of fault simulating the circuit in an exhaustive valid range of conditions **[Publication I, Publication II, Publication III]**.

- A hybrid approach is proposed which combines concurrent checkers for control part with embedded on-line test packets replacing the data-path checkers to outline the trade-off between area-overhead and fault coverage **[Publication IV]**.

- The correlation between the fault coverage obtained from the behavior fault simulation with the qualified assertions and the fault coverage obtained from the gate-level simulation of the checkers which are synthesized from the qualified assertions is studied and validated **[Publication V]**.

- A methodology is proposed for producing a set of high-quality hardware checkers from Register-Transfer Level (RTL) assertions **[Publication VI]**.

1.4 Thesis organization

This thesis consists of 6 Chapters and 6 appendices.

Chapter 1 introduces the thesis, which includes the motivation, problem formulation and the main contributions.

Chapter 2 presents the background information about faults, fault models and fault simulation. An overview of target NoC router architecture is provided, followed by the introduction of the concept of concurrent checkers. Next, the metrics used to evaluate the fault detection capability of the checkers are introduced. At the end, assertions related topics along with metrics to evaluate the assertion quality are provided.

Chapter 3 describes the methodology flow for evaluating online fault detection and minimization of the hardware checkers. It also contains the literature review of related works of the topics. Experimental results are also discussed.

Chapter 4 outlines the linking of assertions and hardware checkers. First the correlation between the fault models is studied, followed by a translation procedure for assertions and then a conversion method to derive hardware checkers. It also contains the literature review of related works of the topics.

Chapter 5 describes the qualification method proposed for assertions followed by a minimization procedure to get a minimized set of high-quality assertions. It also contains the literature review of related works of the topics. Experimental results are also discussed.

Chapter 6 summarizes the conclusion and discusses the future research direction.

The appendices 1 to 6 present research papers that form the basis of this thesis.

2 Background

This chapter provides the background for the topics that form the basis of the developments in this thesis. The topics include faults, fault models, fault simulation, fault analysis, levels of abstraction, overview of NoC router architecture, concurrent checkers, metrics to evaluate fault detection capability, assertion-based verification, data mining and assertion mining, metrics for data mining.

2.1 Faults

2.1.1 Defect, fault, error

Defects are caused by process variations or random localized manufacturing imperfections. A *fault* is a representation of a defect reflecting a physical condition that causes a circuit to fail to perform in a required manner. A circuit *error* is a wrong output signal produced by a defective circuit. A *failure* is a deviation in the performance of a circuit or system from its specified behavior and represents an irreversible state of a component such that it must be repaired for it to provide its intended design function. A circuit defect may lead to a fault, a fault can cause a circuit error, and a circuit error can result in a system failure [11].

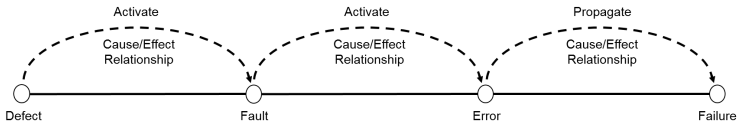


Figure 3 – Threats to digital circuits

2.1.2 Classification of faults

Most faults are caused by defects (e.g. shorts, opens, etc. induced by thermal aging, improper manufacture or misuse) or by environmental influences (e.g. particle radiation, electromagnetic fields etc.). Faults can be classified into Soft and Hard faults [9] as shown in figure 4.

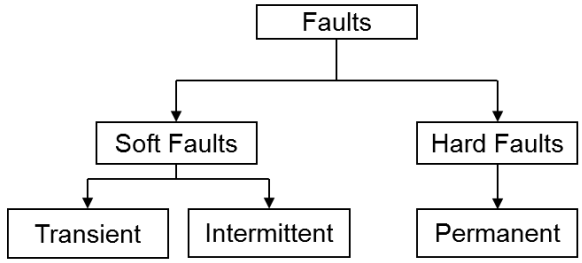


Figure 4 – Classification of faults

- Permanent fault: A permanent fault remains active until a corrective measure is taken. They are irreversible changes in the circuit. These faults are caused by some of the physical defects in the circuit like short circuits, broken interconnections. All permanent faults once have occurred, cannot get vanished and therefore the test to detect them can be easily repeated with the same results.

- Transient fault: A transient fault remains active for a short period of time. The occurrence of transient faults is random in nature and it is difficult to detect. Because

of their short duration, transient faults are often detected through the errors that result from their propagation. A common impact of a transient fault is a change of value in a single bit.

- Intermittent fault: An intermittent fault becomes apparent not continuously, but at irregular intervals. Intermittent faults can be due to implementation flaws, aging and wear-out and to unexpected environmental conditions.

2.2 Fault models

A fault model is necessary for fault simulation, fault analysis and generating and evaluating the set of test vectors. A fault model should accurately reflect the behavior of defects and it should be computationally efficient in terms of fault simulation and test pattern generation. No single fault model can accurately enumerate all possible defects that can occur. As a result, a combination of different fault models is often used in the generation of test vectors.

In this thesis, the goal is to develop error-checker circuitry that is able to detect all Single Event Transient (SET) faults. In the following, the stuck-at fault model, which is the basis of all logic level fault models is presented, followed by the description of single event effects (including SETs) and higher abstraction level fault models.

2.2.1 Stuck-at faults

The Single Stuck-At (SSA) fault model [12] is by far the most widely used fault model in digital testing. The reasons for its popularity lie in its ease of modeling and simulation and its close correspondence with real physical defects in digital integrated circuits. Three properties define a single stuck-at fault:

- Only one circuit line is faulty.
- The faulty line is permanently set to 0 or 1.
- The fault can be at an input or output of a gate.

One of the limitations of SSA is the fact that it models only a single fault at a time and therefore ignores the effects of fault combinations. The reason for resorting to single fault instead of considering multiple faults approach lies in the fact that while there exists 2n SSA faults in the circuit, where n is the number of lines, whereas there are as much as 3^n-1 multiple faults. Although the multiple-fault model is more accurate than the single-fault assumption, the number of possible faults becomes impractically large other than for a small number of fault types and fault sites. Also, it has been shown that high fault coverage obtained under the single-fault assumption will result in high fault coverage under the multiple-fault model [11].

2.2.2 Single Event Effects: SET and SEU fault models

Single Event Effects (SEEs) are faults caused by a single, energetic particle striking a sensitive node in the circuit. SEEs are normally soft errors, which means they do not cause permanent damage to the circuit. The main causes of SEE are from radioactive decay of the packaging materials (alpha particles) or high-speed neutrons from cosmic rays colliding with silicon atoms creating secondary particles, which then create an ionization track where the electrons or holes can get collected on the source or drain of a transistor, causing the soft errors [2]. SEEs can be divided into single event transients (SETs) and single event upsets (SEUs) as shown in figure 5:

- SET - A glitch caused by single event effect, which travels through combinational logic and is captured into one or several storage elements.
- SEU - SEU is a change in the state of a storage element inside a device or system.

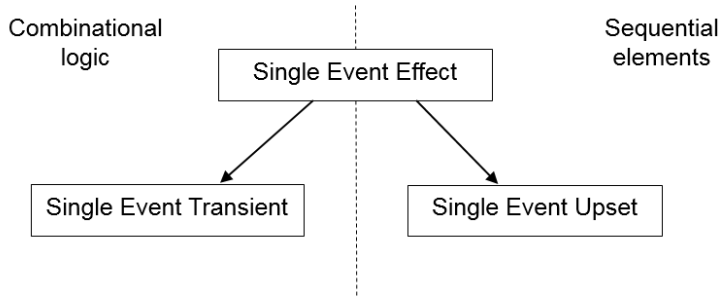


Figure 5 – Classification of Single Event Effects

2.3 Fault simulation

In general, simulating a circuit in the presence of faults is known as fault simulation. The main goal of fault simulation is measuring the effectiveness of the test patterns. Any input pattern or sequence of input patterns, that produces a different output response in a faulty circuit from that of a fault-free circuit is a test vector. Fault simulation is an essential process for reliable design. Fault simulation is typically used to evaluate the fault coverage obtained by the set of test vectors which is defined as the fraction (or percentage) of modeled faults detected by test vectors divided by the set of total faults [11].

$$\text{Fault Coverage} = \frac{\text{Number of Detected Faults}}{\text{Total number of faults}} \tag{1}$$

A fault simulation requires a fault model which provides a quantitative measure of the fault detection capabilities of a given set of test vectors.

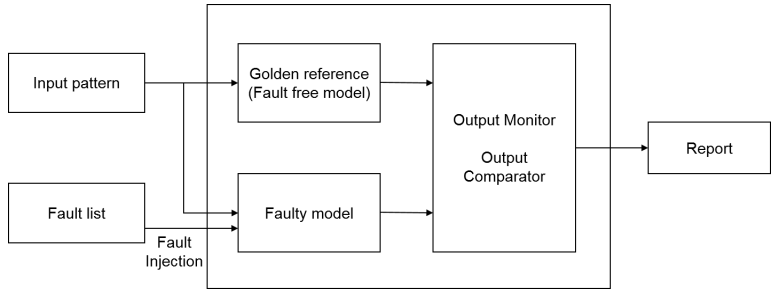


Figure 6 – Concept of Fault simulation

2.4 Mutation-based fault analysis

In this thesis, we consider mutation-based fault analysis only. *Mutation analysis* induces many simple faults, called mutations into program (HDL design) to create a set of mutant programs [13]. A mutation is a single syntactic change that is made to the program statement. Each mutant program should differ from the original program by one mutation. Each mutant is executed with the test cases and when a mutant produces an incorrect

output on a test case, that mutant is said to be killed by that test case. When this happens, the mutant is considered dead and no longer needs to remain in the testing process since the faults represented by that mutant have been detected. Mutants are limited to simple changes on the basis of the coupling effect, which says that complex faults are coupled to simple faults in such a way that a test data set that detects all simple faults in a program will detect most complex faults [14].

The quality of the test cases is measured by the percentage of faults detected (i.e., mutants that they kill) against the total number of injected faults (i.e., injected mutants) which shows how good the verification environment is at detecting faults.

$$\text{Fault coverage} = \frac{\text{Detected faults}}{\text{Total number of injected faults}} \tag{2}$$

2.5 Levels of abstraction

Due to extreme down-scaling, chip density reaches hundreds of millions of transistors per die. A key method of managing the ever increasing complexity is to describe a system at several levels of abstraction [15]. An abstraction is the simplified model of the system, showing only the selected features and ignoring the associated details. A higher level of abstraction focuses on the most vital data like functional specifications. Whereas the lower level of abstraction are more complex but it is more accurate and is closer to the real circuit. Four levels of abstraction considered in digital system development as shown in figure 7 are

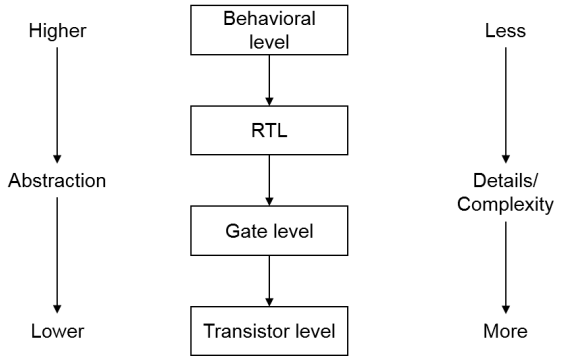


Figure 7 – Levels of abstraction

- Behavioral Level
- Register Transfer Level (RTL)
- Gate Level
- Transistor Level

2.6 Network-on-Chip router architecture

A System on Chip (SoC) refers to a single-integrated circuit composed of all the components of an electronic system. A SoC is heterogeneous, in addition to classical digital components: processor, memory, bus, etc.; it may contain analog and radio frequency components [16]. With the increasing number of on-chip components and further advances in semiconductor technologies, the communication complexity increases and there is a need for alternatives

to the traditional bus-based or point-to-point communication architectures. Although these architectures have the advantages of simple topology, extensibility, and low area cost, these do not scale the system performance with the number of cores attached. As the number of cores gets increasing, it causes high latency which in turn decreases the system performance. Also, long wires cause high power consumption. Network-on-chip (NoC) has emerged as the viable alternative for the design of modular and scalable communication architectures [17].

In NoC, the cores communicate with each other using a router-based packet-switched network. A SoC can be composed of processing elements (resources). A processing element can be memory, processor core, DSP or any IP block. Instead of connecting them by dedicated point-to-point channels, an interconnection network is implemented as a set of shared routers and communication links between the routers. A processing element can communicate through the network with any other module connected to the communication infrastructure, not only with its neighbors. This leads to advantages in terms of structure, performance and modularity.

Figure 8 provides an example of a typical NoC communication infrastructure. The network consists of routers (R) connected by interconnect lines. A processing element (Core) is attached to a router through a network interface (NI) module [10] enabling seamless communication between various cores and the network. The way the routers are connected to each other defines the network topology. It is worth noting that the NoC router used for experimental case study throughout the thesis was implemented in-house. The author was solely responsible for developing the entire NoC router design.

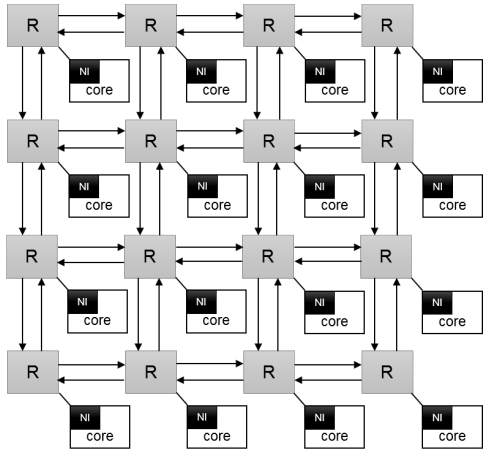


Figure 8 – An example of NoC based SoC

A NoC router consists of a control part and a data-path. The NoC router architecture used in this thesis for case study is implemented as follows: Data-path responsible for transmission of the actual data to destination includes

- input buffers, implemented as First-In-First-Out (FIFO), one for each input port
- crossbar switch, implemented with MUXs, one for each output port
- output buffers, implemented as simple single-slot registers, one for each output port

The control part affecting the flow of data through the data-path comprises of

- routing computation unit, which is based on Logic-Based Distributed Routing (LBDR) [18], one for each input port
- arbitration unit, which is based on Round Robin starvation free priority and one-hot encoding of the state, one for each output port.

The considered NoC topology used in this thesis is shown in figure 9 where the following considerations are made such as 4x4, 2D mesh and the target Design Under verification (DUV) is router number **5**, counting from the left top corner, from left to right and from top to bottom, starting from number **0**. Each router has 5 input/output ports, one for each direction. North – N, East – E, South – S, West – W and Local - L. Local port is connected to the Processing Element (PE) associated with router.

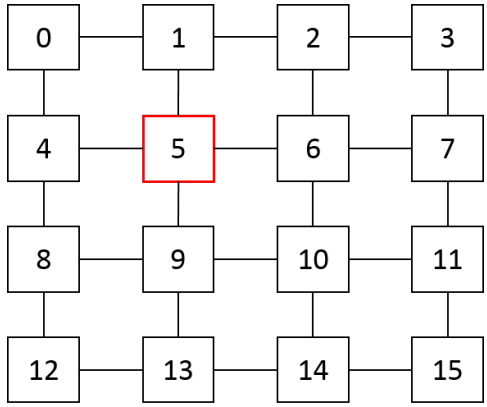


Figure 9 – Target NoC router number 5 in considered 4X4 2D mesh topology

The NoC router utilizes wormhole switching. Therefore, packets are sent in form of flits, consisting of a header flit, body flit(s) and a tail flit. Whenever a flit arrives at a NoC router, first it will be stored in the corresponding input buffer. Next, the routing computation logic estimates the appropriate output port(s) based on the destination address stored in the header flit of a packet acquired from the input buffer and signals the arbiter. It is worth noting that the routing logic only becomes active upon receiving a header flit. The role of the arbiter is to solve the contention when multiple input ports want to access the same output port, which is done based on prioritization algorithm. Since at the same time multiple requests might be sent to different output ports, one arbiter is instantiated for each output port. An additional role of the arbiter is to control the data-path, that is, when the grant is given to an input port, arbiter allows data to be sent to its corresponding output port from the granted port, at the same time opening the correct path through the crossbar switch.

Figure 10 depicts the high-level overview of the NoC router architecture used in this thesis. The architecture will be described in the following sections.

2.6.1 Routing computation unit: LBDR

The Logic-Based Distributed Routing (LBDR) mechanism [18] is used as the routing computation unit in the NoC router. The design of scalable and reliable interconnection networks for multicore chips introduces new design constraints like power consumption, area, and ultra-low latencies. Usually routers can be easily configured to support most routing algorithms and topologies by using routing tables, but the routing table does not scale in terms of latency and area.

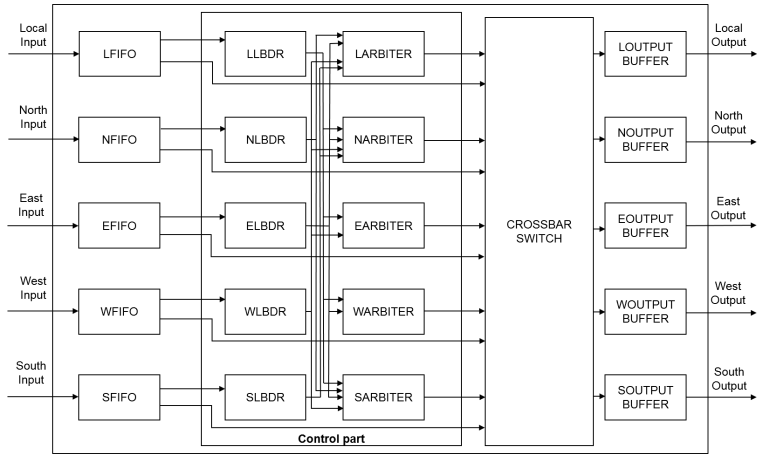


Figure 10 – High-level overview of NoC Router Architecture

LBDR mechanism has been introduced to support different routing algorithms in 2D NoC. The mechanism relies on the use of only three bits per output port (excluding local, 12 bits per router) which are grouped in two sets.

- 4 *Connectivity bits*, one for each output port excluding local, describing the topology by indicating the connection of each router to its possible neighbor.
- 8 *Routing bits* describes the routing algorithm by considering whether packets can change direction at next router.

This way of describing the routing logic with routing and connectivity bits ensures both easy reconfigurability and scalability of the routing computation unit. LBDR removes the need of a routing table and therefore it would be more scalable, as it only depends on a limited set of registers and bits in each router. The logic of LBDR is shown in Figure 11.

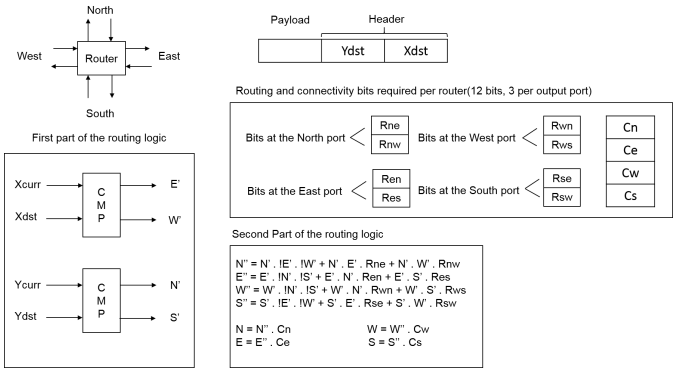


Figure 11 – LBDR mechanism [18]

LBDR is a distributed routing mechanism, thus, for routing computation it only relies on the current address of the router and the address of the destination node included within the header flit of a packet. On the other hand, in source routing, the source node computes the whole path and stores it in the packet header flit. So, LBDR avoids consuming significant network bandwidth. The LBDR becomes active (only) when a header flit is received and takes a decision regarding the direction to be followed by the packet at the next hop, based only on destination address and address of the current router.

The following constraints are considered while designing the LBDR logic for NoC router

- Router 5 in a 4x4 2D mesh topology
- XY Dimension-ordered routing algorithm
- no 180° turn restriction, i.e., a packet coming from a port cannot be forwarded to the same port

Based on the above constraints, the connectivity and routing bits are configured as follows:

- C_n, C_e, C_w, C_s = 1
- R_{ne}, R_{nw}, R_{se}, R_{sw} = 0
- R_{en}, R_{es}, R_{wn}, R_{ws} = 1

A connectivity bit C_x describes the absence (0) or presence (1) of a neighbor router in a certain direction. A routing bit R_{xy} describes the permission to forward the packet to x direction and take y direction at the next hop (1) or not (0). A simplified version of LBDR logic based on the above constraints is shown in figure 12, for instance for **East** input port. It is worth noting that XY routing algorithm allows at most one port at time to be selected for forwarding a packet, furthermore the output port corresponding to the direction to which LBDR is related cannot be selected. In order to cover a wide range of fault occurrences, one-hot encoding is considered for the flit type in the structure of flits.

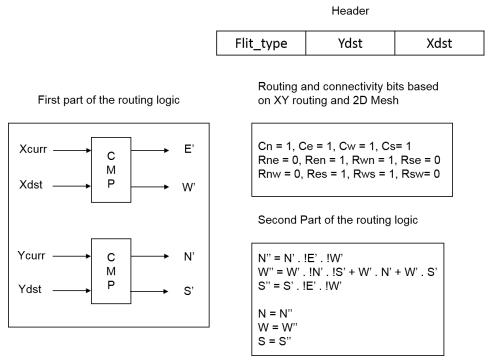


Figure 12 – East LBDR logic for NoC router

LBDR is a sequential design, including flip-flops to store the current values of the output requests for the arbitration units. The proposed methodology for evaluating the fault detection capability of the checker described in section 3 requires the extraction of pseudo-combinational circuit from the original sequential circuit. The pseudo-combinational version of the circuit is extracted by breaking the registers in two different set of signals, one representing the current values of the output request, now fed as inputs to the pseudo-combinational version, the other representing the newly evaluated values of the output request, according to the current values and functional inputs of the logic. The pseudo-combinational version for ELBDR is shown in figure 13.

Table 1 presents the checkers introduced for ELBDR logic. The checkers were devised for the pseudo-combinational version of the design. After optimization, the checkers are extended to the actual sequential version of the design.

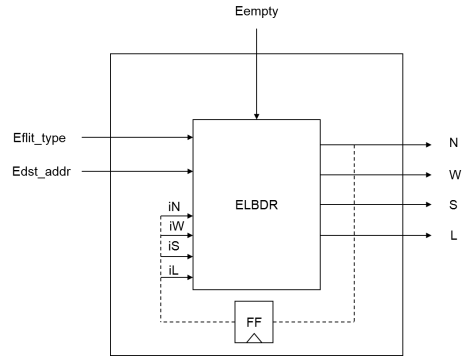


Figure 13 – Pseudo-combinational version of ELBDR logic

Valid LBDR output	If there is a request to the routing logic (the corresponding input buffer is not empty and flit type is valid), LBDR has to compute at least one valid output direction (according to XY routing).
No LBDR output	If no flit arrives (the corresponding input buffer is empty), all the output port signals of LBDR should remain zero.
Single LBDR output	If the corresponding input buffer is not empty (there is a request to LBDR), because of using XY routing, at most only one output port signal of the LBDR logic can become active.
Switch LBDR output	If the corresponding input buffer is not empty (there is a request to LBDR) and a non-header flit has arrived, LBDR outputs should remain the same.
Local port output	If the corresponding input buffer is not empty (there is a request to LBDR) and a header flit has arrived, the local output should become active only if the packet has reached its destination.

Table 1 – Checkers for LBDR logic

2.6.2 Arbitration unit: Round-Robin arbiter

Arbitration unit plays an important role in NoC router by serving simultaneous request from multiple input ports and granting access to one of them based on a scheduling algorithm implemented to send flits through a single output port. There are many different implementations for an arbitration unit, and most of the designs are implemented in a sequential way, describing a Finite State Machine (FSM), in which any different state represents a different possible granting condition.

The arbitration unit should guarantee the fairness in scheduling, to avoid starvation, and to provide high throughput [19]. Round Robin (RR) arbitration satisfies fairness and for this reason a RR arbiter is considered in this work. RR is based on dynamic prioritizing, giving highest priority to the L input port, then N, E, W and finally S and again it starts from L input port. This way of circular prioritization can guarantee that there would not be any starvation and all input ports will eventually get access to their requested output port. Also, each input port cannot hold an output port for more than a specific threshold period during each arbitration round, therefore, the arbiter controls this by setting a timer to threshold period. It is noteworthy that, protecting the timer by checkers has not been

considered in this thesis. In the pseudo-combinational version of the arbiter circuit, the timer logic was not considered. While in the sequential actual version of the arbitration unit, the timer didn't contribute to significant loss of fault detection coverage.

A high-level overview of the functionality of a RR arbiter is shown in figure 14. The inner FSM of the arbitration unit present 6 different states, corresponding to the following 6 possible granting conditions, in which the potential input requests are considered in the decreasing order of priority.

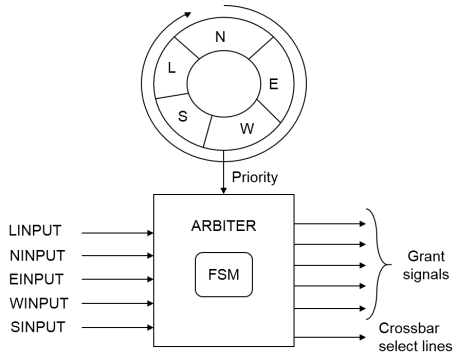


Figure 14 – Overview of Round-Robin Arbiter

- *IDLE* - grant is given to none of the input requests
- *GRANT_L* - grant is given to the local input request
- *GRANT_N* - grant is given to the north input request
- *GRANT_E* - grant is given to the east input request
- *GRANT_W* - grant is given to the west input request
- *GRANT_S* - grant is given to the south input request

Different encoding style can be chosen for the states of the FSM in the arbiter. In this thesis, one-hot encoding style is considered. Such that, it would be easier for the checkers to check if, for example, a fault occurs in the state register and violates the one-hot rule. The state variable of the arbiter FSM is used to generate grant signals for the input ports and select lines of the crossbar switch. Thus, grant signals and select lines follow the one-hot encoding as well, in a way that at most one signal can be high during each arbitration, i.e. only one input port at time can send its data towards a certain output port. It is important to emphasize the choice made for selecting the encoding style of the state-variable. Binary encoding style would lead to reduced area for both the arbitration unit itself and its checker logic. On the other hand, in case of single Stuck-At Fault (SAF), it could be possible that the state variable changes from a valid value to another valid value, but incorrect. With one-hot encoding of the state variable, any single SAF would lead to an invalid value for the variable, thus making fault detection much easier and more effective.

A simplified version for the arbitration unit is considered for **south** output port arbiter. The south arbiter cannot provide grant to the south input port because of no 180° turn restriction. The pseudo-combinational version of South arbiter is shown in figure 15 by omitting clock and reset signals. To extract the pseudo-combinational version of arbiter, the register for the state variable is broken into two different set of signals. One fed as

input representing the current value of the state variable, the other considered as output, representing the newly evaluated value of the state variable, according to the current state and the functional inputs of the logic.

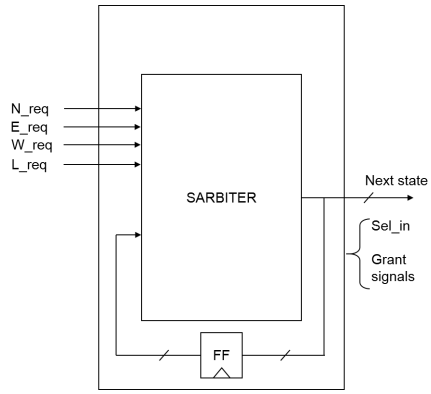


Figure 15 – Pseudo-combinational version of SARBITER logic

Table 2 presents the checkers introduced for the RR arbitration unit. As previously stated for the LBDR logic, the checkers were devised for the pseudo-combinational version of the arbitration unit. Once optimization has been accomplished, the final set of checkers has been later extended to the actual sequential version of the arbiter.

Valid Grant output	If there is a request from LBDR, arbiter must assert at least one of the grant signals for the corresponding output direction.
No Grant output	If there is no request to the arbiter, it should not assert any of the grant signals for any direction.
Invalid Grant output	Whenever there is a request to the arbiter, the grant signal corresponding to that specific requested direction should go active, and invalid direction should not be chosen.
Invalid Arbiter state	State variable of the arbiter's FSM cannot possess invalid values according to the one-hot encoding.
Invalid IDLE state	If the arbiter logic is in IDLE state, and there is a request for arbitration from LBDR, the circuit should not remain in IDLE state, i.e. a grant signal should be asserted.
Priority Grant	In case there are requests to the arbiter, it should follow the correct prioritization (Local, North, East, West and then South), according to the current value of the state variable.

Table 2 – Checkers for Round-Robin Arbiter logic

2.6.3 Input buffer: FIFO

The input buffer is implemented as First-In-First-Out (FIFO) for the targeted NoC router architecture. Since the amount of sequential logic is large for a buffer, it is not feasible to extract the pseudo-combinational version of the circuit. Thus the checkers were devised for actual sequential version of the circuit from the beginning.

The FIFO has been implemented using two pointers: *Write pointer* and *Read pointer*

along with two status signals: *Empty* and *Full*, which signals a neighbor router regarding the status of the input buffer. Additionally, to accept the request for writing to the FIFO or reading from it, it has two more input signals: *Write Enable* and *Read Enable* respectively.

Normally the data are written into FIFO if it is not full and the Write Enable signal (which comes from a neighbor router in the network) is set to high. When a flit is written to the buffer, the write pointer is incremented by one step and then points to the location not yet written to. In the same fashion, the read pointer is incremented whenever there is a request for reading from the buffer i.e., Read Enable is set to high and the FIFO is not empty. The FIFO is considered as a circular buffer i.e., once a pointer reaches the end of the buffer, the pointer just wraps around at the next increment and will point to the first memory location.

Similar to [20], the buffer module is implemented with:

- one-hot encoding style for read and write pointers rather than binary encoding.
- a set of registers to store the data instead of an array of memory locations.

Like arbitration unit, one-hot encoding style is used in FIFO to make the design more robust to single SAFs, aiming to avoid the faulty but legal output situations i.e., one of the pointers changes its value from a valid one to another one which is valid but incorrect. The depth of the FIFO is set to 4 slots, corresponding to 4 registers to store data coming from the input port. Thus, 4 bits are used to describe the values of the pointers – 0001, 0010, 0100 and 1000 as per one-hot encoding style. Each bit of the write pointer represents the enable signal for the one of the memory registers. To pass the data to output of the buffer, a one-hot multiplexer is implemented at the output which is in-turn controlled by the read pointer.

Initially, both write and read pointers will point to the same address in the buffer (0001). In case of FIFO write, the incoming data are written in the location currently addressed by the write pointer and then the write pointer is left shifted by one bit. Similarly, after a read operation, the read pointer is left shifted by one bit. The following conditions are used to identify FIFO full or empty conditions:

- the Empty signal goes high whenever read and write pointer addresses the same location in the FIFO registers.
- the Full signal goes high whenever the read pointer is equivalent to the write pointer but shifting one bit to the left.

It is worth noting that, to generating Full and Empty signals, only 3 registers of the FIFO can be used at the same time for storing incoming flits.

Table 3 presents the proposed checkers for 4-flits depth FIFO. Only the control part of the FIFO design was addressed, because the data-path is protected by parity checkers which will be described later.

2.6.4 Crossbar switch

Crossbar switch is implemented to connect inputs and outputs of the NoC router. Crossbar switch can establish multiple parallel data paths. Thus, Crossbar switch must be designed in a way that can guarantee connection between every input port and every output port and connections realized by the crossbar are determined by arbiter. Each arbiter decides which of the input ports can access its corresponding output port. Crossbar switch is implemented in the form of multiplexers shown in figure 16, one for each output port.

Reset Checker	Whenever reset goes high, at the next clock cycle Empty flag should be high (reading and writing pointer are reset to the same value).
Flags Checkers	Empty and Full flags should never be high at the same time. Whenever the defining condition occurs, the corresponding flag should go high at the next clock cycle.
One-hot pointers checkers	Reading and writing pointers must respect one-hot encoding.
Registers enable DMR checker	Duplication and comparison for the logic enabling the writing operation in data registers.
Reading pointer update checker 1	Whenever Read Enable is high and the FIFO is not empty, at the next clock cycle the reading pointer should be updated.
Reading pointer update checker 2	If either Read Enable is low or the FIFO is empty, at the next clock cycle the reading pointer should preserve its value.
Writing pointer update checker 1	Whenever Write Enable is high and the FIFO is not full, at the next clock cycle the writing pointer should be updated.
Writing pointer update checker 2	If either Write Enable is low or the FIFO is full, at the next clock cycle the writing pointer should preserve its value.

Table 3 – Checkers for FIFO Control part

The select lines of the crossbar switch which are generated by the arbiters are encoded in the form of one-hot fashion to make the fault detection easier. Also, the considered router is limited to XY routing, therefore connections between some inputs and outputs are not needed. For example, for the **East** crossbar switch, turns from North, South, East (no 180° turn) input ports to East output ports are restricted, only input connections from Local and West input ports are allowed. The similar design constraint is applied for the other crossbar switches for other output ports.

Since the checkers devised for Crossbar switch have resulted in unbearable area overhead, it was decided to follow Double Modular Redundancy (DMR) approach. Therefore, the MUXs in the crossbar switch are duplicated, then real and duplicated outputs are compared.

2.6.5 Even parity checkers

Parity computation provides a simplest means of detecting single event faults. A parity bit may be generated from the various bits of the incoming data entering each input port. A parity bit can be even or odd based on design consideration. In this thesis, even parity checker is considered. The parity checker evaluates the data before data leave the router and indicate a parity error if odd parity is received. Figure 17 shows the high-level overview of the targeted NoC router including parity generation and checker. Parity checkers are effective in detecting the faults in the data-path, i.e., in the registers of the input and output buffers.

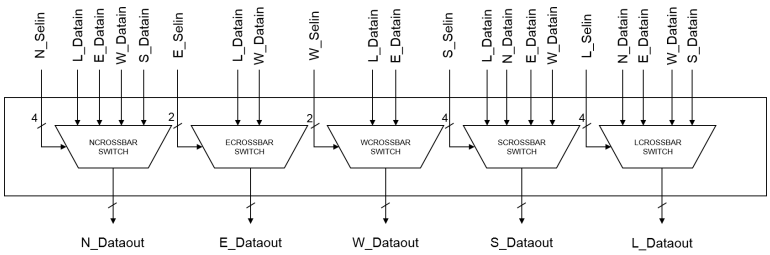


Figure 16 – Crossbar switch architecture

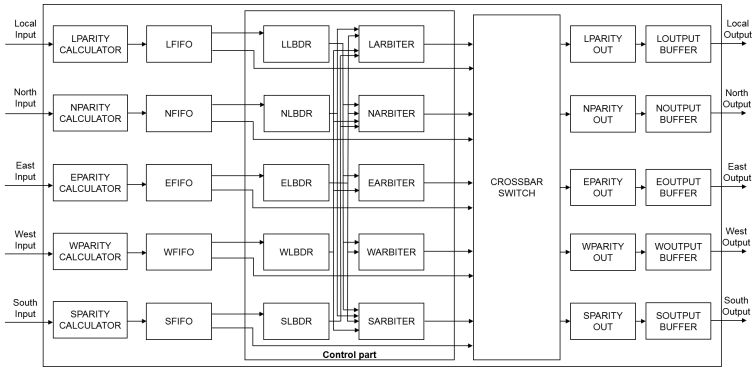


Figure 17 – Overview of NoC Router with embedded parity checking

2.6.6 Infrastructure of the complete router

In the previous subsections, each block composing the considered NoC router has been considered separately. Each block has been studied as a stand-module in checkers evaluation and minimization computation in the first place. When all the blocks are put together to build the whole NoC router, new undetected faults popped up. Most of these faults are related to the infrastructure of the router, i.e., the logic dedicated to correctly establish the communication between different modules, especially those belonging to the control part.

Table 4 list the checkers devised to deal with infrastructure of the control part of the router. The first one checks one-hot encoding for the flit type input of the LBDR, passed to the routing computation logic from the corresponding input FIFO. The second and third checkers address the simple logic which deals with the use of the grant signal produced by the arbiters, both to enable the reading from the FIFOs and writing to the output buffers. The logic is duplicated since the structure of the logic is simple (group of 5 OR gates).

Flit type LBDR error	Flit type field of a flit must respect one-hot encoding.
FIFOs Read Enable DMR checker	Logic producing Read Enable signals for the FIFOs (5 OR gates) is duplicated, then real and duplicated outputs are compared.
Output registers enable DMR checker	Logic producing enable signals for the output registers (5 OR gates) is duplicated, then real and duplicated outputs are compared.

Table 4 – Checkers for the control part infrastructure

2.7 Concurrent checkers

In this section, the concept of concurrent checkers is introduced. A set of checkers (Checker

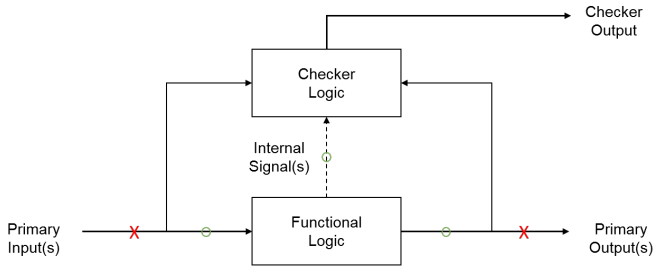


Figure 18 – The concept of Concurrent checker

logic) is generated and then connected to the functional (primary) inputs and outputs of the circuit. Figure 18 shows the functional logic augmented with checker logic. These checkers are introduced based on functional assertions derived from relationships between variables corresponding to inputs and outputs and also possibly internal signals of the circuit. The checker logic targets the faults at lines within the functional logic (marked by green circles). On the other hand the checkers are not designed to detect faults occurring at

- functional inputs preceding the checkers input
- function outputs succeeding the checkers inputs

marked with red cross. In the first case the checker logic would normally not be able to detect that a functional input has been altered by a fault, indeed the behavior of the checkers is based on the assumptions that the functional inputs are correct. In the latter case the situation is mirrored, the eventual fault on a functional output would occur after the scope of the checker logic.

2.8 Metrics to evaluate fault detection capability

In this section, the metrics which are used to evaluate the fault detection capability of the checkers are explained.

Traditionally the most significant index to describe fault detection is fault coverage, but its conventional definition cannot be considered when checker logic is introduced to achieve online fault detection [3]. Fault detection is generally evaluated feeding the considered circuit with a suitable test patterns set, considering SAFs in all the possible locations, and marking a fault as detected when it's made observable to the circuitry outputs by a certain vector. Finally, the index is obtained as the ratio between detected and undetected faults. This conventional definition makes absolutely no sense in the evaluation of checkers applied to functional logic. The behavior of each checker strongly depends on the considered input vector, i.e. a checker could detect a fault with a certain test pattern, while missing it with a different one. For this reason, it makes no sense to consider a fault detected by checker logic as soon as there is a checker flagging it.

In traditional fault detection evaluation, given a fault at a line within the functional logic and a set of input stimuli, the possible scenarios are

- fault is detected (i.e. it is observable at the outputs)
- fault is not detected (i.e. it is not propagated to the output)

In this thesis, a classification of scenarios, using the following terminology to describe the possible different situations in detection of an injected fault is given:

- Case 1: Fault occurs at an internal line and is visible at functional output(s) and checker logic flags a violation. The term **True Detection** is used to describe this situation, since a critical fault is effectively detected by the checker.

- Case 2: Fault occurs at an internal line but is not visible at primary output(s). Checker catches the fault and flags a violation. The term **False Positive** is used to describe this situation. False positive is not harmful because an error is flagged which did not have any effect. However, it has negative impact on design's performance because normally it causes re-execution of the task.

- Case 3: Fault occurs at internal line but is not visible at primary output(s) and the checker logic does not detect the violation. The term **Benign Miss** is used to describe this situation. Benign miss shows correct operation by the checker.

- Case 4: Fault occurs at internal node and is visible at primary output(s). Checker does not detect violation. The term **True Miss** is used to describe this situation, which is the worst possible case. True miss means that the fault propagates to the functional outputs and further propagates to the system. However, the system has no information that a critical fault has occurred.

With *visible* means the situation in which, given a fault and an input test pattern, the fault is propagated to the functional outputs of the considered logic, i.e. the values of these outputs are different from those of the fault-free simulation. Table 5 summarizes the four possible scenarios. Here ✓ means fault is visible at the output and *X* means not detected at the output.

Case	*Functional Logic*	*Checker Logic*
True Detection	✓	✓
False Positive	*X*	✓
Benign Miss	*X*	*X*
True Miss	✓	*X*

Table 5 – Checkers Evaluation Metrics

It is worth noting that the class of faults described as False Positives must be carefully considered. Even though harmless on a functional ground, due to the fact the fault is not propagated to the primary outputs of the considered circuit, if checker logic detecting faults are evaluated to repeat operations, this may lead to useless re-executions of tasks, causing undesired delay.

The following three metrics are introduced to evaluate the fault detection quality of the checkers:

- *FC* - Fault coverage

- *FPR* - False Positive Ratio

- *CEI* - Checkers Efficiency Index

The fault coverage metric is redefined according to the concurrent online detection capability of the checker logic compared to the conventional fault coverage metric.

Let D be the number of occurrences of true detections, F be the number of occurrences of false positives, W be the number of occurrences of true misses and X be the number of occurrences of benign misses detected by checker logic with the considered set of input stimuli.

FC is defined as

$$FC = \frac{D+X}{D+X+W} \tag{3}$$

FC can be considered as the probability of checkers behaving correctly on a larger set of situations.

FPR is defined as

$$FPR = \frac{F}{F+X} \tag{4}$$

FPR describes the ratio of false positives among those faults which are masked towards the primary outputs of the considered functional design.

CEI is defined as

$$CEI = \frac{D}{D+W} \tag{5}$$

CEI can be considered as the probability that checkers detect critical faults, those effectively leading to a faulty output behaviour in the considered design. It is fundamental to underline that this newly devised index cannot be straightforwardly compared with traditional fault coverage evaluation, thus it is important not to be narrow-minded towards results that may not immediately track the unity value, avoiding discarding them, as it is commonly done towards fault coverage results which are not close to 100%. Before concluding about a checker as ineffective, or on the other hand extremely effective, it is fundamental to devise the right set of input patterns to be used in evaluating the efficiency of the detection granted by the checker logic, because the CEI value may change drastically, since the behaviour of the checker logic strictly depends on the considered set of stimuli.

2.9 Assertion based verification

Assertion-based verification (ABV) [21], has gained popularity in verification process by providing a more powerful and easy way to verify complex digital systems. ABV has been successfully applied at multiple levels of verification abstraction ranging from high-level assertions within transaction-level testbenches down to implementation-level assertions synthesized into hardware. An assertion is a conditional statement that checks for specific behavior and displays a message if it occurs. Assertions are generally used as monitors looking for bad behavior but may be used to create an alert for desired behavior as well. Assertions can be used to verify the functional correctness of the design with respect to the expected behavior. Some of the benefits of assertions are reducing verification time, catching errors earlier, focussing the design effort and pinpointing sources of error.

2.10 Data mining and assertion mining

Some definitions and concepts concerning data mining and assertion mining are reported initially. Data mining [22] [23] deals with item sets, transactions and association rules, which are defined as follows .

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of items. Let $D = \{d_1, d_2, \ldots, d_m\}$ be a data set, i.e., a set of observations, called transactions, with respect the set of items I. Each element in D

contains a subset of the items in I. An association rule is defined as an implication of the form $X \rightarrow Y$ where $X, Y \subseteq I$ and $X \cap Y = \emptyset$. X and Y are called item sets.

Figure 19a shows an example of a data set which describes the behaviours of customers in a supermarket with respect to a set of items (i.e., milk, bread, ..., coffee). Data mining approaches are generally intended to extract association rules from data sets, which are then used to predict non trivial, implicit, previously unknown and potential useful information, like, for example, "when milk is bought bread and coffee are generally bought too", which is expressed by the association rule $Milk \rightarrow Bread \wedge Coffee$.

Assertion mining deals instead with execution traces and assertions. Figure 19b shows, an example of an assertion in Linear Time Logic is $always(p_1 \rightarrow next(p_2 \wedge p_3))$ which states it always happens that p_2 and p_3 are satisfied one simulation instant later than p_1 becomes true. Assertions are generally considered as a formula in the form of $A \rightarrow C$, where the antecedent A and the consequent C are composed of propositions, logic connectives, and temporal operators according to the selected temporal logic.

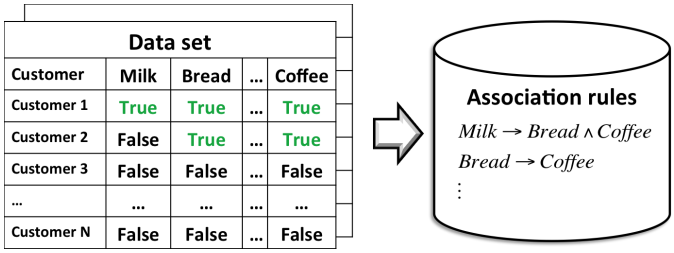
The overall goal of data mining is to extract information from a data set and transform it into an understandable and useful structure. This structure allows user analyzing data from many different dimensions, categorizing them and summarizing correlations between items in a database. For example, analyzing data from behaviours of different customers as reported in Figure 19 leads to obtain useful information and helps analyzers to decide which trend is more interesting for marketing. Association rules can also be extracted when data are referred to time sequences. In this case, temporal data mining strategies are adopted, whose goal is to discover hidden relations between sequences and sub-sequences of events [24]. In any case, the mined (temporal) association rules are a prediction for future behaviours, which may be true or not. Metrics are thus used to estimate the probability that rules extracted from past observations can be valid also in the future.

On the contrary, the main goal of assertion mining consists of extracting formulas that exactly describe the functionality implemented in the DUV, which is not ambiguous and does not vary in the future, except in the case the implementation is changed. Assertion mining is thus not intended to predict the future, but to formalize the actual set of DUV behaviours.

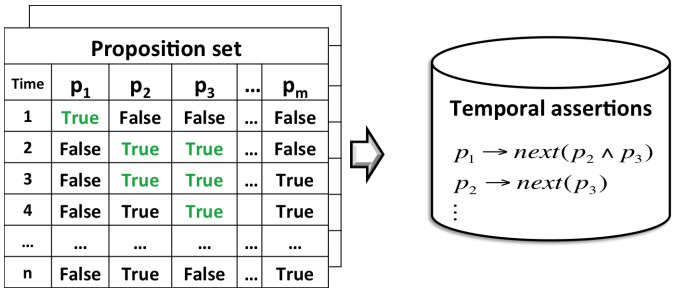
Summarizing, main similarities among data mining and assertion mining are the presence of a set of data that represents observations with respect to past behaviours exposed by the observed target (customers, DUV, ...), and the need of extracting association rules that formalize such observations. Items, data sets, and association rules in data mining correspond, respectively, to propositions, execution traces, and temporal assertions in assertion mining. Meanwhile, the main difference between data mining and assertion mining is represented by the concept of transaction (i.e., a row in a data set), which does not have a direct correspondence with a row of an execution trace, because an assertion is composed by one antecedent and one consequent that are true in different instants inside the execution trace. This difference impacts on the way metrics typically adopted for evaluating association rules in data mining can be reused for measuring the quality/interestingness of assertions. Finally, another difference is related to the final goal of the mining: in one case the prediction of future behaviours, in the other the formalization of actual (except in the case the DUV functionality is changed) behaviours.

2.11 Metrics for data mining

In the context of assertion qualification [25], metrics that provide information about the degree of accuracy of a rule with respect to the probability it will hold in the future (like for example, confidence, which estimates the joint probability between occurrences of the antecedent and the consequent in the data set) are not relevant, since the assertions under



(a) Example of data mining



(b) Example of assertion mining

Figure 19 – Similarities between data mining and assertion mining

analysis are always true on the DUV. Rather, the metrics that measure the interestingness of an assertion with respect to covered behaviours, number of activations, and correlation between antecedent and consequent and etc. are worth interested. For this reason, the following metrics *support*, *correlation coefficient* and *strength measure* are identified and their definition in the context of data mining is hereafter reported together with considerations related to how they can be adapted to be suited for assertion evaluation [26].

Definition 1 *Given a set of items I, and the corresponding set of transactions D, a rule $X \rightarrow Y$ has support S if X and Y occur concurrently in S percent of transactions in D.*

In practice, to compute the support of an association rule, it is necessary to count how many rows in the transaction set table contain both X and Y. In case of temporal assertions, the support corresponds instead to the number of times a temporal assertion occurs (i.e., its antecedent is fired and consequently its consequent is satisfied) in the execution traces with respect to the total number of occurrences corresponding to the other temporal assertions under analysis. For example, let us consider a temporal assertion $A \rightarrow C$ that occurs 10 times in a set of execution traces. If it belongs to a set of temporal assertions that globally occur 1000 times in the same execution traces, then the support of $A \rightarrow C$ is 10/1000 = 0.01.

Definition 2 *Given a set of items I, and the corresponding set of transactions D, the correlation coefficient of the rule $X \rightarrow Y$ is the co-variance of X and Y divided by the product of their individual standard deviations.*

More informally, the correlation coefficient can determine if antecedent and consequent are related or not by observing whether occurrences of the antecedent depend on oc-

currences of the consequent and vice-versa. The higher the correlation coefficient is, the higher is the interestingness of the analyzed rule.

Definition 3 *Strength Measure is a product of quantities such as Support (Definition 1) and Correlation Coefficient (Definition 2) but with giving priority in the region of rules/assertions with low occurrences but highly correlated with other rules/assertions.*

3 Online fault detection and minimization of the checkers

Online fault detection in digital systems is of paramount importance for reliable operation of the system. An online fault detection mechanism aims to monitor the digital systems at run-time and detect the undesired behavior while the system is in operation. The online fault detection can be achieved with the help of hardware checker infrastructure. The checker infrastructure runs concurrently with the system operation and performs near instant fault detection. In [27, 28], end-to-end error detection is discussed, the use of an end-to-end, epoch-based detecting scheme results in significantly delayed fault detection. Also, any faults that do not cause a functional error at the output (like packet loss, flit drop) will never be detected.

This thesis focuses on the faults in the Network on Chip (NoC) routers. Components inside the NoC router can be divided into two parts, the control part and data path. Faults in the control part are very crucial, hard to be detected. Faults in the control part may lead to flit drop, packet loss, packet mixing, misrouting of packets, deadlock, livelock. Checkers usually perform a simple comparison and so checkers are usually comprised of simple combinational circuits. A trade-off between the fault detection capabilities and the area consumption of the checkers can be eventually outlined. Also, several checkers might overlap with each other in terms of fault coverage. Therefore, there is a need to minimize the checkers to get maximum fault coverage with low area overhead.

Fault detection in the data-path can be achieved through error detection and correction mechanisms. Simple parity checks will detect and may even correct errors affecting the contents of in-flight packets [29]. Careful consideration must be made while implementing the fault detection mechanisms for the data-path against the benefits it can bring. [27, 30] shows that the complex Error Detection and Correction (EDC) schemes may require unacceptably high area overhead. Alternatively, test packets generated by a dedicated hardware structure to detect faults in the data-path require very little area and provides high fault coverage.

This chapter is based on the following publications:

- Behrad Niazmand, Ranganathan Hariharan, Vineeth Govind, Gert Jervan, Thomas Hollstein, and Jaan Raik. Extended checkers for logic-based distributed routing in network-on-chips. In *2014 14th Biennial Baltic Electronic Conference (BEC)*, pages 77–80. IEEE, 2014 [**Publication I**]

- Pietro Saltarelli, Behrad Niazmand, Jaan Raik, Ranganathan Hariharan, Gert Jervan, and Thomas Hollstein. A framework for comprehensive automated evaluation of concurrent online checkers. In *2015 Euromicro Conference on Digital System Design*, pages 288–292. IEEE, 2015 [**Publication II**]

- Pietro Saltarelli, Behrad Niazmand, Ranganathan Hariharan, Jaan Raik, Gert Jervan, and Thomas Hollstein. Automated minimization of concurrent online checkers for network-on-chips. In *2015 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, pages 1–8. IEEE, 2015 [**Publication III**]

- Pietro Saltarelli, Behrad Niazmand, Jaan Raik, Vineeth Govind, Thomas Hollstein, Gert Jervan, and Ranganathan Hariharan. A framework for combining concurrent checking and on-line embedded test for low-latency fault detection in noc routers. In *Proceedings of the 9th International Symposium on Networks-on-Chip*, page 6. ACM, 2015 [**Publication IV**]

3.1 Literature review

Several research works have been carried out in the field of fault detection and fault recovery to increase the fault tolerance and reliability of NoC. With extreme down scaling of the feature size, SoC that have NoC interconnects are becoming more vulnerable to faults during the run-time. Thus, detecting a fault at run-time is rapidly becoming a necessity. Efficient, accurate and quick- responding fault detection is also a prerequisite of successful fault recovery. However, the fault recovery approaches are beyond the scope of this thesis.

Fault tolerance enables the system to function correctly in presence of one or more faults within the system. Usually fault tolerance is achieved with some sort of redundancy techniques and one such technique is Hardware redundancy. Hardware redundancy approaches such as Triple Modular Redundancy (TMR) and Duplication with Comparison (DwC) are helpful in detecting the faults, but along with that it brings a lot of drawbacks such as increase in area, power consumption. Also, the reliability of these redundancy techniques decreases as the lifetime of the system increases [9]. Though these approaches are helpful in detecting the faults, it lacks the information regarding the fault location within the system. An alternative to minimize these overheads is the selective TMR approach that identifies Single Event Upset (SEU) sensitive sub-circuits that are to be protected [31]. Even this approach lacks the ability to localize the faults in the system. On the other hand, coding techniques such as Berger [32] or Bose-Lin [33] along with information redundancy address fault detection. In many works the coding techniques are combined with synthesis [34, 35, 36]. However, these approaches suffer from significant area overhead, and they require alteration of the original circuit to generate the codes. Hardware redundancy is also applied in detecting the faults in NoC. Since NoC may integrate hundreds of switches in a single chip, hardware redundancy imposes a high area overhead.

Concurrent on-line Built-In Self-Test (BIST) techniques such as Built-In Concurrent Self-Test (BICST) [37] and Reduced Observation Width Replication (ROWR) [38] provide high fault coverage at low area overhead, but only consider a limited subset of pre-computed input test vectors. Hence these approaches are likely to miss faults occurring in a normal circuit operation, addressing a limited set of the possible errors. [39, 40, 41, 42] deals with BIST strategies to test NoC architectures. However, BIST solution is not suitable to detect transient faults.

Multiple different alternatives, which do not require changes in the addressed logic, based on concurrent monitors - checkers - were presented. Authors of [43] propose a framework to extract checkers from logical implications derived from the structural description of the considered circuit, which may seem comfortable and is feasible, but suffers low coverage and extremely huge area overhead, often exceeding Double Modular Redundancy (DMR) solutions.

Grecu et al. introduced a method for on-line fault detection and location in NoC communication fabrics [44]. The faults in the communication links and the faults in the NoC switches can be distinguished by the proposed method. The work is based on the utilization of code-disjoint routing elements, combined with parity check encoding for the inter-switch links. Because of design constraints, this method targets the fault detection and location in the data path of the routing unit only. In [45], a hybrid method is presented to synthesize a fault-secure NoC switch employing data encoding at data path (data flits) and concurrent error detection structure for dealing with faults not covered by the flit encoding using multiple parity trees. However, the work still results in more than 50% area overhead.

In [28], SafeNoC, an end-to-end runtime error detection and recovery technique has been introduced to ensure the functional correctness of Chip Multi Processors (CMP)

interconnects. SafeNoC overlay the interconnect with a lightweight and simple checker network. A look-ahead signature is delivered through the checker network for each data packet which is sent through primary network. SafeNoC detects functional errors by computing and comparing the signature of every received data packet with the signature delivered through the checker network. SafeNoC does not provide protection against dropped packets or flits and it cannot recover from errors arising from aliasing of signatures. SafeNoC also does not localize the fault, rather it detects and recovers the interconnects from the fault. The reconstruction and recovery time depend on the severity of the functional error. In case a greater number of packets are affected by the error then SafeNoC can take up to 39M execution cycles to recover the system from the fault. However, the fault recovery approaches are beyond the scope of this thesis. SafeNoC is limited to interconnects only.

Park et al. [46] have examined the impact of transient faults on the reliability of on-chip interconnects and have developed an approach to either protect against or recover from them. For the inter-router link faults, they use Hop-By-Hop (HBH) retransmission method. However, the retransmission buffer can add latency to the system in case of a fault occurrence. Moreover, it is not mentioned whether the retransmission buffer itself is protected against SEUs or not. Regarding the control part of router, an Allocation Comparator (AC) unit is proposed, which provides full error protection to the Virtual Channels (VCs) and Switch Allocation (SA) units at minimal cost, without affecting the router's critical path. Like [28], the approach is limited to interconnects.

Several works have proposed utilization of concurrent checkers for checking faults in the control part of NoC router. In [47], the Inherent Information Redundancy (IIR) in the control path of NoC routers is utilized to manage transient errors, preventing packet loss and misrouting. The goal is to capture faults that might happen in the routing computation unit or in the arbitration unit of NoC router. However, the proposed method is effective only in routers using XY routing. Yu et al. [48] have proposed a set of error detection units for the NoC routing blocks implemented using Logic-Based Distributed Routing (LBDR) for topologies with high-radix (hr). The IIR in LBDRhr logic is exploited to manage transient errors in the routers. LBDRhr provides better scalability compared to routing tables. Despite the advantages, the proposed checkers for LBDRhr logic cannot reach 100% fault coverage. As the number of faulty gates increases, the error detection rate slightly decreases. Thus, the proposed method achieves a high error detection rate in smaller and simple NoCs only. Furthermore, the work only focuses on the routing logic of a NoC router and not considering the full control part.

In [1], the set of checkers introduced in [48] are extended for the baseline LBDR logic to increase the fault coverage (up to 64.9%). LBDR mechanism [18] is a scalable solution for the routing computation unit compared to routing tables. The mechanism describes the topology and the routing function in form of fixed sets of connectivity and routing bits, therefore, the logic can be easily re-configured. The proposed checkers cover most single stuck-at faults occurring in the LBDR circuitry. Fault injection experiments have shown that the proposed method allows increasing the fault coverage 3 times (compared to [48]), of course at the price of 26.8% checker area overhead. However, still 100% fault coverage is not reached, and the area overhead minimization aspect of the checkers is also not addressed.

Alaghi et al. [49] have presented a method based on high level fault model for online error detection and diagnosis of NoC switches. The proposed method deals with routing faults that cause NoC packets to be forwarded to output ports that are not intended to. However, this work targets only functional level fault coverage and does not guarantee a

high coverage for structural faults. The fault model does not detect nor diagnose when the packets are dropped, misrouted or lost-destination.

Parikh et al. [50] have proposed ForEVeR, a solution that complements the use of formal methods and runtime verification to ensure functional correctness in NoCs. Formal verification, due to its scalability limitations, is used to verify the smaller modules, such as individual router components. To protect against escaped design errors with a runtime technique, a network-level error detection and recovery solution, which monitors the traffic in the NoC and protects it against escaped functional bugs that affect the communication paths in the network. To this end, ForEVeR augments the baseline NoC with a lightweight checker network that alerts destination nodes of incoming packets ahead of time. The use of an end-to-end, epoch-based scheme, ForEVeR results in significantly delayed fault detection.

Authors of [51] have proposed NoCAlert, a comprehensive on-line and real-time fault detection mechanism that demonstrates 0% false negatives within the interconnect for the fault model and stimulus set used. It employs a group of lightweight micro-checker modules that collectively implement real-time hardware assertions based on the concept of invariance checking. The checkers operate concurrently with the normal NoC operation and can detect a wide range of faults instantaneously. Low overhead checkers were used to detect faults without the need of periodic or triggered-based testing. The faults that are not covered correspond to non-catastrophic failures. However, the minimization aspect of the area overhead of the checkers is not addressed.

In [52], an online checking mechanism, designed specifically for the switch allocator of a NoC router is proposed. The proposed checkers for the switch allocator of the router have self-checking property. The authors did not mention the faults that can occur at the gate-level structure of the switch allocator; therefore, it is not possible to verify whether the proposed checking mechanism can eliminate all single points of failure which can occur at the gate-level.

The use of embedded test configurations for testing the data-path of NoC routers has been proposed in [53], with design for-testability structures included in [54] and BIST application in [42]. However, all the mentioned approaches are targeting the global network and not a concrete router. Furthermore, only off-line test scenarios have been considered in [53, 54, 42].

3.1.1 Thesis contributions

This thesis proposes a framework for formal qualification of checkers and for minimizing the area overhead with the given fault coverage constraints. The goal is to achieve low-latency, low area overhead checkers with high fault coverage for NoC routers. Different from the above-mentioned approaches, the online fault detection and checker minimization method proposed in this thesis provides the following novelties:

- Formally proving the absence or presence of true misses over all possible valid inputs for a checker, whereas in the case of traditional fault injection only statistical probabilities can be calculated without providing the user with full confidence of fault detection capabilities **[Publication I, Publication II, Publication III]**.
- Targeting the minimum fault detection latency of a single clock-cycle. This is achieved by representing the circuit under test as a pseudo combinational design and concentrating on combinational checkers **[Publication I, Publication II, Publication III]**.
- Providing accurate, automated evaluation for the fault detection characteristics of the checkers. It allows finding cost-efficient trade-offs between the fault detection

capabilities and the required overhead area **[Publication I, Publication II, Publication III]**.

- Combination of concurrent checkers with embedded on-line test packets to enable cost-effective trade-offs between area-overhead and fault coverage **[Publication IV]**.

In the following sections, the methodology for devising, evaluating and minimizing concurrent checkers is discussed. Followed by experimental results regarding the application of framework to the control logic of the NoC router is studied. Finally, a short summary of the chapter is provided.

3.2 Checkers' evaluation and minimization flow

This section focuses on the proposed methodology flow for devising, evaluating and minimizing concurrent checkers. The details regarding the concept of concurrent checkers is already covered in section 2.7.

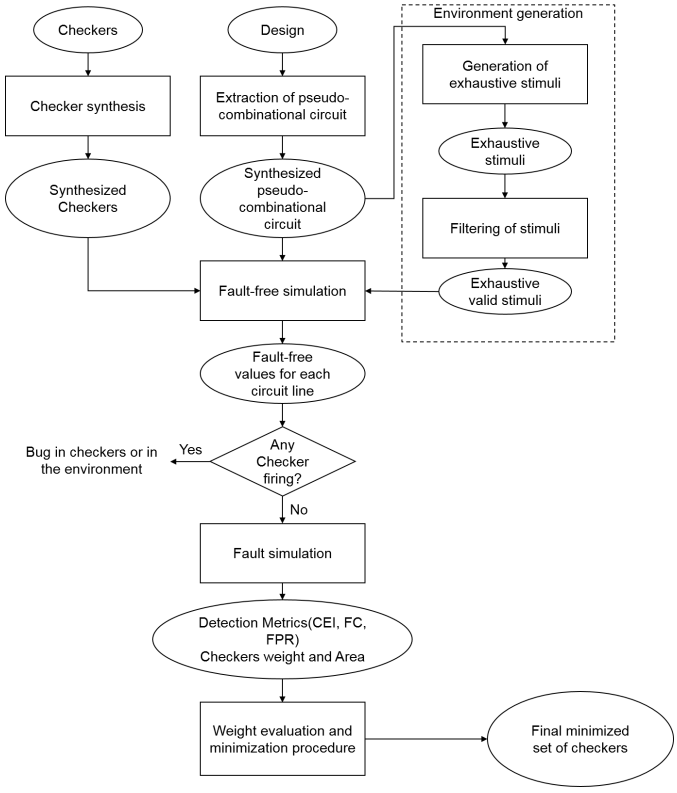


Figure 20 – Checkers Evaluation and Minimization flow

Figure 20 depicts the proposed methodology flow. Initially the flow starts by considering a design under verification. Next, the pseudo-combinational version of the circuit is extracted, and it is synthesized. An initial set of checkers are devised from a set of combinational assertions. Additional checkers are also added to describe the relations on pseudo-primary inputs/outputs to the checker suite to increase the fault coverage. The initial set of checkers include set of structural as well as functional checkers which are then synthesized to be used in fault simulation. Next, the checker evaluation environment is

created during the environment generation step by generating an exhaustive valid set of input stimuli. A fault free simulation is performed to verify the correctness of environment and checkers. Once the correction is made, using the bug-free checkers, the checkers evaluation is performed to measure the fault detection capability of checkers using the metrics discussed in section 2.8. In addition to fault detection capability estimation, the checkers weight information i.e., the number of true detections and its area consumption is also calculated. Using a *greedy heuristics*, a minimization procedure is carried out by considering the fault detection capability, weight, area of a checker as an input. The final step of the methodology flow is the minimized set of checkers with high fault detection capability and low area consumption.

Traditionally, to evaluate the fault detection capability of the checkers, fault injection has been applied. Fault injection refers to injecting fault into a circuit at a certain time step and simulating it with the input stimuli to see whether any functional output of the circuit changes and whether any of the checkers detect the fault. Since it is generally impossible to inject and simulate all the faults at each circuit line at each time step, a statistically significant sample of random faults would normally be injected and simulated. This in turn would introduce the risk to evaluate the checkers in an incomplete scenario, providing superficial and probably misleading results concerning fault coverage.

The proposed methodology flow requires the extraction of a *pseudo-combinational* version of the design i.e., a version of the circuit where the feedback loops of the sequential circuits are broken. The proposed methodology

- is complete, i.e. it allows proving the absence or presence of true misses.
- provides minimal detection latency, because a pseudo-combinational version of the circuit is extracted, thus single snapshots of time corresponding to single clock cycles are considered.
- allows minimization of the initial devised set of checkers for the considered design, based on the weights output information provided by fault simulator tool, opening prospective of seamless trade-offs in between fault detection coverage and area overhead.

The *completeness* of the approach is a key feature: pseudo-combinational extraction guarantees the possibility to fault simulate the circuit in an exhaustive valid range of conditions, overcoming the feasibility issues of traditional fault injection approaches. Moreover, considering the pseudo-combinational version of the design provides *single clock cycle latency* detection capabilities for the checkers, since single instances of time corresponding to a clock cycle are considered. Furthermore, analysis of sequential circuits, with temporal checkers would not be feasible because of the combinatorial explosion of considering all possible input sequence combinations.

3.2.1 Extraction of pseudo-combinational version of the circuit

Extraction of pseudo-combinational version of the design is achieved by breaking the sequential elements of the design such as Flip-flops, Memory(registers). The corresponding input and output connected to the sequential elements are fed to the overall design as *pseudo-primary inputs/outputs* as shown in figure 21. Figure 21a shows the digital design with combinational and sequential circuits with primary inputs and outputs whereas figure 21b shows the pseudo-combinational equivalent circuit, which has additional pseudo-inputs and outputs in addition to primary inputs and outputs. In the pseudo-combinational circuit, the current state signals are converted to pseudo-primary inputs and next state

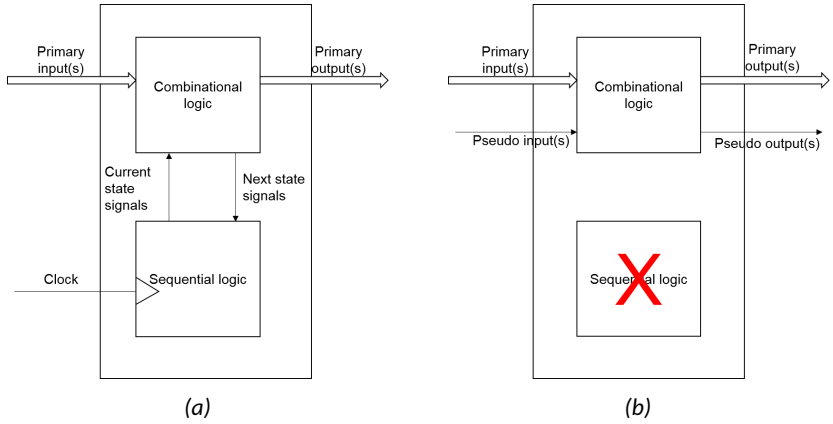


Figure 21 – (a) Design with combinational and sequential circuit (b) Equivalent pseudo-combinational circuit

signals are converted to pseudo-primary outputs. This way it is possible to take the logic to some desired situation in the considered single clock cycle and also feeding the values of the registers through input vectors.

It has to be noted that even though the extraction of pseudo-combinational circuit leads to creation of additional inputs/outputs, at the end of the proposed flow, the checkers are integrated in the original design and therefore, the final structure of the DUV is not altered. Devising concurrent checker is discussed in the following subsection.

3.2.2 Synthesizing the checkers

Checkers monitor the inputs and outputs of the considered design, and they evaluate if the values of the output do not match those expected from the values of the input. The flow begins with the synthesis of an initial set of devised checkers for the considered design, starting from a set of combinational assertions.

It is possible to outline a set of implications as relationships in between output and inputs by studying a digital circuit and based on these, assertions can be extracted, which are later mapped to the checker logic. The specification of the considered design is used while devising the checkers. The checkers are not automatically devised, and it need the expertise of a verification engineer. Most of times checker logic is based on relationships existing in between values of the inputs and the outputs, of the considered logic, but in some cases, especially when concerning encoding of the information, they could monitor only the outputs. Some examples of functionality-related violations, on which checker logic could be based, can be easily provided:

- empty and full signals of a buffer cannot be high at the same time.
- if a meaningful packet reaches a routing unit in a switch, at least one output direction must be selected by the unit.

The devised checkers might be redundant in terms of fault coverage metrics. It is worth noting that, in this step, it could be very challenging to spot any overlap or inconsistence of some considered checkers, or their effectiveness, and this is the main reason why it is fundamental to correctly evaluate the checker logic. After performing the evaluation and minimization flow using the proposed methodology, the checkers which satisfy the area

constraints are kept while ensuring the target fault coverage. It is also noteworthy that this section focus on devising checkers for control logic of NoC routers, it is assumed that the data path is already protected by Error Detecting/Correcting code (EDC/ECC) [55, 36] techniques, which can alert the system whenever there is any unexpected changes in the data packets.

3.2.3 Environment generation for checkers' evaluation

This section focuses on the generation of environment required to perform fault simulation for checkers evaluation as shown in figure 20. Once the initial set of checkers for the DUV is devised and the pseudo-combinational version of the circuit is extracted, a set of input test vectors is needed in order to perform fault simulation for checkers evaluation.

In the traditional verification of the circuit, an exhaustive set of input test patterns are considered which would guarantee that the design is fully evaluated. On the contrary, for checkers evaluation this would lead to both erroneous results and useless computational effort. The behavior of the checkers strongly depend on the applied input patterns. If incorrect patterns are applied to the checkers logic, the checker would detect fault even though there is no actual fault causing fallacious results as an outcome of the experiment. At the same time, performing a fault simulation would cause additional run-time. Therefore, it is imperative to identify a set of valid and meaningful input patterns to be used during fault simulation for checkers evaluation.

For exhaustive set of input patterns, 2^n input patterns are considered initially where n is the overall number of inputs (functional and pseudo-primary) of the considered pseudo-combinational version of the design. Then a set of filtering *constraints* is devised based on the functional behavior of the considered DUV and those input vectors which do not satisfy those conditions are discarded from the initial set of exhaustive input patterns. This step is fundamental to consider only those patterns corresponding to correct and meaningful behavior of the logic, so that checkers can be evaluated only in realistic conditions. It is important to avoid those patterns that would cause the checker logic to fire in a fault-free situation.

3.2.4 Fault-free simulation and debugging checkers/environment

To perform fault simulation for checkers evaluation, first a fault-free simulation is performed with valid set of input stimuli obtained from previous step (see section 3.2.3) along with the synthesized checkers and pseudo-combinational version of the circuit. This allows to find bugs either in the devised set of checkers or in the simulation environment generated. If the output of the simulation points out that a checker is firing even though no fault is injected, this suggests the presence of the either one of the following bugs:

- The checker is not correctly implemented.
- The input vector for which a checker detects fault are not valid and have to be discarded.

The fault-free simulation results allow to spot and fix single or multiple bugs in the checker and environment before starting the actual fault simulation for checkers' evaluation. It is to be noted the fault-free simulation results are not related to the evaluation of the detection capability of the checkers.

3.2.5 Fault simulation based evaluation of checkers

The evaluation of a checker is performed using a fault simulator tool developed as an extension of a freeware test system Turbo Tester [56]. Turbo Tester (TT), which is a diag-

nostic software package that contains a variety of tools related to testing and diagnosis of integrated circuits. TT can read the schematic entries of various EDA tools and produce a representation of the circuit in terms of Structurally Synthesized Binary Decision Diagram (SSBDD) [57]. TT tool suite consists of the following set of related tools for performing fault simulation

- Test generation by different algorithms
- Test program optimization
- Fault simulation for combinational and sequential circuits
- Fault diagnosis

The baseline fault simulator has been extended to deal with the presence of concurrent checkers (described in section 2.7), rather than only with the functional design, producing additional output information on the online detection effectiveness of the checker logic. This way the injection of faults can be applied only to the functional design, thus allowing to evaluate the detection capability of the introduced checker logic. The main differences between the extended and baseline fault simulator are

- no parallelization of test patterns is considered during fault simulation, to avoid memory issues and grant the possibility to evaluate large designs.
- no fault dropping, every fault is simulated with every input vector.
- concurrent fault simulation is considered.

To evaluate the fault detection capabilities of concurrent checkers, fault dropping cannot be considered. In a traditional fault simulation, a fault once detected may be optionally dropped from the list of active faults. This is intrinsically meaningless while studying checker logic, because the behavior of each checker is strictly dependent on the considered test pattern, i.e. a checker could flag a fault with a certain test vector, while missing it with a different one.

The metrics described in section 2.8 are expressed in more generic ways which cannot distinguish the fault model considered. Since, single stuck-at fault model is considered for fault simulation, it is assumed that only one net of the circuit can have a fault at a time. This consideration led to the introduction of new paradigm of statistics to describe fault detection capabilities of concurrent checker logic, including a reformulation of fault coverage, described in section 2.8. The possible detection outcomes after the injection of a fault, described in table 5 were extended to seven symbols, to be used in the computation and evaluation of the output of the fault simulation, listed as follows:

- 0 - stuck-at-zero fault is detected by the circuit and by the checkers
- 1 - stuck-at-one fault is detected by the circuit and by the checkers
- w - stuck-at-zero fault is detected by the circuit and not by the checkers
- W - stuck-at-one fault is detected by the circuit and not by the checkers
- o - stuck-at-zero fault is not detected by circuit and is detected by the checkers
- i - stuck-at-one fault is not detected by circuit and is detected by the checkers

Symbols	*Detection Outcomes*
0	True Detection
1	True Detection
w	True Miss
W	True Miss
o	False Positive
i	False Positive
X	Benign Miss

Table 6 – Symbols and detection outcome correspondence

- X - none of the stuck-at-faults is detected, nor by the circuit, neither by the checkers

Table 6 represents the correspondence in between the alphabet of symbols and the previously introduced classification of possible outcomes.

Based on the introduced alphabet of symbols, the newly devised metrics for coverage - CEI, FC and FPR defined in section 2.8 can be rewritten as follows

$$CEI = \frac{\sum[0,1]}{\sum[0,1,w,W]} \tag{6}$$

$$FC = \frac{\sum[0,1,X]}{\sum[0,1,w,W,X]} \tag{7}$$

$$FPR = \frac{\sum[o,i]}{\sum[o,i,X]} \tag{8}$$

where each symbol represents the occurrences of the symbol itself, i.e. of the corresponding situation.

In addition to the newly devised statistics CEI, FC and FPR, the tool also generates some useful output information.

- **Fault table** presents one row for each input vector and one column for each node in the SSBDD representing the considered design. For the corresponding input pattern, the detection outcome for each node where faults are injected are marked with any one of the previously introduced alphabet of symbols extensively.

- **Nodes' detection information** is extracted from the fault table, for those nodes where faults are injected, while the remaining nodes are simply listed. For any node the number of occurrences of the introduced alphabet is reported, omitting zero entries. This information allows to spot those nodes which eventually present a large amount of true misses (w and W occurrences), the worst situations in which checkers do not detect critical faults, thus suggesting where to act in order to increase the detection effectiveness of the checking logic.

- **Checkers' detection information** portrays the capabilities of each checker in detecting faults, in form of a table, with a row for each checker and a double column for each node. Each double column gives information for stuck-at-0 fault on the left side and stuck-at-1 fault on the right side, for the corresponding node, if faults are injected in that node. Each intersection between a row and a double column provides the numbers of detection for stuck-at-0 and stuck-at-1 faults, injected in the node corresponding to the column, provided by the checker corresponding to

the row. This table is related to the fault table, indeed, for instance, the sum of stuck-at-0 fault detections provided by the whole set of checkers, reading the left side of a double column, is equal to the occurrences of 0 symbols in the column of the fault table corresponding to the considered node. Checkers detection table could also be used to spot eventual overlapping or independence in the action of different checkers.

- **Checkers' detection absolute weights** is evaluated for each checker, as the number of provided detections over the considered set of stimuli, giving a first hint on the capabilities of detection of the checker itself. On one hand it would be wrong to suppose a checker better than another one only considering the number of provided detections, because a checker firing for a limited number of faults could be the only one detecting those faults. On the other hand this information may suggest some optimization work, based on a heuristic approach, trying to use a limited set of checkers derived from the whole. For instance it could be interesting to start using at first only the checker providing the highest number of detections, and gradually increasing the size of the used set of checkers, keeping trace of the evolution of both coverage information and area overhead.

3.2.6 Checkers' evaluation and minimization

All the checkers are assumed to be functionally verified. The checker evaluation is performed based on the metrics described in section 2.8. The goal is to reach 100% or another target value for both CEI and FC. A 100% CEI would mean there were no True Misses during the fault simulation and thus checkers are able to capture all Single Event Transient (SET) faults at different location in the design. At this point, each checker is weighted according to the number of true detections accomplished on the considered input test vectors set, and this information can be used as starting point for some algorithm of minimization of devised set of checkers. Once a minimized set of checkers is produced, the described flow can be rerun, to display the eventual loss of coverage due to the reduction of the considered set of checkers, through the value of the introduced metrics.

In the proposed methodology, the minimization procedure is a greedy weight-based heuristic procedure as shown in Algorithm 1. First, it selects the heaviest checker, i.e., the checker with the highest number of detections in the considered environment. Then the second heaviest checker is included in the set and so on. At each step, coverage metrics and area consumption are evaluated, expected to reach the target value for coverage metrics. Based on the results at each step, coverage metrics and area overhead, trade-offs in between them can be eventually outlined. The final output is the optimized set of checkers, supposedly matching coverage and area overhead target values.

Algorithm 1 Checker minimization

```
initialization; // Evaluates individual checkers and stores them in a list with number of true
detections
sorted_checkers = sort_checker_based_on_weight()
minimized_set = []
for checker in sorted_checkers do
    temp_set = minimized_set + checker
    if (check_area(temp_set) ≤ target_area) then
        CEI = calculate_CEI(temp_set)
        minimized_set.append(checker)
        if (CEI ≥ target_CEI) then
            break
        end
    end
end
return minimized_set
```

A greedy heuristic (described in Algorithm 1) is proposed for finding minimized set of checkers with low-area high fault coverage. The algorithm sorts the checkers based on the weight (number of True Detection). Area of the chosen checker along with the temporary set of checkers is calculated to check whether it violates the target area constraint. If there isn't any area violation, then CEI of the chosen checker along with the temporary set of checkers is calculated. If the CEI of the new set is providing any improvement to the temporary set, then the checker would be added to the final minimized set. If the chosen checker either violates area constraint or does not improve the CEI then it is discarded. Once the CEI reaches 100% or target coverage, the process terminates.

3.3 Embedded online test packets

In order to exploit the strong reduction of area overhead due to the removal of data-path checkers, while at the same time ensuring high coverage of faults, a hybrid solution could be considered, introducing what is proposed in [53, 54, 42], i.e. the use of online embedded test packets. The checkers can be complemented by embedded online test packets which are to be applied as a periodic routine during the idle periods in router operation.

In the case of the control part of a NoC router, where embedded test packets based approaches have proven inefficient [54], low area concurrent checkers could be applied, as described in previous section. Differently, according to[53, 54, 42], the embedded test yields full fault coverage in data-path modules, whereas error correcting codes would be more expensive in terms of area consumption. The embedded test packets-based approaches propose that, whenever a router in a mesh-like NoC is in idle condition for a certain temporal window, neighbor routers can send to it test patterns, in order to detect eventual faults in its data-path. Neighbor routers have the duty both to inject packets and to check the outputs of the router under test.

The functional fault model that is applied to cover the stuck-at faults in the data-path of the NoC router is based on the idea proposed for functional testing of mesh-like NoC networks in [53, 54, 42]. The three test configurations considered to cover the entire mesh-like NoC router is shown in figure 22 and they include *Straight paths*, *Turning paths* and *Processing element connections*. A configuration is set up by adjusting the corresponding destination address fields of the transmitted packets to the last row (column) of the

network. A fault model proposed in [53, 54, 42] is applied, where the value at a selected router input is distinguished from the values at other inputs of the router. To fully cover the structural faults in the multiplexers of the crossbar, tests for each address value must be performed. An additional constraint is that all turns must be covered by the distinguishing tests.

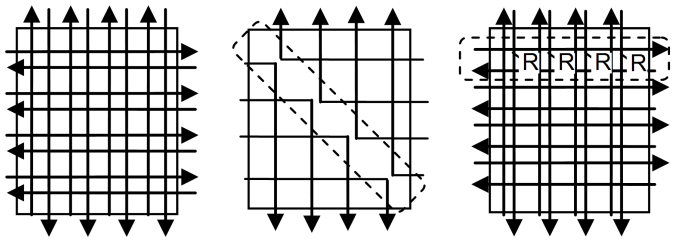


Figure 22 – Test configurations for mesh-like NoCs [54]

Straight path configuration will be set up by letting the packets pass straight through the network matrix. This will cover the faults in the straight links of the network. A constraint is that each bit in the data bus must be traversed with a 0 and a 1 (i.e. toggle coverage must be 100%). Additionally, vertical and horizontally sent data must be distinguished from each other. This is necessary to cover faults in multiplexer addressing of the crossbar switch. In turning paths configuration, based on the deterministic XY routing implemented in the switches, the packets will be sent by the X axis of the network and will meet at a diagonal of the switch. Here, the diagonal will be shifted over the entire network matrix until all the switches have been covered. Processing element configuration is needed to cover the links to resources. This configuration can be achieved by providing a loop-back between the resource and links.

The embedded online test packets will be applied whenever there are idle periods or slacks in scheduling with length K for the send/receive resources, K test patterns will be applied from them. This will be done periodically fetching K next tests from the test set in a circular manner, i.e. if the end of the test is reached then it starts again from the beginning. This scenario provides online test capabilities for regularly checking the health of the data-path of the router.

3.4 Experimental results

In this section, the experiments ran on the target NoC router architecture explained in section 2.6 using the methodology flow introduced in section 3.2 were discussed. First, the evaluation and minimization flow based on the extraction of pseudo-combinational version of the considered design, was applied to stand-alone East input port LBDR module as shown in figure 13. Next, the experiment for the pseudo-combinational version of the control part of the NoC router specifically East input port LBDR module connected with South output port Arbiter module is discussed. In the end, the evaluation of checkers was ran on the entire router which includes all the modules introduced in section 2.6.

3.4.1 ELBDR experiment

Table 1 describes the initial devised set of checkers for the routing computation unit corresponding to the East input port. Based on the pseudo-combinational version of ELBDR design shown in figure 13, it has 11 input bits:

- 2 flit type bits

- 4 destination address bits
- 4 previous output port values bits
- 1 empty bit coming from the corresponding input FIFO buffer

It is worth noting that there is no East output port for ELBDR due to no 180° turn restriction. Also, when the routing computation logic has been integrated in the whole router, the number of bits for the flit type field was extended to three, to implement one-hot encoding.

The exhaustive set of input stimuli represent 2^{11} = 2048 vectors. A filtering scheme based on the following constraints was devised to extract the valid set of input patterns for fault simulation:

- If Empty signal coming from the input buffer is set high, then the rest of the input bits are ineffective and therefore any value is allowed.
- If the incoming flit is a header flit, then the destination address must be valid according to the XY routing and turn restrictions.
- If the incoming flit is a body or tail flit, the previous output values must be valid, i.e., only one output direction must be set high according to XY routing.

Based on the above constraints, valid set of stimuli consisting of 1536 vectors which is around 75% of the exhaustive set of stimuli is extracted through filtering scheme.

Once both the initial set of checkers and the valid set of input stimuli were available, fault free simulation is performed to find any bugs in the devised checkers or in the verification environment before starting the actual fault simulation. During the fault simulation, the initial set of checkers guaranteed full coverage and the results of the evaluation process in terms of true detection weights are reported in figure 23.

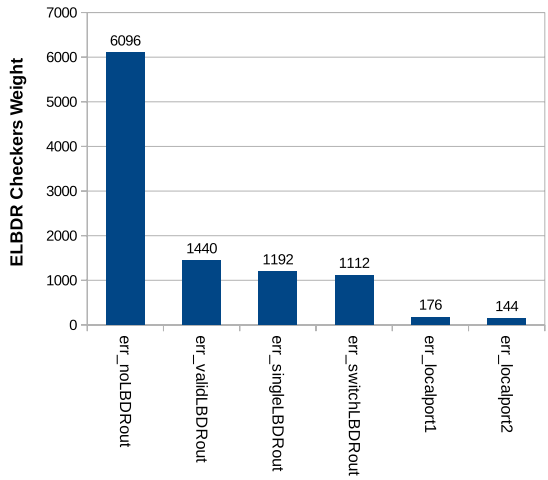


Figure 23 – Weights of checkers proposed for ELBDR

Based on true detection weight information, the optimization process has been executed, and by greedy heuristics minimization procedure, the minimized set of checkers was obtained. The heaviest checker (*err_noLBDRout*) is considered as the starting point, then the following heaviest checkers were added one by one, evaluating coverage metrics and area overhead for each different set of checkers. Results are shown in figure 24, reporting

CEI, *FC* and *area overhead* increase at each step. Full coverage (i.e., 100%) is achieved when the first three heaviest checkers are considered, resulting in the minimized set of checkers. The three selected checkers dominate the others present in the initial set, i.e., they cover all the faults that the discarded checkers would cover. The area overhead with the minimized set of checkers has been reduced to 78.57% over the ELBDR circuit which is far lower than 185.71% imposed by the initial set of checkers. No false positives (FPR is zero) were encountered during this experiment.

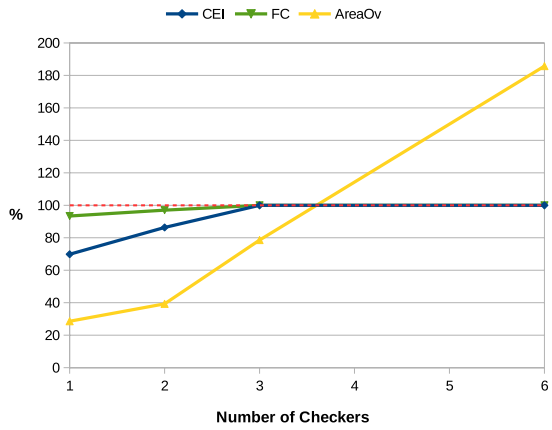


Figure 24 – ELBDR checkers optimization results

3.4.2 ELBDR and SARBITER experiment

The design considered for the second experiment is shown in figure 25. East input port routing computation logic which is evaluated in the previous section 3.4.1 is connected to South output port arbitration unit and they are evaluated together. The initial set of checkers for the arbiter logic is introduced previously in the table 2. The overall design presents 19 input bits, 11 bits for ELBDR and additional 8 bits for SARBITER.

- 3 inputs requests bits (from North, West and Local input ports, East input port is provided by ELBDR)
- 5 previous state bits which are one-hot encoded

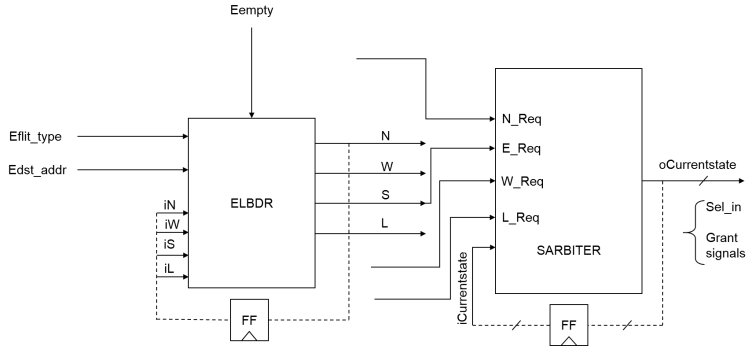


Figure 25 – ELBDR + SARBITER experiment

Due to routing restrictions (no 180° turn), there is no request from South input port and therefore, grant cannot be given to South direction. It is noteworthy to stress that, the set of checkers devised for two modules, routing computation and arbitration units are independent to each other, i.e., they cover faults for different and separate parts of the circuit without any overlap. Therefore, the minimized set of checkers obtained from the previous experiment described in section 3.4.1 is considered for the routing computation unit and the focus is on optimizing the initial set of 28 checkers devised for the arbitration unit.

The exhaustive set of input patterns considered for the pseudo-combinational circuit would be 2^{19} = 524288 input stimuli. The filtering scheme used will be an extension of the one used for ELBDR experiment along with one-hot encoding restraint for the 5 previous state value bits of the arbitration unit. After applying the filtering constraint, the exhaustive set of input patterns has been reduced to 61440 valid set of input patterns which is around 12% of the initial set.

Once both the initial set of checkers and the valid set of input stimuli were available, fault free simulation is performed to find any bugs in the devised checkers or in the verification environment before starting the actual fault simulation. During the fault simulation, the initial set of checkers guaranteed full coverage and the results of the evaluation process in terms of true detection weights are reported in figure 26. From the figure 26 it can be clearly noticed that the two checkers which are both related to one-hot encoding of the state variable of the arbiter logic are effective in detecting single SAFs.

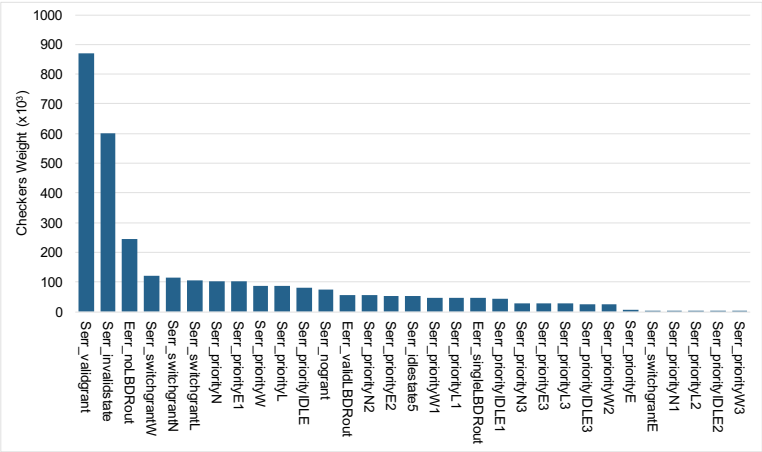


Figure 26 – Weights of checkers proposed for ELBDR+SARBITER scenario

Figure 27 represents the results of the optimization process on the ELBDR + SARBITER scenario, considering the minimized set of checkers of the routing computation logic obtained from the previous section 3.4.1, the greedy heuristics is applied on the set of checkers of the arbitration unit. The two heaviest checkers from the figure 26, clearly dominate all the other checkers for the arbitration unit, ensuring full coverage (i.e., 100%). Thus, the area overhead of a total 3 ELBDR and 2 SARBITER checkers over the considered partial control path circuit is limited to 56.82%. While the initial set of 28 checkers for the SARBITER would lead to 170.45% area overhead. It is interesting to observe that the minimized set of 5 checkers corresponds to one third of the whole 31 checkers considered for ELBDR + SARBITER scenario. No false positives (FPR is zero) were encountered during this experiment.

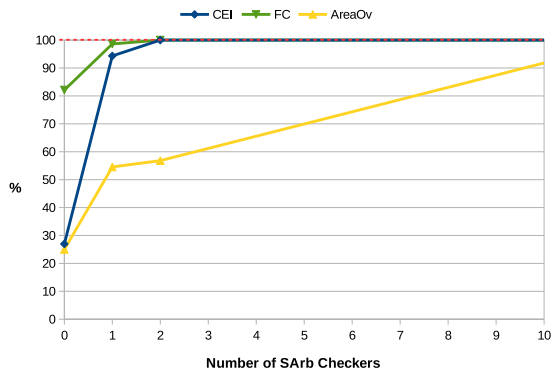


Figure 27 – SARBITER checkers optimization results for ELBDR + SARBITER scenario

3.4.2.1 Importance of the independence of checkers As previously mentioned, the set of checkers devised for ELBDR and SARBITER modules are independent to each other, i.e., they cover faults for different and separate parts of the circuit, without any overlap. Therefore, 100% fault coverage for SARBITER achieved with the two heaviest checkers shown in figure 26 does not mean that they have also covered all the faults occurring in ELBDR.

Table 7 illustrates the importance of considering the independence information for the sets of checkers of the two different modules. As it can be observed that the weights of the ELBDR checkers are far less than those of the SARBITER, but they are still needed to achieve full coverage for the considered design.

Checker	*Weight*
Serr_validgrant	871552
Serr_invalidstate	600512
Eerr_noLBDRout	243840
Eerr_validLBDRout	57600
Eerr_singleLBDRout	47680

Table 7 – Weights of the minimized set of 5 checkers for ELBDR + SARBITER scenario

Figure 28 shows the inefficiency of the greedy-heuristic approach due to the lack of the independence information. The number of steps in the greedy-heuristics procedure is heavily increased, before reaching the target (i.e., 100%) upper bound for CEI and FC. 19 steps are needed, and full coverage is reached only when Eerr_singleLBDRout checker is considered. However, when partitioning the fault set to different parts of the design has been considered, the overall minimization procedure requires only 5 steps.

3.5 Experiments on the whole router

Evaluation of the whole NoC router shown in figure 10 is carried out by considering different values of data-path bit-width: 10, 32, 64, 128, 256. Three different evaluation experiments are carried out for the whole NoC router.

- Considering all the checkers devised for the different modules of the router, addressing both control part and data-path
- Considering the checkers devised for the control part

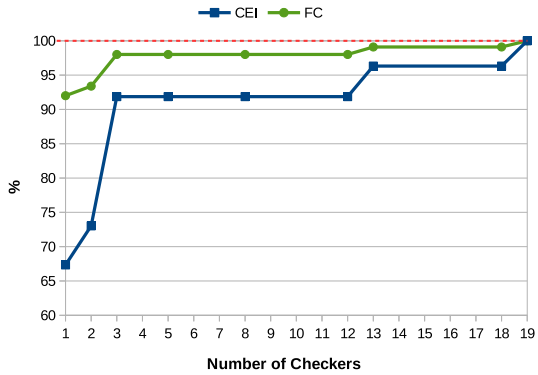


Figure 28 – Results without considering independent set of checkers for ELBDR + SARBITER scenario

- Hybrid solution which includes the introduction of online embedded test packets replacing data-path checkers along with control part checkers.

3.5.1 Experiment considering the overall set of checkers

In this experiment, all checkers previously introduced in section 2.6 are considered

- LBDR Checkers
- RR arbiter Checkers
- FIFO buffer control part Checkers
- Cross bar switch DMR
- Even parity Checkers
- Checkers for infrastructure of the router

It is worth mentioning that for routing computation and arbitration logic, the minimized set of checkers obtained through previous experiments (described in sections 3.4.1, 3.4.2) were considered. Also, these minimized set of checkers has to be adapted for each port of the router.

Table 8 lists the results for the considered scenario, comprising the evaluation metrics and run time. It can be noticed that *CEI* and *FC* slightly increase with data-path bit-width. Also, FPR decreases, suggesting that the majority of false positive occurs in the control part of the NoC router. Simulation time noticeably increases when the bit-width is increased, as the number of possible locations for the injection of faults becomes wider.

bit-width	CEI (%)	FC (%)	FPR (%)	sim_time (s)
10	99.10	99.71	16.91	64.16
32	99.62	99.67	8.87	233.4
64	99.79	99.92	5.23	671.5
128	99.89	99.96	2.84	2196.9
256	99.94	99.98	1.49	8081.2

Table 8 – Evaluation result for the whole NoC Router

Figure 29 displays the area information for considered NoC router design for different data widths along with the checkers logic for different modules of NoC router. Area consumption is provided in terms of number of NAND2 gates of the considered synthesis library. As seen from the figure 29, the control part checkers present constant area with the increase of the data-path bit-width, while the area consumption of the data-path checkers (XBAR DMR and parity) grows proportionally to the router size. Table 9 shows the numerical information for the area consumption of the considered NoC router for different data widths along with the checkers logic for different modules of NoC router. The table 9 also contains the area overhead increase in percentage.

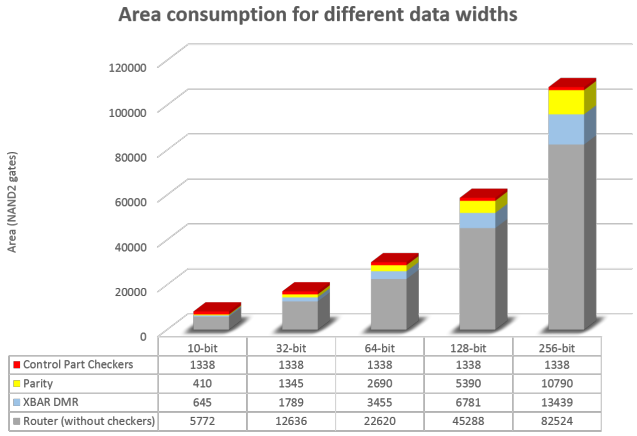


Figure 29 – Area information for different bit-widths of NoC router with checker logic

	$bit-width$				
	10	32	64	128	256
Area (NAND2)					
Router	5772	12636	22620	42588	82524
Checker Logic	2393	4472	7483	13509	25567
Control Part checkers	1338	1338	1338	1338	1338
Data-Path checkers	1055	3134	6145	12171	24229
XBAR DMR Checker	645	1789	3455	6781	13439
Parity Checkers	410	1345	2690	5390	10790
Area Overhead (%)					
All checkers	41.45	35.39	33.08	31.72	30.98
Control part Checkers only	23.18	10.59	5.92	3.15	1.62

Table 9 – Area information for different bit-widths of NoC router with checker logic

3.5.2 Experiment considering the control part checkers only

Table 10 shows the evaluation metrics of the checkers considered only for the control part of the NoC router. It can be observed that the coverage value significantly drops, stressing the effectiveness of the data-path checkers. However, if the fault injection is limited only to control part of the router, both CEI and FC results in 100% value. Interestingly, FPR values reported in table 10 are really close to the ones reported in table 8 from the previous

experiment, showing that the most of the false positive occurrences are related to the control part of the NoC router.

bit-width	CEI (%)	FC (%)	FPR (%)	sim_time (s)
10	60.51	87.17	15.54	47.9
32	25.99	74.87	7.78	152.6
64	14.31	70.66	4.49	408.5
128	7.47	67.92	2.44	1300.8
256	3.84	66.62	1.27	4629.8
10 (No fault injection in Data-path)	99.68	99.87	21.18	22.3

Table 10 – Evaluation result for the whole NoC Router (control part checkers only)

3.5.3 Experiment considering the hybrid solution

Based on area consumption information shown in table 9 and evaluation metrics shown in table 8 and 10, coverage is dramatically reduced when data-path checkers are removed, also area consumption is significantly dropped, ranging in between 23.18% and 1.62% depending on the different data widths. To exploit the strong reduction of area-overhead due to the removal of data-path checkers while at the same time ensuring high fault coverage, a hybrid solution can be considered, i.e., the use of online embedded test packets proposed in [53, 54, 42] combining with control part checkers. Using this hybrid solution, full fault coverage for the NoC router can be achieved with a minor area overhead. As it has been shown by experiments in [42] an embedded test of length K=196 clock cycles will achieve FC=100% within the NoC router data path.

3.6 Chapter summary

In this chapter, a methodology is proposed to evaluate the fault detection capabilities of the concurrent checkers devised for the considered design and to minimize the set of checkers to ensure high fault coverage with low area overhead. Initially, a set of checkers has been devised for different modules of NoC router mentioned in the section 2.6. Then a pseudo-combinational version of the design is extracted to ensure the completeness of the approach. The pseudo-combinational extraction guarantees the possibility of fault simulating the circuit in an exhaustive valid range of conditions, overcoming the feasibility issues of traditional fault injection approaches.

New metrics have been introduced to evaluate the fault detection capabilities of the checkers devised. A greedy heuristics-based minimization procedure was introduced to derive the minimized set of checkers, outlining the eventual trade-off between the target fault coverage and area overhead. It was also described how to deal with the clustering of different units into the whole router, that led to minor changes in some part of the checker logic (see table 4), as some of the faults cannot be spotted while studying each logic as a standalone module.

Experiments were run, applying the proposed methodology flow, at first, to the standalone modules and then to the whole router. It is worth mentioning that value of coverage metrics tracing 100% upper bound where achieved in almost every experiment with limited area overhead, but decreasing as wider bit-widths were considered in the data-path. Finally, a hybrid approach based on the embedded online test packets for the data-path

along with the control part checkers was considered, to ensure full coverage while at the same time ensuring limited area overhead for the data-path.

4 Linking verification assertions and concurrent hardware checkers

The underlying hypothesis of the checker minimization approach proposed in this thesis was to move to the higher abstraction level (behavioral) to gain productivity and scalability. For this purpose, it was required that the high-level fault model would correlate with the low-level (structural) fault model. A fault model is a representation of one or more faults in the design for which the test generation process must generate test vectors. The results obtained from test generation and fault simulation - fault coverage, reflects the quality of test vectors with respect of the underlying fault models. On the other hand, designing a hardware checker infrastructure is a manual and error-prone work. Even for verification experts, it can be extremely time-consuming to define the required hardware checkers. A possible solution to automate the synthesis of concurrent error checkers is to derive them from verification assertions. Moreover, creating checkers automatically based on logic implications derived from the circuit structure [58] is feasible but suffers from low fault coverage and high area overhead, often exceeding the duplex solutions. However, deriving checkers from functional assertions, or reusing verification assertions, is similarly known to yield low coverage of structural faults as it is difficult to correlate functional coverage to structural one [59]. Thus, in this chapter first, we describe the results of our investigation in correlation of assertion and hardware checkers in detecting faults, second a method to translate liveness assertions to safety assertions in order to reuse them as hardware checkers.
Typically, two different types of assertions must be proved essentially to prove the correctness of the design. They are safety and liveness assertions. A safety assertion stipulates that 'undesired things' do not happen during execution of a program and a liveness assertion stipulates that 'desired things' do happen eventually [60]. Design specifications can be described as set of assertions and most of them can be more naturally formulated as liveness assertions [61, 62]. The automatically generated assertions that are derived from systems behavior are almost exclusively liveness assertions [63] [64]. Some of the assertions generated can be invalid and must be removed. On the other hand, in practical applications, safety assertions are prevalent [61, 62] and also, verification using liveness assertion is known to be significantly less scalable [65]. Therefore, a translation procedure is required to translate the valid liveness assertions to safety assertions. Generating checkers from verification assertion is a solved problem where commercial tools are available (for example IBM FoCs [66]).

In this thesis, the linking between the verification assertions and the hardware checkers is accomplished by the following steps:

- Correlation between behavioral fault model and structural (gate-level) fault model
- Translation of liveness assertions to safety assertions
- Conversion of safety assertions to safety checkers

Correlation of fault detection capability of high-quality assertions versus the fault detection capability of the synthesized checkers is studied. A translation scheme is proposed that translates liveness assertions into safety assertions. Finally, the safety assertions are synthesized into hardware checkers which in turn can be used for providing a cost-effective checker infrastructure.

This chapter is based on the following publications:

- Ranganathan Hariharan, Behrad Niazmand, and Jaan Raik. On fault detection efficiency of reliability checkers obtained by verification assertion qualification. In *RESCUE 2017 Workshop on Reliability, Security and Quality European Test Symposium (ETS) Fringe Workshop, May 25-26*. IEEE, 2017 [**Publication V**]

- Ranganathan Hariharan, Tara Ghasempouri, Behrad Niazmand, and Jaan Raik. From rtl liveness assertions to cost-effective hardware checkers. In *2018 Conference on Design of Circuits and Integrated Systems (DCIS)*, pages 1–6. IEEE, 2018 [**Publication VI**]

4.1 Literature review

Several studies have been carried out to show the correlation between high-level and low-level fault models.

In [67], the authors have proposed a Register Transfer (RT)-level single-bit stuck-at fault model which is correlated with Gate-level fault coverage. A set of rules which identifies the redundancies removed by synthesis are used to compute a fault list that exhibits good correlation with stuck-at faults. The degree of correlation is measured as the percentage of gate-level stuck-at faults that are detected by fault simulating the test sequences generated at RT-level.

Brera et al [68] have proposed a methodology to improve the behavioral fault model by including information concerning the synthesis flow. The improved behavioral fault model which is having a high correlation with the structural fault model coupled with an efficient behavioral test generator able to achieve good gate-level fault coverage. Compared to fault coverage at gate-level obtained by a commercial test pattern generator, the behavioral test generator able to obtain a higher fault coverage at gate-level using the improved behavioral fault model.

In [69], a high-level fault models referred to as behavioral fault models has been proposed for complex combinational digital designs described in a high-level hardware description language. The proposed fault models are based on multiple-input stuck-at faults, actual observed complex failures, and failures of the software structures of a generic hardware description language that is used to describe high-level designs. The proposed fault models are validated through a correlation of the fault coverage obtained from behavior fault simulation of the design in the presence of such faults with the coverage from the gate-level simulation of an equivalent gate-level representations in the presence of stuck-at faults.

There exist some research works which convert liveness assertions to safety assertions. For example, [61] uses finite state machine and reachability analysis to translate liveness assertions to safety assertions. In [70] the same technique as [61] has been implemented but using an unbounded state machine. [60] describes a technique to distinguish between a liveness and a safety assertion but no specific method for translation of liveness to safety assertions has been proposed.

In [71], the authors have presented a methodology to synthesize monitors from assertions written in the Property Specification Language (PSL). The method implements both the weak and strong versions of PSL Foundation Language (FL) operators. The authors have proved the correctness of the monitors using Prototype Verification system (PVS) theorem prover. Monitors can be connected to the DUV for formal verification, simulation, emulation or online testing. Such monitors read the system's signals and report if the property has either failed, completed successfully or its status is still pending.

Finally, IBM developed a commercial tool called Formal Checkers (FoCs) [66] which takes formal specifications as input and translate them into VHDL checkers which are then linked with design in a simulation-based verification environment. The checker essentially implements a state machine which will enter an error state in a simulation run if the property fails to hold in the run.

4.1.1 Thesis contributions

This thesis studies the correlation between fault detection capabilities of checkers and high-quality assertions. Particularly, contributions of this thesis are:

- Studied the capability of fault coverage of set of assertions at RTL and fault coverage of these assertions when they are converted to hardware checkers at gate level **[Publication VI]**.

- Studied the correlation between the fault coverage of assertions and fault coverage of the hardware checkers **[Publication V]**.

For the sake of scalability, this thesis proposes the following light weight method to generate hardware checker from verification assertion:

- Converting liveness assertions to safety assertions. Subsequently, the safety assertions are converted to RTL hardware checkers by using checker synthesis software like IBM FoCs [66], which in turn can be synthesized by any logic synthesis tool **[Publication VI]**.

In the following sections, the correlation between behavioral fault model and structural fault model is discussed. Followed by, the translation of liveness assertions to safety assertions is explained and then conversion of safety assertions to safety checkers is studied. Finally, an experimental result and a short summary of the chapter are provided.

4.2 Correlation between behavioral fault model and structural fault model

The focus of this section is studying correlation between the fault coverage of set of assertion/checkers at different level of abstraction i.e, Behavioral. Figure 30 shows the framework for estimating the correlation between the evaluation of fault coverage at behavioral level and evaluation of fault coverage at gate-level. Looking at the figure, liveness assertions and the design are given as inputs to the *Assertion Qualification and Minimization* phase. A mutation-based qualification software called Certitude injects faults based on mutation analysis into considered design and then determines whether the assertions can detect these faults. Based on number of faults detected by an assertion, the fault detection capability of the assertion can be calculated. Similarly, at gate-level, hardware checkers and the design are given as inputs to the *Fault Simulation, Evaluation and Minimization* phase. A diagnostic software called Turbo tester performs the fault simulation on the considered design along with the checkers devised. Based on the metrics introduced in section 2.8, the fault detection capabilities of the checkers are evaluated. And finally, the correlation between fault coverage of the both level is calculated. Experimental results reports this correlation. More details about evaluation of fault detection capabilities of the checkers at gate-level are described previously in section 3.2.6 and the evaluation of fault detection capabilities of the assertions at behavioral level will be discussed in section 5.2.2.

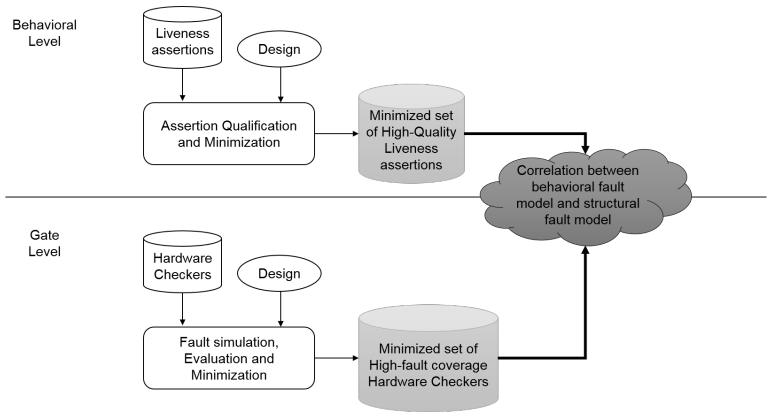


Figure 30 – Correlation between behavioral fault model and structural fault model

4.3 Translation of liveness assertions to safety assertions

In this section, the translation of liveness assertions to safety assertions is discussed in detail. To verify the expected behavior of a design, it is important to specify the sequence of events that happen in the design [72]. This can be expressed by the following notation: $Antecedent(A) \rightarrow Consequent(C)$, where $\rightarrow$ refers to implication operator. Antecedent is a behavior that occurs before the Consequent. The result of the implication is either true or false. A liveness assertion is one which checks for the behavior that must happen eventually. For consequent to be true, the antecedent must be true first. In some cases, the liveness assertion stays true even though the antecedent is false. It means that the verification environment never stimulates the design in such a way to make antecedent true. This is known as vacuous success and this success does not carry any weight as far as the verification of the design is concerned. The truth table of the liveness assertion is shown in table 11. From table 11, the liveness assertion output can be expressed as $\overline{A}$ || C.

Antecedent	Consequent	Liveness assertion output
0	0	1
0	1	1
1	0	0
1	1	1

Table 11 – Truth table of liveness assertion

A safety assertion is one which checks for the behavior that must not happen. The safety assertion used for hardware checker circuitry is concerned when antecedent is true, but consequent is false. The truth table of the safety assertion is shown in table 12. From table 12, the safety assertion output can be expressed as A & $\overline{C}$.

Antecedent	Consequent	Safety assertion output
0	0	0
0	1	0
1	0	1
1	1	0

Table 12 – Truth table of safety assertion

Based on the output derived from table 11 and table 12, the liveness assertion and safety assertion are expressed by the following notations respectively.

- Liveness assertion = A ? ($\overline{C}$? 0 : 1) : 1;
- Safety assertion = (A & $\overline{C}$) ? 1 : 0;

Here, A is the antecedent and C is the undesired consequent.

The following is a set of examples where liveness assertions generated for the design ELBDR are translated into its corresponding safety assertions.

- Whenever there is a request (i.e., the HEADER flit contains the destination address) LBDR must compute at least one valid output direction (XY Routing) to pass the flits from the input buffer to the respective output port.
  - E_validLBDRout_liveness = (!Eempty && (flit_id == HEADER || flit_id == PAYLOAD || flit_id == TAIL)) ? ((!oNport && !oWport && !oSport && !oLport) ? 1'b0 : 1'b1) : 1'b1;
  - E_validLBDRout_safety = ((!Eempty && (flit_id == HEADER || flit_id == PAYLOAD || flit_id == TAIL)) && (!oNport && !oWport && !oSport && !oLport)) ? 1'b1 : 1'b0;

Here, E_validLBDRout_liveness is the liveness assertion generated, E_validLBDRout_safety is the translated safety assertion, (!Eempty && (flit_id == HEADER || flit_id == PAYLOAD || flit_id == TAIL)) is the antecedent and (!oNport && !oWport && !oSport && !oLport) is the undesired consequent.

- LBDR routing logic works on HEADER flit alone and it maintains the same port direction for PAYLOAD and TAIL flit of the same packet, until a new HEADER flit arrives. If there is none of these flits arrive then output should be zero.
  - E_noLBDRout_liveness = (Eempty || (flit_id != HEADER) && (flit_id != PAYLOAD) && (flit_id != 'TAIL)) ? ((oNport || oWport || oSport || oLport) ? 1'b0 : 1'b1) : 1'b1;
  - E_noLBDRout_safety = ((Eempty || (flit_id != HEADER) && (flit_id != PAYLOAD) && (flit_id != 'TAIL)) && (oNport || oWport || oSport || oLport)) ? 1'b1 : 1'b0;

Here, E_noLBDRout_liveness is the liveness assertion generated, E_noLBDRout_safety is the translated safety assertion, (Eempty || (flit_id != HEADER) && (flit_id != PAYLOAD) && (flit_id != 'TAIL)) is the antecedent and (oNport || oWport || oSport || oLport) is the undesired consequent.

- Only one direction port or Local port can become output at an instant.
  - E_singleLBDRout_liveness = (!Eempty && (flit_id == HEADER || flit_id == PAYLOAD || flit_id == TAIL)) ? (((oNport && oWport) || (oNport && oSport) || (oNport && oLport) || (oWport && oSport) || (oWport && oLport) || (oSport && oLport)) ? 1'b0 : 1'b1) : 1'b1;
  - E_singleLBDRout_safety = ((!Eempty && (flit_id == HEADER || flit_id == PAYLOAD || flit_id == TAIL)) && ((oNport && oWport) || (oNport && oSport) || (oNport && oLport) || (oWport && oSport) || (oWport && oLport) || (oSport && oLport))) ? 1'b1 : 1'b0;

Here, E_singleLBDRout_liveness is the liveness assertion generated, E_singleLBDRout_safety is the translated safety assertion, (!Eempty && (flit_id == HEADER || flit_id == PAYLOAD || flit_id == TAIL)) is the antecedent and ((oNport && oWport) || (oNport && oSport) || (oNport && oLport) || (oWport && oSport) || (oWport && oLport) || (oSport && oLport)) is the undesired consequent.

- A non-header flit (PAYLOAD or TAIL) arrives but there is a request to build a new input-output port connection or the request of the intended output port is muted while there is a simultaneous request for another output port. Although one output is computed by LBDR, this erroneous output results in packet misrouting and even deadlock.
  - E_switchLBDRout_liveness = (!Eempty && (flit_id == PAYLOAD || flit_id == TAIL)) ? (((oNport != iNport) || (oWport != iWport) || (oSport != iSport) || (oLport != iLport)) ? 1'b0 : 1'b1) : 1'b1;
  - E_switchLBDRout_safety = (!Eempty && (flit_id == PAYLOAD || flit_id == TAIL)) && ((oNport != iNport) || (oWport != iWport) || (oSport != iSport) || (oLport != iLport)) ? 1'b1 : 1'b0;

Here, E_switchLBDRout_liveness is the liveness assertion generated, E_switchLBDRout_safety is the translated safety assertion, (!Eempty && (flit_id == PAYLOAD || flit_id == TAIL)) is the antecedent and ((oNport != iNport) || (oWport != iWport) || (oSport != iSport) || (oLport != iLport)) is the undesired consequent.

- Local port should be triggered when current address matches with the destination address
  - E_localport1_liveness = (!Eempty && (flit_id == HEADER)) ? (((oLport == 1) && (cur_addr != dst_addr)) ? 1'b0 : 1'b1) : 1'b1;
  - E_localport1_safety = ((!Eempty && (flit_id == HEADER)) && ((oLport == 1) && (cur_addr != dst_addr))) ? 1'b1 : 1'b0;

Here, E_localport1_liveness is the liveness assertion generated, E_localport1_safety is the translated safety assertion, (!Eempty && (flit_id == HEADER)) is the antecedent and ((oLport == 1) && (cur_addr != dst_addr)) is the undesired consequent.

  - E_localport2_liveness = (!Eempty && (flit_id == HEADER)) ? (((oLport == 0) && (cur_addr == dst_addr)) ? 1'b0 : 1'b1) : 1'b1;
  - E_localport2_safety = ((!Eempty && (flit_id == HEADER)) && ((oLport == 0) && (cur_addr == dst_addr))) ? 1'b1 : 1'b0;

Here, E_localport2_liveness is the liveness assertion generated, E_localport2_safety is the translated safety assertion, (!Eempty && (flit_id == HEADER)) is the antecedent and ((oLport == 0) && (cur_addr == dst_addr)) is the undesired consequent.

4.4 Conversion of safety assertions to hardware checkers

In this section, the conversion of safety assertions to hardware checkers is discussed. Checking tools, such as [66] are available commercially which converts the assertions to checkers, which can be integrated into the verification environment. In this thesis, the safety assertions derived from liveness assertions as described in section 4.3 are synthesized using Synopsys design compiler tool [73] to generate hardware checkers. For a safety assertion, the output of the hardware checker reports whenever the condition is violated.

4.5 Experimental results

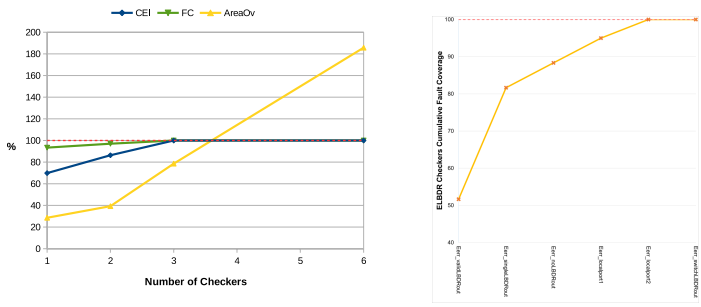
In this section, the experiments ran on the East input port LBDR module explained in section 2.6.1 and South output port Arbiter module explained in section 2.6.2 using the methodology flow introduced in section 3.2 and assertion qualification and minimization introduced in section 5.2.2, 5.3 respectively were discussed. First, the correlation between the fault detection capabilities of checkers and assertions devised for ELBDR module is studied. Next, the same procedure is carried out for SARBITER module.

4.5.1 ELBDR experiment

Table 1 describes the initial devised set of checkers for the routing computation unit corresponding to the East input port. First, the fault simulation is performed with the initial set of checkers. The results of the evaluation process in terms of true detection weights are already reported in figure 23. Based on true detection weight information, the optimization process has been executed, and by greedy heuristics minimization procedure, the minimized set of checkers was obtained. Results are shown in figure 31a, reporting *CEI*, *FC* and *area overhead* increase at each step. Full coverage (i.e., 100%) is achieved when the first three heaviest checkers are considered (Eerr_noLBDRout, Eerr_validLBDRout, Eerr_singleLBDRout), resulting in the minimized set of checkers. At behavioral level, fault analysis is performed with initial set of assertions. The result is the fault table containing which assertions cover which faults. Based on the fault table information, the minimized set of assertions are obtained by the greedy heuristics minimization procedure. Full coverage (i.e., 100%) is achieved when all the assertions are considered except E_switchLBDRout-_liveness. Results are shown in figure 31b reporting *Cumulative Fault Coverage* increase at each step.

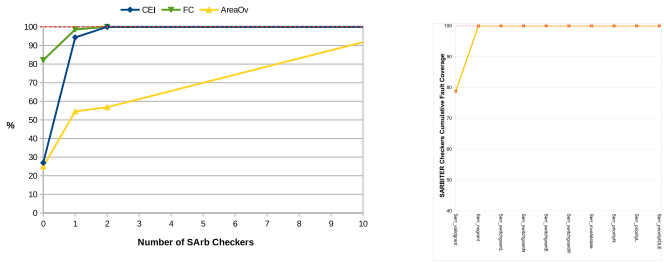
4.5.2 SARBITER experiment

Table 2 describes the initial devised set of checkers for the arbitration unit corresponding to the South output port. First, the fault simulation is performed with the initial set of checkers. The results of the evaluation process in terms of true detection weights are already reported in figure 26. Based on true detection weight information, the optimization process has been executed, and by greedy heuristics minimization procedure, the minimized set of checkers was obtained. Results are shown in figure 32a, reporting *CEI*, *FC* and *area overhead* increase at each step. Full coverage (i.e., 100%) is achieved when the first two heaviest



(a) *(b)*

Figure 31 – (a) ELBDR checkers optimization results (b) ELBDR Cumulative fault coverage using Assertion Qualification and Minimization procedure



(a) *(b)*

Figure 32 – (a) SARBITER checkers optimization results (b) SARBITER Cumulative fault coverage using Assertion Qualification and Minimization procedure

checkers are considered (Serr_validgrant, Serr_invalidstate), resulting in the minimized set of checkers. At behavioral level, fault analysis is performed with initial set of assertions. The result is the fault table containing which assertions cover which faults. Based on the fault table information, the minimized set of assertions are obtained by the greedy heuristics minimization procedure. Full coverage (i.e., 100%) is achieved when the first two assertions with high quality fault coverage (Serr_validgrant, Serr_nogrant) are considered. Results are shown in figure 32b reporting *Cumulative Fault Coverage* increase at each step.

4.6 Chapter summary

In this chapter, the correlation between the fault detection capabilities of the concurrent checkers and the fault detection capabilities of the high-quality verification assertions are studied. Next an assertion translation was proposed which converts the liveness assertions into its safety equivalents. Finally, the safety assertions are synthesized further to hardware checkers which are to be evaluated at the gate-level to provide cost-effective checker infrastructure.

Experiments are carried out on the East input port LBDR module and South output port arbiter module. The results showed the feasibility of assessing the fault detection capabilities of the concurrent checkers by applying assertion qualification. Although assertion quality was not directly proportional to the fault coverage of the checker, a heuristic-based minimization procedure indicates that the optimal solution in terms of area and fault coverage was achieved without the need to move to the lower level of abstraction.

5 Qualification and minimization of assertions

Due to the increasing complexity of today's digital systems, the amount of time and manpower that is invested in finding and removing bugs is growing. To overcome this problem and to develop systems without bugs, verification techniques have arisen which check if a system meets its specification and thereby fulfills its intended purpose [74]. Among all these techniques, Assertion Based Verification (ABV) has become a popular means for catching and eliminating errors. At the same time, due to the growing failure-rates, process variations and time-dependent degradation of modern chip technologies, it is imperative to develop cost-effective means for protecting systems against faults occurring in the field, during their life-time.

Thus, concurrent on-line checker circuitry is required to monitor the fault-free functioning of the system hardware. Such checkers are normally designed ad-hoc or by synthesizing them from verification assertions. However, the number of assertions in the verification environment is generally far too high to allow for area-efficient checking infrastructure. Moreover, the number of liveness checkers generated by automated methods (e.g. [75, 76, 63]) may be too high even for verification purposes. Therefore, there is a need for qualification and minimization of liveness assertions with a prospect of reusing them as hardware safety checkers.

A verification environment consists of a set of assertions that collectively can detect a range of design bugs. However, not all the assertions are essential to detect this range: some assertions are dominated by others, or by a set of other assertions, some assertions are equivalent in terms of bug detection capabilities, etc. Discarding such assertions which do not detect any unique bugs leads to obtaining a set of *minimized* assertions. Of course, it is not possible to enumerate all possible bugs, and therefore, fault models are applied to estimate the coverage of different assertions. This assertion quality estimation task is called assertion *qualification*. While there exist several works that address assertion qualification and minimization [75, 76, 63] as well as qualification and minimization of checkers at the gate-level [3, 77]. This chapter proposes a methodology that applies high-level assertion qualification and minimization with the goal of generating low-area high-quality checker circuitry.

This chapter is based on the following publications:

- Ranganathan Hariharan, Tara Ghasempouri, Behrad Niazmand, and Jaan Raik. From rtl liveness assertions to cost-effective hardware checkers. In *2018 Conference on Design of Circuits and Integrated Systems (DCIS)*, pages 1–6. IEEE, 2018 [**Publication VI**]

5.1 Literature review

With the advent of standardized assertion languages and assertion libraries, several research works witnessed an increased interest in adopting assertion-based techniques.

Assertions may be generated manually or by assertion mining tools. Manual definition of assertions requires high expertise, and it is an error-prone, time-consuming activity. Both manual as well as automatic way of generating assertions suffer from certain problems. These problems are related to the risk of defining assertion sets that are incomplete (i.e., unable to cover all expected behaviors of the DUV), inconsistent (i.e., with contradicting assertions), redundant (i.e., with assertions that are logical consequence of others), and including vacuous assertions (i.e., assertions that are true independently from the DUV, and thus irrelevant). As a result, a false sense of security is induced by ABV[21] campaign conducted with a low-quality set of assertions. As opposite to manual definition, some

works have been done recently to automatically generate assertions from the DUV implementation [78, 64]. In these approaches, execution traces obtained by simulating the DUV are dynamically analyzed to mine significant assertions. Independent from the abstraction level of the DUV (e.g. Transaction-Level Modeling [TLM], RTL, gate-level) execution traces pass through an assertion miner, whose output is a set of candidate assertions capturing the behaviors exposed by the DUV during simulation, according to a set of predefined temporal patterns. Extracted assertions may highlight shortcomings of the original specifications, which may lead to distinguish design's error and unpredictable behaviors implemented in the DUV. While vacuity and inconsistency in the set of generated assertions are generally avoided by the mining approach itself, assertion incompleteness and redundancy may still affect the outcome of assertion mining. Thus, a qualification phase for evaluating the degree of interestingness of extracted assertions and minimize the irrelevant ones is still necessary.

As the number of mined assertions can be very high, their manual qualification is almost impractical. For this reason, a strategy to automatically evaluate the interestingness of extracted assertions and rank them accordingly is necessary. Unfortunately, current approaches for assertion mining are still unsatisfactory from this point of view.

In [78], a stressing phase is proposed to obtain the candidate assertions which in turn are converted into checkers, by using for example, IBM FoCs [66], and connecting these checkers to the DUV. During the stressing phase, a much larger set of testbenches are used compared to the initial set of testbenches used to generate the execution traces. In stressing phase when a checker fails, the corresponding candidate assertion is discarded. Only assertions that survive this stressing phase are collected. The stressing phase is applied to increase the likelihood that the surviving assertions are satisfied by the DUV independently from the execution traces adopted for their extraction. This approach can verify the likelihood that mined assertions are globally satisfied (and not only for the execution traces analyzed by the miner), but no strategy is proposed to measure their interestingness in covering DUV behaviors.

In [64], interestingness estimation is based on the number of propositions included in the antecedent of the assertion, according to the fact that an assertion with a lower number of propositions in its antecedent has a higher input space coverage than one with many propositions in its antecedent. However, the correlation between the antecedent and the consequent of an assertion is not considered. To solve this drawback, in [79] a ranking function is proposed that evaluates the quality of the mined assertions in terms of cause-effect relationship between antecedent and consequent of an assertion.

In [22] the quality of assertions is estimated based on their amount of frequencies and correlation during the simulation. However, the work does not consider assertions with low number of frequencies which may cover the corner cases of a design. In [25], a metric is introduced to rank assertions based on their ability to cover corner cases. Moreover, it does not consider assertions which cover the general behavior of the design. In [75], mined assertions are said to be generally ranked according to their frequency of occurrences and time of first occurrence, which is too general metric for the ranking purpose.

As an opposite class of approaches, coverage metrics have been widely studied for qualification of assertions [80, 81, 82, 83] . Most of these works rely on mutation analysis, which requires perturbing the DUV implementation by injecting mutations (faults) to check, either statically [81, 82] or dynamically [83], whether they change the truth values of the assertions. Mutations that cause a change are said to be detected. Assertions that detect a few mutations are less interesting than assertions detecting a higher number of mutations. Not detected mutants generally highlight area/behaviours of the DUV

that are not covered by any of the defined assertions showing a hole on the coverage. Dynamic approaches like [83] scale better with respect to static techniques, however, they still require long simulation runs for checking each assertion for each mutation with a significant set of testbenches. When the number of assertions is very high, as in the case of assertion extracted automatically, evaluating their interestingness through mutation analysis becomes a very time-consuming activity.

5.1.1 Thesis contributions

This thesis proposes a framework for selecting a set of high-quality and minimized liveness assertions from initial set of liveness assertions by combining a data mining technique with fault analysis approach. Different from the above-mentioned approaches, the qualification and minimization method proposed in this thesis provides the following contributions:

- First, an advanced assertion ranking approach has been introduced based on combining three different metrics adapted from data mining, to evaluate assertions' quality from different conditions. Compared to similar approaches in the state of the art, it provides higher correlation of assertion quality with fault coverage of the obtained checkers **[Publication VI]**.
- Second, combining a fault analysis approach along with a data mining approach for assertion qualification to get the advantages of both techniques, the former providing high accuracy and the latter very short execution time **[Publication VI]**.
- Third, proposing an innovative automatic technique to estimate quality of assertions by applying a data mining based qualifier, according to probabilistic metric typically adopted in the context of data mining. This metric can distinguish the assertions with limited number of activation which are very effective in covering the corner cases of a design **[Publication VI]**.

In the following sections, the methodology for qualifying, minimizing the considered set of assertions is discussed. Followed by experimental results regarding the application of framework to the control logic of the NoC router is studied. Finally, a short summary of the chapter is provided.

5.2 Assertion qualification

In order to derive cost effective hardware checkers from a large number of liveness assertions, a methodology is proposed (as shown in figure 33), first to estimate the quality of the assertions based on data mining metrics and rank them based on their quality i.e., Assertion Qualification step. Second, these highly ranked assertions are subjected to fault analysis to determine the fault coverage of each of them. Some of the assertions are discarded based on the output of fault analysis to derive minimized set of high-quality liveness assertions i.e., Assertion Minimization step. In the following these two steps are discussed.

Assertion qualification consists of two main phases as shown in figure 34.

1. Assertion ranking
2. Assertion fault analysis

In *Assertion Ranking* phase, a data mining-based tool called Shayan [22] is applied on liveness assertions to estimate their *quality* based on data mining metrics. The high-quality assertions selected by Shayan go through *Assertion Fault Analysis* phase utilizing

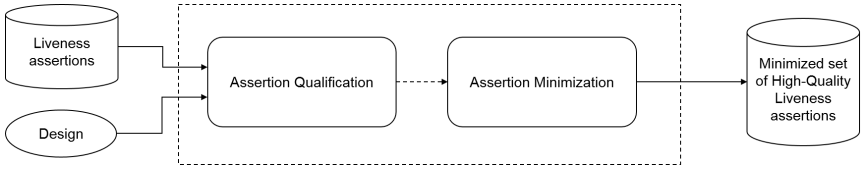


Figure 33 – Assertion Qualification and Minimization

the Synopsys Certitude qualification tool [84]. The hypothesis is that assertions with higher degree of quality are more effective in the verification process. Thus, Shayan selects assertions with the degree of quality above a preset threshold and forwards them to the certitude for fault analysis. The main drawback of fault analysis approaches is their long simulation time, since the effect of a fault that has been injected needs to be evaluated by simulation. The above-mentioned preliminary selection by Assertion Ranking leads to reduction of this simulation time. In the following subsections it will be explained how assertions are ranked based on the data mining metrics and also how fault coverage of each assertion is estimated.

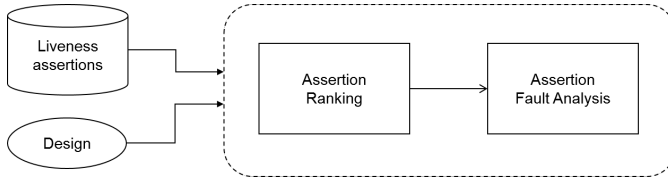


Figure 34 – Assertion Qualification

5.2.1 Assertion ranking

From the point of view of general concept, data mining [85] and assertion ranking share the same idea (extracting rules from data), but they have several differences that make it practically different how these metrics are computed and interpreted for evaluating the quality of assertions. Shayan calculates a metric called Q which is calculated individually for each assertion. Q is the linear combination of Support (Definition 1), Correlation Coefficient (Definition 2) and Strength Measure (Definition 3). The higher the value of Q, the higher the quality of the assertion would be. Figure 35 shows the internal design of Shayan in three steps:

1. *Occurrence counting*
2. *Contingency table creation*
3. *Metric calculation*

- Occurrence counting: Liveness assertions and the DUV are inputs of the work flow. In the first step, set of valid input sequences are connected to a simulator to extract information about occurrences of assertions during the simulation. It is worth to be noted that valid input sequences used are extracted from the exhaustive set of input pattern. The method used to extract the valid input sequences are discussed already in section 3.2.3. The number of times an assertion is holding in the valid input sequences is computed. Then, each assertion is decomposed into *antecedent* and *consequent* and their respective frequencies in the valid input sequences are computed.

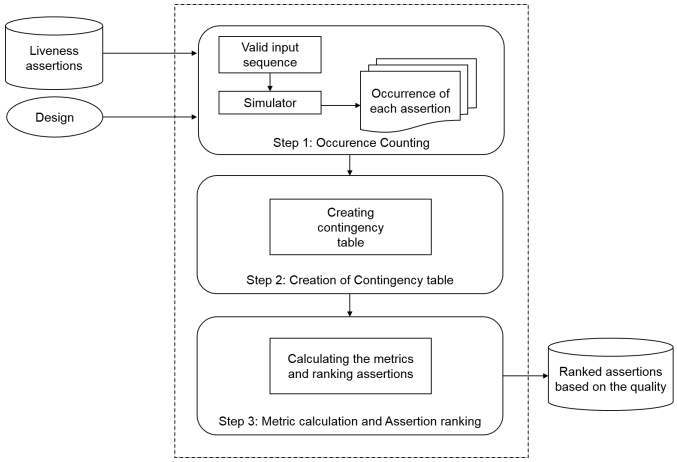


Figure 35 – Overview of Assertion ranking

- Creation of contingency table: At this stage, the necessary ingredients are ready for Creating contingency tables, see (Table 13). The computation of the contingency

	C	$\bar{C}$	
A	f_{11}	f_{10}	f_{1X}
$\bar{A}$	f_{01}	f_{00}	f_{0X}
	f_{X1}	f_{X0}	f_{XX}

Table 13 – Contingency table for $A \rightarrow C$.

table is based on counting occurrences of antecedent, consequent and the assertions respectively. Given an assertion $A \rightarrow C$, its contingency table represents the relation between A and C.

The cells of contingency table contain the following information (Table 13):

  - Cell f_{11} represents the number of times where A is true and C is true in the valid input patterns.
  - Cell f_{10} represents the number of times where A is true but C is false in the valid input patterns.
  - Cell f_{01} is the dual of f_{10}, *i.e.*, it is the number of times where A is false and C is true in the valid input patterns, *i.e.*, it is the sum of occurrences of assertions $A' \rightarrow C$ included in the considered assertion set with $A \neq A'$. In this case, A and A' can also be conflicting because this does not represent an inconsistency for the assertion set.
  - Cell f_{00} is the number of times an assertion is not true in the valid input patterns. f_{00} is obtained by summing the occurrences of f_{11}, f_{10} and f_{01} and subtracting them to the total number of valid input patterns.
  - Cell f_{1X} is the sum of cells f_{11} and f_{10}.
  - Cell f_{0X} is the sum of cells f_{01} and f_{00}.
  - Cell f_{X1} is the sum of cells f_{11} and f_{01}.
  - Cell f_{X0} is the sum of cells f_{10} and f_{00}.

- Cell f_{XX} is the grand total. f_{XX} is same as the total number of valid input patterns considered in the simulation.

- Metric calculation and assertion ranking: Contingency tables provide basic ingredients for computation of Support [S], Correlation Coefficient [CC], Strength Measure [Strength] and their linear combination Q. Concerning support, according to (Definition 1), it is computed using the following formula:

$$Support = \frac{f_{11}}{f_{XX}} \tag{9}$$

The Correlation Coefficient for an assertion according to (Definition 2) is computed using the following formula:

$$CC(A,C) = \frac{f_{11}f_{XX} - f_{1X}f_{X1}}{\sqrt{f_{1X}f_{0X}f_{X1}f_{X0}}} \tag{10}$$

The computation of the Strength Measure for an assertion according to (Definition 3) is computed by:

$$Strength\ Measure = \sqrt{\frac{f_{11}^2}{|f_{X1} - f_{X0}| \cdot |f_{1X} - f_{0X}|}} \tag{11}$$

According to equation 9, the support ranks in the highest positions assertions that occur frequently in the execution traces. On the other hand, the correlation coefficient (equation 10) privileges assertions where the number of occurrences of the antecedent better matches the number of occurrences of the consequent, but assertions where these numbers are low could be extracted by chance without representing a real behavior of the DUV. However, there are also some specific assertions that occur very rarely because they refer to the corner cases and thus equation 11 has been proposed. A combination of support, correlation coefficient and strength measure provide a more accurate estimation of assertion interestingness (equation 12).

Thus, the quality of an assertion A can be measured through the following formula:

$$Q(A) = \alpha * s_n(A) + (1-\alpha) * \rho_n(A) + (1-\alpha) * strength_n(A) \tag{12}$$

where, $\alpha \in [0,1]$, and $s_n(A)$ and $\rho_n(A)$ are the value obtained by normalizing, respectively, the support s, the correlation coefficient ρ and Strength measure $strength$ of A with respect to the whole set of analyzed assertions. By varying the value of α the role of support becomes important with respect to the role of the correlation coefficient and strength in determining the final estimation of assertion quality.

5.2.2 Assertion fault analysis

In this section, the fault detection capability the output from the assertion ranking phase (section 5.2.1) is discussed. The overview of *Assertion Fault analysis* phase is shown in figure 36. Shayan, which was described in section 5.2.1, can rank the assertions based on data mining metrics but it cannot provide any information whether two assertions have the same set of covered faults (i.e. the assertions are *equivalent*) or one is subset of the other assertions' covered faults (i.e. *dominated* by the other assertion) etc. Such equivalence and dominance relationships between assertions allow minimizing the set of assertions selected for synthesis and gate-level evaluation. As shown in Figure 36, the

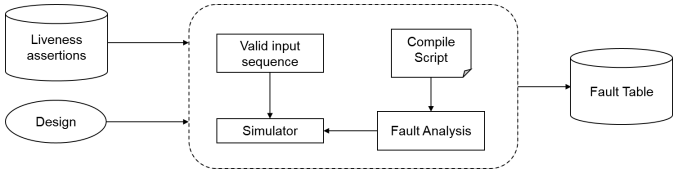


Figure 36 – Assertion Fault analysis phase

DUV and selected liveness assertions provided by *Assertion Ranking* phase together with the valid input sequence (i.e. the verification environment) are fed into the Certitude tool that performs the fault analysis.

Certitude injects faults based on mutation analysis into considered design and then determines whether the liveness assertions can detect these faults. Based on number of faults detected by an assertion, the fault detection capability of the assertion can be calculated. Once the fault is injected into the design, certitude requires a simulator to run the fault simulation along with valid input patterns. The output is the fault table showing which assertion a_j covers which faults f_i. The fault table presents one row for each assertion and one column for each fault injected. For the corresponding input pattern, the assertion detecting a particular fault will be marked as *1* and the undetected faults as well as the faults which have not been propagated are marked as *0*. A sample fault table is shown in figure 37.

Assertions	Faults						
	f1	f2	f3	f4	f5	f6	f7
a1	0	0	1	1	0	0	1
a2	1	0	0	0	0	1	0
a3	0	1	0	0	0	0	0
a4	1	1	1	1	0	0	0
a5	0	0	1	1	1	0	0

Figure 37 – Sample Fault Table

5.3 Assertion minimization

In assertion minimization step, a minimization algorithm (see algorithm 2) is implemented on the fault table information of each assertion and the output is the minimized set of high-quality assertion as shown in figure 38. The minimized set of liveness assertions are of high-quality, with good fault coverage and which in turn can be synthesized to be reused as hardware checker circuitry.

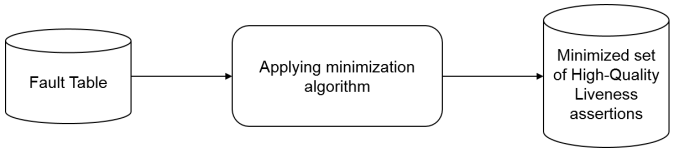


Figure 38 – Assertion Minimization

The algorithm is based on iterative $implication$ and *greedy selection* operations. Two types of implications are used. First, $unique$ assertion a_j, which cover some fault f_i that is not covered by any other assertions are identified and removed from the table. Second, it is said that assertion a_j $dominates$ assertion a_k, if all the faults covered by a_k is a subset of the faults covered by a_j. Note, that equivalence of two assertions a_j and a_k is a special

Algorithm 2 Fault table minimization

```
while exist faults uncovered by assertions do
    while implications provide new assertions do
        Select unique assertions
        Remove dominated assertions
    end
    Make a greedy selection
end
```

case of dominance, where a_j and a_k mutually dominate each another. If after performing the implications the set of selected assertions are not covering all the faults in the fault table, a *greedy selection* operation is performed. The algorithm selects an assertion that covers the greatest number of faults not yet covered by the set of selected assertions. The algorithm will complete when the selected assertions cover all the faults in the fault table.

5.4 Experimental results

The efficiency of the proposed methodology has been evaluated by considering liveness assertions for East input port routing computation unit and South output port arbitration unit of NoC router which is already discussed in section 2.6. By applying filtering constraints for the exhaustive set of input patterns considered initially for East input port LBDR design, 2144 is extracted as valid input patterns. In the similar fashion, for South output port Arbiter design, 80 valid input patterns are extracted from an exhaustive set of input patterns.

Initially the information about the occurrences of assertions for the set of valid input sequence during the simulation is extracted. Based on the counting occurrences of antecedent, consequent and assertions, a contingency table is created. The contingency table for the ELBDR design is shown in Table 14. For example, for assertion E_validLBDRout-_liveness, f_{11} corresponds to the total number of occurrences of E_validLBDRout_liveness in the analyzed valid input sequences. Assertion E_validLBDRout_liveness was activated in 96 different sequences and assertion E_noLBDRout_liveness was activated in 2048 other sequences and exactly at different clock cycle compare to E_validLBDRout_liveness. f_{10} is equal to 0, since antecedent A does not appear in none of the other assertions. f_{01} is 0 since consequent of the assertion does not appear in none of the other assertions. And finally, f_{00} is obtained by summing the occurrences of f_{11}, f_{10} and f_{01} and subtracting them to the total number of valid input patterns (in this case 2144). Next the combination columns like f_{X1}, f_{X0}, f_{1X} and f_{0X} are obtained by summing the values of either of f_{11} or f_{10} or f_{01} or f_{00} accordingly. f_{XX} is same as the total number of valid input patterns considered in the simulation. Similar considerations allow computing values for all the other cells of Table 14.

Assertion	f_{11}	f_{10}	f_{01}	f_{00}	f_{X1}	f_{X0}	f_{1X}	f_{0X}	f_{XX}
E_noLBDRout_liveness	2048	0	0	96	2048	96	2048	96	2144
E_validLBDRout_liveness	96	0	0	2048	96	2048	96	2048	2144
E_switchLBDRout_liveness	64	0	137	1943	201	1943	64	2080	2144
E_singleLBDRout_liveness	96	0	0	2048	96	2048	96	2048	2144
E_localport1_liveness	4	16	0	2124	4	2140	20	2124	2144
E_localport2_liveness	28	0	2116	0	2144	0	28	2116	2144

Table 14 – Contingency table for ELBDR design assertions

Based on the contingency table information, the metrics described in section 2.11 for each assertion of ELBDR design is calculated. The output is shown in table 15. The best result for the quality estimation is obtained with $\alpha = 0.4$ in comparison with fault detection capability of the assertions.

Assertion	*S*	*CC*	*Strength*	*Q*
E_noLBDRout_liveness	0.96	1.0	1.05	1.61
E_validLBDRout_liveness	0.04	1.0	0.05	0.65
E_switchLBDRout_liveness	0.03	0.55	0.03	0.36
E_singleLBDRout_liveness	0.04	1.0	0.05	0.65
E_localport1_liveness	0.001	0.45	0.001	0.27
E_localport2_liveness	0.01	0	0.01	0.01

Table 15 – Calculated metrics for each assertion of ELBDR design

The assertions are ranked based on the value Q. According to the preset threshold (here 75%), assertions, E_noLBDRout_liveness, E_validLBDRout_liveness, E_singleLBDRout-_liveness and E_switchLBDRout_liveness are selected and rest are discarded. Similarly for SARBITER design, 19 assertions out of 28 assertions are selected and given as input to Assertion fault analysis phase.

In Assertion fault analysis phase, a mutation-based qualification software, Certitude does fault analysis on each of the selected set of assertions for ELBDR design and outputs the result in the form of fault table information which contains the details of which assertion detects which fault injected in the design. Based on the fault table information provided by Assertion fault analysis phase for ELBDR design, the minimization algorithm is applied. Assertions reduced to 3 based on the fault detection capability. Similarly, for SARBITER design, based on the fault table information provided by Assertion fault analysis phase, the minimization algorithm selects 2 assertions out of 19.

Table 16 shows the number of assertions considered initially at the beginning of each phase - Assertion ranking phase (described in Section 5.2.1) and a minimized set of assertions subsequent to the Assertion fault analysis phase (described in Section 5.2.2). As it can be seen, for ELBDR design the initial number of assertions were minimized to 50% and for SARBITER design, the initial number of assertions were reduced to 7.14%.

	Initial		Assertion Ranking		Assertion Fault Analysis	
Design	#	%	#	%	#	%
ELBDR	6	100	4	66.7	3	50
SARBITER	28	100	19	66.7	2	7.14

Table 16 – Minimization of the number of assertions for ELBDR and SARBITER design

The minimized set of checkers are synthesized to hardware checkers via Synopsys design compiler using a class library, leading to area consumption results in terms of number of NAND2 gate equivalents. As shown in table 17, the area consumption of initial set of checkers were 60 and 128 (number of NAND2 gates) for ELBDR and SARBITER respectively. After minimization, the area consumption has been reduced to 29 and 33 (number of NAND2 gates) respectively. This minimization leads to 48.3% and 25.7% of reduction in area consumption.

Design	Initial set	Minimized set	Reduced Area (in %)
ELBDR	60	29	48.3%
SARBITER	128	33	25.7%

Table 17 – Area consumption of Checkers (number of NAND2 gates)

5.5 Chapter summary

In this chapter, a framework is proposed for selecting a minimal set of high-quality assertions which in turn to be implemented as hardware checkers. Experiments were run, applying the proposed framework, to ELBDR and SARBITER as a standalone module. It is worth mentioning that the area consumption of the synthesized checkers were reduced to 25.7% and 48.3% for ELBDR and SARBITER design respectively. These checkers provided by this proposed framework are able to cover 99.83% of single event transient faults.

6 Conclusion

This thesis has addressed a set of timely issues in reliability by developing a methodology for generating cost-effective concurrent hardware checkers. A framework has been proposed for evaluating the fault detection capabilities of concurrent checkers for NoC routers. The goal was to achieve low-latency, low area overhead and high fault coverage checkers. Also, a framework has been developed for selecting a set of high-quality assertions by combining a data mining technique with the fault-analysis approach allowing the reuse of the verification assertions in hardware checkers synthesis. The quality of assertions was validated by studying the correlation between the fault detection capabilities of the checkers and assertions.

The four main contributions of this thesis are summarized below:

- A framework was provided to evaluate the fault detection capability of the concurrent checkers by formally proving the absence or presence of true misses over all possible valid inputs for a checker and targeting the minimum fault detection latency of a single clock-cycle. Pseudo-combinational extraction guaranteed the possibility of fault simulating the circuit in an exhaustive valid range of conditions. The following results were achieved concerning minimization of concurrent checkers:
  - Full set of checkers was devised for control part of the target NoC router architecture reaching the target (i.e., 100%) SET fault coverage and detecting the faults with a single clock cycle latency.
  - The area overhead of 185.71% imposed by the initial set of checkers over the ELBDR module has been reduced to 78.57% with the minimized set of checkers after the minimization procedure. While the initial set of checkers for the SARBITER module would lead to 170.45% area overhead, which is limited to 56.82% after the minimization procedure.
  - The result is by far more cost-effective in terms of area when compared to a doubling or triplicating of the respective modules applied in traditional fault tolerance approaches.
- A hybrid approach was proposed which combines concurrent checkers for control part with embedded on-line test packets replacing the data-path checkers. The trade-off between area-overhead and fault coverage was outlined. The approach led to the following results:
  - When considering the checkers for both control part and data-path of the whole NoC router, the fault coverage metrics resulted in 100% value with the considerable area overhead ranging from 30.98% to 41.45% depending on the data width of the router.
  - By using the hybrid approach, the area overhead was reduced to 1.62% to 23.18% depending on different data widths.
  - Despite, it comes at the expense of loss of fault coverage for soft errors in the data path, which is however less critical since those faults can be corrected by means of software in an end-to-end correction setup.
- The correlation between the fault coverage obtained from the behavioral fault analysis with the qualified assertions and the fault coverage obtained from the gate-level fault simulation of the checkers which were synthesized from the qualified assertions was studied and validated yielding the following results:

  - Experiments carried out on the routing computation logic of NoC router showed the feasibility of assessing the fault detection capabilities of checkers by applying assertion qualification. Although assertion quality was not directly proportional to the checker coverage, experiments implementing a heuristic assertion minimization indicate that the optimal solution in terms of coverage/area was achieved without the need to descend to tedious gate-level analysis.

- A framework was proposed for selecting a minimal set of high-quality assertions to be implemented as hardware checkers by combining a data mining technique with a fault analysis approach. An assertion conversion methodology was proposed which converted liveness assertions into their safety equivalents. The safety assertions were further synthesized to hardware checkers to be evaluated at the gate level to provide a cost-effective checking infrastructure.

  - The area consumption of the synthesized hardware checkers were reduced to 25.7% and 48.3% for ELBDR and SARBITER design respectively. These checkers provided by this proposed framework were able to cover 99.83% of single event transient faults.

6.1 Future work

This research paves the way for future work in multiple directions such as:

- Considering multiple-cycle temporal logic assertion which addresses inputs, internal signals and outputs of the design in different clock cycles.
- Combining the online fault detection based on the concept of end-to-end detection.
- Combining with effective recovery and reconfiguration mechanism enabling the system to continue operating properly in the presence of faults.

List of Figures

List of Tables

References

[1] Behrad Niazmand, Ranganathan Hariharan, Vineeth Govind, Gert Jervan, Thomas Hollstein, and Jaan Raik. Extended checkers for logic-based distributed routing in network-on-chips. In *2014 14th Biennial Baltic Electronic Conference (BEC)*, pages 77–80. IEEE, 2014.

[2] Pietro Saltarelli, Behrad Niazmand, Jaan Raik, Ranganathan Hariharan, Gert Jervan, and Thomas Hollstein. A framework for comprehensive automated evaluation of concurrent online checkers. In *2015 Euromicro Conference on Digital System Design*, pages 288–292. IEEE, 2015.

[3] Pietro Saltarelli, Behrad Niazmand, Ranganathan Hariharan, Jaan Raik, Gert Jervan, and Thomas Hollstein. Automated minimization of concurrent online checkers for network-on-chips. In *2015 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, pages 1–8. IEEE, 2015.

[4] Pietro Saltarelli, Behrad Niazmand, Jaan Raik, Vineeth Govind, Thomas Hollstein, Gert Jervan, and Ranganathan Hariharan. A framework for combining concurrent checking and on-line embedded test for low-latency fault detection in noc routers. In *Proceedings of the 9th International Symposium on Networks-on-Chip*, page 6. ACM, 2015.

[5] Ranganathan Hariharan, Behrad Niazmand, and Jaan Raik. On fault detection efficiency of reliability checkers obtained by verification assertion qualification. In *RESCUE 2017 Workshop on Reliability, Security and Quality European Test Symposium (ETS) Fringe Workshop, May 25-26*. IEEE, 2017.

[6] Ranganathan Hariharan, Tara Ghasempouri, Behrad Niazmand, and Jaan Raik. From rtl liveness assertions to cost-effective hardware checkers. In *2018 Conference on Design of Circuits and Integrated Systems (DCIS)*, pages 1–6. IEEE, 2018.

[7] Pietro Saltarelli, Behrad Niazmand, Ranganathan Hariharan, Jaan Raik, Gert Jervan, and Thomas Hollstein. A framework for area-efficient concurrent online checkers design. In *MEDIAN 2015 Workshop on Manufacturable and Dependable Multicore Architectures at Nanoscale, November 10-11*, 2015.

[8] Way Kuo. Challenges related to reliability in nano electronics. *IEEE Transactions on Reliability*, 55(4):569–570, 2006.

[9] Elena Dubrova. *Fault-tolerant design*. Springer, 2013.

[10] L. Benini and G. De Micheli. Networks on chips: a new soc paradigm. *Computer*, 35(1):70–78, Jan 2002.

[11] Laung-Terng Wang, Cheng-Wen Wu, and Xiaoqing Wen. *VLSI test principles and architectures: design for testability*. Elsevier, 2006.

[12] Richard D Eldred. Test routines based on symbolic logical statements. *Journal of the ACM (JACM)*, 6(1):33–37, 1959.

[13] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints on test data selection: Help for the practicing programmer. *Computer*, 11(4):34–41, April 1978.

[14] A. J. Offutt, G. Rothermel, and C. Zapf. An experimental evaluation of selective mutation. In *Proceedings of 1993 15th International Conference on Software Engineering*, pages 100–107, May 1993.

[15] Pong P Chu. *RTL hardware design using VHDL: coding for efficiency, portability, and scalability*. John Wiley & Sons, 2006.

[16] Tanguy Risset. *SoC (System on Chip)*, pages 1837–1842. Springer US, Boston, MA, 2011.

[17] Santanu Kundu. *Network-on-Chip: The Next Generation of System-on-Chip Integration*. CRC Press, July 2017.

[18] J. Flich and J. Duato. Logic-based distributed routing for nocs. *IEEE Computer Architecture Letters*, 7(1):13–16, Jan 2008.

[19] Zhizhou Fu and Xiang Ling. The design and implementation of arbiters for network-on-chips. In *2010 2nd International Conference on Industrial and Information Systems*, volume 1, pages 292–295, July 2010.

[20] I. Miro Panades and A. Greiner. Bi-synchronous fifo for synchronous circuit communication well suited for network-on-chip in gals architectures. In *First International Symposium on Networks-on-Chip (NOCS'07)*, pages 83–94, May 2007.

[21] Harry Foster, David Lacey, and Adam Krolnik. *Assertion-Based Design*. Kluwer Academic Publishers, Norwell, MA, USA, 2 edition, 2003.

[22] T. Ghasempouri and G. Pravadelli. On the estimation of assertion interestingness. In *2015 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 325–330, Oct 2015.

[23] Pang-Ning Tan, Vipin Kumar, and Jaideep Srivastava. Selecting the right In *ACM SIGKDD*, pages 32–41, 2002.

[24] AL Oliveira and CM Antunes. Temporal data mining: An overview. In *KDD Workshop on Temporal Data Mining*, 2001.

[25] T. Ghasempouri, S. Payandeh Azad, B. Niazmand, and J. Raik. An automatic approach to evaluate assertions' quality based on data-mining metrics. In *2018 IEEE International Test Conference in Asia (ITC-Asia)*, pages 61–66, Aug 2018.

[26] Pang-Ning Tan and Vipin Kumar. Interestingness measures for association patterns: A perspective. In *Proc. of Workshop on Postprocessing in Machine Learning and Data Mining*, 2000.

[27] S. Murali, T. Theocharides, N. Vijaykrishnan, M. J. Irwin, L. Benini, and G. De Micheli. Analysis of error recovery schemes for networks on chips. *IEEE Design Test of Computers*, 22(5):434–442, Sep. 2005.

[28] R. Abdel-Khalek, R. Parikh, A. DeOrio, and V. Bertacco. Functional correctness for cmp interconnects. In *2011 IEEE 29th International Conference on Computer Design (ICCD)*, pages 352–359, Oct 2011.

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

[30] D. Bertozzi, L. Benini, and G. De Micheli. Low power error resilient encoding for on-chip data buses. In *Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition*, pages 102–109, March 2002.

[31] Praveen K Samudrala, Jeremy Ramos, and Srinivas Katkoori. Selective triple modular redundancy (stmr) based single-event upset (seu) tolerant synthesis for fpgas. *IEEE transactions on Nuclear Science*, 51(5):2957–2969, 2004.

[32] Jay M Berger. A note on error detection codes for asymmetric channels. *Information and Control*, 4(1):68–73, 1961.

[33] Bose and Der Jei Lin. Systematic unidirectional error-detecting codes. *IEEE Transactions on Computers*, C-34(11):1026–1032, Nov 1985.

[34] D. Das and N. A. Touba. Synthesis of circuits with low-cost concurrent error detection based on bose-lin codes. In *Proceedings. 16th IEEE VLSI Test Symposium (Cat. No.98TB100231)*, pages 309–315, April 1998.

[35] K. Mohanram, E. S. Sogomonyan, M. Gossel, and N. A. Touba. Synthesis of low-cost parity-based partially self-checking circuits. In *9th IEEE On-Line Testing Symposium, 2003. IOLTS 2003.*, pages 35–40, July 2003.

[36] S. Ghosh, N. A. Touba, and S. Basu. Synthesis of low power ced circuits based on parity codes. In *23rd IEEE VLSI Test Symposium (VTS'05)*, pages 315–320, May 2005.

[37] R. Sharma and K. K. Saluja. An implementation and analysis of a concurrent built-in self-test technique. In *[1988] The Eighteenth International Symposium on Fault-Tolerant Computing. Digest of Papers*, pages 164–169, June 1988.

[38] P. Drineas and Y. Makris. Concurrent fault detection in random combinational logic. In *Fourth International Symposium on Quality Electronic Design, 2003. Proceedings.*, pages 425–430, March 2003.

[39] Raimund Ubar and Jaan Raik. Testing strategies for networks on chip. In *Networks on chip*, pages 131–152. Springer, 2003.

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

[41] K. Petersen and J. Oberg. Toward a scalable test methodology for 2d-mesh network-on-chips. In *2007 Design, Automation Test in Europe Conference Exhibition*, pages 1–6, April 2007.

[42] J. Raik and V. Govind. Low-area boundary bist architecture for mesh-like network-on-chip. In *2012 IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, pages 95–100, April 2012.

[43] K. Nepal, N. Alves, J. Dworak, and R. I. Bahar. Using implications for online error detection. In *2008 IEEE International Test Conference*, pages 1–10, Oct 2008.

[44] C. Grecu, A. Ivanov, R. Saleh, E. S. Sogomonyan, and Partha Pratim Pande. On-line fault detection and location for noc interconnects. In *12th IEEE International On-Line Testing Symposium (IOLTS'06)*, pages 6 pp.–, July 2006.

[45] A. Dalirsani, M. A. Kochte, and H. Wunderlich. Area-efficient synthesis of fault-secure noc switches. In *2014 IEEE 20th International On-Line Testing Symposium (IOLTS)*, pages 13–18, July 2014.

[46] Dongkook Park, C. Nicopoulos, Jongman Kim, N. Vijaykrishnan, and C. R. Das. Exploring fault-tolerant network-on-chip architectures. In *International Conference on Dependable Systems and Networks (DSN'06)*, pages 93–104, June 2006.

[47] Q. Yu, M. Zhang, and P. Ampadu. Exploiting inherent information redundancy to manage transient errors in noc routing arbitration. In *Proceedings of the Fifth ACM/IEEE International Symposium*, pages 105–112, May 2011.

[48] Q. Yu, J. Cano, J. Flich, and P. Ampadu. Transient and permanent error control for high-end multiprocessor systems-on-chip. In *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, pages 169–176, May 2012.

[49] A. Alaghi, N. Karimi, M. Sedghi, and Z. Navabi. Online noc switch fault detection and diagnosis using a high level fault model. In *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT 2007)*, pages 21–29, Sep. 2007.

[50] R. Parikh and V. Bertacco. Formally enhanced runtime verification to ensure noc functional correctness. In *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 410–419, Dec 2011.

[51] A. Prodromou, A. Panteli, C. Nicopoulos, and Y. Sazeides. 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*, pages 60–71, Dec 2012.

[52] Giorgos Dimitrakopoulos and Emmanouil Kalligeros. Low-cost fault-tolerant switch allocator for network-on-chip routers. In *Proceedings of the 2012 Interconnection Network Architecture: On-Chip, Multi-Chip Workshop*, pages 25–28. ACM, 2012.

[53] J. Raik, V. Govind, and R. Ubar. An external test approach for network-on-a-chip switches. In *2006 15th Asian Test Symposium*, pages 437–442, Nov 2006.

[54] J. Raik, V. Govind, and R. Ubar. Design-for-testability-based external test and diagnosis of mesh-like network-on-a-chips. *IET Computers Digital Techniques*, 3(5):476–486, Sep. 2009.

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

[56] Margit Aarna, Eero Ivask, Artur Jutman, Elmet Orasson, Jaan Raik, Raimund Ubar, V. Vislogubov, and Heinz-Dietrich Wuttke. Turbo tester – diagnostic package for research and training. 2003.

[57] A. Jutman A. Peder J. Raik M. Tombak R. Ubar Tallinn. Structurally synthesized binary decision diagrams. 2004.

[58] N. Alves, Y. Shi, J. Dworak, R. I. Bahar, and K. Nepal. Enhancing online error detection through area-efficient multi-site implications. In *29th VLSI Test Symposium*, pages 241–246, May 2011.

[59] M. Boule, J. Chenard, and Z. Zilic. Assertion checkers in verification, silicon debug and in-field diagnosis. In *8th International Symposium on Quality Electronic Design (ISQED'07)*, pages 613–620, March 2007.

[60] Bowen Alpern and Fred B Schneider. Recognizing safety and liveness. *Distributed computing*, 2(3):117–126, 1987.

[61] Armin Biere, Cyrille Artho, and Viktor Schuppan. Liveness checking as safety checking. In *7th Int. Workshop on Formal Methods for Industrial Critical Systems (FMICS'02)*, volume 66, 2002.

[62] Viktor Schuppan and Armin Biere. Efficient reduction of finite state model checking to reachability analysis. *International Journal on Software Tools for Technology Transfer*, 5(2):185–204, Mar 2004.

[63] Alessandro Danese, Francesca Filini, Tara Ghasempouri, and Graziano Pravadelli. Automatic generation and qualification of assertions on control signals: A time window-based approach. In Youngsoo Shin, Chi Ying Tsui, Jae-Joon Kim, Kiyoung Choi, and Ricardo Reis, editors, *VLSI-SoC: Design for Reliability, Security, and Low Power*, pages 193–221, Cham, 2016. Springer International Publishing.

[64] S. Hertz, D. Sheridan, and S. Vasudevan. Mining hardware assertions with guidance from static analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(6):952–965, June 2013.

[65] P. K. Nalla, R. K. Gajavelly, H. Mony, J. Baumgartner, and R. Kanzelman. Effective liveness verification using a transformation-based framework. In *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*, pages 74–79, Jan 2014.

[66] Yael Abarbanel, Ilan Beer, Leonid Gluhovsky, Sharon Keidar, and Yaron Wolfsthal. Focs–automatic generation of simulation checkers from formal specifications. In *International Conference on Computer Aided Verification*, pages 538–542. Springer, 2000.

[67] F. Corno, G. Cumani, M. Sonza Reorda, and G. Squillero. An rt-level fault model with high gate level correlation. In *Proceedings IEEE International High-Level Design Validation and Test Workshop (Cat. No.PR00786)*, pages 3–8, Nov 2000.

[68] M. Brera, F. Ferrandi, D. Sciuto, and F. Fummi. Increase the behavioral fault model accuracy using high-level synthesis information. In *Proceedings 1999 IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (EFT'99)*, pages 174–180, Nov 1999.

[69] T. Chakraborty and S. Ghosh. On behavior fault modeling for combinational digital designs. In *International Test Conference 1988 Proceeding@m_New Frontiers in Testing*, pages 593–600, Sep. 1988.

[70] Viktor Schuppan and Armin Biere. Liveness checking as safety checking for infinite state spaces. *Electronic Notes in Theoretical Computer Science*, 149(1):79–96, 2006.

[71] K. Morin-Allory and D. Borrione. Proven correct monitors from psl specifications. In *Proceedings of the Design Automation Test in Europe Conference*, volume 1, pages 1–6, March 2006.

[72] Ieee standard for systemverilog–unified hardware design, specification, and verification language. *IEEE Std 1800-2017 (Revision of IEEE Std 1800-2012)*, pages 1–1315, Feb 2018.

[73] Synopsys design compiler. `http://www.synopsys.com/`, 1994.

[74] H.D.K. Foster and D.J. Lacey. *Assertion-based design 2nd edition*. Springer, 2004.

[75] W. Li, A. Forin, and S. A. Seshia. Scalable specification mining for verification and diagnosis. In *Design Automation Conference*, pages 755–760, June 2010.

[76] Lingyi Liu and Shobha Vasudevan. Automatic generation of system level assertions from transaction level models. *Journal of Electronic Testing*, 29(5):669–684, 2013.

[77] S. P. Azad, B. Niazmand, A. K. Sandhu, J. Raik, G. Jervan, and T. Hollstein. Automated area and coverage optimization of minimal latency checkers. In *2017 22nd IEEE European Test Symposium (ETS)*, pages 1–2, May 2017.

[78] A. Danese, T. Ghasempouri, and G. Pravadelli. Automatic extraction of assertions from execution traces of behavioural models. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 67–72, March 2015.

[79] M. Bertasi, G. Di Guglielmo, and G. Pravadelli. Automatic generation of compact formal properties for effective error detection. In *2013 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 1–10, Sep. 2013.

[80] Sagi Katz, Orna Grumberg, and Danny Geist. " have i written enough properties?"-a method of comparison between specification and implementation. In *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, pages 280–297. Springer, 1999.

[81] Y. Hoskote, T. Kam, Pei-Hsin Ho, and Xudong Zhao. Coverage estimation for symbolic model checking. In *Proceedings 1999 Design Automation Conference (Cat. No. 99CH36361)*, pages 300–305, June 1999.

[82] N. Jayakumar, M. Purandare, and F. Somenzi. Do's and don'ts of ctl state coverage estimation. In *Proceedings 2003. Design Automation Conference (IEEE Cat. No.03CH37451)*, pages 292–295, June 2003.

[83] A. Fedeli, F. Fummi, and G. Pravadelli. Properties incompleteness evaluation by functional verification. *IEEE Transactions on Computers*, 56(4):528–544, April 2007.

[84] `https://www.synopsys.com/verification/simulation/certitude.html`.

[85] Pang-Ning Tan, Vipin Kumar, and Jaideep Srivastava. Selecting the right interestingness measure for association patterns. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 32–41, New York, NY, USA, 2002. ACM.

Acknowledgements

I would like to express my gratitude to my supervisor Prof. Jaan Raik for guiding me throughout the PhD studies. I would also like to thank my co-supervisor Dr. Tara Ghasempouri. I have always appreciated their support, comments and open-door approach. I could not have imagined having a better supervisors for my PhD study. It has been a great pleasure to do research work with them.

Special thanks to Dr. Margus Kruus, the Head of Department of Computer Systems for his support with many administrative issues.

I would also like to thank all the people in Department of Computer Systems who helped me during my PhD studies. I would like to express special thanks to my colleague Behrad Niazmand. Also, I would thank my friends for their valuable support.

Furthermore, I like to acknowledge the organizations that have supported my PhD studies: Tallinn University of Technology, Estonian IT Academy program, EU's H2020 RIA IMMORTAL, the Estonian Center of Excellence in IT (EXCITE) and EU's Twinning Action TUTORIAL project.

Finally, I would like to thank my family: my parents Padmavathi and Hariharan, my brother Subramaniyan and his family, my in-laws, whose love and guidance are with me in whatever I pursue. Most importantly, I wish to thank my loving wife, Neeraja and our sweet daughter Ankita for supporting and providing unending inspirations.

Abstract

Cost-Effective Concurrent Hardware Checkers for Network on Chip based System on Chip

Extreme down-scaling of semi-conductor technologies causes a rapid increase of life-time issues in digital circuits. Consequently, detecting faults at run-time is becoming imperative. An on-line fault detection mechanism aims to monitor the digital circuits at run-time and detect the undesired behavior while the device is in operation. This kind of online fault detection can be achieved with the help of concurrent hardware checkers. However, designing checkers by hand can be a tedious and error-prone task.

At the same time the complexity of integrated circuits is growing and the underlying architectures have been moving towards multi-/many-core and System-on-Chip (SoC) paradigms. With the number of cores increasing, the on-chip communication efficiency has become one of the bottlenecks determining the overall system performance and cost. A packet based, on-chip intercommunication network known as Network on Chip (NoC) is emerging as an alternative solution to address the increasing interconnect complexity. However, NoC based interconnects, because of advanced router architectures, complex operation and concurrent communication are highly susceptible to faults during the runtime of the system. Without taking an appropriate run-time solution to ensure that such faults do not affect the operation of NoCs based interconnects, there could be possibility of data getting misrouted, dropped, corrupted, deadlocked or even several on-chip communication performance degradation.

To address the above issues, this thesis proposes a methodology for producing a set of cost-effective concurrent checkers from verification assertions. It is known that the number of assertions is generally too high to allow for area-efficient checking infrastructure. Therefore, there is a need for qualification and minimization of assertions with a prospect of reusing them as hardware checkers. To derive low-area, high fault coverage hardware checkers from many assertions, this thesis proposes a framework for selecting a set of high-quality and minimized assertions by combining a data mining technique with the fault analysis approach along with an assertion conversion methodology that converts liveness assertions into safety assertions. The framework then synthesizes these safety assertions into hardware checkers to be evaluated at the gate level to provide a cost-effective checking infrastructure.

Experimental results evaluating the methodology proposed in this thesis show that it is capable of synthesizing checker circuitry whose area overhead lies in a 60-80% range while guaranteeing 100% of single-event transient fault coverage. This is by far more area efficient than what is required by the traditional duplication and triplication based fault tolerant architectures. Moreover, a hybrid solution combining concurrent checkers with online test packets can further minimize the requirements of the area overhead down to less than 2 percents.

Kokkuvõte
Kulutõhusad süsteemiga paralleelsed rikkemonitorid kiipvõrkudel põhinevatele kiipsüsteemidele

Pooljuhtide tehnoloogiate ekstreemne miniaturiseerimine on põhjustanud digitaalsüsteemide eluea jooksul toimuvate rikete plahvatusliku kasvu. Seega on häirete avastamine seadme eluea jooksul hädavajalik. Rikke tuvastamise mehhanismi eesmärk on jälgida digitaalsüsteeme ja tuvastada soovimatu käitumine seadme töötamise ajal. Sellist tõrke tuvastamist on võimalik saavutada süsteemiga paralleelsete rikkemonitoride abil. Kuid sarnaste monitoride projekteerimine käsitsi on üldjuhul aeganõudev ja vea-aldis tegevus.

Samal ajal on kasvanud integraallülituste keerukus ja selle aluseks olevad arhitektuurid on liikunud mitme- ja paljutuumaliste süsteemide ning kiipsüsteemide paradigmade suunas. Tuumade arvu kasvades on kiibi kommunikatsioon muutunud üheks kitsaskohtaks, mis määrab süsteemi üldise jõudluse ja maksumuse. Paketipõhine kiibil olev sidevõrk, mida tuntakse kui kiipvõrku, on kujunemas alternatiivseks lahenduseks üha suureneva ühenduste keerukuse lahendamiseks. Kuid kiipvõrgu-põhised ühendused on tänu keerukatele ruuteri arhitektuuridele muutunud äärmiselt tundlikeks rikete suhtes. Adekvaatsete, seadme töö ajal rakendatavate lahenduste puudumine võib põhjustada andmete ruutimisvigu, kadunud või rikutud andmepakette, võrgu ummikseisu või kiibi kommunikatsiooni jõudluse halvenemist.

Ülaltoodud küsimuste lahendamiseks pakutakse käesolevas töös välja metoodika verifitseerimisväidetest kulu-efektiivsete rikkemonitoride sünteesiks. On teada, et väidete arv on üldjuhul liiga suur, et võimaldada kompaktsete monitoride väljatöötamist. Seetõttu on tarvis verifitseerimise väiteid eelnevalt minimeerida, et neid riistvaramonitoridena taaskasutada. Madala pindala, suure rikete kattega riistvaraliste monitoride sünteesiks paljudest veriftseerimise väidetest pakub väitekiri välja raamistiku kvaliteetsete ja minimeeritud väidete väljavalimiseks, kombineerides andmekaevandustehnikat veaanalüüsi meetodiga. Seejärel sünteesib raamistik saadud minimiseertud väidete hulga riistvaralisteks rikkemonitorideks, mida hinnatakse loogikalülituste tasemel, et tagada kulutõhus rikete monitooring.

Käesolevas töös välja pakutud metoodika hindamiseks läbi viidud eksperimendid näitavad, et see on võimeline sünteesima rikkemonitooringu süsteemi, mille pindala lisavajadus on 60-80% , tagades samas 100% transientsete rikete katvuse. Seega on pindala lisavajadus oluliselt väiksem traditsioonilisest dubleerimisel ja kolmekordistamisel põhinevatest tõrkekindlatest arhitektuuridest. Peale selle võimaldab dissertatsioonis välja pakutud hübriidlahendus, mis ühendab riistvaralisi rikkemonitore online-testipakettidega veelgi vähendada pindala nõudeid viies need vähem kui 2 protsendini.

Appendix 1

Publication I

Behrad Niazmand, Ranganathan Hariharan, Vineeth Govind, Gert Jervan, Thomas Hollstein, and Jaan Raik. Extended checkers for logic-based distributed routing in network-on-chips. In *2014 14th Biennial Baltic Electronic Conference (BEC)*, pages 77–80. IEEE, 2014

Extended Checkers for Logic-Based Distributed Routing in Network-on-Chips

Behrad Niazmand, Ranganathan Hariharan, Vineeth Govind, Gert Jervan, Thomas Hollstein, Jaan Raik
Department of Computer Engineering
Tallinn University of Technology
Tallinn, Estonia
{bniazmand, ranga, vineeth, gert, thomas, jaan}@ati.ttu.ee

***Abstract*—Network on Chips (NoCs) are composed of routers, whose task is to dispatch packets within the communication network according to the routing algorithm implemented. However, the extreme scaling of emerging nanometer technologies makes the routers vulnerable to wear-out and environmental effects. In order to contain this issue, development of online testing capabilities for the NoC routers is a must. This paper proposes concurrent online checkers for structural faults in the NoC routing algorithms utilizing the Logic-Based Distributed Routing (LBDR) concept. We show by fault injection experiments that the fault coverage of existing checking mechanisms for LBDR faults is very low. We propose an extended set of concurrent checkers that increase the coverage more than threefold facilitating detection of the majority of structural faults within the LBDR.**

Keywords—network-on-a-chip, logic-based distributed routing, concurrent online checking.

I. INTRODUCTION

Network on Chips (NoCs) have emerged as a scalable and predictable alternative to the bus-based and ad-hoc interconnect seen in the System-on-a-Chip designs in the past. The NoCs consist of routers, whose task is to dispatch packets within the communication network according to the routing algorithm implemented.

However, the extreme scaling of new nanometer technologies makes the electronic systems vulnerable to wear-out (e.g. aging) and environmental effects (e.g. soft errors, electro-magnetic interference). These are issues occurring during the life time of the system and cannot be filtered out by manufacturing testing. Thus, online solutions for detecting faults during circuit life time are needed. Preferably, these solutions should be concurrent, i.e. functioning concurrently to the normal circuit operation.

This paper proposes concurrent online checkers targeting structural faults in the NoC routing algorithms utilizing the Logic-Based Distributed Routing (LBDR) concept [1]. We show by fault injection experiments that the fault coverage of existing checking mechanisms for LBDR faults is very low [2]. We propose an extended set of concurrent checkers that increase the coverage threefold facilitating detection of the majority of structural faults occurring within the LBDR. This comes at the expense of area overhead requirements, which however are less than having the circuit duplicated.

The paper is organized as follows. Section 2 presents an overview of related works. Section 3 explains the basic concept of LBDR. In Section 4, the concurrent online checkers for LBDR are presented. Section 5 provides the fault injection experiments. Finally, Section 6 concludes the paper.

II. RELATED WORKS

Several works on online checking for NoCs have been developed in the past. Grecu et al. have introduced a method for online fault detection and location in NoC communication fabrics [3], which is able to distinguish between faults in the communication links and the ones in NoC switches. This work is based on the utilization of code-disjoint routing elements, combined with parity check encoding for the inter-switch links. However, the method targets faults in the data part only.

In [4], an end-to-end error detection and recovery solution, SafeNoC, has been introduced for ensuring the functional correctness of CMP interconnects. In this solution, a lightweight checker network is added to the existing interconnect, that guarantees to deliver messages correctly. Therefore, for each data message, a look-ahead signature is transmitted over the checker network, which is used for detecting errors in the corresponding data message. The solution does not provide checking for faults within the routers.

Several works have proposed utilization of concurrent online checkers for checking router faults [5,6,2]. In [5], the authors propose a lightweight checker for faults in routing algorithm implementation. However, only faults manifesting themselves as erroneous routing to the local port are targeted. [6] proposes checkers synthesized from a set of 32 assertions. The checkers detect most of the injected faults. The faults that are not covered correspond to non-catastrophic failures.

Yu et al. [2] have proposed a set of checkers for the NoC routing algorithmic blocks implemented as LBDR. In this paper we extend the set of checkers in order to increase the fault coverage.

978-1-4673-9539-7/14/$31.00 ©2014 IEEE

14th Biennial Baltic Electronics Conference (BEC2014)
Tallinn, Estonia, October 6-8, 2014

III. LOGIC-BASED DISTRIBUTED ROUTING (LBDR)

The proposed approach combines the concepts used in [6] and [2] with the aim of introducing checker modules for different components related to the combinational logic of the routing algorithm. In this work we have utilized the concept mentioned in [1], which proposes a mechanism for implementing routing algorithms in form of combinational logic at each router in the network and not making use of routing tables at all. The proposed logic, named Logic-Based Distributed Routing (LBDR), relies on two sets of configuration bits referred to as the *connectivity bits* and the *routing bits*, respectively. The former describes how the network topology looks like, for example being a 2D mesh, torus, plus, d and P. The latter describes the limited turns in the routing algorithm, so for example based on the routing algorithm used, such as XY (Dimension-Ordered) routing, Segment-based routing, Turn-Model routing algorithms, the corresponding routing bits are set to specific values accordingly in each router.

The LBDR logic accepts as input the following [1]:

- The ID of the current router (which is stored in a register at the current router)
- The ID of the destination router (which is extracted from the header flit of the packet)
- The connectivity bits of the current router (4 bits in total for a 2D Mesh, corresponding to four main directions)
- The routing bits of the current router (8 bits)

Fig. 1 shows the logic for LBDR as proposed in [1].

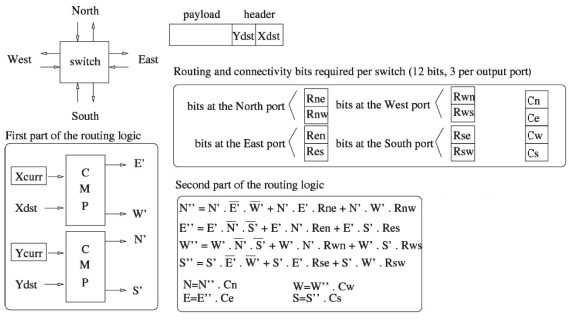


Fig. 1. Logic for LBDR

It is worth mentioning that since only the header flit of a packet includes the destination router ID then this is the flit that LBDR uses for setting the appropriate output port signal. The main LBDR logic (as depicted in [1]), provides a set of five output signals: N, E, W, S and L. That means, for example if N is set, then the flit can be forwarded to the North output port. For the North output port to be considered, the destination router can either be on the same column as the current router, or it can be located on the NE (North-East) quadrant with respect to the current router and the routing bit R_{ne} should allow for the turn from North to East at the next hop, or it can be located on the NW (North-West) quadrant with respect to the current router and the routing bit R_{nw} should allow for the turn from North to West at the next hop.

Depending on the routing algorithm used, one or two output signals of the LBDR logic can become active. In case of using Dimension-Ordered routing algorithms (such as XY routing), since there is always one output direction chosen as a candidate for forwarding the flit, only one output signal can be set to one in the LBDR logic. However, in case of adaptive routing algorithms (such as Turn Model routing algorithms), there might be a maximum of two candidate output ports, so two output signals can be set to one in the LBDR logic. In such cases, there is a possibility to utilize a selection function that can decide which output port should finally be chosen for forwarding the flit.

IV. CONCURRENT CHECKERS FOR THE LBDR

In [2], the comparator units (CMP) of the LBDR are implemented using subtractors. The output of subtraction Xcurr-Xdist is denoted as A and the output of subtraction Ycurr-Ydist is denoted as B, respectively.

Table I presents the properties implemented by checkers in the current paper. The first two properties ERR_{CMP} and $ERR_{LBDR\text{-}out}$ were introduced by [2]. As an extension, in this work, we have added new checkers to the LBDR logic, making it possible to detect faults in connectivity bits, faults in the value of Node IDs that do not exist and faults that are related to the router's local (L) port.

As mentioned in [2], the LBDR logic utilizes two sets of configuration bits for computing the appropriate output signal(s): namely the *connectivity bits* and the *routing bits*. The connectivity bits describe the network topology and indicate which output ports exist and which output ports do not exist or have become faulty and cannot be used for routing. According to the logic of LBDR whenever an output port is expected to be chosen as a candidate, its corresponding connectivity bit should also be set to one (the output port should actually exist), otherwise, the output signal for that port cannot be enabled. For instance, if the N output signal is set to one, according the LBDR's logic for N port, Cn (connectivity bit for the North port) should have been set to one, otherwise this indicates a fault. In order to capture faults that might occur regarding connectivity bits, we have made use of Equation (1) which describes the corresponding checker logic for this purpose:

$$(1)\quad ERR_{connectivity} = (N \,\&\, \sim Cn) \mid (E \,\&\, \sim Ce) \mid (W \,\&\, \sim Cw) \mid (S \,\&\, \sim Cs)$$

There are also some situations in which the destination node ID or the current node ID, which are inputs to the LBDR logic, might be influenced by faults and therefore the value of the node ID might change to an invalid number that either does not exist in the topology or its links are disabled. Such conditions might have effect on the local output signal of LBDR (L) and make it active inadvertently. However, the Local output signal in LBDR logic cannot become active when at least one of the *prime signals* (N', E', W' and S') has been set to one. Also, as long as the flit has not reached its destination, the local output signal (L) cannot be set to one.

TABLE I. CONCURRENT ONLINE CHECKERS FOR THE LBDR

Previous Work [2]		
$\mathbf{ERR_{CMP}}$	(E' & W') | (N' & S') | (~A & (E' | W')) | (A & ~E' & ~W') | (~B & (N' | S')) | (B & ~N' & ~S')	• Two opposite direction signals can not be set to one at the same time • Inconsistency should not exist between inside the CMP logic and its output signals (between Node A and E' and W' signals and between Node B and N' and S' signals)
$\mathbf{ERR_{LBDR\text{-}out}}$	(N & S) | (E & W)	• Two opposite output port signals can not be set to one at the same time
Our Proposal (Extension of Checker Logics)		
$\mathbf{ERR_{connectivity}}$	(N & ~C_n) | (E & ~C_e) | (W & ~C_w) | (S & ~C_s)	• The output port signal of a direction can not be set to one when the corresponding output link does not exist
$\mathbf{ERR_{Invalid\text{-}Node}}$	(iN | iE | iW | iS | iL) & (~N & C_n) & (~E & C_e) & (~W & C_w) & (~S & C_s) & ~L	• The Local output signal in LBDR logic can not become active when at least one of the Prime signals has been set to one. Also, as long as the flit has not reached its destination, the Local output signal can not be set to one.
$\mathbf{ERR_{Local}}$	((N' | E' | W' | S') & L) | ((iN | iE | iW | iS | iL) & ~N' & ~E' & ~W' & ~S' & ~L)	• The Local output signal in LBDR logic can not become active when at least one of the Prime signals has been set to one. Also, as long as the flit has not reached its destination, the Local output signal can not be set to one.

Equation (2) describes the checker logic introduced for detection of such conditions:

$$(2)\quad ERR_{Invalid\text{-}Node} = (iN \mid iE \mid iW \mid iS \mid iL)\ \&\ (\sim N\ \&\ Cn)\ \&\ (\sim E\ \&\ Ce)\ \&\ (\sim W\ \&\ Cw)\ \&\ (\sim S\ \&\ Cs)\ \&\ \sim L$$

Furthermore, if one of the output signals of the CMP units becomes enabled, that means the destination node does not have the same coordinates as the current node and therefore if the L output port of LBDR also becomes active, this signals a fault. In addition, if there is a request to the LBDR logic, but none of the output signals of the CMP neither the L output port of LBDR become active, this indicates an occurrence of fault as well. Equation (3) represents the checker logic proposed for detection of such conditions:

$$(3)\ ERR_{Local} = ((N' \mid E' \mid W' \mid S')\ \&\ L) \mid ((iN \mid iE \mid iW \mid iS \mid iL)\ \&\ \sim N'\ \&\ \sim E'\ \&\ \sim W'\ \&\ \sim S'\ \&\ \sim L)$$

Table I presents the checkers proposed by [2] as well as the additional checkers introduced by this paper.

V. SYNTHESIS AND FAULT INJECTION EXPERIMENTS

In order to evaluate the fault detection capabilities of the checkers proposed in the current work, fault injection experiments were carried out. An equivalent circuit consisting of an LBDR, its duplicate and the checkers' module was synthesized (see Fig. 2). The outputs of the LBDR and its duplication were connected to a miter circuit with an output signal *error_at_output*. This signal became 1 only in the case the fault injected to LBDR propagated to the circuit output. The outputs of all the checkers were OR-ed together to generate the signal *error_at_checker*. Finally, the two respective signals were AND-ed together to produce the primary output of the equivalence circuit. The faults were injected to the LBDR module only. The faults were detected by the equivalent circuit iff they propagated to the LBDR output AND they were detected by at least one of the checkers.

Table II presents the results of the fault injection experiments comparing the fault detection power as well as required area overhead of the work proposed in [2] and the proposed method. The experiments showed that the faults detectable in LBDR by the given set of checkers in [2] was very low, merely 21.6%. The checkers described in the previous Section implementing the conditions (1)-(3) allowed increasing the fault coverage more than threefold, i.e. up to 64.9%.

The area overhead with respect to the area of the LBDR in the case of the proposed checkers was higher than in [2]. However, the LBDR circuitry is forming only a small portion (1-2%) of the total router area. Therefore the required area overhead is small in terms of absolute area. Devising efficient checkers for the LBDR based routing circuitry is nevertheless imperative because it is a challenge to check for errors in the routing control as opposed to checking for errors in the data intensive parts of the router.

TABLE II. EXPERIMENTAL RESULTS

	[2]	Proposed
Fault coverage	21.6%	64.9%
Overhead area	26.8%	60.6%

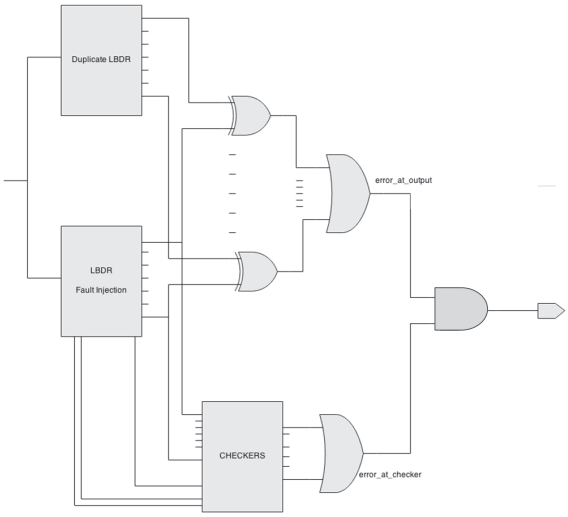


Fig. 2. An equivalent circuit for fault injection experiments

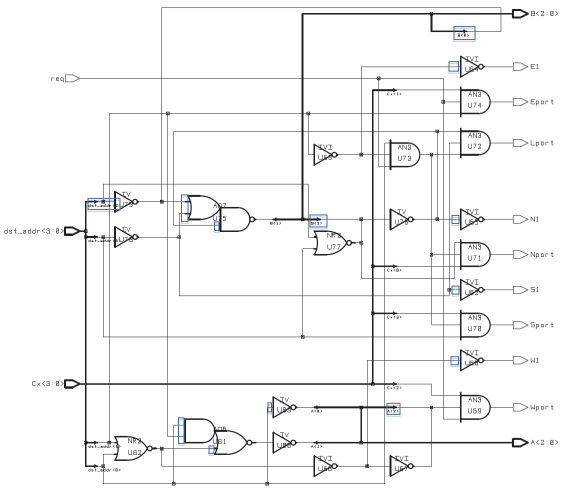


Fig. 3. Not covered faults in the LBDR schematic.

DISCUSSION OF LIMITATIONS

Figure 3 presents the schematic of the LBDR circuit, where lines with no faults covered have been marked by blue rectangles. As it can be seen from the Figure, none of the prime signals, i.e. N', W', E', S' (denoted by N1, W1, E1, S1, respectively) are detected by the checkers. In addition, signals related to the comparator output signals A and B are not detected. The latter may be addressed by developing LBDR architectures, where comparators are designed in a different manner.

CONCLUSIONS AND FUTURE WORK

The paper presented a set of checkers for concurrent online testing of both temporary and permanent faults in LBDR based routing logic of the network on chips. We proposed a set of five checkers which cover the majority of faults occurring in the LBDR circuitry of NoC routers.

Fault injection experiments showed that the proposed method allowed increasing the fault coverage 3 times with a still acceptable checker area overhead. In future work we foresee development of new checkers to increase the fault coverage as well as minimization of the checker set in order to save the required overhead area.

ACKNOWLEDGMENT

The work has been supported by EU FP7 STREP BASTION, Estonian institutional research grant IUT 19-1, research grants 8478, 9429, funded by Estonian Ministry of Education and Research, and by EU through the European Structural and Regional Development Funds.

REFERENCES

[1] J. Flich, J. Duato, Logic-Based Distributed Routing for NoCs, *IEEE Computer Architecture Letters*, Vol. 7, No. 1, January-June 2008.

[2] Yu, Qiaoyan; Cano, J.; Flich, J.; Ampadu, P., "Transient and Permanent Error Control for High-End Multiprocessor Systems-on-Chip," *2012 Sixth IEEE/ACM International Symposium on Networks on Chip (NoCS)*, vol., no., pp.169,176, 9-11 May 2012.

[3] C. Grecu, A. Ivanov, R. Saleh, E. S. Sogomonyan, P. P. Pande, On-line fault detection and location for NoC interconnects. *12th IEEE IOLTS 2006*, 6 pp., 10-12 July 2006.

[4] Abdel-Khalek, R.; Parikh, R.; DeOrio, A.; Bertacco, V., "Functional correctness for CMP interconnects," *2011 IEEE 29th International Conference on Computer Design (ICCD)*, pp. 352,359, 9-12 Oct. 2011.

[5] Alaghi, A.; Karimi, N.; Sedghi, M.; Navabi, Z., "Online NoC Switch Fault Detection and Diagnosis Using a High Level Fault Model," *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems, 2007. DFT '07.*, vol., no., pp.21,29, 26-28 Sept. 2007.

[6] Prodromou, A.; Panteli, A.; Nicopoulos, C.; Sazeides, Y., "NoCAlert: An On-Line and Real-Time Fault Detection Mechanism for Network-on-Chip Architectures," *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 60-71, 1-5 Dec. 2012.

Appendix 2

Publication II

Pietro Saltarelli, Behrad Niazmand, Jaan Raik, Ranganathan Hariharan, Gert Jervan, and Thomas Hollstein. A framework for comprehensive automated evaluation of concurrent online checkers. In *2015 Euromicro Conference on Digital System Design*, pages 288–292. IEEE, 2015

2015 Euromicro Conference on Digital System Design

A Framework for Comprehensive Automated Evaluation of Concurrent Online Checkers

Pietro Saltarelli[1,2], Behrad Niazmand[1], Jaan Raik[1], Ranganathan Hariharan[1], Gert Jervan[1], Thomas Hollstein[1]
[1]Tallinn University of Technology
Tallinn, Estonia
[2]Università degli Studi di Ferrara
Ferrara, Italy
pietro.saltarelli@student.unife.it
{bniazmand, jaan, ranga, gert, thomas}@ati.ttu.ee

***Abstract*— This paper proposes a framework for automated evaluation of concurrent online checkers. The novelty of the underlying approach lies in its completeness (i.e. ability of formally proving the presence or absence of true misses), minimal fault detection latency and accurate, fully automated evaluation of the fault detection characteristics of the checkers. The methodology consists of creating a pseudo-combinational version of the circuit under test, specifying the environment in terms of valid input stimuli and providing the assertions for generating the checkers, which will thereafter be evaluated by the framework. In this paper, a case-study on the control part (routing and arbitration) of a Network-on-Chip (NoC) router has been carried out. It shows on a realistic application that the framework is capable of accurately and formally evaluating the quality of individual concurrent checkers which constitutes an important task in fault tolerant system design. The case study shows that the proposed approach helps achieving high fault coverage in a single clock-cycle.**

Keywords- concurrent online checking, Network-on-Chip, routing logic, arbitration.

I. INTRODUCTION

Extreme scaling of nanometer technologies has made the electronic systems increasingly vulnerable to wear-out and environmental effects (e.g. soft errors, electro-magnetic interference). These are issues occurring during the life-time of the system and cannot be filtered out by manufacturing testing. Thus, online solutions for detecting faults are needed. These solutions should preferably be concurrent to the normal circuit operation.

One of the possible solutions for concurrent online test is the use of checkers for monitoring faults occurring within the circuit. In this paper, we introduce a framework for accurate, automated evaluation of concurrent online checkers. The methodology includes preparation of the checkers in the form of verification assertions (or reuse of existing assertions), creation of a pseudo-combinational version of the circuit under test and specifying the environment in terms of valid input stimuli for it.

Subsequently, the set of obtained checkers, together with the stimuli and the circuit are given to the framework that accurately evaluates the fault detection characteristics of the given checkers. The underlying approach in the framework is complete, i.e. it allows proving the absence or presence of true misses by the checkers. In addition, it provides minimal fault detection latency due to the fact that the circuit is transformed into a pseudo-combinational one and therefore only checkers with a single clock cycle latency are considered.

The proposed approach is applicable to control-oriented designs. In this paper, a case-study on the control part (routing and arbitration) of an NoC router has been carried out. It shows on a realistic application that the framework is capable of evaluating the quality of individual concurrent checkers which constitutes an important task in fault-tolerant system design.

The paper is organized as follows. Section 2 provides an overview of related works in concurrent online testing. Section 3 gives an overview of the concurrent online checking concept. In Section 4, the proposed framework and the corresponding methodology are presented. Section 5 presents the target architecture of the control part of a Network-on-Chip (NoC) router. Section 6 provides the checker evaluation experiments. Finally, Section 7 concludes the paper.

II. RELATED WORKS

Online detection of errors in logic is a thoroughly studied research area. Traditional Triple-Modular Redundancy (TMR) and duplication-based approach are too costly in terms of multiplying the area and correspondingly the power consumption. An alternative to minimize this overhead is the selective TMR that identifies Single Event Upset (SEU) sensitive sub-circuits that are to be protected [1].

In addition, there exists a variety of solutions based on coding techniques such as Berger [2] or Bose-Lin [3] codes. In many works the coding techniques are combined with synthesis [4,5]. The approaches suffer from significant area overhead as well as require alteration of the original circuit in order to generate the codes.

Concurrent on-line built-in self-test techniques such as Built-In Concurrent Self-Test (BICST) [6] and Reduced Observation Width Replication (ROWR) [7] provide high fault coverage at low area overhead but only consider a limited subset of pre-computed test vectors. Hence these approaches are likely to miss faults occurring in a normal circuit operation.

Several alternatives based on fault monitors and checkers that do not require modification of the circuit under test have been developed. Creating checkers automatically based on

978-1-4673-8035-5/15 $31.00 © 2015 IEEE
DOI 10.1109/DSD.2015.15

logic implications derived from the circuit structure [8] is feasible but suffers from low fault coverage and high area overhead, often exceeding the duplex solutions. On the other hand, deriving checkers from functional assertions, or reusing verification assertions, is similarly known to yield low coverage of structural faults as it is difficult to correlate functional coverage to structural one [9].

The framework and methodology presented in this paper exceed the existing state-of-the-art in concurrent online checking in the following aspects:

- It allows formally proving the absence or presence of true misses over all possible valid inputs for a checker, whereas in the case of traditional fault injection only statistical probabilities can be calculated without providing the user with full confidence of fault detection capabilities.
- The methodology targets the minimum fault detection latency of a single clock-cycle. This is achieved by representing the circuit under test as a pseudo-combinational design and concentrating on combinational checkers.
- The framework provides accurate, fully automated evaluation for the fault detection characteristics of the checkers. It allows finding cost-efficient trade-offs between the fault detection capabilities and the required overhead area.

III. THE CONCEPT OF CONCURRENT CHECKERS

Fig. 1 presents the role of concurrent on-line checkers in detecting faults within a circuit. In addition to the original circuit (functional logic), a set of checkers (checker logic) will be connected to functional inputs/outputs of the circuit. These checkers are introduced based on functional assertions derived from relationships between variables corresponding to inputs and outputs of the circuit. The checker logic targets the faults at lines within the functional logic (marked by green circles). The lines at the functional outputs succeeding the checker inputs (marked by a red cross) can not be detected by the checker. In addition, the checkers are not targeting the faults at functional inputs preceding checker inputs since the checker may not detect that the input value has been altered by a fault. (Such functional input lines are also marked by a red cross in Fig. 1). In this paper, we consider the single stuck-at fault model. However, due to the fact that concurrent checkers are implemented and a single time-frame is targeted, the model also covers timing related faults.

Given a fault at a line within the functional logic and a set of input stimuli, four possible scenarios may occur:

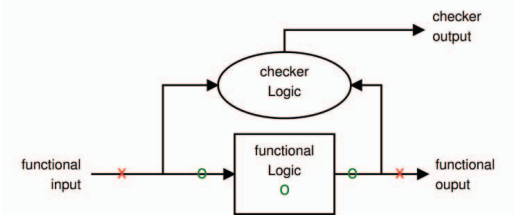


Figure 1. The concept of concurrent checking

- ***Case 1***: Fault occurs at an internal line and is visible at functional output(s) and checker logic flags a violation. The term *True Detection* is used to describe this situation, since a critical fault is effectively detected by the checker.
- ***Case 2***: Fault occurs at an internal line but is not visible at functional output(s). Checker catches the fault and flags a violation. The term *False Positive* is used to describe this situation. False positive is not harmful because an error is flagged which did not have any effect. However, it has negative impact on designs performance because normally it causes re-execution of the task. In the experiments in this paper we did not encounter any cases of false positives.
- ***Case 3***: Fault occurs at internal line but is not visible at functional output(s) and the checker logic does not detect the violation. The term *Benign Miss* is used to describe this situation. Benign miss shows correct operation by the checker.
- ***Case 4***: Fault occurs at internal node and is visible at functional output(s). Checker does not detect violation. The term *True Miss* is used to describe this situation, which is the worst possible case. True miss means that the fault propagates to the functional outputs and onwards to the system. However, the system has no information that a critical fault has occurred.

Traditionally, in order to evaluate the fault detection quality of the checkers, *fault injection* has been applied. Fault injection refers to injecting faults into a circuit at a certain time step and simulating it with the input stimuli to see whether any functional output of the circuit changes and whether any of the checker output fires. Due to the fact that it is generally impossible to inject and simulate all the faults at each circuit line at each time step, a statistically significant sample of random faults would normally be injected and simulated.

However, in this paper a methodology is proposed which is based on automated extraction of a pseudo-combinational circuit out of the original functional logic by breaking the flip-flops and converting them to pseudo-primary inputs and pseudo-primary outputs. Further, an exhaustive test for the extracted circuit is fed through a filtering tool in order to derive the exhaustive valid set of input stimuli which will serve as the environment for checker evaluation. This means that in this paper full evaluation of the checkers with all possible stimuli and faults is obtained.

Let D be the number of true detections, X be the number of benign misses and W be the number of true misses over all the injection runs. Then we define the metrics of *Fault Coverage* (*FC*) and *Checkers' Efficiency Index* (*CEI*) as follows.

$$CEI = \frac{D}{D + W} \tag{1}$$

$$FC = \frac{D + X}{D + X + W} \tag{2}$$

Here, FC shows the probability of the checkers behaving correctly over all possible cases and CEI shows the probability of checkers' ability to detect critical faults. As mentioned above, the approach proposed in this paper is able to formally prove the presence or absence of true misses. Due to the fact

that none of the checkers resulted in false positives, this information is excluded from the metrics.

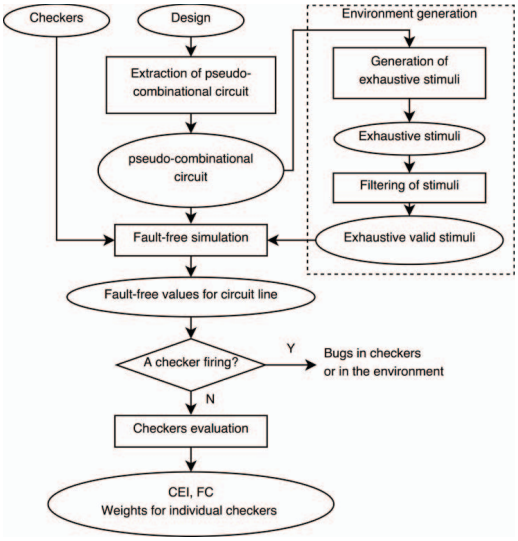


Figure 2. The proposed checker evaluation flow

IV. Framework for Checker Evaluation

Fig. 2 presents the flow of the checker evaluation framework together with the respective methodology. The flow starts with synthesizing the checkers (described in RTL Verilog) from a set of combinational assertions. Thereafter, a pseudo-combinational circuit will be extracted from the circuit of the design under checking. The pseudo-combinational circuit is derived out of the original circuit by breaking the flip-flops and converting them to pseudo-primary inputs and pseudo-primary outputs. Note, that at this point additional checkers that describe relations also on the pseudo-primary inputs/outputs may be added to the checker suite in order to increase the fault coverage.

Subsequently, the checker evaluation environment is created by generating exhaustive test stimuli for the extracted pseudo-combinational circuit. These stimuli are fed through a filtering tool which selects only the stimuli that correspond to functionally valid inputs for the circuit. As a result, the exhaustive valid set of input stimuli which will serve as the environment for checker evaluation is obtained.

The obtained environment, pseudo-combinational circuit and synthesized checkers are applied to fault free simulation. The simulation calculates fault free values for all the lines within the circuit. Additionally, if any of the checkers fires within fault simulation, it means a bug in the checker or an incorrect environment. During the case study presented in Section 5 several bugs were detected by this simulation step.

If none of the checkers are firing in the fault-free mode, then checker evaluation takes place. The tool injects faults to all the lines within all the vectors. As a result, the overall fault detection capabilities for the set of checkers, in terms of FC and CEI metrics will be calculated. In addition, each individual checker will be weighted by summing up the total number of true detections by the checker.

The framework is developed as an extension of a freeware test system Turbo Tester [10]. The system applies Structurally Synthesized Binary Decision Diagram (SSBDD) models [11] for circuit modelling.

V. Case-Study Design: NoC Router

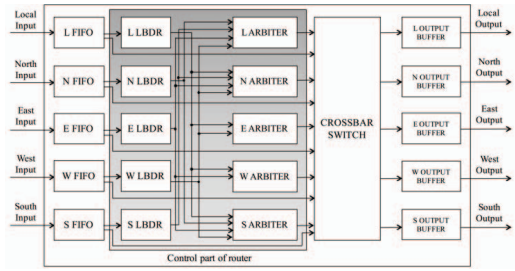


Figure 3. High level overview of an NoC router

Fig. 3 demonstrates the high-level overview of a 5-port 2D NoC router that we have chosen as a target architecture for applying the checkers. Mainly, the router consists of a data path and a control part. The data path is composed of input buffers (implemented as First-In-First-Out (FIFO)), one for each input port, a crossbar switch and an output buffer for each output port. The main responsibility of the data path is to transmit actual data to destination.

The flow of data through the datapath is managed and controlled by the control part, which consists of a routing computation unit for each input port and an arbitration unit (arbiter) for each output port, which prioritizes the requests from different input ports to the corresponding output port. The router has 5 input/output ports, four ports connected to four cardinal directions (North – N, East – E, South – S, West – W) and one Local (L) port connected to the local processing element. The NoC router utilizes wormhole switching. Therefore, packets are sent in form of flits, consisting of header flit, body flit(s) and tail flit.

For the routing computation unit of our target architecture, we have opted for Logic-Based Distributed Routing (LBDR) [12], which is considered as a scalable solution compared to routing tables. The mechanism describes the topology and the routing function in form of connectivity and routing bits, respectively. Therefore, the logic can be easily re-configured. Routing decision is distributed and only requires local and destination addresses for forwarding flits.

In this work we focus on a 2D Mesh topology and we consider XY as the routing algorithm, which is a deterministic dimension-ordered algorithm, and we assume that 180 degrees turns are not allowed. This would in turn lead to further simplification of the logic of LBDR. The basic mechanism of the logic is shown in Fig. 4, customized for the East input port.

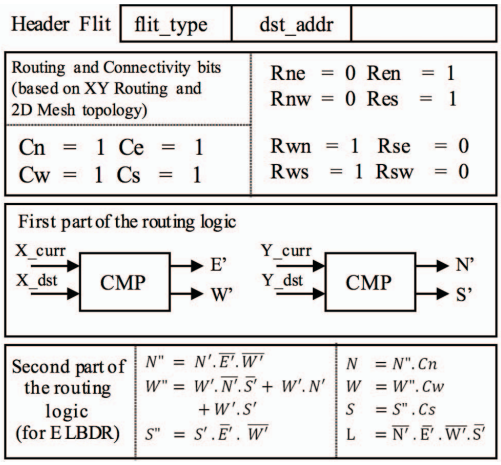


Figure 4. East input port LBDR (ELBDR)

For the arbitration unit (arbiter) we have chosen Round-Robin (RR) policy for prioritizing the requests from the routing logic of different input ports. Prioritization is circular, thus ensuring the absence of starvation, and guaranteeing that eventually any input port will get access to the requested output port.

Arbiter grants the access to the requesting input port winning the eventual contention, allowing data to go from the input FIFO to the corresponding output port, through the crossbar switch. The arbitration mechanism is based on an internal Finite State Machine (FSM). In this work one-hot encoding has been considered for the state variable, in order to improve detections of faults in the logic. Moreover, one-hot encoding is extended to grant signals and select lines for the crossbar switch.

The design decision to implement a one-hot encoded arbiter state machine versus a decimal encoded one did increase the area of the arbiter by 27.7%. However, the CEI nearly doubled from 58.55% to 100% and the fault coverage increased from 93.69% to 100%.

Extracting the pseudo-combinational circuit

In the control part of the router, we have limited our focus to the case in which the LBDR and arbiter logic have the most number of connected signals, more specifically considering ELBDR and SArbiter. Since for ELBDR the existing output port signals are N, W, S and L and for SArbiter, request and grant signals exist for N, E, W and L. Due to the routing algorithm and restrictions, other cases could have a smaller number of connections. The checkers that cover faults for such scenario, are symmetrical to the other cases (different connections between each LBDR logic to arbiter logics). The considered scenario for the connection between ELBDR and SArbiter is shown in Fig. 5. As it can be seen, connectivity and routing bits and also the current address are set to fixed values according to the scenario under consideration: 2D Mesh topology, XY routing algorithm, 180 degree turns not allowed and focusing on router with ID 5 in a 4x4 network. This scenario allows minimizing the number of circuit inputs and previous state values to be considered to as low as 22 bits:

- 2 flit type bits;
- 4 destination address bits;
- 4 ELBDR previous state bits;
- 3 SArbiter request signals bits;
- 4 SArbiter empty signals bits (from FIFOs);
- 5 SArbiter previous state bits.

This, in turn, makes the exhaustive approach in checker evaluation fully feasible.

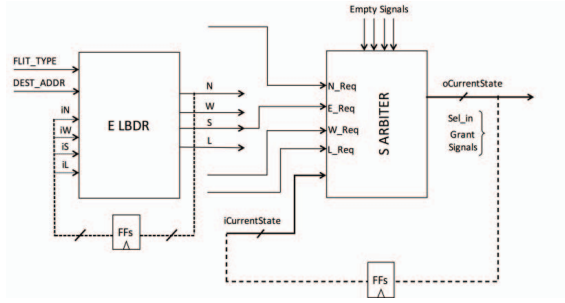


Figure 5. The pseudo-combinational circuit

Filtering the stimuli

The exhaustive test for the pseudo-combinational circuit would require 2^{22}=4,194,304 input stimuli. However, in order to minimize the stimuli, and more important, to avoid checkers being evaluated in non-realistic conditions, the exhaustive set of stimuli has to be filtered to contain only the functionally feasible values.

For the pseudo-combinational partial control part of an NoC router studied here the filtering step is based on the implemented routing algorithm and restrictions in the routing logic, as well as on invalid conditions for the state and the stimuli of the arbiter logic. Its use allowed us to shrink the exhaustive set of 2^{22} stimuli to a valid and complete set consisting of 40,960 input vectors, which is less than 1% of the initial number. It is important to stress the fact that none of the checkers fires in fault free simulation with any of the considered input stimuli.

Preparing the checkers

The set of checkers consists of 37 checkers, based on the functionality of the considered circuit, 3 of them focusing on the ELBDR logic, 34 (12 types) focusing on the SArbiter logic. Due to spatial limitations, we have not explicitly reported the list of checkers individually in this paper.

VI. EXPERIMENTAL RESULTS

Experiments for the checker evaluation framework were carried out on the circuit, set of checkers and test stimuli described in previous section. As a result, the Fault Coverage (FC) of 99.777% and the Checkers' Efficiency Index (CEI) of

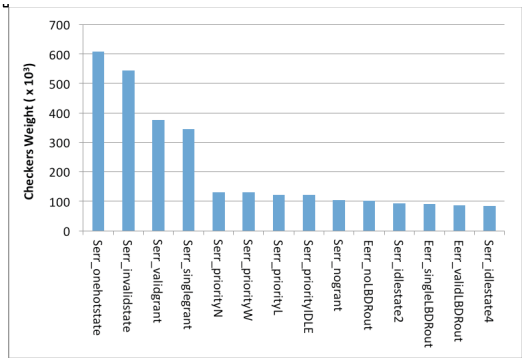


Figure 6. Checkers weighted according to true detections

99.320% were obtained. Each individual checker was weighted by the tool by summing up the total number of true detections by the checker. Fig. 6 lists the 14 checkers with the highest weights in a descending order. As it can be seen, four checkers are detecting considerably more faults than the others.

The proposed framework produced checkers that achieve 99.777% fault coverage for the NoC control part in a single clock cycle. Similar number for NoCAlert [14] was 97%, and the technique reached 100% after 28 cycles. The respective numbers for ForEVeR [13] were 30% and 11950 cycles. This gain in the proposed approach was achieved due to the facts that there were checkers devised for arbiter states and that the implemented state encoding was one-hot.

The area overhead of the initial set of checkers was relatively high, doubling the size of the LBDR and arbiter combined. However, the size of the checkers compared to the entire router area was negligible. Moreover, the checkers' weighting data allows to further compact the number of checkers in a straightforward manner.

VII. Conclusions and Future Work

The paper presented a framework for automated evaluation of concurrent online checkers, which is formal (able of proving the presence or absence of true misses), yields minimal fault detection latency and enables accurate, fully automated evaluation of the fault detection characteristics of a given set of checkers.

A case-study on the control part (routing and arbitration) of a Network-on-Chip (NoC) router showed on a realistic application the feasibility and efficiency of the framework and the underlying methodology. Experimental results also showed that the proposed approach allows reaching higher fault coverage within a single clock-cycle compared to previously published approaches.

As a future work we consider extending the framework with the support of temporal checkers in order to further increase the fault coverage, for those designs where pseudo-combinational extraction is either not feasible or not sufficient. In addition, we plan to develop algorithms for minimization of checkers based on the weights calculated by the proposed framework.

Acknowledgements

The work has been supported by EU's FP7 STREP BASTION, EU's H2020 RIA IMMORTAL, Estonian Science Foundation grant ETF9429, Estonian institutional research grant IUT 19-1, funded by Estonian Ministry of Education and Research, and by EU through the European Structural and Regional Development Funds.

References

[1] R. Sedmak and H. Liebergot. Selective triple modular redundancy (STMR) based single-event upset (SEU) tolerant synthesis for FPGAs. *IEEE Transactions on Nuclear Science*, 51:2957-2969, 2005.

[2] J. M. Berger. A note on an error detection code for asymmetric channels. *Information and Control*, 4:68-73, 1961.

[3] D. Das and N. A. Touba. Synthesis of circuits withlow-cost concurrent error detection based on Bose-Lin codes. *In VLSI Test Symposium*, pages 309-315, 1998.

[4] K. Mohanram, E. Sogomonyan, M. Gossel, and N. Touba. *Synthesis of low-cost parity-based partially self-checking circuits*, 2003.

[5] S. Ghosh, N.A. Touba, and S. Basu. Synthesis of low power ced circuits based on parity codes. *In VLSI Test Symposium*, pages 315-320, 1-5 May 2005.

[6] R. Sharma and K.K. Saluja. An implementation and analysis of a concurrent built-in self-test technique. *In Digest of Papers Eighteenth International Symposium on Fault-Tolerant Computing FTCS-18*, pages 164- 169, June 1988.

[7] P. Drineas and Y. Makris. Concurrent fault detection in random combinational logic. *In Proceedings Fourth International Symposium on Quality Electronic Design ISQED*, pages 425-430, March 2003.

[8] Alves, N.; Shi, Y.; Dworak, J.; Bahar, R.I.; Nepal, K. "Enhancing online error detection through area-efficient multi-site implications", *IEEE 29th VLSI Test Symposium (VTS)*, pp. 241 – 246, 2011.

[9] Marc Boule, Jean-Samuel Chenard, and Zeljko Zilic. Assertion checkers in verification, silicon debug and infield diagnosis. *In Proceedings of the ISQED '07*.

[10] M Aarna, E Ivask, A Jutman, E Orasson, J Raik, R Ubar, V Vislogubov, HD Wuttke. Turbo Tester-Diagnostic Package for Research and Training. *The 1st East-West Design and Test Conference*, Alushta, 2003.

[11] Artur Jutman, A Peder, J Raik, M Tombak, R Ubar. Structurally synthesized binary decision diagrams. *6th International Workshop on Boolean Problems.* pp. 271-278, 2004.

[12] J. Flich, J. Duato, Logic-Based Distributed Routing for NoCs, *IEEE Computer Architecture Letters*, Vol. 7, No. 1, January-June 2008.

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

[14] Prodromou, A.; Panteli, A.; Nicopoulos, C.; Sazeides, Y., "NoCAlert: An On-Line and Real-Time Fault Detection Mechanism for Network-on-Chip Architectures," *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 60-71, 1-5 Dec. 2012.

Appendix 3

Publication III

Pietro Saltarelli, Behrad Niazmand, Ranganathan Hariharan, Jaan Raik, Gert Jervan, and Thomas Hollstein. Automated minimization of concurrent online checkers for network-on-chips. In *2015 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, pages 1–8. IEEE, 2015

Automated Minimization of Concurrent Online Checkers for Network-on-Chips

Pietro Saltarelli[1,2], Behrad Niazmand[1], Ranganathan Hariharan[1], Jaan Raik[1], Gert Jervan[1], Thomas Hollstein[1]
[1]Tallinn University of Technology, Estonia
[2]Università degli Studi di Ferrara, Italy
pietro.saltarelli@student.unife.it
{bniazmand, jaan, ranga, gert, thomas}@ati.ttu.ee

***Abstract*— The paper introduces automated minimization of a set of concurrent online checkers for Network-on-Chips (NoCs) under given fault detection quality constraints. The proposed framework allows accurate and complete evaluation of the fault detection capabilities of checkers, which in turn enables finding seamless trade-offs between the overhead area of the checkers and the fault detection quality. The features of the automated minimization approach include formal proof for the absence or presence of true misses in checkers and a minimal fault detection latency. The minimization technique is based on a divide-and-conquer approach of partitioning the checkers' fault table into independent clusters. The checkers within the cluster are weighted and the set of checkers is minimized based on a heuristic method. Experiments on the control part (routing and arbitration) of an NoC router show that 100% fault coverage with very low overhead area will be achieved by the proposed minimization approach.**

Keywords—Network-on-Chip, routing logic, arbitration, concurrent online checking.

I. INTRODUCTION

Network-on-Chip (NoC) has been introduced as a solution to overcome the scalability and performance constraints of previous on-chip communication architectures such as bus-based networks. One of the challenges in the design of NoC routers is that as more cores get integrated on the same die and nanometer technologies get extremely scaled down, the probability of vulnerability of the components to wear-out and environmental effects increases. These are effects occurring during the life time of the system and cannot be filtered out by manufacturing testing. Thus, concurrent online fault monitors (i.e. *checkers*) for detecting faults during circuit's life time are needed. These checkers would report errors within routers and would allow reconfiguration of the routing infrastructure.

In this paper, we introduce an automated tool flow for obtaining a minimized list of checkers for checking on-chip communication architectures. The flow is based on accurate, automated evaluation of concurrent online checkers. The methodology includes preparation of the checkers in the form of verification assertions (or reuse of existing assertions), creation of a pseudocombinational version of the circuit under test and specifying the environment in terms of valid input stimuli for it. Subsequently, the set of the fault detection characteristics for the checkers, together with the stimuli and the circuit are applied to accurate evaluation. As a result, weights for individual checkers belonging to the set are obtained.

Finally, the number of checkers within the set will be minimized. The minimization technique is based on a divide-and-conquer approach of partitioning the checkers' fault table into independent clusters. Further, weight information of the checkers within the cluster is applied in a heuristic minimization method. The ultimate result will be a minimal selection of checkers to achieve a target fault coverage level.

The underlying approach is complete, i.e. it allows proving the absence or presence of true misses by the checkers. In addition, it provides minimal fault detection latency due to the fact that the circuit is transformed into a pseudo-combinational one and therefore only checkers with a single clock cycle latency are considered. Experiments on the control part (routing and arbitration) of a Network-on-Chip (NoC) router show that 100% fault coverage with very low overhead area will be achieved by the proposed minimization approach.

The paper is organized as follows. Section 2 provides an overview of related works in concurrent online testing. Section 3 explains the concurrent online checking concept. In Section 4, the automated tool flow and the corresponding methodology for checkers' minimization are presented. Section 5 presents the target architecture of the control part of an NoC router. Section 6 discusses application of the checker evaluation and minimization framework to the NoC Router design. Section 7 provides the checkers' evaluation and minimization experiments. Finally, Section 8 concludes the paper.

II. RELATED WORKS

Online detection of errors in logic is a thoroughly studied research area. Traditional Triple-Modular Redundancy (TMR) and duplication based approaches are too costly in terms of multiplying the area and correspondingly the power consumption. An alternative to minimize this overhead is the selective TMR that identifies Single Event Upset (SEU) sensitive sub-circuits that are to be protected [1].

In addition, there exists a variety of solutions based on coding techniques such as Berger [2] or Bose-Lin [3] codes. In many works the coding techniques are combined with synthesis [4,5]. The approaches suffer from significant area overhead as well as require alteration of the original circuit in order to generate the codes.

Concurrent on-line built-in self-test techniques such as Built-In Concurrent Self-Test (BICST) [6] and Reduced Observation Width Replication (ROWR) [7] provide high fault coverage at low area overhead but only consider a limited subset of pre-

978-1-4673-7942-7/15/$31.00 ©2015 IEEE

computed test vectors. Hence these approaches are likely to miss faults occurring in a normal circuit operation.

Several alternatives based on checkers that do not require modification of the circuit under test have been developed. Creating checkers automatically based on logic implications derived from the circuit structure [8] is feasible but suffers from low fault coverage and high area overhead, often exceeding the duplex solutions. On the other hand, deriving checkers from functional assertions, or reusing verification assertions, is similarly known to yield low coverage of structural faults as it is difficult to correlate functional coverage to structural one [9].

Many previous works have focused on addressing faults in the control logic of NoC routers. In [15], Yu et al. have addressed fault tolerance for NoC topologies and proposed an error control method for detecting transient errors in routing logic implemented using Logic-Based Distributed Routing (LBDR) mechanism and its extension for high-radix topologies, LBDRhr. The proposed error control method utilizes the inherent information redundancy (IIR) to reduce the error control overhead. However, the method does not guarantee full fault coverage.

Authors of [16] have presented a method for online error detection and diagnosis of NoC switches. The proposed method deals with routing faults that cause NoC packets to be forwarded to output ports that are not intended to. Regarding modeling routing faults in switches, a high-level fault model has been introduced in this work. The fault coverage is measured only at the functional level and there is no estimate of correlation to gate-level fault coverage.

Parikh et al. have proposed ForEVeR [13], where in order to deliver correctness guarantees for the complete network, a network-level detection and recovery solution is devised that monitors the traffic in the NoC and protects it against functional bugs that were not detected during design time. To this end, ForEVeR augments the baseline NoC with a lightweight checker network that alerts destination nodes of incoming packets ahead of time and is used for the recovery process. The approach suffers from extremely high latency. Only 30% of the faults will be detected during the first clock cycle by the approach.

[14] proposes checkers synthesized from a set of 32 verification assertions. The checkers detect most of the injected faults. The faults that are not covered correspond to non-catastrophic failures. The work proposed in [14] lacks the completeness and minimization aspects present in the current paper.

This paper exceeds the existing state-of-the-art in concurrent online checking by proposing a tool flow for automated evaluation and minimization of the verification checkers. We show that starting from a realistic set of verification assertions a minimal set of checkers will be synthesized that provide 100% fault coverage at a low area overhead and the minimum fault detection latency of a single clock-cycle. The latter is especially crucial for enabling rapid fault recovery in reliable real-time systems.

An additional feature of the proposed approach is that it allows formally proving the absence or presence of true misses over all possible valid inputs for a checker, whereas in the case of traditional fault injection only statistical probabilities can be calculated without providing the user with full confidence of fault detection capabilities.

The formal proof as well as the minimal fault detection latency will be guaranteed by reasoning on a pseudo-combinational version of the circuit and by the application of exhaustive valid set of input stimuli as the verification environment.

III. The Concept of Concurrent Checkers

Fig. 1 presents the role of concurrent on-line checkers in detecting faults within a circuit. In addition to the original circuit (functional logic), a set of checkers (checker logic) will be connected to functional inputs/outputs of the circuit. These checkers are derived based on functional assertions obtained from relationships between variables corresponding to inputs and outputs of the circuit. The checker logic targets the faults at lines at the inputs of each gate within the functional logic (marked by green circles). The lines at the functional outputs succeeding the checker inputs (marked by a red cross) cannot be detected by the checker. In addition, the checkers are not targeting the faults at functional inputs preceding checker inputs, since the checker may not detect that the input value has been altered by a fault (such functional input lines are also marked by a red cross in Fig. 1). In this paper, we consider the single stuck-at fault model. However, due to the fact that concurrent checkers are implemented and a single time-frame is targeted, the model also covers timing related faults.

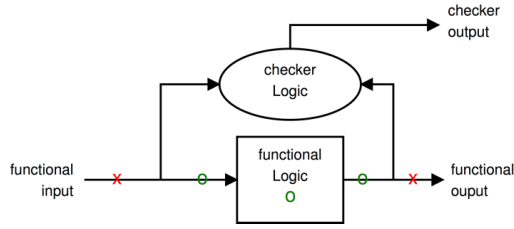


Fig. 1. The concept of concurrent checking

Given a fault at a line within the functional logic and a set of input stimuli, four possible scenarios may occur:

- ***Case 1***: Fault occurs at an internal line and is visible at functional output(s) and checker logic flags a violation. The term *True Detection* is used to describe this situation, since a critical fault is effectively detected by the checker.

- ***Case 2***: Fault occurs at an internal line but is not visible at primary output(s). Checker catches the fault and flags a violation. The term *False Positive* is used to describe this situation. False positive is not harmful because an error is flagged which did not have any effect. However, it has negative impact on design's performance because normally it causes re-execution of the task. In the experiments in this paper we did not encounter any cases of false positives.

- ***Case 3***: Fault occurs at internal line but is not visible at primary output(s) and the checker logic does not detect the violation. The term *Benign Miss* is used to describe this situation. Benign miss shows correct operation by the checker.

- ***Case 4***: Fault occurs at internal node and is visible at primary output(s). Checker does not detect violation. The term *True Miss* is used to describe this situation, which is the worst possible case. True miss means that the fault propagates to the functional outputs and onwards to the system. However, the system has no information that a critical fault has occurred.

Traditionally, in order to evaluate the fault detection quality of the checkers, *fault injection* has been applied. Fault injection refers to injecting faults into a circuit at a certain time step and simulating it with the input stimuli to see whether any functional output of the circuit changes and whether any of the checker output fires. Due to the fact that it is generally impossible to inject and simulate all the faults at each circuit line at each time step, a statistically significant sample of random faults would normally be injected and simulated.

However, in this paper a methodology is proposed which is based on automated extraction of a pseudo-combinational circuit out of the original functional logic by breaking the flipflops and converting them to pseudo primary inputs and pseudo primary outputs. Further, an exhaustive test for the extracted circuit is fed through a filtering tool in order to derive the complete valid set of input stimuli which will serve as the environment for checker evaluation. This means that in this paper full evaluation of the checkers with all the valid stimuli and faults is obtained.

Let D be the number of true detections, X be the number of benign misses and W be the number of true misses over all the injection runs. Then we define the metrics of *Fault Coverage* (*FC*) and *Checkers' Efficiency Index* (*CEI*) as follows.

$$FC = \frac{D + X}{D + X + W} \quad (1)$$

$$CEI = \frac{D}{D + W} \quad (2)$$

Here, FC shows the probability of the checkers behaving correctly over all possible fault cases while CEI shows the probability of checkers ability to detect critical faults. Due to the fact that none of the checkers resulted in false positives, this information is excluded from the metrics.

IV. Checkers Evaluation and Minimization Flow

Fig. 2 presents the evaluation and minimization flow for the checkers. The flow starts with synthesizing the checkers from a set of combinational assertions. Thereafter, a pseudo-combinational circuit will be extracted from the circuit of the design under checking. The pseudo-combinational circuit is derived out of the original circuit by breaking the flipflops and converting them to pseudo primary inputs and pseudo primary outputs. Note, that at this point additional checkers that also describe relations on the pseudo primary inputs/outputs may be added to the checker suite in order to increase the fault coverage.

Subsequently, the checker evaluation environment is created by generating exhaustive test stimuli for the extracted pseudo-combinational circuit. This stimuli are fed through a

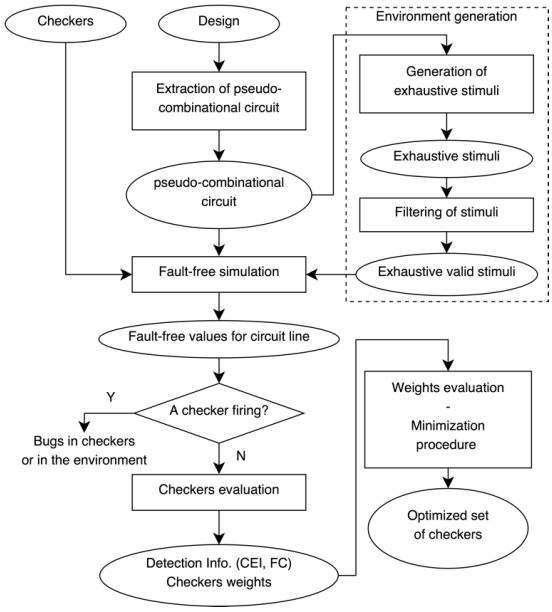


Fig. 2. Checkers' Evaluation and Minimization Flow

filtering tool that selects only the stimuli that correspond to functionally valid inputs of the circuit. As a result, the complete valid set of input stimuli that will serve as the environment for checker evaluation is obtained.

The obtained environment, pseudo-combinational circuit and synthesized checkers are applied to fault free simulation. The simulation calculates fault free values for all the lines within the circuit. Additionally, if any of the checkers fires during fault-free simulation it means a bug in the checker or an incorrect environment. During the case study presented in Section 5 several bugs were detected by this simulation step.

If none of the checkers is firing in the fault-free mode then checker evaluation takes place. The tool injects faults to all the lines within the circuit one-by-one and this step is repeated for each input vector. As a result, the overall fault detection capabilities for the set of checkers, in terms of FC and CEI metrics will be calculated. In addition, each individual checker will be weighted by summing up the total number of true detections by the checker.

Finally, the weighting information will be exploited in minimizing the number of checkers, eventually allowing to outline a trade-off between CEI, or FC, and the area overhead due to the introduction of checker logic.

The framework is developed as an extension of a freeware test system Turbo Tester [10]. The system applies Structurally Synthesized Binary Decision Diagram (SSBDD) models [11] for circuit modeling.

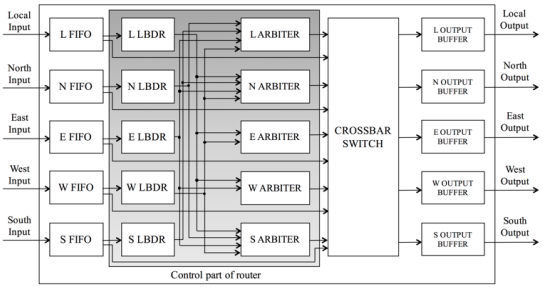


Fig. 3. High-level overview of an NoC router

V. Target Architecture: NoC Router

Fig. 3 demonstrates the high-level overview of a 5-port 2D NoC router that we have chosen as a target architecture for applying the checkers. Mainly, the router consists of a datapath and a control part. The datapath is composed of input buffers (implemented as First-In-First-Out (FIFO)), one for each input port, a crossbar switch and an output buffer for each output port.

The flow of data through the data path is managed and controlled by the control part, which consists of a routing computation unit for each input port and an arbitration unit (arbiter) for each output port, which prioritizes the requests from different input ports to the corresponding output port. The router has 5 input/output ports, four ports connected to four cardinal directions (North – N, East – E, South – S, West – W) and one Local (L) port connected to the local processing element. The NoC router utilizes wormhole switching. Therefore, packets are sent in form of flits, consisting of header flit, body flit(s) and tail flit.

For the routing computation unit of our target architecture, we have opted for Logic-Based Distributed Routing (LBDR) [12], which is considered as a scalable solution compared to routing tables. The mechanism describes the topology and the routing function in form of connectivity and routing bits, therefore the logic can be easily re-configured. Routing decision is distributed and only requires local and destination addresses for forwarding flits.

In this work we focus on a 2D Mesh topology, we consider XY as the routing algorithm, which is a deterministic dimension-ordered algorithm, and we assume that 180 degrees turns are not allowed. This would in turn lead to further simplification of the logic of LBDR. The basic mechanism of the logic is shown in Fig. 4, for instance for the East input port.

For the arbitration unit (arbiter) we have chosen Round-Robin (RR) policy for prioritizing the requests from the routing logic of different input ports. Prioritization is circular, thus ensuring the absence of starvation, and guaranteeing that eventually any input port will get access to the requested output port.

Arbiter grants the access to the requesting input port winning the eventual contention, allowing data to go from the input FIFO to the corresponding output port, through the crossbar switch. The arbitration mechanism is based on an internal Finite State Machine (FSM). In this work one-hot encoding has been considered for the state variable, in order to improve detections of faults in the logic. Moreover, one-hot encoding is extended to grant signals and select lines for the crossbar switch.

The design decision to implement a one-hot encoded arbiter state machine versus a decimal encoded one did increase the area of the arbiter by 27.7%. However, the CEI nearly doubled from 58.55% to 100% and the fault coverage increased from 93.69% to 100%, respectively.

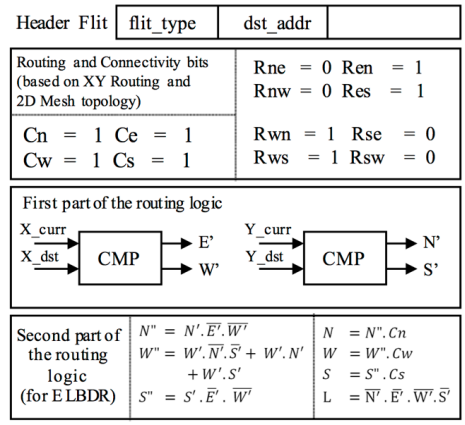


Fig. 4. Logic-based Distributed Routing (LBDR) logic for the East input port

VI. Application of the Framework to the Design

In the control part of the router, we have limited our focus to the case in which the LBDR and arbiter logic have the most number of connected signals, more specifically considering ELBDR and SArbiter. For ELBDR the existing output port signals are N, W, S and L and for SArbiter, request and grant signals exist for N, E, W and L. Such scenario provides the case with the most number of connectivities between LBDR and arbiter logic. The checkers that cover faults for such scenario, are symmetrical to the other cases (different connections between each LBDR logic to arbiter logics).

From the output of the checker evaluation tool it can be observed that the two set of checkers for the ELBDR and the SArbiter are *independent*, i.e. they cover faults for different and separate parts of the circuit, without any overlap. Therefore the fault table will be partitioned into two clusters. First, the ELBDR alone will be considered. Secondly, the circuit under study will be expanded, interconnecting the routing logic with the SArbiter. The second considered scenario is depicted in Fig. 5.

Connectivity and routing bits and also the current address are set to fixed values according to the scenario under consideration: 2D Mesh topology, XY routing algorithm, 180 degrees turns not allowed, focus on router with ID 5 in a 4x4 network. This scenario allows minimizing the number of circuit inputs and previous state input bits that together form the inputs for the pseudo-combinational circuit to be considered in both

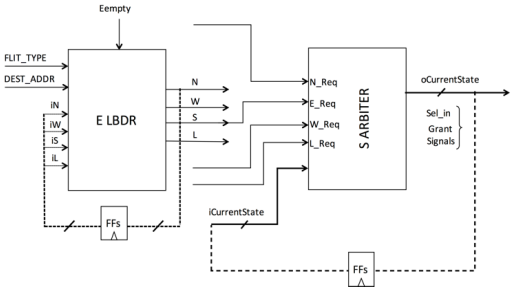


Fig. 5. The pseudo-combinational circuit for the full scenario

experiments. When ELBDR only is considered, the amount of inputs is limited to 11 bits:

- 2 flit identifier bits;
- 4 destination address bits;
- 4 ELBDR previous output values bits;
- 1 empty bit (coming from East input buffer).

With the interconnection to the SArbiter in the second experiment, the number of input bits is increased to 19, introducing:

- 3 SArbiter request signals bits;
- 5 SArbiter previous state bits.

This, in turn, makes the exhaustive approach in checker evaluation fully feasible.

Once the pseudo-combinational circuit to be studied is extracted, a set of checkers can be devised from the functional behaviour of the considered circuit, evaluating the possible implications existing in between input and output signals. It is interesting to underline that *a priori* it may be very difficult to outline the effectiveness of a single checker or the overlap of different checkers in detection.

Together with the considered pseudo-combinational circuit and its set of checkers, a set of input patterns is needed for performing fault simulation. The exhaustive test would require 2^{11}=2,048 and 2^{19}=524,288 input stimuli, respectively for the ELBDR and for the East-South control path experiments. However, in order to minimize the stimuli, and more important, to avoid checkers being evaluated in non-realistic conditions, the exhaustive set of stimuli has to be filtered to contain only the functionally feasible values.

The filtering step is based on the implemented routing algorithm (i.e. allowed destinations from the current router), restrictions in the routing logic (e.g. no 180 degrees turns) and emptiness condition of the input buffer, as well as on invalid conditions for the state of the arbiter logic (i.e. violation of one-hot encoding - only for the second experiment). It is important to stress the fact that none of the checkers is firing in fault free simulation with any of the considered input stimuli, in neither of the scenarios.

TABLE I. PROPOSED CHECKERS FOR ELBDR

Checkers for Routing Logic (LDBR)		
1	Valid LBDR output	If there is a request to the routing logic (the corresponding input buffer is not empty), LBDR has to compute at least one valid output direction (according to XY routing).
2	No LBDR output	If no flit arrives (the corresponding input buffer is empty), all the output port signals of LBDR should remain zero.
3	Single LBDR output	If the corresponding input buffer is not empty (there is a request to LBDR), because of using XY routing, at most only one output port signal of the LBDR logic can become active.
4	Switch LBDR output	If the corresponding input buffer is not empty (there is a request to LBDR) and a non-header flit has arrived, LBDR outputs should remain the same.
5	Local Port output	If the corresponding input buffer is not empty (there is a request to LBDR) and a header flit has arrived, the local output should become active only if the packet has reached its destination.

VII. EXPERIMENTAL RESULTS

Experiments for the checker evaluation and minimization framework were carried out on the scenarios described in previous section, first on the ELBDR circuit only, then on its interconnection with the SArbiter, as displayed in Fig. 5. In both cases an initial set of checkers was devised a priori, together with a filtering scheme to obtain a valid set of input stimuli. Each individual checker was weighted by the tool by summing up the total number of true detections by the checker, and this information was used in a heuristic way to minimize the initial set of checkers, with the final aim of achieving highest possible CEI and FC, and at the same time with the lowest possible area overhead. These quantities were evaluated iterating the fault

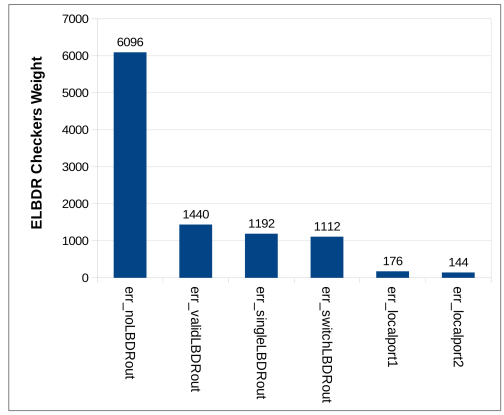


Fig. 6. Weights of checkers proposed for EBLDR

simulation, including at each step the next heaviest checker still not included in the currently considered set of checkers, initialized only with the first heaviest checker.

ELBDR experiment

All the experiments in this paper were carried out on an Asus ux32vd-r4002v computer with a 1.9 GHz Intel Core i7-3517U processor and 10 GB RAM. Table I lists the initial a priori set of checkers for ELBDR, devised from the functionality of the logic. The pseudo-combinational circuit for ELBDR has 11 input bits, as mentioned in the previous section, thus the exhaustive set of stimuli presents 2^{11}=2,048. A filtering scheme based on the following statements was devised:

- if input buffer's empty signal is high, any other input bit is meaningless, and therefore any value is allowed for it;
- if the incoming flit is a header, the destination address has to be valid according to the XY routing and turns restrictions;
- if the incoming flit is a body or tail flit, the previous output values must be valid, they must follow a one-hot fashion, according to XY routing.

This allowed to obtain a valid and complete set of stimuli consisting of 1536 vectors, which forms 75% of the exhaustive set. The run-time for generating the stimuli was 2 seconds.

Fig. 6 displays the weight information output of the tool, on the initial set of checkers for the ELBDR. The checker, *err_noLBDRout* (checker 2 in Table I) is considerably detecting more faults than any other checker. The 5 remaining checkers, in descending order of weights are *err_validLBDRout* (checker 1), *err_singleLBDRout* (checker 3), *err_switchLBDRout* (checker 4), and finally the two *err_localport* checkers (entry 5). The checkers' analysis required 10 ms of run-time from the proposed framework.

Fig. 7 depicts the results obtained with the weight-based greedy heuristic approach applied to the ELBDR and its initial set of checkers, in terms of achieved CEI, FC and area overhead. Considering at first only the heaviest weight, and adding at each step the next heaviest checker still not included in the considered set, all the quantities gradually increase. When the three most significant checkers are used, CEI and FC reach 100%. This result shows that minimization of the set of considered checkers is achieved, with the three heaviest checkers *dominating* the three lightest, i.e. the three considered checkers cover all the faults detected by the other checkers. Reducing the used set of checkers to the three most significant ones allows to limit the area overhead to 78.57% over the ELBDR circuit, far lower than 185.71% imposed by the initial non-minimized set of checkers.

TABLE II. PROPOSED CHECKERS FOR THE ARBITER LOGIC

Checkers for the Arbiter logic		
6	Valid Grant output	If there is a request from LBDR, arbiter has to assert at least one of the grant signals for the corresponding output direction.
7	No Grant output	If there is no request to the arbiter, it should not assert any of the grant signals for any direction.
8	Invalid Grant output	Whenever there is a request to the arbiter, the grant signals should go active corresponding to that specific requested direction and invalid direction should not be chosen.
9	Invalid arbiter output state	Output state variable (oScurrentState – which represents the grant signals) in arbiter's pseudo-combinational circuit can not possess invalid values due to the one-hot coding.
10	Invalid IDLE state for arbiter input state	If the input previous state variable (iScurrentstate) is in IDLE state and there is a request for arbitration from LBDR, oScurrentstate should not remain in IDLE state i.e. a grant signal should be asserted.
11	Priority Grant	In case there is one or multiple request(s) to the arbiter, it should follow the correct prioritization (Local, North, East and then West) according to the input previous state variable (iScurrentstate).

ELBDR + SArbiter combined scenario experiment

ELBDR is connected to SArbiter according to Fig. 5, thus providing the East request signal to the arbitrating logic. Table II lists the a priori initial set of checkers for the arbiter. Multiple individual checkers are grouped to the same table entry according to types. The initial set amounts to 28 checkers.

The exhaustive test for the considered pseudo-combinational circuit would require 2^{19}=524,288 input stimuli. The test stimuli were generated in 270 seconds run time. The considered filtering scheme is an extension of the one used for the ELBDR experiment valid input patterns set, adding the one-hot encoding restraint to the 5 previous state value bits of the arbitrating pseudo-combinational unit. This allowed to shrink the exhaustive set of 2^{19} input stimuli to a valid and complete set consisting of 61,440 input vectors, which is less that 12% of the initial number. This may be considered as a proof of the effectiveness of the one-hot encoding for the arbiter state variable.

First, the evaluation tool was run considering the whole set of checkers for the SArbiter, altogether with the minimized set of 3 checkers for the ELBDR. This analysis required 1 second of run time by the framework. Figure 8 lists the considered 31 checkers, with their corresponding weights in a descending order. Focusing on the arbitrating unit, two checkers look to be far more significant than the others, *Serr_validgrant* (checker 6 in Table II), *Serr_invalidstate* (checker 9), both of them monitoring different aspects of the one-hot encoding condition for the arbiter's state variable.

From the output of the evaluation tool it can be observed that the two set of checkers for the ELBDR and the SArbiter are *independent*, i.e. they cover faults for different and separate parts of the circuit, without any overlap. For this reason the minimized set of ELBDR checkers is used, and the previously introduced weight-based greedy minimization heuristic is applied to the SArbiter checkers set.

Fig. 9 displays the obtained results. As it could have been expected from the weighting information in Fig. 8, the two most significant checkers dominate all the lightest checkers, ensuring

100% CEI and FC. Thus, considering a total of 3 ELBDR and 2 SArbiter checkers, area overhead over the partial control path circuit is limited to 56.82%, while using the whole initial set of 28 checkers for the SArbiter would lead to 170.45% area overhead. It is interesting to observe that the minimized set of 5 checkers corresponds to one third of the whole 31 checkers set area.

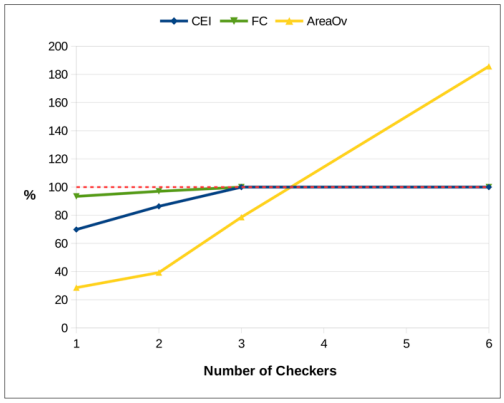


Fig. 7. ELBDR scenario results

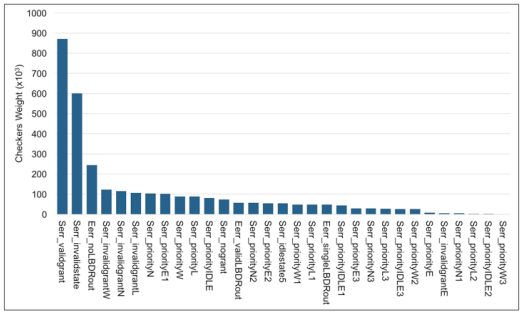


Fig. 8. ELBDR + SArbiter checkers ranked by weights

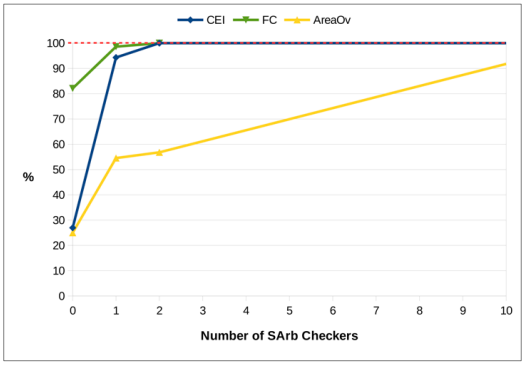


Fig. 9. ELBDR + SArbiter scenario results

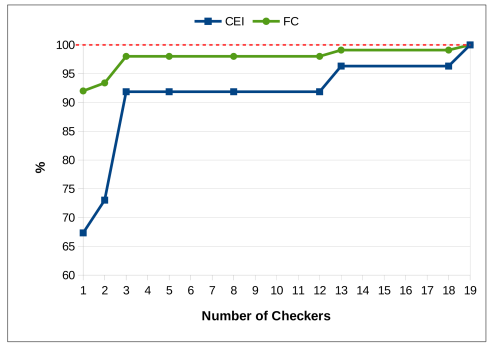


Fig. 10. Results without considering independent clusters

Checker	Weight
Serr_validgrant	871552
Serr_invalidstate	600512
Eerr_noLBDRout	243840
Eerr_validLBDRout	57600
Eerr_singleLBDRout	47680

Fig. 11. Weights for minimized set of checkers

Impact of clustering the faults

Assuming that we had no information of the overlap of faults detected by the checkers for ELBDR and SArbiter, the weight-based greedy heuristic, starting from the heaviest checker *Serr_validgrant*, would add at each step the next heaviest checker still not considered in the current set of checkers, based on the weight information displayed in Fig. 8. Fig. 10 shows the inefficiency of the heuristic approach caused by the lack of the clustering information. The number of steps in the greedy procedure is heavily increased, and only after 19 steps, when the *Eerr_singleLBDRout* checker is considered, the 100% upper bound for CEI and FC is reached.

However, when partitioning of the fault set to clusters is taken into account and minimization is performed on the clusters separately then total of five checkers are needed. Fig. 11 illustrates the importance of considering the clustering information. It can be observed that the weights of the ELBDR checkers are far less than those of the SArbiter, but they are still needed to achieve full coverage for the considered design.

VIII. CONCLUSIONS

The paper proposes a new tool providing an automated flow for evaluation and minimization of concurrent online checkers, which is formal (able of proving the presence or absence of true misses), yields minimal fault detection latency and enables accurate, fully automated evaluation of the fault detection characteristics of a given set of checkers.

Experiments carried out on the control part (routing and arbitration) of a Network-on-Chip (NoC) router showed on a realistic application the feasibility and efficiency of the framework and the underlying methodology. Experimental results showed that the approach allowed selecting the minimal

set of 5 checkers out of 31 verification assertions with the fault coverage of 100% and area overhead of only 56.82%.

ACKNOWLEDGEMENT

The work has been supported by EU FP7 STREP BASTION, EU's H2020 RIA IMMORTAL, Estonian Science Foundation grant ETF9429, Estonian institutional research grant IUT 19-1, funded by Estonian Ministry of Education and Research, and by EU through the European Structural and Regional Development Funds.

REFERENCES

[1] R. Sedmak and H. Liebergot. Selective triple modular redundancy (STMR) based single-event upset (SEU) tolerant synthesis for FPGAs. *IEEE Transactions on Nuclear Science*, 51:2957-2969, 2005.

[2] J. M. Berger. A note on an error detection code for asymmetric channels. *Information and Control*, 4:68-73, 1961.

[3] D. Das and N. A. Touba. Synthesis of circuits withlow-cost concurrent error detection based on Bose-Lin codes. *In VLSI Test Symposium*, pages 309-315, 1998.

[4] K. Mohanram, E. Sogomonyan, M. Gossel, and N. Touba. *Synthesis of low-cost parity-based partially self-checking circuits*, 2003.

[5] S. Ghosh, N.A. Touba, and S. Basu. Synthesis of low power ced circuits based on parity codes. *In VLSI Test Symposium*, pages 315-320, 1-5 May 2005.

[6] R. Sharma and K.K. Saluja. An implementation and analysis of a concurrent built-in self-test technique. *In Digest of Papers Eighteenth International Symposium on Fault-Tolerant Computing FTCS-18*, pages 164- 169, June 1988.

[7] P. Drineas and Y. Makris. Concurrent fault detection in random combinational logic. *In Proceedings Fourth International Symposium on Quality Electronic Design ISQED*, pages 425-430, March 2003.

[8] Alves, N.; Shi, Y.; Dworak, J.; Bahar, R.I.; Nepal, K. "Enhancing online error detection through area-efficient multi-site implications", *IEEE 29th VLSI Test Symposium (VTS)*, pp. 241 – 246, 2011.

[9] Marc Boule, Jean-Samuel Chenard, and Zeljko Zilic. Assertion checkers in verification, silicon debug and infield diagnosis. *In Proceedings of the ISQED '07*.

[10] M Aarna, E Ivask, A Jutman, E Orasson, J Raik, R Ubar, V Vislogubov, HD Wuttke. Turbo Tester-Diagnostic Package for Research and Training. *The 1st East-West Design and Test Conference*, Alushta, 2003.

[11] Artur Jutman, A Peder, J Raik, M Tombak, R Ubar. Structurally synthesized binary decision diagrams. *6th International Workshop on Boolean Problems.* pp. 271-278, 2004.

[12] J. Flich, J. Duato, Logic-Based Distributed Routing for NoCs, *IEEE Computer Architecture Letters*, Vol. 7, No. 1, January-June 2008.

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

[14] Prodromou, A.; Panteli, A.; Nicopoulos, C.; Sazeides, Y., "NoCAlert: An On-Line and Real-Time Fault Detection Mechanism for Network-on-Chip Architectures," *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 60-71, 1-5 Dec. 2012.

[15] Yu, Qiaoyan; Cano, J.; Flich, J.; Ampadu, P., "Transient and Permanent Error Control for High-End Multiprocessor Systems-on-Chip," *2012 Sixth IEEE/ACM International Symposium on Networks on Chip (NoCS)*, vol., no., pp.169,176, 9-11 May 2012.

[16] Alaghi, A.; Karimi, N.; Sedghi, M.; Navabi, Z., "Online NoC Switch Fault Detection and Diagnosis Using a High Level Fault Model," *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems*, 2007. DFT '07., vol., no., pp.21,29, 26-28 Sept. 2007.

Appendix 4

Publication IV

Pietro Saltarelli, Behrad Niazmand, Jaan Raik, Vineeth Govind, Thomas Hollstein, Gert Jervan, and Ranganathan Hariharan. A framework for combining concurrent checking and on-line embedded test for low-latency fault detection in noc routers. In *Proceedings of the 9th International Symposium on Networks-on-Chip*, page 6. ACM, 2015

A Framework for Combining Concurrent Checking and On-Line Embedded Test for Low-Latency Fault Detection in NoC Routers

Pietro Saltarelli
Università degli Studi di Ferrara
Via Savonarola, 9
44121 Ferrara FE, Italy
pietro.saltarelli@student.unife.it

Behrad Niazmand, Jaan Raik, Vineeth Govind, Thomas Hollstein, Gert Jervan
Tallinn University of Technology
Department of Computer Engineering
Akadeemia 15a, 12618 Tallinn, Estonia
Phone: +372 6202257
<behrad|jaan>@ati.ttu.ee

Ranganathan Hariharan
Nokia Solutions and Networks Oy
Hatanpää
Hatanpään valtatie 30
33100 Tampere, P.O. Box 785
Finland
ranganathanh87@gmail.com

ABSTRACT

The focus of the paper is detection of faults in NoC routers by combining concurrent checkers with embedded on-line test to enable cost-effective trade-offs between area-overhead and test coverage. First, we propose a framework of tools for formally evaluating the quality of the checkers and for optimizing the overhead area with given fault coverage constraints. The stress is in particular on the minimization of the error detection latency, which is a crucial aspect in order to eliminate (or limit) error propagation. Second, the concurrent checkers will be complemented by embedded on-line test packets which are to be applied as a periodic routine during the idle periods in router operation. The framework together with the corresponding methodology has been successfully applied to a realistic case-study of a fault tolerant NoC router design. The case study shows that combining concurrent routers with embedded test allows reducing the area overhead of the checkers from 31-35% down to 1.5-10% without sacrificing the fault coverage.

Keywords

Network-on-chip, fault tolerant router design, concurrent online checking, embedded test, test packets.

NOCS '15, September 28 - 30, 2015, Vancouver, BC, Canada

 ISBN 978-1-4503-3396-2/15/09…$15.00
DOI: http://dx.doi.org/10.1145/2786572.2788713

1. INTRODUCTION

One of the main challenges related to the design of Network-on-Chip (NoC) routers is the extreme down-scaling of modern technologies that increases the probability of the components to wear-out as well as their vulnerability towards environmental effects. These are phenomena occurring during the life-time of the system and cannot be screened out by manufacturing testing. Thus, cost-efficient mechanisms for detecting faults during system's life-time are needed. These mechanisms should detect errors within routers and enable reconfiguration of the routing network in order to isolate the problem and provide graceful degradation for the system. In this paper, we propose combining concurrent checkers with embedded on-line test packets in order to achieve early and cost-effective detection of faults in NoC routing infrastructure.

Regarding the development of on-line checkers, we introduce a new framework and a methodology with a stress on the level of automation, fault coverage, detection latency and area-efficiency. The methodology consists of four main steps. The first step of the methodology is formal checker qualification which includes identification of control-intensive parts of the router architecture, converting them to pseudo-combinational counterparts, preparation of the checkers synthesized from verification assertions and specifying the environment in terms of valid input stimuli for the pseudo-combinational circuit. As a result, the faults detected by each individual checker will be calculated.

Second, the number of checkers within the set will be minimized by applying the checker optimization step. As a starting point is the fault detection characteristics for each individual checker as well as their weights in terms of silicon area. Further, a heuristic minimization method is applied resulting in a minimal selection of checkers to achieve a target fault coverage level. The minimization technique is based on a divide-and-conquer approach of partitioning the checkers' fault table into independent clusters. This approach is very effective as the checkers devised for different modules normally do not have overlapping fault sets.

Third, and optional, step of the methodology includes devising additional checkers from temporal assertions for modules that do

not achieve 100% fault detection. For these checkers the formal qualification step described above is not possible and traditional fault injection experiments are carried out by a sequential fault simulation tool included to the framework.

Finally, the checkers for the control part of the router are to be complemented by embedded on-line test packets which are to be applied as a periodic routine during the idle periods in router operation, e.g. slacks in the task scheduling. The framework together with the corresponding methodology has been successfully applied to a realistic case-study of a fault tolerant NoC router design. The case study shows that combining concurrent checkers with embedded test packets allows reducing the area overhead of the checkers from 31-35% down to 1.5-10%, depending on the router bitwidth, without sacrificing the fault coverage.

The paper is organized as follows. Section 2 provides an overview of related works in concurrent online testing and embedded test for NoC routers. Section 3 explains the concurrent online checking concept. Section 4 discusses application of embedded test packets. In Section 5, the automated framework and the corresponding methodology for checkers' minimization combined with the embedded test are presented. Section 6 discusses application of the framework and the underlying methodology to the NoC router design. Section 7 provides the experiments. Finally, Section 8 concludes the paper.

2. RELATED WORKS

Online detection of errors in logic is a thoroughly studied research area. Traditional Triple-Modular Redundancy (TMR) and duplication based approaches are too costly in terms of multiplying the area and correspondingly the power consumption. An alternative to minimize this overhead is the selective TMR that identifies Single Event Upset (SEU) sensitive sub-circuits that are to be protected [1].

In addition, there exists a variety of solutions based on coding techniques such as Berger [2] or Bose-Lin [3] codes. In many works the coding techniques are combined with synthesis [4,5]. The approaches suffer from significant area overhead to the design to be checked.

Concurrent on-line built-in self-test techniques such as Built-In Concurrent Self-Test (BICST) [6] and Reduced Observation Width Replication (ROWR) [7] provide high fault coverage at low area overhead but only consider a limited subset of pre-computed test vectors. Hence these approaches are likely to miss faults occurring in a normal circuit operation.

Several alternatives based on checkers that do not require modification of the circuit under test have been developed. Creating checkers automatically based on logic implications derived from the circuit structure [8] is feasible but suffers from low fault coverage and high area overhead, often exceeding the duplex solutions. On the other hand, deriving checkers from functional assertions, or reusing verification assertions, is similarly known to yield low coverage of structural faults as it is difficult to correlate functional coverage to structural one [9].

Many previous works have focused on addressing faults in the control logic of NoC routers. In [16], Yu et al. have addressed fault tolerance for NoC topologies and proposed an error control method for detecting transient errors in routing logic implemented using Logic-Based Distributed Routing (LBDR) mechanism and its extension for high-radix topologies, LBDRhr. The proposed error control method utilizes the inherent information redundancy (IIR) to reduce the error control overhead. However, the method does not guarantee full fault coverage.

Authors of [17] have presented a method for online error detection and diagnosis of NoC switches. The proposed method deals with routing faults that cause packets to be forwarded to unintended output ports. Regarding modeling routing faults in switches, a high-level fault model has been introduced in this work. The fault coverage is measured only at the functional level and there is no estimates on correlation to gate-level fault coverage.

In order to deliver correctness guarantees for the complete network, Parikh et al. have proposed a network-level detection and recovery solution ForEVeR [14] that monitors the traffic in the NoC and protects it against functional bugs that were not detected during design time. To this end, ForEVeR augments the baseline NoC with a lightweight checker network that alerts destination nodes of incoming packets ahead of time and is used for the recovery process. The approach suffers from extremely high latency. Only 30% of the faults will be detected during the first clock cycle by the approach.

The work in [15] proposes checkers synthesized from a set of 32 verification assertions. The checkers detect most of the injected faults. The faults that are not covered correspond to non-catastrophic failures. The work proposed in [15] is not automated and lacks the completeness and minimization aspects present in the current paper.

In [18] a hybrid method is introduced for synthesis of fault-secure NoC switches utilizing error detecting codes for the data path (data flits) and a concurrent error detection structure for dealing with faults not covered by the flit encoding (using multiple parity trees). However, the work still results in more than 50% area overhead.

The use of embedded test configurations for testing the datapath of NoC routers has been proposed in [19], with design-for-testability structures included in [20] and built-in self-test application in [21]. However, all the mentioned approaches are targeting the global network and not a concrete router. Furthermore, only off-line test scenarios have been considered in [19-21].

This paper exceeds the existing state-of-the-art in fault tolerant router design by proposing:

- a framework for *formal checker qualification*. The underlying approach is complete, i.e. it allows proving the absence or presence of true misses by the checkers. In addition, it provides minimal fault detection latency due to the fact that the circuit is transformed into a pseudo-combinational one and therefore only checkers with a single clock cycle latency are considered.
- *automated minimization of checkers*. The formal qualification of the combinational checkers provides the fault detection capabilities for them. These, along with the checker area requirements are applied in an automated minimization process resulting in a minimal area overhead checker solution under certain fault coverage constraints.
- complementing the resulting checkers withtemporal checkers and *on-line embedded test packets*. This enables combining best

of both worlds. In the case of NoC control part, where embedded test packet based approaches have proven inefficient, low area concurrent checkers are applied. On the other hand, in the datapath, the embedded test yields full fault coverage whereas error correcting codes would be expensive.

Experimental results on a realistic NoC router design demonstrate the efficiency of the proposed approach.

3. THE CONCEPT OF CONCURRENT CHECKERS

Fig. 1 presents the role of concurrent on-line checkers in detecting faults within a circuit. In addition to the original circuit (functional logic), a set of checkers (checker logic) will be connected to functional inputs/outputs of the circuit. These checkers are derived based on functional assertions obtained from relationships between variables corresponding to inputs and outputs of the circuit. The checker logic targets the faults at lines at the inputs of each gate within the functional logic (marked by green circles). The lines at the functional outputs succeeding the checker inputs (marked by a red cross) cannot be detected by the checker. In addition, the checkers are not targeting the faults at functional inputs preceding checker inputs, since the checker may not detect that the input value has been altered by a fault (such functional input lines are also marked by a red cross in Fig. 1). In this paper, we consider the single stuck-at fault model. However, due to the fact that concurrent checkers are implemented and at-speed embedded test packets are applied, the model also covers timing related faults.

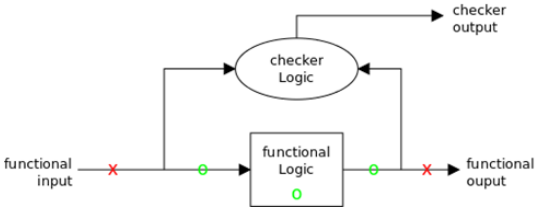


Figure 1. The concept of concurrent checking

Given a fault at a line within the functional logic and a set of input stimuli, four possible scenarios may occur:

Case 1: Fault occurs at an internal line and is visible at functional output(s) and checker logic flags a violation. The term *True Detection* is used to describe this situation, since a critical fault is effectively detected by the checker.

Case 2: Fault occurs at an internal line but is not visible at primary output(s). Checker catches the fault and flags a violation. The term *False Positive* is used to describe this situation. False positive is not harmful because an error is flagged which did not have any effect. However, it has negative impact on design's performance because normally it causes re-execution of the task.

Case 3: Fault occurs at internal line but is not visible at primary output(s) and the checker logic does not detect the violation. The term *Benign Miss* is used to describe this situation. Benign miss shows correct operation by the checker.

Case 4: Fault occurs at internal node and is visible at primary output(s). Checker does not detect violation. The term *True Miss* is used to describe this situation, which is the worst possible case. True miss means that the fault propagates to the functional outputs and onwards to the system. However, the system has no information that a critical fault has occurred.

Traditionally, in order to evaluate the fault detection quality of the checkers, *fault injection* has been applied. Fault injection refers to injecting faults into a circuit at a certain time step and simulating it with the input stimuli to see whether any functional output of the circuit changes and whether any of the checker output fires. Due to the fact that it is generally impossible to inject and simulate all the faults at each circuit line at each time step, a statistically significant sample of random faults would normally be injected and simulated.

However, in this paper a methodology is proposed which is based on automated extraction of a pseudo-combinational circuit out of the original functional logic by breaking the flipflops and converting them to pseudo primary inputs and pseudo primary outputs. Further, an exhaustive test for the extracted circuit is fed through a filtering tool in order to derive the complete valid set of input stimuli which will serve as the environment for checker evaluation. This means that in this paper full formal qualification of the combinational checkers with all possible stimuli and faults can be obtained.

Let D be the number of true detections, X be the number of benign misses, F be the set of false positives and W be the number of true misses over all the injection runs. In order to evaluate the fault detection capabilities of the checkers we define the metrics of *Fault Coverage* (*FC*), *Checkers' Efficiency Index* (*CEI*) and *False Positive Ratio* (*FPR*) as follows.

$$FC = \frac{D + X}{D + X + W} \quad (1)$$

$$CEI = \frac{D}{D + W} \quad (2)$$

$$FPR = \frac{F}{F + X} \quad (3)$$

Here, FC shows the probability of the checkers behaving correctly over all possible fault cases, CEI shows the probability of checkers ability to detect critical faults whereas FPR reports the ratio of false positives over all the cases a fault did not propagate to circuit outputs. The mentioned three metrics are calculated for checkers by the automated checker qualification framework proposed in this paper.

4. EMBEDDED ONLINE TEST PACKETS

The functional fault model that is applied to cover the stuck-at faults in the datapath of the NoC router is based on the idea proposed for fuctional testing of mesh-like NoC networks in [19-21]. However, in this paper the fault model is applied to a "localized" approach, where resources (i.e. processing elements) connected to neighbouring routers West (W), East (E), North (N), South (S), and Local (L) are utilized as senders/receivers of test packets to test the central router as the Circuit Under Test (CUT). Figure 2 visualizes the overall setup of the sending/receiving resources and the CUT.

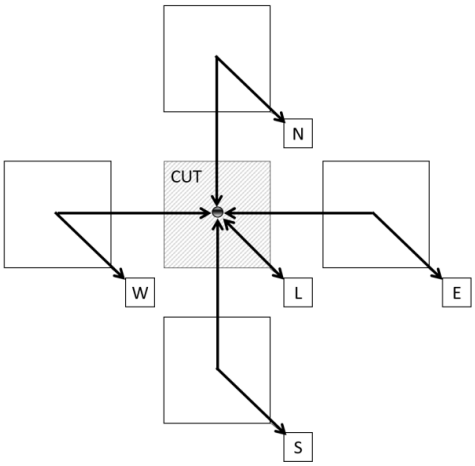


Figure 2. The setup for sending/receiving test packets

In the proposed setup, whenever there are idle periods or slacks in scheduling with length K for the send/receive resources, K test patterns will be applied from them. This will be done periodically fetching K next tests from the test set in a circular manner, i.e. if the end of the test is reached then it starts again from the beginning. This scenario provides online test capabilities for regularly checking the health of the datapath of the routing infrastructure.

A fault model proposed in [19-21] is applied, where the value at a selected router input is distinguished from the values at other inputs of the router. In order to fully cover the structural faults in the multiplexers of the crossbar, tests for each address value have to be performed. An additional constraint is that all turns must be covered by the distinguishing tests. In [19] it was shown that by applying them, near 100% fault coverage for the crossbar switch and the I/O buffers comprising the datapath of the NoC router is achieved.

5. FRAMEWORK AND METHODOLOGY

This Section presents the framework for fault tolerant NoC router design that has been developed as an extension of the Turbo Tester test framework [10]. The proposed methodology of combining concurrent checkers with embedded online test consists of three main steps:

1. Checkers' qualification and minimization (combinational checkers);
2. Checkers' evaluation by fault injection (temporal checkers);
3. Fault simulation of the embedded online test packets.

In the following, these steps are explained in more detail.

5.1 Checker Qualification and Minimization

Fig. 3 presents the qualification and minimization flow for the checkers. The flow starts with synthesizing the checkers from a set of combinational assertions. Thereafter, a pseudo-combinational circuit will be extracted from the circuit of the design under checking. The pseudo-combinational circuit is derived out of the original circuit by breaking the flipflops and converting them to pseudo primary inputs and pseudo primary outputs. Note, that at this point additional checkers that also describe relations on the pseudo primary inputs/outputs may be added to the checker suite in order to increase the fault coverage.

Subsequently, the checkers' qualification environment is created by generating exhaustive test stimuli for the extracted pseudo-combinational circuit. This stimuli are fed through a filtering tool that selects only the stimuli that correspond to functionally valid inputs of the circuit. As a result, the complete valid set of input stimuli that will serve as the environment for checkers' qualification is obtained.

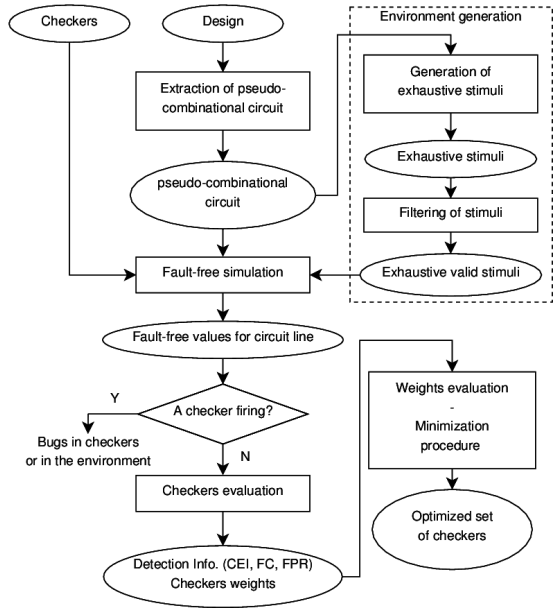


Figure 3. Checkers' qualification and minimization flow

The obtained environment, pseudo-combinational circuit and synthesized checkers are applied to fault free simulation. The simulation calculates fault free values for all the lines within the circuit. Additionally, if any of the checkers fires during fault-free simulation it refers either to a bug in the checker or an incorrect environment.

If none of the checkers is firing in the fault-free mode then checkers' qualification takes place. The tool injects faults to all the lines within the circuit one-by-one and this step is repeated for each input vector. As a result, the overall fault detection capabilities for the set of checkers, in terms of FC, CEI and FPR metrics will be calculated. In addition, each individual checker will be weighted by summing up the total number of true detections by the checker.

The weighting information will then be exploited in minimizing the number of checkers, eventually allowing to outline a trade-off

between the fault coverage, and the area overhead due to the introduction of checker logic.

5.2 Checkers' Evaluation by Fault Injection

There are cases when a module under checking cannot be handled by the combinational checker qualification and minimization approach. For example the module may have a large number of inputs so that the set of generated valid input stimuli would be too large (e.g. datapath modules) and/or the fault coverage reached by the combinational checkers is too low.

In those cases, the checkers are to be evaluated by traditional fault injection. Here a test bench is created for the design and the circuit with the checkers is simulated by a sequential fault simulator with a sufficiently large random sample of faults injected into the circuit. In this paper, all the datapath checkers and the FIFO checkers were evaluated using this approach.

5.3 Fault Simulation of the Embedded Test

Finally, the stuck-at fault coverage of the online embedded test packets for the datapath of the NoC router is measured by a fault simulator belonging to the framework. As experimental results show, full fault coverage for the datapath with the test application time of 196 clock cycles is achieved.

6. EXPERIMENTAL RESULTS

Fig. 4 demonstrates the high-level overview of a 5-port 2D NoC router that we have chosen as a target architecture for applying the checkers. The router consists of a datapath and a control part. The datapath is composed of input buffers (implemented as FIFO), one for each input port, a crossbar switch and an output buffer for each output port. The control part contains routing units, arbiters and FIFO control. For the routing unit of our target architecture, we have opted for Logic-Based Distributed Routing (LBDR)[13], which is considered as a scalable solution compared to routing tables. As an arbiter, round-robbin arbitration was implemented.

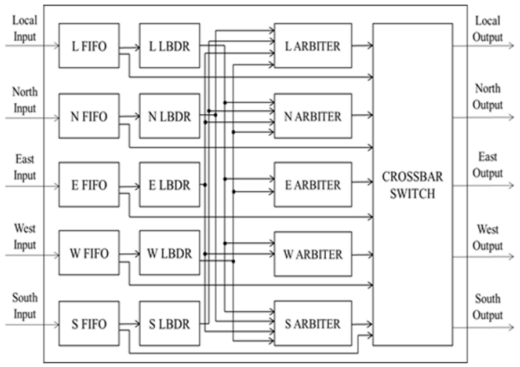


Figure 4. High level architecture of the NoC router

6.1 Checker Qualification/Minimization for LBDR/Arbiter

The pseudo-combinational circuit for ELBDR has 11 input bits, as mentioned in the previous section, thus the exhaustive set of stimuli presents 2^{11}=2,048. A filtering scheme based on the following statements was devised:

- if input buffer's empty signal is high, any other input bit is meaningless, and therefore any value is allowed for it;
- if the incoming flit is a header, the destination address has to be valid according to the XY routing and turns restrictions;
- if the incoming flit is a body or tail flit, the previous output values must be valid, they must follow a one-hot fashion, according to XY routing.

This allowed to obtain a valid and complete set of stimuli consisting of 1536 vectors, which forms 75% of the exhaustive set. The run-time for generating the stimuli was 2 seconds. (All the experiments in this paper were carried out on an Asus ux32vd-r4002v computer with a 1.9 GHz Intel Core i7-3517U processor and 10 GB RAM.)

Table 1 lists the obtained minimized set of three checkers for the LBDR. Reducing the set of checkers to the three most significant ones allows to limit the area overhead to 78.57% over the ELBDR circuit, far lower than 185.71% imposed by the initial non-minimized set of checkers, while the CEI and FC remain at 100%.

Table 1. A minimized list of checkers for the LBDR

Checkers for Routing Logic (LDBR)		
1	Valid LBDR output	If there is a request to the routing logic (the corresponding input buffer is not empty), LBDR has to compute at least one valid output direction (according to XY routing).
2	No LBDR output	If no flit arrives (the corresponding input buffer is empty), all the output port signals of LBDR should remain zero.
3	Single LBDR output	If the corresponding input buffer is not empty (there is a request to LBDR), because of using XY routing, at most only one output port signal of the LBDR logic can become active.

Similarly, Table 2 lists the minimized set of two checkers for the Arbiter that was obtained from an initial set of 28 verification checkers by applying the checker qualification and minimization framework.

Table 2. A minimized list of checkers for the Arbiter

Checkers for Arbiter logic		
4	Valid Grant output	If there is a request from LBDR, arbiter has to assert at least one of the grant signals for the corresponding output direction.
5	Invalid arbiter State	State variable of the arbiter FSM has to respect one-hot encoding.

6.2 Fault Injection Experiments for the FIFO

Table 3 lists the set of 8 checkers generated from the verification assertions for the FIFO control part. The checkers were evaluated by the fault injection tool of the framework. A set of input stimuli for the FIFO was devised, aiming to cover all the possible situations for the control logic. The following conditions were considered in the pattern generation procedure:

- reset condition;
- filling the FIFO, followed by reading up to empty condition;
- smooth traffic condition, i.e. concurrent writing and reading operations, avoiding the FIFO to get full;
- idle condition, i.e. write and read enable signals low, during reading and writing operations, in different conditions of fulfillment of the buffer.

100% CEI and FC were achieved on the control part of the FIFO, considering the patterns derived from the previously listed conditions, amounting to 134. Run time for the experiment was 0.06 s. No false positives were encountered in this experiment.

Table 3. Checkers for the FIFO Control Part

Checkers for FIFO control part		
6	Reset checker	Whenever reset goes high, at the next clock cycle empty flag should be high (reading and writing pointer are reset to the same value).
7	Flags checkers	Empty and full flags should never be high at the same time. Whenever the defining condition occurs, the corresponding flag should go high at the next clock cycle.
8	One-hot pointers checkers	Reading and writing pointers have to respect one-hot encoding.
9	Registers enable DMR checker	Duplication and comparison for the logic enabling the writing operation in data registers.
10	Reading pointer update checker 1	Whenever read enable is high and the FIFO is not empty, at the next clock cycle the reading pointer should be updated.
11	Reading pointer update checker 2	If either read enable is low or the FIFOis empty, at the next clock cycle the reading pointer should preserve its value.
12	Writing pointer update checker 1	Whenever write enable is high and the FIFO is not full, at the next clock cycle the writing pointer should be updated.
13	Writing pointer update checker 2	If either write enable is low or the FIFO is full, at the next clock cycle the writing pointer should preserve it value.

Table 4 lists the set of 3 additional checkers which were included in order to achieve the full fault coverage after fault injection experiments for the control part identified uncovered faults in the interconnections of control part modules.

Table 4. Control Part Infrastructure Checkers

Control Part Infrastructure Checkers		
14	FIFOs read enable DMR checker	Logic producing read enable signals for the FIFOs (5 OR gates) is duplicated, then real and duplicated outputs are compared.
15	Output registers enable DMR checker	Logic producing enable signals for the output registers (5 OR gates) is duplicated, then real and duplicated outputs are compared.
16	Flit type LBDR error	Flit type field of a flit has to respect one-hot encoding.

6.3 Checkers for the Datapath

In order to fully cover the faults in the NoC datapath two types of concurrent checkers were introduced (listed in Table 5). First, for each input port an even parity bit is included, whereas each output port has a checker evaluating the even parity. Second, since fault injection experiments for the whole router identified undetected faults within the crossbar multiplexers, dedicated checkers for the crossbar were devised.

Table 5. Checkers for the NoC Datapath

Datapath Checkers		
17	Even parity checker	An even parity bit is computed and added to data entering each input port, which is later evaluated before data leaves the router through any of the output ports.
18	Crossbar checker	Crossbar MUXs are duplicated, then real and duplicated outputs are compared.

6.4 Putting It All Together

Fig. 5 reports the area overhead required by the checkers for routers of varying bitwidth (from 32 bits to 256 bits). It can be observed from the Figure that the required area for the control part checkers stays constant while the overhead area of datapath checkers (parity and crossbar) grow proportionally to the router size.

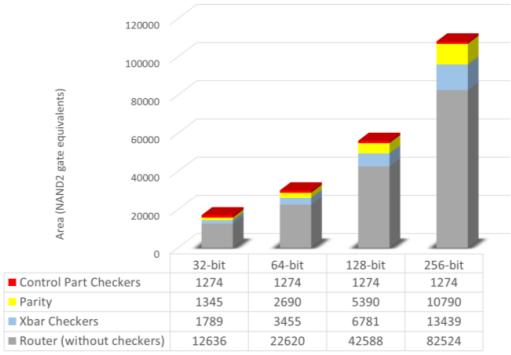


Figure 5. Area consumption for different datawidhts

The same trend is revealed in Table 6. It can be seen that if datapath checkers are included then the required area overhead would be in the range of 31-35%. Whereas, the control part checker circuitry demands significantly less area, especially for larger bitwidths.

Table 6. Overhead Area for Different Datawidths

	32-bit	64-bit	128-bit	256-bit
Router (w/o checkers)	12636	22620	42588	82524
Control part checkers	1274	1274	1274	1274
Xbar Checkers	1789	3455	6781	13439
Parity	1345	2690	5390	10790
Area overhead (contr. p. checkers), %	10.08	5.63	2.99	1.54
Area overhead (all checkers), %	34.88	32.80	31.57	30.90

However, when combining the control part checkers with embedded online test packets presented in Section 4, full fault coverage for the NoC router can be achieved with a minor area overhead. As it has been shown by experiments in [21] an embedded test of length K=196 clock cycles will achieve FC=100% within the NoC router datapath. Thus, combining the concurrent checkers for the control with embedded test solution for the datapath results in a cost-effective solution for fault tolerant NoC routers.

7. CONCLUCIONS

The paper proposes a framework for formal qualification of checkers and for minimizing the overhead area with the given fault coverage constraints. The goal is to achieve low-latency, low area overhead checkers for network on chip routers. In addition the paper proposes complementing the concurrent checkers with embedded on-line test packets which are to be applied as a periodic routine during the idle periods in router operation.

The framework together with the corresponding methodology has been successfully applied to a realistic case-study of a fault tolerant NoC router design. The case study shows that combining concurrent routers with embedded test allows reducing the area overhead of the checkers from 31-35% down to 1.5-10% without sacrificing the fault coverage.

8. ACKNOWLEDGMENTS

The work has been supported in part by EU's FP7 STREP project BASTION and H2020 RIA IMMORTAL, by Estonian ICT program project FUSETEST, by Research Centre CEBE funded by European Union through the European Structural Funds and by Estonian SF grant 9429.

9. REFERENCES

[1] R. Sedmak and H. Liebergot. Selective triple modular redundancy (STMR) based single-event upset (SEU) tolerant synthesis for FPGAs. *IEEE Transactions on Nuclear Science*, 51:2957-2969, 2005.

[2] J. M. Berger. A note on an error detection code for asymmetric channels. *Information and Control*, 4:68-73, 1961.

[3] D. Das and N. A. Touba. Synthesis of circuits withlow-cost concurrent error detection based on Bose-Lin codes. *In VLSI Test Symposium*, pages 309-315, 1998.

[4] K. Mohanram, E. Sogomonyan, M. Gossel, and N. Touba. *Synthesis of low-cost parity-based partially self-checking circuits*, 2003.

[5] S. Ghosh, N.A. Touba, and S. Basu. Synthesis of low power ced circuits based on parity codes. *In VLSI Test Symposium*, pages 315-320, 1-5 May 2005.

[6] R. Sharma and K.K. Saluja. An implementation and analysis of a concurrent built-in self-test technique. *In Digest of Papers Eighteenth International Symposium on Fault-Tolerant Computing FTCS-18*, pages 164- 169, June 1988.

[7] P. Drineas and Y. Makris. Concurrent fault detection in random combinational logic. *In Proc. Fourth International Symposium on Quality Electronic Design ISQED*, pages 425-430, March 2003.

[8] Alves, N.; Shi, Y.; Dworak, J.; Bahar, R.I.; Nepal, K. "Enhancing online error detection through area-efficient multi-site implications", *IEEE 29th VLSI Test Symposium (VTS)*, 2011.

[9] Marc Boule, Jean-Samuel Chenard, and Zeljko Zilic. Assertion checkers in verification, silicon debug and infield diagnosis. *In Proceedings of the ISQED '07*.

[10] M Aarna, E Ivask, A Jutman, E Orasson, J Raik, R Ubar, V Vislogubov, HD Wuttke. Turbo Tester-Diagnostic Package for Research and Training. *The 1st East-West Design and Test Conference*, Alushta, 2003.

[11] Artur Jutman, A Peder, J Raik, M Tombak, R Ubar. Structurally synthesized binary decision diagrams. *6th International Workshop on Boolean Problems*. pp. 271-278, 2004.

[12] Raimund Ubar, Sergei Devadze, Jaan Raik, Artur Jutman. Parallel X-fault simulation with critical path tracing technique. *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pp. 879-884, 2010.

[13] J. Flich, J. Duato, Logic-Based Distributed Routing for NoCs, *IEEE Computer Architecture Letters*, Vol. 7, No. 1, January-June 2008.

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

[15] Prodromou, A.; Panteli, A.; Nicopoulos, C.; Sazeides, Y., "NoCAlert: An On-Line and Real-Time Fault Detection Mechanism for Network-on-Chip Architectures," *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 60-71, 2012.

[16] Yu, Qiaoyan; Cano, J.; Flich, J.; Ampadu, P., "Transient and Permanent Error Control for High-End Multiprocessor Systems-on-Chip," *2012 Sixth IEEE/ACM International Symposium on Networks on Chip (NoCS)*, vol., no., pp.169,176, 9-11 May 2012.

[17] Alaghi, A.; Karimi, N.; Sedghi, M.; Navabi, Z., "Online NoC Switch Fault Detection and Diagnosis Using a High Level Fault Model,"

22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems, vol., no., pp.21,29, 26-28 Sept. 2007.

[18] Dalirsani, A.; Kochte, M.A.; Wunderlich, H.-J., "Area-efficient synthesis of fault-secure NoC switches," *IEEE 20th International On-Line Testing Symposium* (*IOLTS*), pp.13,18, 7-9 July 2014.

[19] J. Raik, V. Govind, R. Ubar. An External Test Approach for Network-on-a-Chip Switches. Proc. of the IEEE Asian Test Symposium, pp. 437-442, Nov. 2006

[20] J. Raik, V. Govind, R. Ubar. Design-for-Testability- Based External Test and Diagnosis of Mesh-like NoCs. *IET Computers and Digital Techniques*, Vol. 3, Issue 5, pp. 476-486, September 2009.

[21] Raik, J.; Govind, V. Low-area boundary BIST architecture for mesh-like network-on-chip, *IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, pp. 95-100, 2012.

Appendix 5

Publication V

Ranganathan Hariharan, Behrad Niazmand, and Jaan Raik. On fault detection efficiency of reliability checkers obtained by verification assertion qualification. In *RESCUE 2017 Workshop on Reliability, Security and Quality European Test Symposium (ETS) Fringe Workshop, May 25-26*. IEEE, 2017

On Fault Detection Efficiency of Reliability Checkers Obtained by Verification Assertion Qualification

Ranganathan Hariharan, Behrad Niazmand, Jaan Raik

Tallinn University of Technology, Estonia

***Abstract*— This paper assesses correlation between verification assertion qualification results and gate-level fault detection capabilities of concurrent error checkers synthesized from these assertions. Generating checkers from verification is a solved problem where commercial tools are available (e.g. IBM FoCs). Furthermore, there exists several academic and industrial solutions for assertion qualification (e.g. Certitude by Synopsys). Such qualification tools are traditionally applied in order to assess the verification power of assertions generated in an automated way. However, the number of assertions selected by qualification is still far too high and result in unacceptable area overheads when implemented as checker circuitry in reliability applications.**

In order to derive low-area but high-coverage checker circuitry from a large number of verification assertions this paper proposes the following flow. First, we use the Certitude tool to qualify verification assertions. Subsequently, we propose a minimization algorithm to allow generation of low-area high quality checkers. Finally, the fault detection capabilities of the obtained checkers are evaluated using the framework developed by the authors of the paper. This is the first work to consider correlation of assertion quality versus fault detection capabilities of the synthesized checkers. It is also the first time when qualified assertions are minimized with considerations of the checker coverage and overhead area.

The paper includes a preliminary study of the proposed methodology on an example of a network-on-chip routing block.

Keywords—assertions, assertion qualification, checkers, fault coverage.

I. INTRODUCTION

There exist many assertion qualification methods, both academic [1-3] and industrial ones (e.g. Synopsys Certitude [4]). They have been used mainly for two scenarios. First, qualification allows assessment of the quality of the verification environment. Second, it enables selection of high quality assertions during the assertion mining process [5].

At the same time, reliability checkers have been automatically generated from verification assertions. Probably the best-known software for performing this is the IBM FoCs [6]. However, the number of verification assertions is too high and it result in unacceptable area overheads when implemented as checker circuitry in reliability applications.

The authors of this paper have proposed methods to carry out automated qualification and minimization of checkers using gate-level fault injection [7]. Although accurate, these methods do not scale as well as assertion qualification taking place at the register-transfer level.

In order to derive low-area but high-coverage checker circuitry from a large number of verification assertions this paper proposes the following flow. First, we use the Certitude tool to qualify verification assertions. Certitude, like many of the qualification tools, rely on mutation analysis to assess the quality of assertions.

Subsequently, we propose a minimization algorithm based on greedy heuristics to allow generation of low-area high quality checkers. Finally, the fault detection capabilities of the obtained checkers are evaluated using the framework developed by the authors of the paper in [7] and a discussion of the correlation between verification qualification and checker evaluation results is also provided.

II. ASSERTION QUALIFICATION AND MINIMIZATION

This Section proposes the algorithm for minimizing a set of verification assertions minimizations $c_i \in C$ under a given size constraint W. The algorithm is based on greedy heuristics to iteratively select assertions $c_i \in C$ with maximum additional mutation coverage $f(c_i)$ with regards to the set of already detected faults F' (a subset of the full fault set F), taking into account the size (i.e. overhead area) of the assertion $w(c_i)$ and the size constraint L.

```
Minimize_Set_of_Assertions(C, L)
begin
    C'=∅
    F'=F
    W=0
    while F'≠∅ and W<L do
        Select ci with max f(ci) ∩ F'
        C'=C'∪{ci}
        F'=F' \ f(ci)
        W=W+w(ci)
    end while
    return C'
end
```

Algorithm 1. Greedy minimization of verification assertions

III. Target Design and Its Verification Assertions

The paper includes a preliminary study of the proposed methodology on an example of a network-on-chip routing block based on Logic-Based Distributed Routing (LBDR) [8], which is a scalable solution compared to routing tables. The mechanism describes the topology and the routing function in form of connectivity and routing bits, therefore the logic can be easily re-configured. Routing decision is distributed and only requires local and destination addresses for forwarding flits.

Table 1 presents the six verification assertions developed for the routing block with the purpose of the experiments.

TABLE I. Proposed checkers for ELBDR

Checkers for Routing Logic (LDBR)		
C1	No LBDR output	If no flit arrives (the corresponding input buffer is empty), all the output port signals of LBDR should remain zero.
C2	Valid LBDR output	If there is a request to the routing logic (the corresponding input buffer is not empty), LBDR has to compute at least one valid output direction (according to XY routing).
C3	Single LBDR output	If the corresponding input buffer is not empty (there is a request to LBDR), because of using XY routing, at most only one output port signal of the LBDR logic can become active.
C4	Switch LBDR output	If the corresponding input buffer is not empty (there is a request to LBDR) and a non-header flit has arrived, LBDR outputs should remain the same.
C5, C6	Local Port output	If the corresponding input buffer is not empty (there is a request to LBDR) and a header flit has arrived, the local output should become active only if the packet has reached its destination.

IV. Experimental Results

Fig. 1 shows the results of assertion qualification by Certitude, as well as the minimization results by Algorithm 1.

FAULT ID	CHECKER1 [valid_out]	CHECKER2 [noLBDRout]	CHECKER3 [single_out]	CHECKER4 [switch_out]	CHECKER5 [localport1]	CHECKER6 [localport2]
1	+	-	-	+	-	-
2	-	+	+	+	-	-
3	+	+	+	+	-	-
4	+	-	-	+	-	-
5	-	+	+	+	-	-
6	+	+	+	+	-	-
7	+	-	-	+	-	-
8	-	+	+	+	-	-
9	+	+	+	+	-	-
10	+	-	-	+	-	+
11	-	+	+	+	+	-
12	+	+	+	+	+	+
13	-	-	-	-	-	+
14	-	-	-	-	+	-
15	-	-	+	-	-	+
16	-	-	+	-	+	+
17	-	-	+	-	-	-
18	-	-	-	-	+	-
19	-	-	-	-	-	+
20	-	-	-	-	+	+
21	-	-	-	-	-	-
22	-	-	-	-	+	-
23	-	-	+	-	-	+
24	-	-	+	-	+	+
25	-	-	+	-	-	-
26	-	+	-	-	-	-
27	+	-	-	+	-	+
28	-	+	-	-	-	-
29	+	+	-	+	-	+
30	+	-	-	-	-	+
31	-	+	-	+	-	-
32	+	+	-	+	-	+
33	+	-	-	+	-	-
34	+	-	-	-	-	-
35	-	-	+	-	-	-
36	+	-	+	-	-	-
37	+	-	-	-	-	-
38	+	-	-	-	-	-
39	+	-	-	-	-	-
40	-	-	+	-	-	-
41	-	-	+	-	-	-
42	+	-	-	-	-	-
43	+	-	-	-	-	-
44	-	-	+	-	-	-
45	-	-	+	-	-	-
46	+	-	-	-	-	-
47	-	-	+	-	-	-
48	+	-	+	-	-	-
49	+	-	-	-	-	+
50	-	-	+	-	+	-
51	-	-	+	-	+	-
52	+	-	+	-	+	+
53	+	-	+	-	+	+
54	+	-	+	-	+	+
55	+	-	-	+	-	-
56	-	+	-	-	-	-
57	+	+	-	+	-	-
58	+	-	-	+	-	-
59	+	+	-	+	-	-
60	+	+	-	+	-	-
SUM	31	17	27	22	13	17

Figure 1. Fault table of assertion qualification

Fig. 2 shows the results of assertion qualification by Certitude, as well as the minimization results by Algorithm 1.

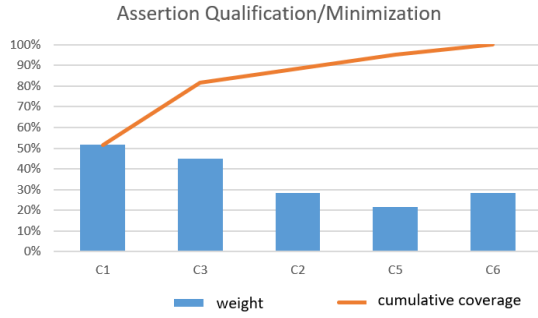


Figure 2. Assertion qualification and minimization

Fig. 3 displays the weight information in terms of the gate-level injection coverage reported by the checker analysis tool [], on the initial set of checkers for the ELBDR. The checker, *err_noLBDRout* (checker C1 in Table 1) is considerably detecting more faults than any other checker. The 5 remaining checkers, in descending order of weights are *err_validLBDRout* (checker C2), *err_singleLBDRout* (checker C3), *err_switchLBDRout* (checker C4), and finally the two *err_localport* checkers (entry C5, C6).

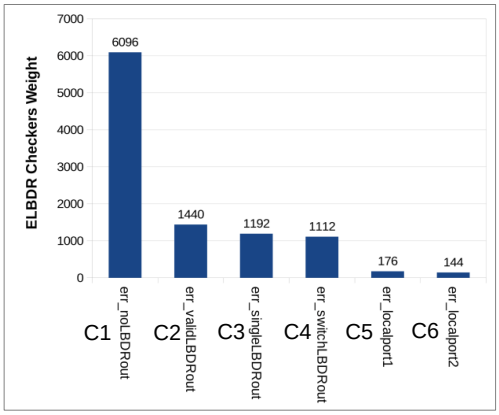


Figure 3. Gate-level qualification results

Minimization of the set of considered checkers by the tool in [7] showed that the three heaviest checkers were *dominating* the three lightest, i.e. the three considered checkers cover all the faults detected by the other checkers. Reducing the used set of checkers to the three most significant ones allows to limit the area overhead to 78.57% over the ELBDR circuit, far lower than 185.71% imposed by the initial non-minimized set of checkers generated from the six verification assertions.

V. Discussion on Preliminary Results

It is important to stress a couple of main aspects indicated by the qualification and minimization experiments presented in the previous section. First, as it can be seen from Figures 1 and 2, the assertion qualification and the subsequent minimization by Algorithm 1 was able to prove that C4 is dominated by other assertions. Therefore, the search space has been pruned already at the level of verification assertions.

Second, similar to gate-level checker qualification presented in Fig. 3, C1-3 are the most significant assertions, although C2, C3 appear in a switched order. Thus, there is a relatively good correlation between the ranking provided by assertion qualification versus the one by checker qualification.

Moreover, if the weight (i.e. size) constraints of the assertions were included then the minimization process may end up with the same result as [7], however without the need to delve to tedious gate-level analysis.

The results presented in this workshop paper is the first insight to assess the feasibility of the proposed methodology. As the next step, we plan to extend the experiments to the entire control part of the network-on-chip router, including also FIFO control and arbitration logic.

VI. Conclusions

The paper investigated the correlation between gate-level fault detection capabilities of concurrent error checkers synthesized from verification assertions and the high-level qualification results for these assertions. For the first time, correlation of assertion quality versus fault detection capabilities of the synthesized checkers was considered and qualified assertions were minimized with considerations of the checker coverage and overhead area.

Experiments carried out on the routing block of a Network-on-Chip (NoC) router showed the feasibility of assessing the fault detection capabilities of checkers by applying assertion qualification. Although assertion quality was not directly proportional to the checker coverage, experiments implementing a heuristic assertion minimization indicate that the optimal solution in terms of coverage/area was achieved without the need to descend to tedious gate-level analysis.

References

[1] S Katz, O Grumberg, and D Geist. Have I written enough properties?—A method of comparison between specification and implementation. *In Proc. of ACM CHARME*, pages 280–297, 1999..

[2] Andrea Fedeli, Franco Fummi, and Graziano Pravadelli. Properties incompleteness evaluation by functional verification. *IEEE Trans. on Computers*, 56(4):528–544, 2007.

[3] N Jayakumar, M Purandare, and F Somenzi. Dos and don'ts of CTL state coverage estimation. *In Proc. of ACM/IEEE DAC*, pages 292–295, 2003.

[4] https://www.synopsys.com/verification/simulation/certitude.html

[5] Jan Malburg, Tino Flenker, Görschwin Fey, "Property Mining using Dynamic Dependency Graphs", *Asia and South Pacific Design Automation Conference (ASP-DAC)*, January 2017.

[6] https://www.research.ibm.com/haifa/projects/verification/focs/

[7] Pietro Saltarelli, Behrad Niazmand, Ranganathan Hariharan, Jaan Raik, Gert Jervan, Thomas Hollstein, Automated Minimization of Concurrent Online Checkers for Networks-on-Chip, *Proceedings of the ReCoSoC'15 Conference*, IEEE, 2015

[8] J. Flich, J. Duato, Logic-Based Distributed Routing for NoCs, *IEEE Computer Architecture Letters*, Vol. 7, No. 1, January-June 2008.

Appendix 6

Publication VI

Ranganathan Hariharan, Tara Ghasempouri, Behrad Niazmand, and Jaan Raik. From rtl liveness assertions to cost-effective hardware checkers. In *2018 Conference on Design of Circuits and Integrated Systems (DCIS)*, pages 1–6. IEEE, 2018

From RTL Liveness Assertions to Cost-Effective Hardware Checkers

Ranganathan Hariharan, Tara Ghasempouri, Behrad Niazmand, Jaan Raik
Department of Computer Systems, Tallinn University of Technology, Tallinn, Estonia
{ranganathan.harihara, tara.ghasempouri, behrad.niazmand, jaan.raik}@ttu.ee

***Abstract*—This paper proposes a methodology for producing a set of high quality hardware checkers from Register-Transfer Level (RTL) assertions. Assertion Based Verification (ABV) has become a highly popular area in design verification. On the other hand, extreme down-scaling of modern technologies has significantly increased the probability of faults occurring during the life-time of the system. To overcome this, concurrent cost-effective checker circuitry is required in order to enable fault resilience of systems. Currently, designing such checker infrastructure is a manual and error-prone work. A possible solution to automate the synthesis of concurrent error checkers is to derive them from verification assertions. However, the number of assertions is generally far too high to allow for area-efficient checking infrastructure. Moreover, the number of liveness assertions generated by automated methods may be too high even for verification purposes. Therefore, there is a need for qualification and minimization of liveness assertions with a prospect of reusing them as hardware safety checkers. In order to derive low-area, high fault coverage hardware safety checkers from a large number of liveness assertions, this paper proposes for the first time a framework for selecting a set of high-quality and minimized liveness assertions by combining a new data mining technique with fault analysis approaches along with assertion conversion methodology that converts liveness assertions into safety assertions. The framework then synthesizes these safety assertions into hardware checkers to be evaluated at the gate level to provide a cost-effective checking infrastructure. Experimental results support the effectiveness of the proposed framework.**

I. Introduction

Due to the increasing complexity of today's digital systems, the amount of time and man-power that is invested in finding and removing bugs is growing. To overcome this problem and to develop systems without bugs, verification techniques have arisen which check if a system meets its specification and thereby fulfills its intended purpose [1]. Among all of these techniques, ABV has become a popular means for catching and eliminating errors. At the same time, due to the growing failure-rates, process variations and time-dependent degradation of modern chip technologies, it is imperative to develop cost-effective means for protecting systems against faults occurring in the field, during their life-time.

Thus, concurrent on-line checker circuitry is required to monitor the fault-free functioning of the system hardware. Such checkers are normally designed ad-hoc or by synthesizing them from verification assertions. However, the number of assertions in the verification environment is generally far too high to allow for area-efficient checking infrastructure. Moreover, the number of liveness checkers generated by automated methods (e.g. [2], [3], [4]) may be too high even for verification purposes. Therefore, there is a need for qualification and minimization of liveness assertions with a prospect of reusing them as hardware safety checkers.

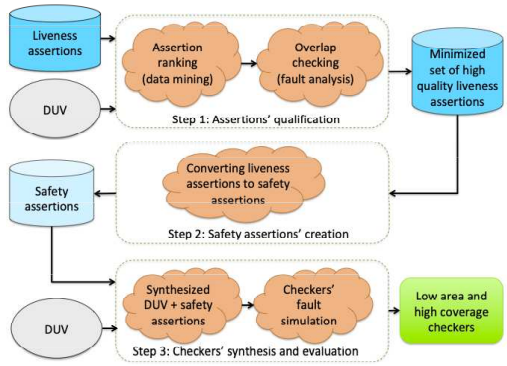


Fig. 1: Overview of the proposed methodology

A verification environment consists of a set of assertions that collectively are capable of detecting a range of design bugs. However, not all the assertions are essential in order to detect this range: some assertions are dominated by others, or by a set of other assertions, some assertions are equivalent in terms of bug detection capabilities, etc. Discarding such assertions which do not detect any unique bugs leads to obtaining a set of *minimized* assertions. Of course, it is not possible to enumerate all possible bugs, and therefore, fault models are applied in order to estimate the coverage of different assertions. This assertion quality estimation task is called assertion *qualification*. While there exist several works that address assertion qualification and minimization [2], [3], [4] as well as qualification and minimization of checkers at the gate-level [5], [6], to the best of our knowledge, this is the first paper that applies high-level assertion qualification/minimization with the goal of generating low-area high-quality checker circuitry.

Figure 1 presents the overview of the proposed methodology. Looking at Step 1 in Figure 1, *Assertion ranking* ranks and selects the high quality assertions passing a preset threshold in order to pass them to *Overlap checking*. Preliminary selection by *Assertion Ranking* gives the framework the advantage of obtaining shorter execution time since the *Overlap checking* phase is based on fault analysis which has high complexity. Application of this methodology prevents the

978-1-7281-0171-2/18/$31.00 ©2018 IEEE

involvement of assertion sets that are incomplete (i.e. unable to cover all expected behaviors of the Design Under Verification (DUV), inconsistent (i.e. with contradicting assertions), redundant (i.e. with assertions that are logical consequence of others), and including vacuous assertions (i.e. assertions that are true, independently from the DUV and thus, irrelevant) into the verification process. Of the different strategies to generate and define assertions there are typically two types of assertions distinguished: safety and liveness assertions. A *safety assertion* stipulates that 'undesired things' do not happen during execution of a program and a *liveness assertion* stipulates that 'desired things' do happen.

Specifications of most systems contain liveness parts [7]. Moreover, automatically generated assertions that are derived from the systems' behavior are almost exclusively liveness assertions. However, function of the checker circuitry is based on safety assertions For these reasons, a conversion procedure that translates liveness assertions into equivalent safety assertions using transposition logic is described in this work. (See Step 2 in Figure 1). The methodology continues with synthesis of the safety assertions into hardware checkers whose quality is evaluated by gate-level fault simulation (Step 3 in Figure 1). As a result, low-area high coverage checker circuitry is obtained.

The rest of the paper is organized as follows. Section II presents related work. Section III introduces preliminary definitions. Section IV presents the methodology. Section V describes the experimental results and finally Section VI concludes the paper.

II. Related Work

There are several techniques in the state of the art for generating assertions, [5] [4], [8], [9] and in many of them minimization of the number of generated assertions is addressed. However in all techniques, the generated assertions suffer from shortcomings such as vagueness, incompleteness and redundancy. As a result, a false sense of security can emerge in the verification flow which is caused by a low quality set of assertions. Moreover, in most cases, the number of generated assertions is very high and verifying a system through all the assertions needs a long simulation time. All the above-mentioned problems show that an assertion qualification and minimization phase is necessary in the ABV so that the verification environment would be free from using inconsistent, redundant and vacuous assertions. Current approaches for ABV are still unsatisfactory from this point of view.

In [10], a stressing phase is proposed only to verify the likelihood that mined assertions are globally satisfied (i.e. not only for the execution traces analyzed by the miner). However, no strategy is proposed to measure their quality in covering DUV behaviors or selecting the best minimized assertions. In [8], quality estimation is based on the number of propositions included in the antecedent of the assertion. It is according to the fact that an assertion with a lower number of propositions in its antecedent has a higher input space coverage than one with many propositions in its antecedent. However, the correlation between the antecedent and the consequent of an assertion is not considered. To solve this drawback, in [11] a ranking function is proposed that evaluates the quality of the mined assertions in terms of cause-effect relationships between antecedent and consequent of an assertion. In [12] the quality of assertions are estimated based on their amount of frequencies and correlation during the simulation. However, the work does not consider assertions with low number of frequencies which may cover the corner cases of a design. In [13], instead, a metric is introduced to rank assertions based on their ability to cover corner cases. Moreover, it does not take into account assertions which cover the general behavior of the design. Finally, in [9], mined assertions are said to be generally ranked according to their frequency of occurrences and time of first occurrence, however, no specific approach is presented.

As an opposite class of approaches, coverage metrics have been widely studied for qualification of assertions [14], [15], [16], [17]. Most of these works rely on fault analysis, which requires perturbing the DUV implementation by injecting mutations (faults) to check, either statically [15], [16] or dynamically [17], whether they change the truth values of the assertions; Faults that do not cause a change are said to be *not observed*. Assertions that observe a lower number of faults have less quality than assertions detecting a higher number of faults. Not observed faults generally highlight areas/behaviors of the DUV that are not covered by any of the defined assertions showing a hole within the coverage.

Dynamic approaches such as [17] scale better with respect to static techniques, however, they still require long simulation runs for checking each assertion for each fault with a significant set of test-benches. When the number of assertions is very high, as in the case of assertions extracted automatically, evaluating their quality through fault analysis becomes a very time-consuming task.

Error-checkers are to be generated from safety assertions, whereas automated assertion generation provides mostly liveness assertions. Therefore, a liveness to safety generation step is proposed in this paper. In the state-of-the-art, there exist some works to convert liveness to safety assertions. For example [18] uses finite state machine and reachability checking to translate liveness assertions to safety assertions. In [19] the same technique as [18] has implemented but using an unbounded state machine. [20] describes a technique to distinguish between a liveness and a safety assertion but no specific method for translation of liveness to safety assertions. The technique which is proposed in this work, is translation of liveness assertions to safety with transposition logic which is a fast and cost-effective approach.

Last but not least, there exist previous works in gate-level minimization of checkers at the gate level [5], [6]. However, the scalability of minimization at the gate-level is very low due to the excessive run-times of fault injection, iterative synthesis and minimization at this low level of abstraction.

To fill the gaps described above, the methodology proposed in this paper provides the following contributions:

- First, an advanced assertion ranking approach has been introduced based on combining three different metrics adapted from data mining, to evaluate assertions' quality

from different conditions (See Section IV-A1). Compared to similar approaches in the state of the art, it provides higher correlation of assertion quality with fault coverage of the obtained checkers.
- Second, combining a fault analysis approach along with a data mining approach for assertion qualification to get the advantages of both techniques, the former providing high accuracy and the latter very short execution time.
- Last but not least, different from existing checker synthesis and minimization approaches, the methodology proposed in this paper operates on high level of abstraction, therefore providing better scalability with the circuit size.

III. Preliminaries

Definition 1: A ***fault*** f_i is a local alteration of the DUV's source code that perturbs its functionality.

Definition 2: A fault f_i is called an ***observable fault*** if, in comparison with a fault-free DUV, its effect is visible as an alteration in the DUV's primary outputs. A fault f_i is ***covered*** by an assertion a_j if assertion a_j fails when the fault f_i is observed at primary outputs.

Definition 3: An ***assertion*** is a composition of propositions, connected via temporal operators according to some temporal logic.

Definition 4: An assertion with higher degree of ***Quality*** is the one that observes a higher number of faults.

Definition 5: An assertion is composed of ***Antecedent*** i.e., the left side of an implication and ***Consequent*** i.e., the right side of an implication. An example of an assertion applied in the Experimental Results section:

$$Empty = False \;\&\&\; (flit = Header \;||\; flit = Payload \;||\; flit = Tail) \;\rightarrow\; oNport = True \;||\; oWport = True \;||\; oSport = True \;||\; oLport = True$$

where the ***Antecedent*** is:

$$Empty = False \;\&\&\; (flit = Header \;||\; flit = Payload \;||\; flit = Tail)$$

and the corresponding ***Consequent*** is:

$$oNport = True \;||\; oWport = True \;||\; oSport = True \;||\; oLport = True$$

Definition 6: Given a set of items I and the corresponding dataset of D, a rule $X \rightarrow Y$ (X being the *antecedent* and Y being the *consequent*) has ***support*** S if X and Y occur concurrently in S percent of transactions in D.

Definition 7: Given a set of items I and the corresponding dataset of D, the ***Correlation Coefficient*** of the rule $X \rightarrow Y$ is the covariance of X and Y divided by the product of their individual standard deviations.

Definition 8: ***Strength Measure*** is a product of quantities such as Support (Definition 6) and Correlation Coefficient (Definition 7) but with giving priority in the region of rules/assertions with low occurrences but highly correlated with other rules/assertions.

IV. Methodology

In order to derive cost effective hardware safety checkers from a large number of liveness assertions, a methodology based on a tool framework is shown in Figure 1. This methodology is applicable independently from the way assertions are defined/generated. The assumption is that they are represented in the form of $A \rightarrow C$ where A and C are *antecedent* and *consequent* (Definition 5), respectively. Assertions go through an *Assertion ranking* phase to evaluate their quality based on data mining metrics. The highest ranked assertions are further forwarded to an *Overlap checking* phase implementing mutation based assertion qualification to analyze the observed faults by each assertion. As a result, a set of high quality minimized assertions are sent to the liveness to safety assertion conversion step. The output of this step is a set of safety assertions. In the last step, DUV and safety assertions are synthesized to be evaluated at the gate-level using fault injection and simulation. So, as stated above, the work flow of the proposed methodology is divided into 3 main steps: assertions' qualification, liveness to safety conversion and checkers' synthesis and evaluation. The three steps are explained in more detail below.

A. Assertion qualification

This step consists of two main phases: *Assertion ranking* and *Overlap checking*. In *Assertion Ranking*, an assertion qualification tool Shayan [12] is applied on liveness assertions to estimate their *quality* (Definition 4) based on data mining metrics. The high quality assertions, selected by Shayan go through the Overlap checking phase for fault analysis utilizing the Synopsys Certitude qualification tool [21]. The hypothesis is that assertions with higher degree of quality are more effective in the verification process. Thus, Shayan selects assertions with the degree of quality above a preset threshold and forwards them to the *Overlap Checking* phase for fault analysis. The main drawback of fault analysis approaches is their long simulation time, since the effect of a fault (Definition 1) that has been injected needs to be evaluated by simulation. The above-mentioned preliminary selection by Assertion Ranking leads to reduction of this simulation time.

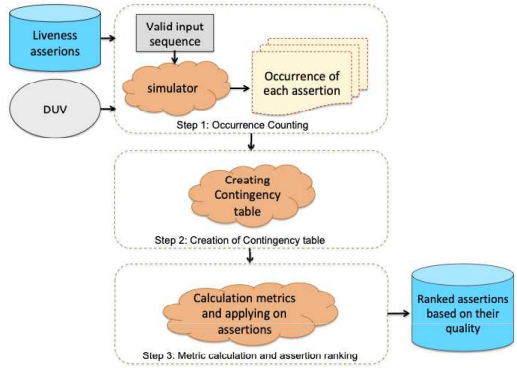


Fig. 2: Overview of Shayan (Assertion ranking)

1) Assertion Ranking: In the core of assertion ranking, a data mining based tool called Shayan is placed. From the point of view of general concept, data mining [22] and assertion ranking share the same idea (extracting rules from data), but they have several differences that make it practically different how these metrics are computed and interpreted for evaluating

the quality of assertions. Shayan calculates a metric called Q which is calculated individually for each assertion. Q is the linear combination of Support (Definition 6), Correlation Coefficient (Definition 7) and Strength Measure (Definition 8). The higher the value of Q, the higher the quality of the assertion would be. Figure 2 shows the internal design of Shayan, composed of three metrics, i.e.,*Occurrence counting*, *Contingency table creation* and *Metric calculation*.

- Occurrence counting: Liveness assertions and the DUV are inputs of the work flow. In the first step, set of valid input sequences are connected to a simulator to extract information about occurrences of assertions during the simulation. The number of times an assertion is holding in the valid input sequences is computed. Then, each assertion is decomposed into *antecedent* and *consequent* and their respective frequencies in the valid input sequences are computed.
- Creation of contingency table: At this stage, the necessary ingredients are ready for Creating contingency tables, see (Table I).

	C	$\bar{C}$	
A	f_{11}	f_{10}	f_{1X}
$\bar{A}$	f_{01}	f_{00}	f_{0X}
	f_{X1}	f_{X0}	fXX

TABLE I: Contingency table for $A \rightarrow C$.

The computation of the contingency table is based on counting occurrences of antecedent/consequent and the assertions respectively. Given an assertion $A \rightarrow C$, its contingency table represents the relation between A and C. The cells of contingency table contain the following information (Table I): Cell f_{11} represents the number of times where A is true and C is true in the valid input patterns; Cell f_{10} represents the number of times where A is true but C is false in the valid input patterns. Cell f_{01} is the dual of f_{10}, *i.e.*, it is the number of times where A is false and C is true in the valid input patterns, *i.e.*, it is the sum of occurrences of assertions $A' \rightarrow C$ included in the considered assertion set with $A \neq A'$. In this case, A and A' can also be conflicting because this doest not represent an inconsistency for the assertion set. Cell f_{00} is the number of times an assertion is not true through the simulation. Cell f_{1X} is the sum of cells f_{11} and f_{10}. Cell f_{0X} is the sum of cells f_{01} and f_{00}. Cell f_{X1} is the sum of cells f_{11} and f_{01}. Cell f_{X0} is the sum of cells f_{10} and f_{00}. Cell f_{XX} is the grand total.

The corresponding contingency tables are reported in Table II. For example, for assertion $validLB$, f_{11} correspond to the total number of occurrences of $validLB$ in the analyzed valid input sequences; f_{10} is equal to 0, since antecedent A does not appear in none of the other assertions; f_{01} is 0 since consequent of the assertion does not appear in none of the other assertions; and finally, f_{00} is obtained by summing the occurrences of all the other assertions except $ValidLB$. Similar considerations allow computing values for all the other cells of Table II.

- Metric calculation and assertion ranking: Contingency tables provide basic ingredients for computation of Support and Correlation Coefficient, Strength Measure and their linear combination Q. Concerning support, according to (Definition 6), it is computed using the following formula:

$AssertionID$	f_{11}	f_{10}	f_{01}	f_{00}
$validLB$	468	0	0	2827
$noLB$	436	0	0	2859
$singleLB$	481	0	0	2814
$switchLB$	361	0	0	2934
$port1$	524	1025	0	1746
$port2$	516	1033	0	1746

TABLE II: Contingency tables of assertions reported in Table I.

$$Support = \frac{f_{11}}{f_{XX}} \tag{1}$$

The Correlation Coefficient for an assertion according to (Definition 7) is computed using the following formula:

$$CC(A,C) = \frac{f_{11}f_{XX} - f_{1X}f_{X1}}{\sqrt{f_{1X}f_{0X}f_{X1}f_{X0}}} \tag{2}$$

The computation of the Strength Measure for an assertion according to (Definition 8) is computed by:

$$Strength\ Measure = \sqrt{\frac{f_{11}^2}{|f_{X1} - f_{X0}| \cdot |f_{1X} - f_{0X}|}} \tag{3}$$

According to equation 1 the support ranks in the highest positions assertions that occur frequently in the execution traces. However, we can have specific assertions that occur very rarely because they refer to the corner cases and thus equation 3 has been proposed. On the other hand, the correlation coefficient privileges assertions where the number of occurrences of the antecedent better matches the number of occurrences of the consequent, but assertions where these numbers are low could be extracted by chance without representing a real behavior of the DUV. For this reason a combination of support, correlation coefficient and strength measure provides a more accurate estimation of assertion interestingness. Thus, we propose measuring the quality of an assertion A through the following formula:

$$Q(A) = \alpha * s_n(A) + (1-\alpha) * \rho_n(A) + (1-\alpha) * strength_n(A) \tag{4}$$

where, $\alpha \in [0,1]$, and $s_n(A)$ and $\rho_n(A)$ are the value obtained by normalizing , respectively, the support s, the correlation coefficient ρ and Strength measure $strength$ of A with respect to the whole set of analysed assertions. By varying of α the role of support becomes more or less important with respect to the role of the correlation coefficient and strength in determining the final estimation of assertion quality. In our experiments best results have been obtained with $\alpha = 0.4$.

In Table III the metrics with their corresponding values for each assertion are represented. Assertions noLB, validLB, singleLB and switchLB are rank better than assertions port2 and port1. Thus, for the next step the best 4 assertions are selected and port1 and port2 are discarded.

TABLE III: calculated metrics for each assertion

	S	CC	Strenght	Q
noLB	0.04	1.0	0.05	0.64
validLB	0.96	1.0	0.98	1.57
singleLB	0.04	1.0	0.05	0.64
switchLB	0.03	0.55	0.03	0.36
port2	0.001	0.45	0.001	0.27
port1	0.01	0	0.01	0.01

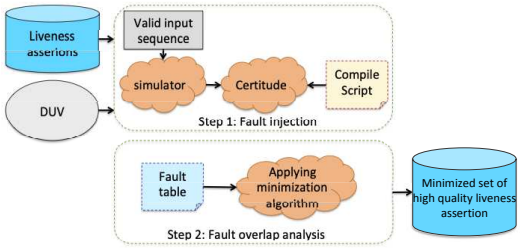


Fig. 3: Overview of overlap checking phase

2) Overlap checking: The *Overlap Checking* phase calculates which faults are covered by which assertions. Shayan, which was described above, has the ability to rank the assertions based on data mining metrics but it can not provide any information whether two assertions have the same set of covered faults (i.e. the assertions are *equivalent*) or one is subset of the other assertions' covered faults (i.e. *dominated* by the other assertion) etc. Such equivalence and dominance relationships between assertions allow minimizing the set of assertions selected for synthesis and gate-level evaluation. Thus, we applied the Overlap Checking framework which consists of two steps, i.e., *Fault injection* and *Fault overlap analysis* (See Figure 3).

As shown in Figure 3, the DUV and selected liveness assertions together with the valid input sequence (i.e. the verification environment) are fed into the Certitude tool that performs the fault analysis. The output of Step 1 is the fault table showing which assertion a_j covers which faults f_i.

The following algorithm is applied for minimizing the number of assertions based on the fault table information. The algorithm is based on iterative $implication$ and *greedy selection* operations. Two types of implications are used. First, $unique$ assertion a_j, which cover some fault f_i that is not covered by any other assertions are identified and removed from the table. Second, it is said that assertion a_j $dominates$ assertion a_k, if all the faults covered by a_k is a subset of the faults covered by a_j. Note, that equivalence of two assertions a_j and a_k is a special case of dominance, where a_j and a_k mutually dominate each another.

If after performing the implications the set of selected assertions are not covering all the faults in the fault table, a *greedy selection* operation is performed. The algorithm selects an assertion that covers the greatest number of faults not yet covered by the set of selected assertions. The algorithm will complete when the selected assertions cover all the faults in the fault table.

while *exist faults uncovered by assertions* **do**
  while *implications provide new assertions* **do**
    Select unique assertions;
    Remove dominated assertions;
  end
  Make a greedy selection;
end

Algorithm 1: Fault table minimization

B. Converting liveness assertions to safety checkers

The next step is converting this set of high quality liveness assertion to safety. During this step liveness assertions in the form $A \rightarrow C$ are converted using transposition into the equivalent safety assertions $A \wedge \overline{C}$.

Consider a real example of a liveness assertion for the design LBDR which is studied in the experimental results. The assertion is describing one of the behaviors of the design as follows: "Whenever there is a request (i.e., the HEADER flit contains the destination address) LBDR has to compute at least one valid output direction (according to XY Routing) to pass the flits from the input buffer to the respective output port":

E_validLBDRout_liveness = !Eempty && (flit_id == HEADER || flit_id == PAYLOAD || flit_id == TAIL => oNport || oWport || oSport || oLport.

Here, !Eempty && (flit_id == HEADER || flit_id == PAYLOAD || flit_id == TAIL is the antecedent A and oNport || oWport || oSport || oLport is the consequent C, respectively. After transposition the liveness assertion is converted to the following safety assertion: Eerr_validLBDRout_safety = !Eempty && flit_id == HEADER) || flit_id == PAYLOAD || flit_id == TAIL && !oNport && !oWport && !oSport && !oLport.

C. Checkers' synthesis and evaluation

During this final step, the safety assertions obtained after the minimization and conversion steps are synthesized to the gate level to obtain concurrent error checkers. The checkers along with the synthesized DUV are fault simulated using the Turbo Tester software [23] in order to evaluate the coverage of gate-level stuck-at faults achieved by the minimized set of checkers.

V. Experimental Results

This section presents the experiments carried out by applying the proposed methodology. To this end, three different modules related to the control part of a 2D Network-on-Chip (NoC) router are considered as examples. Three design modules are taken into consideration for experimental study: Logic Based Distributed Routing (LBDR), Arbiter and a complex Arbiter with timeout. The assertions are manually generated for two design modules: LBDR, Arbiter and are automatically generated for Arbiter with timeout.

The fault simulation experiments in this paper are performed on an IBM System x3500 M3 7380 with two 6-core Intel Xeon E5690 3.47GHz processors and 96 GB RAM.

Table IV shows the number of assertions considered initially at the beginning of each phase - assertion ranking phase (described in Section IV-A1) and a minimized set of checkers subsequent to the overlap checking phase (described in Section IV-A2). As it can be seen, the initial number of assertions was minimized to 7.14-50%.

TABLE IV: Minimization of the number of assertions

	Initial		Assertion Ranking		Overlap Checking	
Example	#	%	#	%	#	%
LBDR	6	100	4	66.7	3	50
Arbiter	28	100	19	66.7	2	7.14
Arbiter with timeout	139	100	94	66.7	18	12.95

The area consumption of the synthesized checkers from the assertions is summarized in Table V. The synthesis of the checkers has been performed via Synopsys Design Compiler [24] using Class library, leading to results in terms of number of NAND2 gate equivalents.

TABLE V: Area consumption of checkers (number of NAND2 gates)

Example	Initial set	Minimized set	
LBDR	60	29	48.3%
Arbiter	128	33	25.7%
Arbiter with timeout	428	91	21.2%

Run times of each phase and the fault coverage calculated in Checkers' synthesis and evaluation step (explained in Section IV-C) are summarized in Table VI. The fault model applied was Single Event Transients (SETs) in the logic of the designs.

TABLE VI: Runtime and Fault Coverage

Example	Assertion Ranking	Overlap Checking	Checker's Simulation	Fault Coverage
LBDR	4.15 s	110 s	0.01 s	99.838%
Arbiter	3.04 s	280 s	> 0.01 s	100%
Arbiter with timeout	5.01 s	1293 s	12.6 s	99.833%

It is observable that the SET fault coverage for the initial set of checkers was 100% and for the minimized sets it exceeded 98.83%. The run time of simulation is 0.01 seconds for the *LBDR* and *Arbiter*, respectively and 12.6s for *Arbiter with timeout*.

VI. Conclusions

This paper proposed a framework for selecting a minimal set of high-quality liveness assertions to be implemented as hardware checkers by combining a new data mining technique with fault analysis approaches. An assertion conversion methodology was proposed which converts liveness assertions into their safety equivalents, which are further synthesized to hardware checkers to be evaluated at the gate level to provide a cost-effective checking infrastructure. Experimental results on example designs show that the initial checker area was minimized to 21.2-48.3% obtaining more than 99.83% single even transient coverage by applying the proposed approach.

VII. Acknowledgement

The work has been supported by H2020 Twinning TUTORIAL, Estonian institutional research grant IUT 19-1, by the Estonian Center of Excellence in IT EXCITE funded by the European Regional Development Fund, and supported by Estonian IT Academy program.

References

[1] H. Foster and D. Lacey, *Assertion-based design 2nd edition*. Springer, 2004.
[2] W. Li, A. Forin, and S. A. Seshia, "Scalable specification mining for verification and diagnosis," in *Proc. of ACM/IEEE CAD*.
[3] L. Liu and S. Vasudevan, "Automatic generation of system level assertions from transaction level models," *Journal of Electronic Testing*, vol. 29, no. 5, pp. 669–684, 2013.
[4] A. Danese, F. Filini, T. Ghasempouri, and G. Pravadelli, *Automatic Generation and Qualification of Assertions on Control Signals: A Time Window-Based Approach*. Springer, 2016.
[5] P. Saltarelli, B. Niazmand, R. Hariharan, J. Raik, G. Jervan, and T. Hollstein, "Automated minimization of concurrent online checkers for network-on-chips," in *ReCoSoC*, 2015.
[6] S. P. Azad, B. Niazmand, A. K. Sandhu, J. Raik, G. Jervan, and T. Hollstein, "Automated area and coverage optimization of minimal latency checkers," in *European Test Symposium (ETS)*, 2017.
[7] P. K. Nalla, R. K. Gajavelly, H. Mony, J. Baumgartner, and R. Kanzelman, "Effective liveness verification using a transformation-based framework," in *International Conference on VLSI Design and International Conference on Embedded Systems*, Jan 2014, pp. 74–79.
[8] S. Hertz, D. Sheridan, and S. Vasudevan, "Mining hardware assertion with guidance from static analysis," *IEEE Trans. on CAD*, vol. 32, no. 6, pp. 952–965, 2013.
[9] W. Li, A. Forin, and S. A. Seshia, "Scalable specification mining for verification and diagnosis," in *Proc. of ACM/IEEE DAC*, 2010.
[10] A. Danese, T. Ghasempouri, and G. Pravadelli, "Automatic extraction of assertions from execution traces of behavioural models," in *Proc. of ACM/IEEE DATE,* 2015.
[11] M. Bertasi, G. Di Guglielmo, and G. Pravadelli, "Automatic generation of compact formal properties for effective error detection," in *Proc. of ACM/IEEE CODES+ISSS*, 2013, pp. 1–10.
[12] T. Ghasempouri and G. Pravadelli, "On the estimation of assertion interestingness," in *2015 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, Oct 2015, pp. 325–330.
[13] T. Ghasempouri, S. P. Azad, B. Niazmand, and J. Raik, "An automatic approach to evaluate assertions' quality based on data-mining metrics," in *2018 International Test Conference in Asia (ITC-Asia)*, August 2018.
[14] S. Katz, O. Grumberg, and D. Geist, "Have I written enough properties? — A method of comparison between specification and implementation," in *Proc. of ACM CHARME*, 1999, pp. 280–297.
[15] H. Hoskote, T. Kam, P. H. Ho, and X. Zao, "Coverage estimation for symbolic model checking," in *Proc. of ACM/IEEE DAC*.
[16] N. Jayakumar, M. Purandare, and F. Somenzi, "Dos and don'ts of CTL state coverage estimation," in *Proc. of ACM/IEEE DAC*.
[17] A. Fedeli, F. Fummi, and G. Pravadelli, "Properties incompleteness evaluation by functional verification," *IEEE Trans. on Computers*, vol. 56, no. 4, pp. 528–544, 2007.
[18] "Liveness checking as safety checking," *Electronic Notes in Theoretical Computer Science*, vol. 66, no. 2, pp. 160 – 177, 2002.
[19] V. Schuppan and A. Biere, "Liveness checking as safety checking for infinite state spaces," *Electronic Notes in Theoretical Computer Science*, vol. 149, no. 1, pp. 79–96, 2006, proceedings of the 7th International Workshop on Verification of Infinite-State Systems (INFINITY 2005).
[20] B. Alpern and F. B. Schneider, "Recognizing safety and liveness," Ithaca, NY, USA, Tech. Rep., 1986.
[21] [Online]. Available: https://www.synopsys.com/verification/simulation/certitude.html
[22] P.-N. Tan, V. Kumar, and J. Srivastava, "Selecting the right interestingness measure for association patterns," in *ACM SIGKDD*.
[23] R. Ubar, S. Devadze, J. Raik, and A. Jutman, "Ultra fast parallel fault analysis on structurally synthesized bdds," in *12th IEEE European Test Symposium (ETS'07)*, May 2007, pp. 131–136.
[24] (1994) Synopsys design compiler. http://www.synopsys.com/.

Curriculum vitae

1. Personal data

Name	Ranganathan Hariharan
Date and place of birth	20 May 1987 Chennai, India
Nationality	Indian

2. Contact information

Address	Tallinn University of Technology, School of Information Technologies, Department of Computer Systems, Akadeemia tee 15A, 12618 Tallinn, Estonia
Phone	+358 408209435
E-mail	ranganathan.harihara@taltech.ee, ranganathanh87@gmail.com

3. Education

2014–2019	Tallinn University of Technology, School of Information Technologies, PhD studies
2010–2011	University of Glasgow, M.Sc. in Telecommunication Electronics
2004–2008	RMK Engineering College (Affiliated to Anna University), B.E. in Electronics and Communication Engineering

4. Language competence

Tamil	native
English	fluent
Hindi	basic

5. Professional employment

2014–2014	Tallinn University of Technology, Early-stage researcher
2014–Present	Nokia, SoC Verification Engineer

Elulookirjeldus

1. Isikuandmed

Nimi	Ranganathan Hariharan
Sünniaeg ja -koht	20.05.1987, Chennai, India
Kodakondsus	India

2. Kontaktandmed

Aadress	Tallinna Tehnikaülikool, Usaldusväärsete arvutisüsteemide keskus, Arvutisüsteemide Instituut, Akadeemia tee 15A, 12618 Tallinn, Estonia
Telefon	+358 408209435
E-post	ranganathan.harihara@taltech.ee, ranganathanh87@gmail.com

3. Haridus

2014–2019	Tallinna Tehnikaülikool, Infotehnoloogia teaduskond, Doktoriõpe, arvuti- ja süsteemitehnika õppekava
2010–2011	Glasgow Ülikool, magistrikraad telekommunikatsiooni elektroonikas
2004–2008	RMK Engineering College (Anna Ülikooli filiaal), Elektroonika ja sidetehnoloogia bakalaureus

4. Keelteoskus

Tamili keel	emakeel
Inglise keel	kõrgtase
Hindi Keel	põhitase

5. Teenistuskäik

2014–2014	Tallinna Tehnikaülikool, nooremteadur
2014–…	Nokia, insener