# Structural Software-Based Self-Test of Network-on-Chip

Atefe Dalirsani, Michael E. Imhof, Hans-Joachim Wunderlich
Institute of Computer Architecture and Computer Engineering, University of Stuttgart, Germany
{dalirsani, imhof}@iti.uni-stuttgart.de, wu@informatik.uni-stuttgart.de

***Abstract*—Software-Based Self-Test (SBST) is extended to the switches of complex Network-on-Chips (NoC). Test patterns for structural faults are turned into valid packets by using satisfiability (SAT) solvers. The test technique provides a high fault coverage for both manufacturing test and online test.**

***Index Terms*—Network-on-Chip (NoC), Software-Based Self-Test (SBST), Automatic Test Pattern Generation (ATPG), Boolean Satisfiability (SAT)**

## I. INTRODUCTION

Network-on-Chips are an alternative for many core System-on-Chips (SoC) enabling high-performance communication, with advantages for bandwidth, latency and dependability. As the technology scales down and the operation frequency increases, systems become more vulnerable to complex defect mechanisms such as latent defects, timing variations and aging. Consequently, low-cost testing methods integrated into the chip become attractive for both production testing and in-field testing.

An NoC comprises a large number of switches which are connected to each other via communication links. Test infrastructure like scan-design may be employed to apply test patterns and reach acceptable test time [1, 2]. Even in presence of defective components, an NoC may still operate due to its implicit redundancy, in the worst case with a degraded performance [3].

Built-In Self-Test (BIST) for in-field testing integrates the test pattern generation and test response evaluation on-chip. *Structural* testing targets faults of a predefined structural fault model like stuck-at faults and allows measuring the fault coverage as a quality metric. Depending on the product quality requirements, more complex fault models like transition faults, delay faults, bridging faults or cross-talk faults may be applied as well. In contrast, *functional* testing targets certain functionalities of a system, for instance the instructions of a microprocessor. As it is impossible to test the functionality completely, e.g. all possible arguments of an operation, or all possible instruction orders, it is hard to estimate the final quality of such a functional test and the structural fault coverage is rather limited [4].

For functional testing of NoC, references [5–7] use data encoding in order to detect faults in the communication links and the switch datapath elements. Test repetition classifies the transient or permanent nature of the fault. In [8], a built-in self-test and self-diagnosis circuit is designed for the mesh topology which traces 20 datapaths (east to west, east to north, etc.) in order to detect and locate faulty FIFOs and MUXes in the switches. Similarly, [9] introduces a functional fault model based on the 20 datapaths in the switch, targeting the multiplexers of the crossbar in a 2D mesh NoC with deterministic XY routing. In [10], based on a system level fault model, an online fault detection for the NoC switches has been proposed. In [6], a fault detection and diagnosis mechanism targeting permanent faults is presented that uses error syndrome collection and packet/flit counting. The method uses error detecting codes in order to deal with data faults, and introduces dropped flits/packets, spurious flits/packets and misrouted packets to describe control faults in the switch. In [11], all link and router level faults are mapped to uni-directional links. Based on this fault model, a reconfiguration scheme is developed that improves the performance and connectivity in presence of faults. These techniques will provide some confidence about the correct functionality of certain parts of the switch, but a high structural fault coverage is not explicitly targeted.

For microprocessors, the benefits of structural testing and functional testing are combined by a so-called structural software-based self-test (SBST) [12–16]. Here, ATPG provides deterministic, structural test patterns which are transformed into arguments of a sequence of valid instructions. In a similar way, the paper at hand transforms deterministic test patterns into valid packets of an NoC.

This paper presents for the first time a structural software-based self-test (SBST) scheme for Networks-on-Chip. Structural faults in NoC switches and interconnects are targeted and tested by valid NoC-packets without the need for dedicated test infrastructure. In fact, the presented SBST scheme combines the advantages of state-of-the-art structural and functional test approaches for NoC infrastructure.

Fig. 1 illustrates the principle of SBST in the scope of NoCs. As an example, in the mesh topology, every switch is connected to four neighboring switches and a Processing Element (PE) is attached to each switch.

The Switch Under Test (SUT) is tested by applying a set of test patterns to its incoming links and observing the test responses at the outgoing links. The test patterns form valid NoC packets, hence they can be fed to the NoC externally as long as they pass the SUT, without requiring to put the system in a non-functional test mode. Here, we assume that the set of test packets is generated by software running on the processing elements (PE) attached to the NoC. The generated test packets target structural faults in the SUT and its links under a single fault assumption. The resulting test

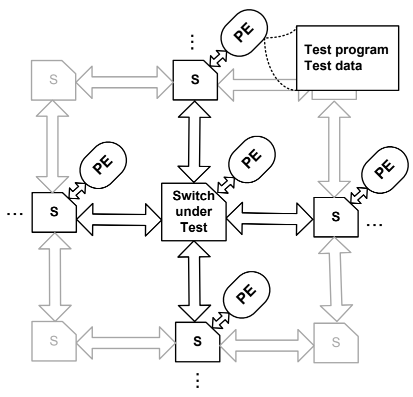

Fig. 1. Target architecture for SBST of a switch

responses are captured and evaluated by the test programs in the ambient PEs. The SBST starts when all PEs surrounding the SUT have sufficient resources to run the test program. A local signal (such as the Ack/Req. signal used for link flow control) can be utilized to synchronize the launch of the test programs running on the PEs involved in testing a SUT (Fig. 1). The switches and PEs give the highest priority to test packets and bypass their caches. Since the switches are identical, the SUT access time through all the incoming links are deterministic. Moreover, once the test begins, normal packets are not routed through the SUT. The complete NoC is tested by consecutively testing all contained switches. Depending on the network topology and the switch location, the SBST pattern generation is adjusted such that only available neighboring PEs contribute in testing. For example, in a 2D mesh a switch at the boundary has three neighbors, consequently its test patterns contain input values for only three input ports of the switch.

The key concept of SBST for NoC switches is the generation of efficient test patterns that achieve a high fault coverage. Since processing elements have only access to the functional input and outputs of the switches, functional patterns are used in SBST. In contrast to scan-based testing, direct controllability and observability of the sequential states of the switch (i.e. pseudo primary inputs and outputs) is not possible. Therefore, achieving a high structural fault coverage with SBST patterns poses a challenge which is tackled in this work.

The SBST pattern generation is modeled as a Boolean satisfiability (SAT) problem in conjunctive normal form (CNF). The resulting SAT instance reflects three aspects inherent to SBST test generation for NoCs:

1) Circuit Model: The combinational logic and interconnect of the Switch under Test is described in CNF using the Tseitin transformation [17]. The sequential behavior is modeled by time-frame expansion.
2) Fault Model: Classic fault models such as stuck-at faults are not sufficient to reason about arbitrary defects in recent process technologies. Hence, the Conditional Line Flip (CLF) calculus [18] is used as a generalized fault model to describe arbitrary defect mechanisms in the switch logic and the links.
3) NoC-Packet Model: Finally, only valid packets are accepted as test patterns in order to utilize the packet-based communication platform of the NoC for SBST.

The rest of the paper will discuss the principle of the SAT-based SBST pattern generation. Section II explains how a sequential NoC switch is modeled in CNF. Section III focuses on representing CLFs in the SAT model. Section IV describes how the syntax of NoC packets is modeled. In section V, the flow of pattern generation by means of the SAT instance is explained. Finally, section VI demonstrates the efficiency of the SBST method through experimental results which is wrapped up by the conclusion in section VII.

## II. Circuit Modeling and Sequential Mapping

The combinational core of the switch is extracted by removing the flipflops of the circuit and replacing the input/output signals of the flipflops by pseudo primary output/input ports, respectively. The structure of the NoC switch is modeled by representing its combinational core in conjunctive normal form (CNF) with the Tseitin transformation [17]. We call the combinational switch model $\Phi_C$.

The sequential behavior of the switch is modeled by time-frame expansion which converts the time domain into the space domain (Fig. 2). As the switch is a sequential

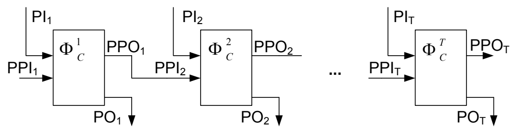

Fig. 2. Time-Frame Expansion for $\Phi_S$: $T$ copies of $\Phi_C$

circuit, some faults may firstly propagate to the internal states and after a few cycles become observable at the switch functional outputs. Hence, to achieve high fault coverage, the test patterns must define the functional inputs and the corresponding test responses for multiple consecutive cycles. The sequential switch $\Phi_S$ is modeled as formulated in Eq. (1) by instantiating multiple copies of the combinational switch instance, $\Phi_C$, one after the other such that the literals of the pseudo primary inputs of each copy are connected to the literals of the pseudo primary outputs of the previous copy in the SAT instance.

$$\Phi_S = \bigwedge_{t=1}^{T} \Phi_C^t \wedge \bigwedge_{t=2}^{T} \{PPI_t \leftarrow PPO_{t-1}\} \qquad (1)$$

In the above equation, $\Phi_C^t$ indicates to the copy $t^{\text{th}}$ of the combinational switch instances in $\Phi_S$.

## III. Modeling of Conditional Line-Flips

To develop a generalized SBST method targeting arbitrary defects in the switch and the interconnects, the Conditional Line Flip (CLF) model is used [18]. It formally describes defects as pairs consisting of a location or victim line $v$ and an activation condition. Whenever the condition evaluates to true, the value of the victim line is inverted. A CLF is noted by the name of the victim line $v$ and an xor-symbol followed by a condition clause as: $v^f := v \oplus [condition]$. In this notation, $v^f$ represents the value of the victim line. The condition is of arbitrary nature (Boolean, temporal, or even random) and is defined as an arbitrary function over time.

With respect to the targeted faults, appropriate values/functions can be used as a condition activating a fault only in some cycles. For example, a static bridge between two signal lines $a$ and $b$ is defined by the following generalized CLF formulation:

$$\begin{aligned} a^f &:= a \oplus [f_a(b) \cdot (a \oplus b)], \\ b^f &:= b \oplus [f_b(a) \cdot (a \oplus b)] \end{aligned}$$

in which $f_a$ and $f_b$ are two Boolean functions which determine the actual behavior of the bridge. There are exactly four basic expressions for each function: $f_a(b) \in \{0, 1, \bar{b}, b\}$ and $f_b(a) \in \{0, 1, \bar{a}, a\}$, resulting in 16 possible configurations for $f_a$ and $f_b$. As an example, for a 4-way bridge in which $a$ and $b$ swap the values, $f_a(b) = f_b(a) = 1$.

In order to generate test patterns for a CLF fault in the switch, two conditions must be satisfied:

- CLF is activated: The CLF condition is true in at least one time frame and leads to a different value in the good and faulty circuit.
- CLF is observed in at least one primary output.

Figure 3 depicts the SAT model used to represent a CLF for a single time-frame, $t$.

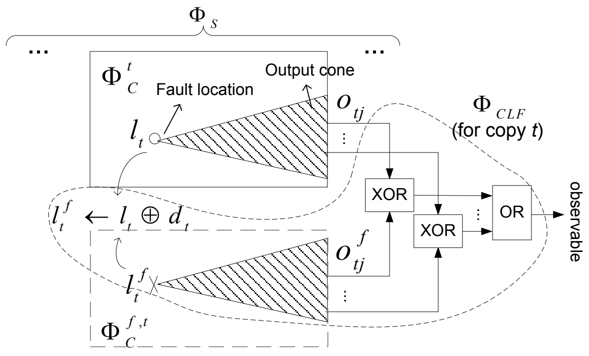

Fig. 3. Modeling a CLF in a single time-frame.

Starting from the model $\Phi_C^t$ representing the good circuit in a single time-frame $t$, a CLF $f$ is defined by the victim line $v$ represented by literal $l_t$ in the good circuit. The faulty circuit $\Phi_C^{f,t}$ is generated by copying the output cone of the fault location, where all internal literals are renamed, and the victim line literal is represented by $l_t^f$. All literals representing signals at the cone boundary are identical to the literals used in $\Phi_C^t$. If the fault propagates to the pseudo primary outputs, $\Phi_C^{f,t}$ includes additionally the gates in the output cone of the equivalent pseudo primary inputs as it is shown in Fig. 4. The cones may have some overlaps. In

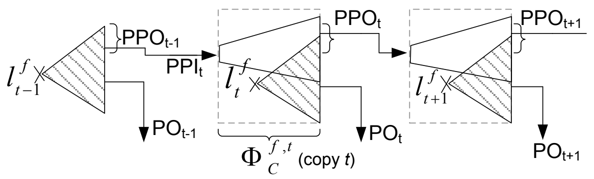

Fig. 4. Faulty instance

order to keep the SAT instance smaller, each gate is included only once in the model.

The condition of a CLF $f$ is represented by the literal $d_t$ that relates the good circuit and the faulty circuit as follows: $l_t^f \leftarrow (l_t \oplus d_t)$. In order to observe a fault effect at a primary output, all output literals $o_{tj}$ of the good circuit are compared to the output literals of the faulty circuit $o_{tj}^f$: $\vee_{j=1}^n (o_{tj} \neq o_{tj}^f)$. In this expression, $n$ is the number of primary outputs to which the fault is propagated. The expression is equivalent to the XOR gates followed by an OR gate in Fig. 3.

To represent a single CLF $f$ in all modeled time-frames $T$, faulty circuits are added for all time-frames and form the faulty instance $\Phi_S^f$:

$$\Phi_S^f = \bigwedge_{t=1}^{T} \Phi_C^{f,t} \tag{2}$$

$\Phi_S^f$ is linked to the good circuits via the condition literals as follows:

$$\Phi_{cond} = \bigwedge_{t=1}^{T} l_t^f \leftarrow (l_t \oplus d_t) \tag{3}$$

Since $d_t$ are free literals in the model, the SAT solver may assign values to them such that the fault is activated in any cycle independently. As it is sufficient to detect a CLF in a single time-frame, the output comparison is modeled as:

$$\Phi_{obs} = \bigvee_{t=1}^{T} \vee_{j=1}^{n} (o_{tj} \neq o_{tj}^f) \tag{4}$$

Therefore, the corresponding clauses for modeling a single CLF $f$ in all time-frames $T$ are denoted by $\Phi_{CLF}$ that is a conjunction of clauses of the faulty instance $\Phi_S^f$, the condition literals $\Phi_{cond}$, and a comparison over the outputs of the good and faulty copy as follows:

$$\Phi_{CLF} = \Phi_S^f \wedge \Phi_{cond} \wedge \Phi_{obs}\,. \tag{5}$$

## IV. Modeling of Valid NoC Packets

To ensure the test patterns are valid NoC packets, the appropriate clauses which define the packet characteristic must be included in the SAT model. Each NoC packet consists of several flits (flow control units). The flits of a packet arrive one after the other via the incoming ports of a switch as shown in Fig. 5. The first flit is called *head* flit and is followed by an arbitrary number of *data* flits. The last flit is called *tail* flit.

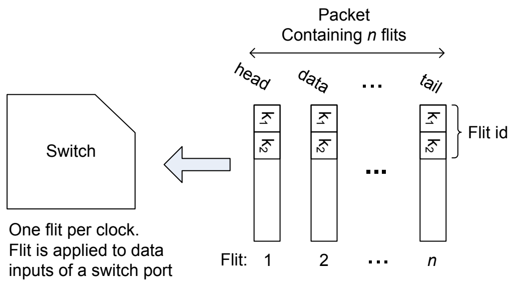

Fig. 5. NoC packet format

A switch has several ports (e.g. five ports in mesh architecture). In order to generate test patterns at port $i$ of the switch that represent valid packets, the input sequence must contain head and tail flits in a manner that every head flit is followed by a tail flit:

$$(PI^i_t = head) \rightarrow (PI^i_{t+fpp-1} = tail),\ 1 \leq t \leq T \quad (6)$$

where $fpp$ is the number of flits per packet, and $PI^i_t$ refers to the switch data inputs at cycle $t$ of the test pattern. Since head and tail flits are mandatory for a packet, $fpp$ is at least 2. However, for $fpp > 2$ the intermediate flits must be defined as data flits:

$$(PI^i_t = head) \rightarrow (PI^i_j = data) \quad (7)$$

for $t < j < (t + fpp - 1)$. According to the packet specification of the NoC, the flits can be identified by means of a few control bits (*Flit id* in Fig. 5), e.g. 01, 10, and 00 can be used as flit id for head, tail, and data flits. Equation (6) and (7) are mapped to clauses by Boolean logic minimization and added to the SAT model for all switch ports: $\Phi_{valid\ packet}$. They ensure that the SAT model is only satisfiable if the test patterns are in the form of NoC packets.

Valid NoC packets satisfy the control flow of the switch (similar to instructions in the SBST of processors). Adding valid packet clauses adjusts the SAT to find the solutions for the faults in the controlling parts. For example, a test pattern may contain several NoC packets which arrive at different switch ports at the same time, requesting the same outgoing port. This activates the scheduler of the switch to decide which packet has the priority. Moreover, as equation (6) does not impose any constraint for the time that the head flit arrives, the packet may arrive at any cycle between 1 and $T$. In this case, if the outgoing port has been already assigned to another packet, the new incoming packet must not overwrite this setting.

## V. Pattern Generation

As explained earlier, the SAT instance for generating SBST patterns for NoCs is constructed as a combination of the clauses of the unrolled switch $\Phi_S$, the clauses describing a CLF fault, and the clauses to ensure only valid packets are accepted as a solution:

$$\Phi_{SBST} \quad = \quad \Phi_S \wedge \Phi_{CLF} \wedge \Phi_{\substack{valid \\ packet}} . \quad (8)$$

To generate SBST patterns for all CLFs in the switch, as shown in Fig. 6 the SBST pattern generation process selects the first undetected CLF location from the fault list and builds $\Phi_{SBST}$ for the fault location as explained before. The process starts with a minimum $T$ of 3, two cycles for head and tail flit to arrive/depart and one cycle for the internal router process. If $\Phi_{SBST}$ is satisfiable, the values of the input literals are stored as the test pattern. Then, the test pattern is applied to the SAT instance as a constraint. The process searches for additional faults detected by this pattern and prunes them from the CLF fault list. The process continues until test patterns are found for all faults detectable under the current sequential depth. To increase the coverage, the faults that are not testable under the sequential depth $T$ are reprocessed by iteratively increasing the sequential depth to $T+1$ and repeating the SBST pattern generation process. This continues up to the maximum sequential depth of the circuit plus one, which is reported by a commercial ATPG tool.

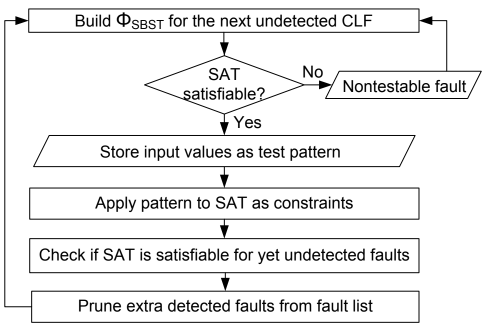
Build $\Phi_{SBST}$ for the next undetected CLF
SAT satisfiable?
No
Nontestable fault
Yes
Store input values as test pattern
Apply pattern to SAT as constraints
Check if SAT is satisfiable for yet undetected faults
Prune extra detected faults from fault list

Fig. 6. SBST pattern generation flow

## VI. Experimental Evaluation

The SBST pattern generation method is independent of the internal architecture of the switch and the topology. The number of input ports of the switch which are available for testing must be known. Besides, in order to introduce valid packet literals, the flit width and the flit format is required.

We evaluate the efficiency of the presented SBST method on a typical switch designed for an NoC mesh topology. The switch consists of five input and output ports, crossbar multiplexers, a router, and additional control logic for the handshake signals. It implements wormhole XY routing and processes the input channels in a round-robin fashion. The switch is implemented in VHDL and synthesized using Synopsys Design Compiler. The target library lsi10k is constrained to basic gate primitives. Since the memory elements are usually equipped with advanced memory BIST, there is no need to consider the channel buffers in the SBST pattern generation process. Table I summarizes the synthesis statistics of the switch for a flit width of 8 bits. The first four columns report the circuit name, the number of primary inputs (PIs), primary outputs (POs), and the number of flipflops (FFs), which is equal to the number of pseudo primary inputs/outputs. The remainder of the table reports the number of gates and the size in cell area units where the cell area of a two input NAND gate in the library is one area unit.

TABLE I. NoC Switch Characteristics

| Name | # PIs | # POs | # FFs | # gates | Cell area |
|---|---|---|---|---|---|
| Switch | 58 | 50 | 497 | 3618 | 8130 |

## A. Stuck-at Faults

For comparison, the fault coverage for stuck-at faults obtained by a scan based test strategy of a commercial ATPG tool was computed. There, 506 faults out of 14355 collapsed faults in the switch are recognized *Unused* by the tool, leading to the fault coverage of 96.47% and fault efficiency of 100%.

Table II shows that the fault coverage of the presented SBST is even higher (96.53%). However, these numbers cannot be directly compared as the number of faults in the scannable model and the unmodified switch do not match exactly. Hence, a commercial sequential ATPG tool without any constraints was applied as well. It reports a fault coverage of 83.29% and a fault efficiency of 86.33%, while 2.39% of the faults are *untestable*. All faults classified as untestable by the ATPG are functionally redundant.

Table II presents the complete process which starts with an initial sequential depth of 3. The second row in the table shows the number of faults which are testable for each sequential depth. The maximum sequential depth of the switch is 8, hence it is sufficient to examine 9 time-frames ($T = 9$) during the pattern generation process. The last column of the table sums up the detailed result and presents the SBST pattern generation statistics for stuck-at faults in the switch. The SBST process is able to generate test patterns for more than 95% of the faults for the initial sequential depth. However, for $T = 4$ and $T = 5$, test patterns are found for 122 additional faults. For larger values of $T$ no additional patterns are found. Among 10994 collapsed faults in the switch, 10613 faults are testable and for the rest, the SAT model proves that there exists no valid NoC packet such that the fault is propagated to the functional switch outputs. As the switch has been unrolled up to its maximum sequential depth, the faults which remain undetectable have no functional influence during the normal operation of the switch.

TABLE II. Pattern Generation for Stuck-at Faults (10994 Collapsed Faults)

| | $T = 3$ | $T = 4$ | $T = 5$ | total |
|---|---|---|---|---|
| Detected faults | 10491 | 121 | 1 | 10613 |
| Undetectable faults | 503 | 380 | 379 | 379 |
| Fault coverage | 95.42% | 96.52% | 96.53% | 96.53% |
| Test patterns | 504 | 32 | 1 | 537 |
| Test volume (bits) | 163296 | 13824 | 540 | 177660 |
| Test time (cycles) | 2016 | 160 | 6 | 2182 |

The last three rows of the table report some statistics of the generated test patterns. Firstly, the number of test patterns generated in each step of the SBST process is listed. Each test pattern includes the input and output values of the switch ports in $T$ consecutive cycles. For the test volume, we compute the size of the test set in bits. Because the test patterns of SBST contain the value of the primary inputs in $T$ cycles as the input pattern and the value of the primary outputs as the test responses, the test volume of the SBST test data is computed as follows:

$$\textit{Test volume}_{\textit{SBST}} = T \times (|\textit{PIs}| + |\textit{POs}|) \times n \tag{9}$$

in which $|\textit{PIs}|$ stands for the number of primary inputs, $|\textit{POs}|$ the number of primary outputs, and $n$ refers to the number of test patterns. The next row reports the test time. Because every SBST test pattern is generated with regard to $T$ cycles of the switch operation, the number of clock cycles required to apply the input patterns and get the responses is $T$. Moreover, one additional cycle is needed to reset the flipflop states before applying the next test pattern. Accordingly, the test time for applying $n$ test patterns is calculated as follows:

$$\textit{Test time}_{\textit{SBST}} = n \times (T + 1) \tag{10}$$

To shed some light on the test volume of the SBST method compared to the conventional test approaches similar to [2], we applied commercial ATPG with test compression targeting stuck-at faults in the switch and extracted the result of the test pattern generation. The number of test patterns is 135. Any scan test pattern consists of the value of both primary inputs and the scan registers as input pattern, and the value of primary outputs and the scan registers as test response. Thus, the test volume of the scan test data is computed as:

$$\textit{Test volume}_{\textit{scan}} = (2 \times |\textit{scan cells}| + |\textit{PIs}| + |\textit{POs}|) \times n \tag{11}$$

in which $|\textit{scan cells}|$ stands for the number of scan cells (i.e. flipflops). The test data volume of the scan test is 148770 bits and the test time varies depending on the number of scan chains in the switch. To achieve a test time comparable to the presented SBST test time, at least 30 scan chains are needed.

## B. Bridging and Crosstalk Faults

Table III presents the result of SBST targeting bridging faults in the switch and the links (4-way bridges), and crosstalk in the communication links. The pairs of suspected signals for the bridging fault are selected according to [19]. It shows that the SBST pattern generation process with the initial sequential depth 3 is able to achieve 98.88% fault coverage with 159 test patterns, resulting in a short test time. To further increase the fault coverage, the process can be repeated for larger $T$.

A crosstalk fault happens where a transition on an aggressor line $a$ causes glitches on a victim line $b$ [18]. In the CLF model, a crosstalk is represented according to the following formula, in which $a_{-1}$ denotes the last value of line $a$:

$$b^f := b \oplus [(a_{-1} \oplus a) \cdot (a \oplus b)] \tag{12}$$

To investigate the crosstalk effect on the communication links, pairs of neighboring signal lines on each input port of the switch are considered as aggressor and victim lines for the crosstalk model. The last value of the aggressor line comes from the equivalent literal in the previous copy of the unrolled switch. For the 8-bit switch, 14 crosstalk faults are injected in each input port. The last column of Table III illustrates the SBST pattern generation result which achieves 100% fault coverage at sequential depth 3. Thus, testing all crosstalk faults in the communication links of the switch requires 22 test patterns applied in 88 cycles.

TABLE III. SBST Pattern Generation Statistics

| | Bridging faults $T = 3$ | Crosstalk $T = 3$ |
|---|---|---|
| Faults | 10834 | 70 |
| Detected faults | 10713 | 70 |
| Undetectable faults | 121 (1.11%) | 0 |
| Fault coverage (%) | 98.88 | 100 |
| Test patterns | 159 | 22 |
| Test volume (bits) | 51516 | 7128 |
| Test time (cycles) | 636 | 88 |

The experiments reveal that SBST pattern generation modeled as a SAT problem is capable to generate test patterns targeting all kind of structural faults in the switches and the links. By only accessing the primary input and outputs of the switch, the test time decreases without imposing any hardware overhead. Besides, as the test patterns are in the form of NoC packet, the test process can be performed infield without putting the system in a non-functional test mode. This, in turn, eliminates the requirement to restore the system states after testing. Storage of the test patterns in the presented SBST method is not a problem, because only a small portion of memory space in the processing elements has to be dedicated to that. Since the SBST patterns target structural faults, the test responses can used to perform structural diagnosis [3].

## VII. Conclusion

A Software-Based Self-Test of switches in Networks-on-Chip was presented. It targets structural faults within NoC switches and NoC links. In order to conduct the test of a switch, the processing elements surrounding the switch under test are reused for test generation and evaluation as well as test access.

SBST pattern generation is mapped to a Boolean satisfiability problem, and only valid NoC packets are accepted as satisfying assignments of the model by additional formula.

The conducted experiments for stuck-at, bridging and crosstalk faults confirm the efficiency of software-based self-testing in the NoC domain with high fault coverage and very low test time. The technique does not impose any hardware overhead to the switch and is applicable for both manufacturing test and online test.

## VIII. Acknowledgment

Part of this work was supported by the German Research Foundation (DFG) under grant WU 245/12-1 (ROCK).

## References

[1] C. Grecu, P. Pande, B. Wang, A. Ivanov, and R. Saleh, “Methodologies and algorithms for testing switch-based NoC interconnects,” in *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, 2005, pp. 238–246.

[2] A. M. Amory, E. Brião, É. Cota, M. Lubaszewski, and F. G. Moraes, “A scalable test strategy for network-on-chip routers,” in *IEEE International Test Conference (ITC)*, 2005, pp. 1–9.

[3] A. Dalirsani, S. Holst, M. Elm, and H.-J. Wunderlich, “Structural Test and Diagnosis for Graceful Degradation of NoC Switches,” *Journal of Electronic Testing*, vol. 28, no. 6, pp. 831–841, 2012.

[4] P. Maxwell, I. Hartanto, and L. Bentz, “Comparing functional and structural tests,” in *IEEE International Test Conference (ITC)*, 2000, pp. 400–407.

[5] 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*, vol. 29, no. 6, pp. 883–896, 2010.

[6] A. Ghofrani, R. Parikh, S. Shamshiri, A. DeOrio, K.-T. Cheng, and V. Bertacco, “Comprehensive online defect diagnosis in on-chip networks,” in *IEEE VLSI Test Symposium (VTS)*, 2012, pp. 44–49.

[7] A. P. Frantz, F. L. Kastensmidt, L. Carro, and E. Cota, “Dependable network-on-chip router able to simultaneously tolerate soft errors and crosstalk,” in *IEEE International Test Conference (ITC)*, 2006, pp. 1–9.

[8] S.-Y. Lin, W.-C. Shen, C.-C. Hsu, C.-H. Chao, and A.-Y. Wu, “Fault-tolerant router with built-in self-test/self-diagnosis and fault-isolation circuits for 2d-mesh based chip multiprocessor systems,” in *International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, 2009, pp. 72–75.

[9] 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*, vol. 3, no. 5, pp. 476–486, 2009.

[10] N. Karimi, A. Alaghi, M. Sedghi, and Z. Navabi, “Online Network-on-Chip Switch Fault Detection and Diagnosis Using Functional Switch Faults,” *Journal of Universal Computer Science*, vol. 14, no. 22, pp. 3716–3736, 2008.

[11] R. Parikh and V. Bertacco, “uDIREC: unified diagnosis and reconfiguration for frugal bypass of NoC faults,” in *Proceedings of IEEE/ACM International Symposium on Microarchitecture*, 2013, pp. 148–159.

[12] L. Chen and S. Dey, “Software-based Self-testing Methodology for Processor Cores,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 3, pp. 369–380, 2001.

[13] A. Paschalis and D. Gizopoulos, “Effective Software-Based Self-Test Strategies for On-line Periodic Testing of Embedded Processors,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 1, pp. 88–99, 2005.

[14] F. Corno, G. Cumani, M. Sonza Reorda, and G. Squillero, “Fully automatic test program generation for microprocessor cores,” in *Design, Automation and Test in Europe (DATE)*, 2003, pp. 1006–1011.

[15] N. Kranitis, A. Paschalis, D. Gizopoulos, and G. Xenoulis, “Software-Based Self-Testing of Embedded Processors,” *IEEE Transactions on Computers*, vol. 54, no. 4, pp. 461–475, 2005.

[16] J. Zhou and H.-J. Wunderlich, “Software-based self-test of processors under power constraints,” in *Design, Automation and Test in Europe (DATE)*, 2006, pp. 430–435.

[17] G. S. Tseitin, “On the Complexity of Derivation in Propositional Calculus,” *Studies in constructive mathematics and mathematical logic*, vol. 2, no. 115-125, pp. 10–13, 1968.

[18] H.-J. Wunderlich, Ed., *Models in Hardware Testing*. Springer, 2010.

[19] E. N. Tran, V. Kasulasrinivas, and S. Chakravarty, “Silicon evaluation of logic proximity bridge patterns,” in *IEEE VLSI Test Symposium (VTS)*, 2006, pp. 1–6.