Exploiting Network-on-Chip Structural Redundancy for A Cooperative and Scalable Built-In Self-Test Architecture

A. Strano[†], C. Gómez[‡], D. Ludovici[†], M. Favalli[†], M.E. Gómez[‡], D. Bertozzi[†]
[†] ENDIF, University of Ferrara, 44100 Ferrara, Italy.
[‡] Dept. of Computer Engineering, Universidad Politecnica de Valencia, Spain.

Abstract—**This paper proposes a built-in self-test/self-diagnosis procedure at start-up of an on-chip network (NoC). Concurrent BIST operations are carried out after reset at each switch, thus resulting in scalable test application time with network size. The key principle consists of exploiting the inherent structural redundancy of the NoC architecture in a cooperative way, thus detecting faults in test pattern generators too. At-speed testing of stuck-at faults can be performed in less than 1200 cycles regardless of their size, with an hardware overhead of less than 11%.**

I. Introduction

On-chip interconnection networks are rapidly becoming the reference communication fabric for multi-core computing platforms both in high-performance processors and in many embedded systems [7], [3]. As the integration densities and the uncertainties in the manufacturing process keep increasing, complementing NoCs with efficient test mechanisms becomes a key requirement to cope with high defect rates [6], [11]. Above all, the NoC testing infrastructure should not be conceived in isolation, but should be coherently integrated into a reliability framework taking care of fault detection, diagnosis and network reconfiguration and recovery to preserve yield [9].

Moreover, wear-out mechanisms such as oxide breakdown, electro-migration and mechanical/thermal stress become more prominent in aggressively scaled technology nodes. These breakdown mechanisms occur over time, therefore the methodology and the infrastructure used for production testing should be designed for re-use during the system lifetime as well, thus enabling graceful degradation of the NoC over time.

The detection and identification of failures is the foundation of any reliability framework. Unfortunately, developing such a testing infrastructure for a NoC is a serious challenge. The controllability/observability of NoC links and sub-blocks is relatively reduced, due to the fact that they are deeply embedded and spread across the chip. Also, pin-count limitations restrict the use of I/O pins dedicated for the test of the different NoC components. A number of other concerns were raised in [10] on the use of external testers for nanoscale chip testing: lack of scalability of test data volumes, high cost for full clock speed testing, poor suitability for the extension of production testing to lifetime testing. As an effect, a migration from external testers to built-in self-test (BIST) infrastructures was envisioned in [10], and was later confirmed by the large amount of works in the open literature targeting scalable BIST architectures for NoC testing [2], [22], [17]. At the same time, the limited fault coverage that functional and pseudo-random testing can achieve on the control path of NoC switches when test generators are outside the switch has further pushed the adoption of BIST units at least for such control blocks [8].

In this direction, this paper relies on a full BIST strategy for NoC testing. A key principle of our approach consists of exploiting the inherent structural redundancy provided by NoCs. Each switch is comprised of input ports, output ports, arbiters and FIFOs that are duplicated for each channel. This feature is used to develop a very effective test strategy which consists of testing multiple identical blocks in parallel and of cutting down on the number of test pattern generators. This is done both at the abstraction level of the switch micro-architecture (e.g., testing of the output port arbiters in parallel) and of the NoC architecture (i.e., testing of all NoC switches in parallel). The inherent parallelism of our BIST procedure makes our testing infrastructure highly scalable and best suited for large network sizes.

Four main features differentiate our testing framework from most previous work. First, we take on the challenge of generating deterministic test vectors on-chip at a limited area overhead. At the same time, this enables us to report much shorter test application times than typical pseudo-random testing frameworks and larger fault coverage in the control path than most functional testing frameworks for NoCs. Second, we account for the tedious problem of faults affecting test pattern generators (TPGs) and provide large coverage for them. This is done without implementing more hardware redundancy but fully exploiting the existing one by means of a cooperative testing framework among switches. Third, our testing framework targets double and triple stuck-at faults from the ground up, and not as an afterthought, in addition to an almost 100% coverage of single stuck-at faults. Fourth, our framework is not limited to regular 2D meshes, but can be applied to a much wider range of network topologies.

Our BIST procedure is suitable both for production and for lifetime testing, and is complemented by a built-in self-diagnosis logic distributed throughout the network architecture able to pinpoint the location of detected faults in each switch. This diagnosis outcome matches the reconfigurability requirements of logic-based distributed routing and is therefore the stepping stone into a novel network reconfiguration strategy that will be developed in future work.

II. Previous Work

Considering the regular and modular structure of on-chip networks, test strategies previously proposed for systems with identical cores [14], [23] can be applied to the NoC. However, both approaches incur a significant overhead for DfT structures (full-scan and IEEE 1500 wrapped cores with registered I/O pins).

It is showed in [13] that traditional full-scan and boundary scan strategies like [18], [21], [15], [17] incur an hardly affordable area overhead. [13] also proposes a partial scan technique in combination with an IEEE 1500-compliant test wrapper. Area overhead is greatly reduced, but test application times amount to tens of thousands of clock cycles and test pattern generation time does not scale.

As opposed to using scan paths and wrappers for test access, [4] considers the case where test patterns are applied at the border I/Os of the network. The method was then extended in [5] to support fault diagnosis, while the DfT infrastructure was developed in [8]. While very high fault coverage was achieved, the time complexity of the test configurations is square with respect to the rank of the NoC matrix. Moreover, in order to apply test patterns from network boundaries at-speed, a large number of test pins are necessary.

In [19], it is proposed to add dedicated logic to enable analysis of response from each FIFO in the switch, however no test data is presented. In [16] the possibility to repair the NoC during testing is envisioned, however error information is

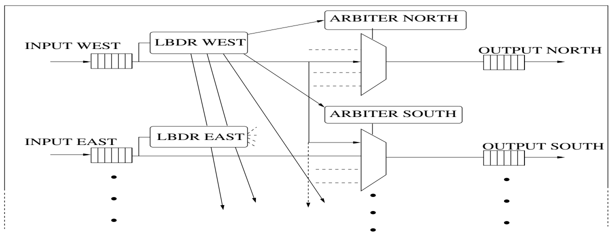


Fig. 1. **Modular structure of the baseline switch architecture. Not all connections are showed.**

computed once for all and thus cannot handle situations where the chip slowly degrades.

[20] proposes a built-in self-test and self-diagnosis architecture to detect and locate faults in the FIFOs and in the MUXes of the switches. Unfortunately, the control path is left out of the framework.

In [2] an automatic go/no-go BIST operation is proposed at start up of a 2D mesh NoC. Low fault coverage is achieved for the switch controller, moreover the methodology applies only to a 2D mesh. That idea is evolved in [12], where a fault coverage close to 100% is documented with a few thousand clock cycles. However, the area cost of the BIST architecture is the main concern of this work.

The pattern based testing section from the more general reliability framework presented in [9] reports a testing methodology relying on random test pattern generation and signature analysis. Unfortunately, testing takes as large as 200000 cycles with 10000 patterns per test.

With respect to previous work, we claim a more efficient use of NoC structural redundancy for testing and diagnosis purposes through the use of a cooperative testing framework. With respect to scan-based approaches, we reduce area overhead while at the same time detecting TPG faults. With respect to functional testing solutions, we provide efficient testing of the control path as well and provide better test time scalability. With respect to pseudo-random testing, we cut down on the test application time. We also take on the challenge pointed by [2] of exploiting architecture behavior knowledge to come up with a set of customized test patterns for NoC components.

III. Target Architecture

Without lack of generality, we use the xpipesLite switch architecture [1] to prove viability of our testing methodology in a realistic NoC setting. The baseline switch architecture is illustrated in Fig.1. It implements both input and output buffering and relies on wormhole switching. The crossing latency is 1 cycle in the link and 1 cycle in the switch itself. Flit width assumed in this paper is 32 bits, but can be easily varied. Without lack of generality, in this paper the size of the output buffers is 6 flits, while it is 2 flits for the input buffers.

This switch relies on a stall/go flow control protocol. It requires two control wires: one going forward and flagging data availability ("valid") and one going backward and signaling either a condition of buffer filled ("stall") or of buffer free ("go").

The switch architecture is extremely modular and exposes a large structural redundancy, i.e., a port-arbiter, a crossbar multiplexer and an output buffer are instantiated for each output port, while a routing module is cascaded to the buffer stage of each input port. This common feature to all switch architectures will be intensively exploited in this work.

We implement distributed routing by means of a route selection logic located at each input port. Forwarding tables are usually adopted for this purpose, although they feature poor area and delay scalability with network size [24]. The possibility to implement logic-based distributed routing (LBDR) while retaining the flexibility of forwarding tables has been recently demonstrated in [26]. In practice, LBDR consists of a selection logic of the target switch output port relying on a few switch-specific configuration bits (namely routing R_{xy}, connectivity C_z and deroute bits dr_t). The number of these bits (14 in this case) is orders of magnitude less than the size of a forwarding table, yet makes the routing mechanism reconfigurable.

The core of LBDR logic is illustrated in Fig.2(a), illustrating the conditions that select the output port north UN' for routing. The pre-processed direction of packet destination $N'/S'/W'/E'$ is an input together with the routing and the connectivity bits. In some cases (see [26] for details), deroutes are needed to properly route packets, and the associated logic is reported in Fig.2(b).

LBDR supports the most widely used algorithms for irregular topologies and can be used on a 2D mesh as well as on roughly 60% of the irregular topologies derived from a 2D mesh, like in Fig.2(c). Its extension to fully irregular topologies with 12 more bits per switch is an ongoing work [25]. Irregularity of the connectivity pattern can be an effect of manufacturing or wearout faults, but also of power management or thermal control decisions or of virtualization strategies. *Switch configuration bits need to be updated whenever the topology evolves from one connectivity pattern to another* (e.g., when a fault is detected).

Our testing and diagnosis framework has been conceived to enable a network reconfiguration strategy leveraging the cost-effective flexibility offered by the LBDR routing mechanism. An algorithm is reported in [26] for computation of the switch configuration bits given the topology connectivity pattern. As an example, updated connectivity bits are illustrated in Fig.2(c). This algorithm might be executed by a centralized NoC manager and in practice *needs the list of failed links to recompute the configuration bits for correct routing with the available communication resources*. Failure of a switch input or output port can be viewed as the failure of the connected link. Our diagnosis strategy will therefore target this requirement and *will provide an indication of whether input and output ports of a switch are operational.*

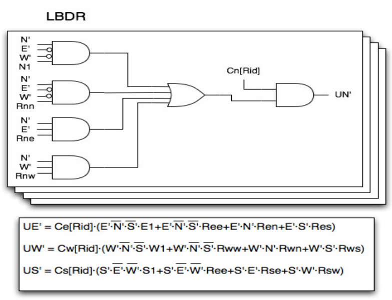
IV. Built-In Self-Test/Diagnosis Framework

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

Each switch can in turn **test its manyfold internal instances of the same sub-blocks (crossbar muxes, communication channels, port arbiters, routing modules) concurrently**. In fact, all the instances are assumed to be identical, therefore they should output the same results if there is no fault. As a consequence, the test responses from these instances are fed to a comparator tree. This makes the successive diagnosis much easier. **There is a unique test pattern generator (TPG) for all the instances** of the same block, thus cutting down on the number of TPGs. Although the principle is similar to what has been proposed in [14], [22], [13], there is a fundamental difference. If the TPG of a set of block instances is affected by a fault, then the comparison logic will not be able to capture this since all instances provide the same wrong response. To avoid this, **a cooperative framework is devised, such that each switch tests the block instances of its neighboring switches.**

As an example, a switch tests the incoming communication channels from its north/south/west/east neighbors (i.e., it feeds their test responses to its local comparator tree), thus checking the responses to distinct instances of the same TPG. This way, a non-null coverage of TPG faults becomes feasible. Fig.3(a) clearly illustrates the cooperative testing framework for communication channels and the need for a single TPG instance per switch to feed test patterns to all of its output ports. Faults in the TPG, in the output buffer, in the link and in the input buffer will be revealed in the downstream switch. Each switch ends up testing its input links, while its output links will be tested by their respective downstream switches.

The same principle can be applied for the testing of switch internal block instances **associated with each output port**:

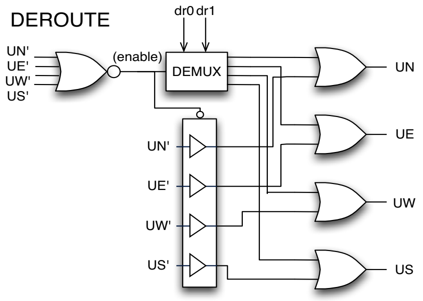


$$UE' = Ce[Rid]\cdot(E'\cdot\overline{N'}\cdot\overline{S'}\cdot E1+E'\cdot\overline{N'}\cdot\overline{S'}\cdot Ree+E'\cdot N'\cdot Ren+E'\cdot S'\cdot Res)$$

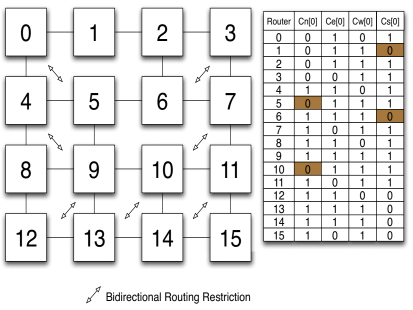
$$UW' = Cw[Rid]\cdot(W'\cdot\overline{N'}\cdot\overline{S'}\cdot W1+W'\cdot\overline{N'}\cdot\overline{S'}\cdot Rww+W'\cdot N'\cdot Rwn+W'\cdot S'\cdot Rws)$$

$$US' = Cs[Rid]\cdot(S'\cdot\overline{E'}\cdot\overline{W'}\cdot S1+S'\cdot\overline{E'}\cdot\overline{W'}\cdot Ree+S'\cdot E'\cdot Rse+S'\cdot W'\cdot Rsw)$$

(a) Core logic for route computation



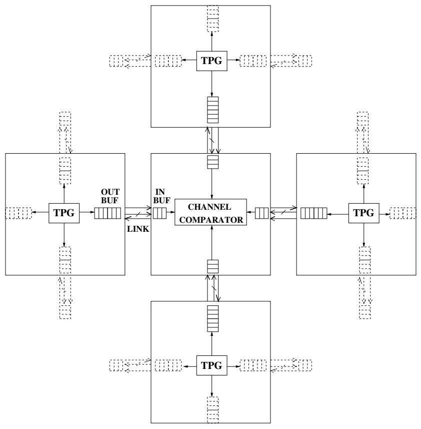
(b) Deroute logic



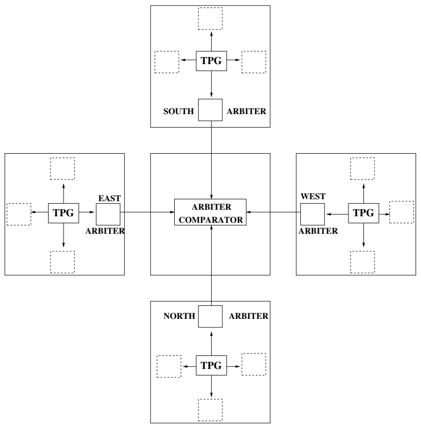
Router	Cn[0]	Ce[0]	Cw[0]	Cs[0]
0	0	1	0	1
1	0	1	1	0
2	0	1	1	1
3	0	0	1	1
4	1	1	0	1
5	0	1	1	1
6	1	1	1	0
7	1	0	1	1
8	1	1	0	1
9	1	1	1	1
10	0	1	1	1
11	1	0	1	1
12	1	1	0	0
13	1	1	1	0
14	1	1	1	0
15	1	0	1	0

(c) Connectivity bit setting because of failed links.

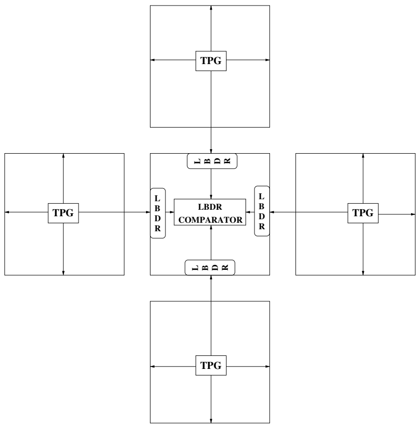
Fig. 2. **LBDR logic and requirements on the diagnosis outcome.**



(a) Testing communication channels.



(b) Testing output port arbiters.



(c) Testing LBDR routing logic.

Fig. 3. **The cooperative and concurrent testing framework saving TPG instances and covering their faults.**

crossbar muxes and output port arbiters. Fig.3(b) shows the case of port arbiters. The main requirement for testing these instances is that the communication channels bringing test responses to the comparators in the downstream switches are working correctly. Clearly, testing these modules can only occur after communication channels have been tested. Therefore, the procedures in Fig.3(a) and Fig.3(b) occur sequentially in time. Should one communication channel result defective, this would not be a problem, since it would not make any sense to test and use a port arbiter when the corresponding port is not operational. Crossbar multiplexers associated with each output port are tested in the same way and are hereafter not illustrated in Fig.3 for lack of space.

Finally, the methodology can be extended to test block instances associated with each switch input port with some modifications. This is the case of the LBDR routing block. The key idea to preserve the benefits of cooperative and concurrent testing is to carry test patterns rather than test responses over the communication channels to neighboring switches, where the LBDR instances are stimulated and their responses compared (see Fig.3(c)). If the channel is not working properly, than testing and use of the downstream routing block is useless, since it is associated with an input port which will not be used.

A BIST engine is embedded into each switch and regulates the testing procedure. This latter is in fact split into four phases in time:
- testing of communication channels
- testing of the crossbar
- testing of the arbiters
- testing of the LBDR routing blocks
The serial execution of test phases for the switch internal components is dictated primarily by the limited flit width, constraining the amount of test patterns that can be transmitted at the same time over the communication channel, and also by the limited availability of comparators, although in our case the former effect comes into play first. As the flit width increases, then we can perform more testing operations in parallel, starting from those components that have a limited amount of primary input/outputs (e.g., the arbiter with the LBDR).

A fundamental difference with respect to a lot of previous work is that we do not rely on pseudo-random testing (like in [9]), which gives rise to large testing times. We use deterministic test patterns instead, which are handcrafted for the specific block under test by exploiting knowledge of the architecture behavior. This way, the reduced number of test patterns enables the serialization of test phases without making test application time skyrocket (see section V-A).

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

In the final implementation, other 5 bits will be needed to code the diagnosis outcome because of practical implementation issues, as discussed in section IV-A.

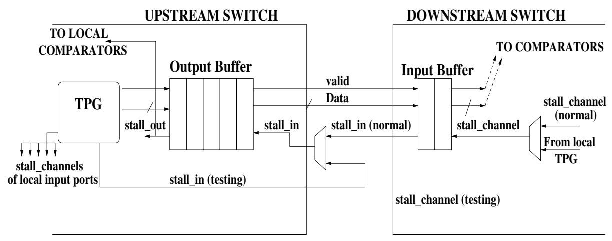


Fig. 4. **Practical implementation of communication channel testing.**

Common to most current NoC testing frameworks, the underlying assumption for correct operation of our BIST/BISD infrastructure is that the reset signal can be synchronously deasserted in all switches of the network at the same time.

A. Testing communication channels

Communication channels include input/output buffers and their intermediate links, as illustrated in Fig.4: all these elements are jointly tested by means of a single TPG and the test patterns are handcrafted for them based on knowledge of their behavior.

Our approach in this direction was to expand the finite state machine (FSM) of the device under test (DUT) into all its possible states. Therefore, we have defined a sequential test pattern that drives the FSM to each of its states. In this way, we can ensure that if the FSM reaches the expected state for all the test patterns there are no faults inside the DUT. As an example, the FSM of the buffers defines that if the Stall signal is asserted and the buffer receives a set of valid flits, the buffer has to store the flits that it receives until it becomes full. One test pattern to check this behavior would fill up the buffer by asserting the Stall signal, and would in the end check whether the output buffer correctly asserts the Full signal. The datapath is obviously much easier to test by means of only few test patterns.

From an implementation viewpoint, there are several practical issues. On one hand, we had to make the stall input of the output buffer directly controllable to the TPG to raise its stuck-at fault coverage to almost 100% (see Fig.4).

On the other hand, the *stall_channel* signal of the input buffer, which lies in the downstream switch, should be driven by the TPG as well. This would require an additional wire in the switch-to-switch link. A similar concern is that the *stall_out* signal from the output buffer should be brought to the comparators in the downstream switch, again requiring an additional wire in the link.

To avoid the extra wires, we opted for the solution in Fig.4: *stall_channel* is driven by the TPG of the downstream switch, while *stall_out* is brought to the comparator tree in the upstream switch. From the testing viewpoint nothing changes, since all channel TPGs inject the same patterns synchronously, and so do the comparators. The only difference lies in the fault coverage of TPG faults, which is likely to be decreased a bit. In fact, those (upstream) TPG faults that can be detected by only monitoring *stall_out* will not be detected, since all the *stall_out* signals brought to the local comparators will be driven by the same TPG. Similarly, some faults in the (downstream) TPG will not be detected, since the comparators compare responses to *stall_channel* signals generated by the same faulty TPG: the responses will look like the same. These implementation variants, needed to adapt the conceptual testing scheme to the constraints of the real implementation, will be proven in section V-A to only marginally decrease fault coverage of the TPGs, while leaving fault coverage for the communication channel obviously unaffected.

The only major implication is that the fault detection framework becomes even more collaborative: some (very few) faults in the channel and/or TPGs are now detected in the upstream switch comparators instead of the downstream ones. Therefore, other 5 additional diagnosis bits are needed, flagging a fault in the output port of a switch. The global controller will combine this (OR operation) with the faults detected at the input port of the downstream switch to get the complete indication of a fault across the entire channel.

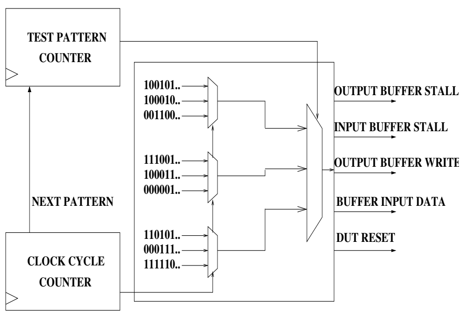


Fig. 5. **TPG for communication channels.**

B. TPG for communication channels

A test pattern can be easily generated in hardware by using a clock cycle counter and some logic to generate the values of the input signals for the DUT. In order to extend this approach to a TPG able to generate all the test patterns for a given DUT, we can include an additional counter. This latter will indicate the current test pattern within the test sequence. Figure 5 depicts the resulting conceptual scheme for the channel TPG. The actual gate-level implementation depends on the logic synthesis tool and on the synthesis constraints. The two counters act as a FSM driving the control signals of two levels of multiplexing: the first one selects the current test pattern, while the second one selects the current clock cycle and associated input vector for the buffer.

It is however possible to easily compact the combinational logic, because there are a lot of test patterns that include other test patterns. For instance, by checking the response not only at the end of the test pattern, but also somewhere in the middle, it is often possible to detect another fault. This perfectly matches with the capability of our BIST framework, which even performs check response at each clock cycle. Therefore, it is possible in our implementation to perform a compaction of test patterns by generating in hardware only those patterns including a subset of the other ones, thus largely saving test time and TPG area.

C. Testing Other Internal Switch Modules

A similar process is followed to generate deterministic test patterns for the port arbiters, the LBDR modules and the crossbar. Also the implementation of their TPGs is identical, and so are the optimization techniques.

Again, the most relevant practical implementation issue concerns the communication of test patterns or responses across the switch-to-switch links for the crossbar and LBDR module. The crossbar outputs 34 bits in response to a test vector: 32 data bits, 1 valid bit and 1 stall bit. The communication channel can only carry 32 bits (the valid bit of the channel needs to be permanently set to 1 during test vector transmission, while the stall signal travels in the opposite direction). The two remaining crossbar signals (valid and stall) which do not fit into the link can be either transmitted by means of additional lines used only during testing, or alternatively checked by local comparators, similarly to what has been done for the communication channel. We took the latter approach, and the results in section V-A again confirm the marginal coverage reduction on TPG faults. Fault coverage of the crossbar is not affected at all by this choice.

Unlike other modules, test vectors for the LBDR modules should be transmitted across the link, and they take 31 lines (the primary inputs of the LBDR module). So, they perfectly match with the current flit width, provided the number of network destinations does not exceed 64. From there on, the test vector width starts growing logarithmically with the number of destinations, and additional lines may be required on the link.

In contrast, the use of a larger flit width in the network (e.g., 64 bits) would automatically solve the problem. In that case, the test patterns of the LBDR block and the test responses of the arbiter could even be communicated at the same time over the link. Also, since LBDR module and arbiters have only few outputs, their response checking could be performed at the same time on the available tree of comparators, thus cutting down on the test application time (see section V-A).

D. Fault detection and diagnosis

The core of the diagnosis unit is given by comparators which can be implemented in two different ways, by:
- using a level of XORs and an OR gate to provide a single output encoding of the equality test;
- using a two-rail checker TRC (with the second word which is negated);

We opted for the TRC approach, which achieves the self-testing and fault-secure properties [27] although leading to a more complex circuit.

In the diagnosis unit we use 10 different comparators to compare data from all the possible pairs of switch input ports. A smaller number of comparators could be used. Unless time multiplexing is exploited, this would trade cost for diagnosis capability. The maximum number of usable comparators also depends on the number of switch I/O ports. In what follows, we will focus on the internal switches of a 2D mesh for the sake of simplicity (featuring 5 I/O ports, including the local connection to the network interface), however *all irregular topologies supported by LBDR and making use of switches with at least 3 I/O ports* are suitable for our methodology. Obviously, the lower the number of ports, the lower the diagnosis capability.

If we denote two faults in different ports under comparison as *equivalent* if they produce the same output sequence in response to the same input stimuli, then our comparator and diagnosis logic is able to:
- diagnose the correct position of 1 or 2 faulty channels affected by equivalent or non-equivalent faults;
- diagnose the correct position of 3 faulty channels affected by non-equivalent faults; [1]
- detect the presence of 4 and 5 faulty channels. Anyway, since a 5x5 switch affected by 4 or 5 faults has to be discarded, we don't distinguish between these two scenarios.

One might argue that when a communication channel fails, then the following testing phases have less inputs available and diagnosis capability reduces. In practice, this effect plays only a minor role, since a fault on a communication channel means that also (say) the arbiter of that channel should be considered faulty (unusable). So, the diagnosis capability reduces, but also the number of input ports to be checked reduces as well.

When a switch features only three I/O ports, then the detection and diagnosis capabilities change as follows. Single stuck-at faults can be diagnosed while double faults can be detected, provided they are not equivalent. If they are equivalent, then diagnosis fails. However, when two faults are detected in two ports out of three, the switch should be discarded anyway.

As regards the possible presence of faulty comparators, let us first note that any input vector producing less than four ones corresponds to faults in less than four comparators (we are neglecting the case where all 5 channels are faulty and 4 of them have equivalent faults, which is very unlikely). In case the number of faulty comparators is larger than 3, some configuration exists which may produce a wrong diagnosis. Let us note, however, that it is sufficient to have a single test vector (not a test sequence) featuring less than four ones to immediately recognize the presence of faulty comparators because no combination of faulty channels may produce such response.

[1]The probability that more than two faulty channels produce the same output sequence in response to the same input stimuli is here neglected.

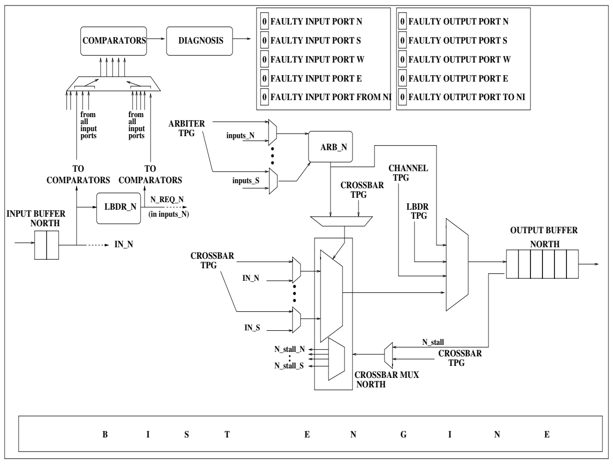


Fig. 6. **BIST-enhanced switch architecture.**

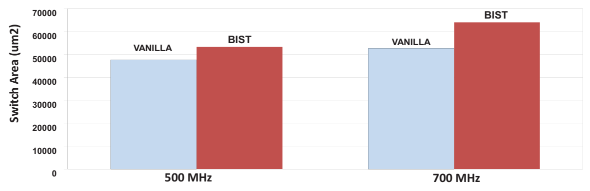


Fig. 7. **Area overhead for BIST implementation as a function of target speed.**

E. BIST-enhanced switch architecture

The switch architecture enriched with the BIST infrastructure is illustrated in Fig.6. Only one section is reported. The figure is necessarily at a high abstraction level, and signal-level connection details previously illustrated in sections IV-A and IV-C are purposely omitted.

A test wrapper consisting of multiplexers can be clearly seen, which enables test pattern injection of TPGs in the modules they test. At the output of the input buffer, test patterns are directly fed to the LBDR module, since they are carried by the communication channel as normal network traffic. A multiplexer in front of each output buffer selects between the switch datapath, the test patterns from the LBDR TPG (feeding the LBDR module of the downstream switch), the channel TPG (directly feeding the channel) and the arbiter test responses (checked in the downstream switch). A BIST engine drives the 4 phases of the testing procedure by acting upon the control signals of the test wrapper.

During the first three phases (communication channel, crossbar, arbiter testing), outputs of the input buffers are selected to feed the comparator tree, while in the last phase (LBDR testing), all LBDR outputs are selected. Test response check and diagnosis are performed at each clock cycle, and result in the setting of 10 bits, indicating whether each input/output port is faulty or not.

V. Experimental results

We performed placement-aware logic synthesis and place&route of a 5x5 switch on an industrial 65nm technology library. The baseline switch architecture of Fig.1 is compared with its BIST-enhanced counterpart. Synthesizing for maximum performance gives approximately the same maximum (post-layout) operating speed of 700 MHz for both architectures, thus proving that our BIST-enabled switch is capable of at-speed testing.

Fig.7 shows the area overhead for BIST implementation as a function of the target speed. Area overhead is 11%, which peaks at 21% when maximum performance is required. In this latter case, the multiplexers on the critical path are primary targets for delay optimization in exchange for more area.

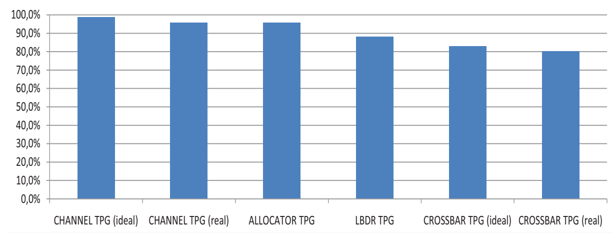


Fig. 8. **Coverage of TPG faults.**

Switch sub-block	Test patterns	Test vectors	Coverage
Comm. channel	58	464	99.4%
Arbiter	82	328	97.1%
Crossbar	72	72	99.8%
LBDR	240	240	98.7%

TABLE I
Coverage for single stuck-at faults.

When considering the BIST infrastructure in isolation (at 700 MHz) most of the overhead comes from the on-chip generation of test patterns (almost 31%) and from the multiplexers (44%) of the test wrapper. Interestingly, although arbiters and LBDR require less test vectors than the communication channel, their TPGs are far more complex due to higher irregularity of their test patterns.

A. Fault Coverage

Tab.1 reports the total number of deterministic test patterns (and test vectors) generated for each tested module, and the associated coverage. This latter was derived by means of an in-house made gate-level fault simulation framework: (one or more) faults are applied to any or selected gate inputs, then our testing procedure is run on the affected netlist and the diagnosis outcome is compared with the expected one.

It can be seen that in all cases the coverage for single stuck-at faults closely tracks 100%. The number of test vectors provides the test application time (in clock cycles). A network with a flit width of 32 bits, as assumed so far, would therefore take 1104 clock cycles for testing, regardless of the network size. If we assume 64 bit flits, then LBDR testing occurs in parallel with arbiter testing and total test time reduces to 864 cycles.

These numbers compare favorably with previous work, as Tab.2 shows. Only [20] and [8] in some cases do better. However, [20] does not test the control path while [8] reports 320 cycles for a 3x3 mesh (made of a simplified switch architecture) which however grow linearly with network size. Also, this latter approach makes additional use of BIST logic for the control path not accounted for in the statistics.

We feel that area overhead is hardly comparable with previous work since whenever numbers are available, features of the testing frameworks are very different (e.g., control path not tested [20], test patterns generated externally [21], [13], diagnosis missing [21], [12], [13], [22], lack of similar test time scalability [4], [8], NoC architecture with overly costly links [12]). Moreover, the impact of synthesis constraints is never discussed.

	Test Cycle	Coverage
Our	864 - 1104	99.3%
[20]	3.88 x 10^2 - 2.89 x 10^3	97.79%
[21]	4.05 x 10^5	95.20%
[12]	2.74 x 10^3	99.89%
[13]	9.45 x 10^3 - 3.33 x 10^4	98.93%
[22]	5 x 10^4 - 1.24 x 10^8	N.A.
[8]	320	99.33%
[9]	200 x 10^3	full (no exact numbers)

TABLE II
Test application time and coverage of different testing methods.

Multiplicity of Fault Injection	2	3	4	5
Coverage	99.2%	96.4%	96.6%	96.6%

TABLE III
Coverage for multiple random stuck-at faults.

Fig.8 reports the coverage of TPG faults. While single stuck-at faults in the allocator and channel TPGs feature a coverage of roughly 95%, worse results are obtained for the LBDR and especially for the crossbar TPGs. We verified that their lower coverage is a direct consequence of the low number of test patterns they generate. The designer can then choose whether increasing crossbar TPG area and having it generate more patterns or dedicating a separate test phase to TPGs. Also, when comparing real vs ideal coverage of channel and crossbar TPGs, it is possible to assess the marginal reduction of TPG fault coverage as an effect of the local (instead of remote) check of some signals of these modules in the switch they belong to (see section IV-A and IV-C).

Since our BIST infrastructure targets multiple stuck-at faults from the ground up, we characterized fault coverage for multiple faults as well. We have injected multiple faults randomly in the gate-level netlist of the switch and checked the diagnosis response. Fault multiplicity was 2,3, 4 and 5 and fault injections for a given multiplicity were repeated 1000 times, as in [9]. As Tab.3 shows, the proposed BIST framework provides a higher than 96% coverage in every scenario. Interestingly, the coverage saturates with 4 and 5 faults since the probability to inject errors in a module already affected by an error becomes high.

VI. Conclusions

This work develops a scalable built-in self-test and self-diagnosis infrastructure for NoCs taking full advantage of their structural redundancy through a cooperative testing and diagnosis framework. Table-less logic based distributed routing is the foundation of our approach, and enables network reconfiguration with only 10 diagnosis bits per switch. We prove the achievement of standard fault coverage targets at an affordable area overhead. However, we do more than that: we quantify coverage for multiple faults and aim at the coverage of faults affecting TPGs as well. This latter is a key step forward to make the move from scan-based approaches to scalable BIST approaches viable.

Acknowledgements

This work has been partially supported by the NANOC European Project (FP7-ICT-248972).

References

[1] S.Stergiou et al., "Xpipes Lite: a Synthesis Oriented Design Library for Networks on Chips", DAC, pp.559-564, 2005.
[2] K.Petersen, J.Oberg, "Utilizing NoC Switches as BIST-Structures in 2D Mesh Network-on-Chip", Future Interconnects and Network on Chip Workshop, 2006.
[3] D.Wentzlaff et al., "On-Chip Interconnection Architecture of the Tile Processor", IEEE Micro, vol.27, no.5, pp.15-31, 2007.
[4] J.Raik, V.Govind, R.Ubar, "An External Test Approach for Network-on-a-Chip Switches", Proc. of the IEEE Asian Test Symposium 2006, pp.437-442, Nov. 2006.
[5] J.Raik, V.Govind, R.Ubar, "Test Configurations for Diagnosing Faulty Links in NoC Switches", Proc. ETS, 2007.
[6] M. Mishra and S. Goldstein, "Defect tolerance at the end of the roadmap", in ITC, pages 1201-1211, 2003.
[7] D. A. Ilitzky, J. D. Hoffman, A. Chun and B. P. Esparza, "Architecture of the Scalable Communications Core's Network on Chip", IEEE MICRO, 2007, pp. 62-74.
[8] J.Raik, V.Govind, R.Ubar, "DfT-based External Test and Diagnosis of Mesh-like NoCs", IET Computers and Digital Techniques, October 2009.
[9] V.Bertacco, D.Fick, A.DeOrio, J.Hu, D.Blaauw, D.Sylvester, "VICIS: A Reliable Network for Unreliable Silicon", DAC 2009, pp.812-817.
[10] Y.Zorian, "Testing the monster chip", IEEE Spectrum, pp.54-60,1999.
[11] Y.Zorian, "Embedded Memory Test and Repair: Infrastructure IP for SoC Yield.", International Test Conference, pp.340-349,2002.
[12] K.Peterson, J.Oberg, "Toward a Scalable Test Methodology for 2D-mesh Network-on-Chip", DATE 2007, pp.75-80, 2007.
[13] A.M. Amory, E.Briao, E.Cota, M.Lubaszewski, F.G.Moraes, "A Scalable Test Strategy for Network-on-Chip Routers", Proc. of ITC 2005.
[14] K.Arabi, "Logic BIST and Scan Test Techniques for Multiple Identical Blocks", IEEE VLSI Test Symnposium, pp.60-68, 2002.
[15] C.Grecu, P.Pande, B.Wang, A.Ivanov, R.Saleh, "Methodologies and Algorithms for Testing Switch-Based NoC Interconnects", IEEE DFT 2005, pp.238-246, 2005.
[16] B.Vermeulen, J.Delissen, K.Goossens, "Bringing Communication Networks on a Chip: Test and Verification Implications", IEEE Communications Magazine, vol.41-9, pp.74-81, 2003.
[17] R.Ubar, J.Raik, "Testing Strategies for Network on Chip", in Book: "Network on Chip", edited by A.Jantsch and H.Tenhunen, Kluwer Academic Publisher, pp.131-152, 2003.
[18] C.Aktouf, "A Complete Strategy for Testing an on-Chip Multiprocessor Architecture", IEEE Design and Test of Computers, vol.19-1, pp.18-28, 2002.
[19] Panda et al., "Design, Synthesis and Test of Networks on Chips", IEEE Design and Test of Computers, vol.22, issue 8, pp.404-413, 2005.
[20] S.Y.Lin, C.C.Hsu, A.Y.Wu, "A Scalable Built-In Self-Test/Self-Diagnosis Architecture for 2D-mesh Based Chip Multiprocessor Systems", IEEE Int. Symp. on Circuits and Systems, pp.2317 - 2320, 2009
[21] M.Hosseinabady, A.Banaiyan, M.N.Bojnordi, Z.Navabi, "A Concurrent Testing Method for NoC Switches", DATE, pp.1171 - 1176, 2006
[22] C.Grecu, P.Pande, A.Ivanov, R.Saleh, "BIST for Network-on-Chip Interconnect Infrastructures", VLSI Test Symposium, page 6, 2006.
[23] Wu, Y. and MacDonald, P., "Testing ASICs with Multiple Identical Cores", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 22-3, 2003, pp. 327-336.
[24] S.Rodrigo, S.Medardoni, J.Flich, D.Bertozzi, J.Duato, "Efficient Implementation of Distributed Routing Algorithms for NoCs", IET-CDT, pp.460-475, vol.3, issue 5, 2009.
[25] Submitted paper under review
[26] S.Rodrigo, J.Flich, A.Roca, S.Medardoni, D.Bertozzi, J.Camacho, F.Silla, J.Duato, "Addressing Manufacturing Challenges with Cost-Effective Fault Tolerant Routing", NOCS 2010, pp.35-32, 2010.
[27] P.K. Lala, "Self-checking and fault tolerant Digital Design", MK Publishers 2001.