# A New Technique to Generate Test Sequences for Reconfigurable Scan Networks

Riccardo Cantoro*, Aleksa Damljanovic†, Matteo Sonza Reorda‡, Giovanni Squillero§
Politecnico di Torino,
Torino, Italy
{ *riccardo.cantoro, †aleksa.damljanovic, ‡matteo.sonzareorda, §giovanni.squillero } @polito.it

***Abstract*—Nowadays, industries require reliable methods for accessing the instrumentations embedded within semiconductor devices. The situation led to the definition of standards, such as the IEEE 1687, for designing the required infrastructures, and the proposal of techniques to test them. So far, most of the test-generation approaches are either too computationally demanding to be applied in complex cases, or too approximate to yield high-quality tests. This paper exploits a recent idea: the state of a generic reconfigurable scan chain is modeled as a finite state automaton and a low-level fault, as an incorrect transition; it then proposes a new algorithm for generating a functional test sequence able to detect all incorrect transitions far more efficiently than previous ones. Such an algorithm is based on a greedy search, and it is able to postpone costly operations and eventually minimize their number. Experimental results on ITC'16 benchmarks demonstrate that the proposed approach is broadly applicable; has limited computational requirements; and the test sequences are order of magnitudes shorter than the ones previously generated by approximate methodologies.**

## I. Introduction

In many of the latest Integrated Circuits (ICs) designers introduced resources whose purpose is not to support the circuit functionality, but rather to support ancillary features such as test, calibration, debug and monitoring. In particular, current ICs often integrate a plethora of sensors and actuators, each associated to a register to be read and/or written from the outside, sometimes at the end of the manufacturing process, sometimes during the operational phase. Many test solutions, such as BIST, also require registers to activate/initialize the test and retrieve results. In order to effectively access all these registers (also called instruments, or TDRs), companies used to include them into a single chain, often accessed through the standard IEEE 1149.1 interface. With the significant rise in complexity and the number of devices, exisiting infrastructure became inefficient. One of the limitations originated from the length of a single scan chain which was constantly increasing; performing an access to communicate with a single device resulted in large time overhead. Moreover, the reliability of such structure became an issue, since a problem on a single bit would lead to a catastophic breakdown. Another possibility involving architecture for accessing each instrument individually, apart from limited flexibility it provides, requires infeasible number of instructions to be implemented. To tackle these issues, solutions based on so called *Reconfigurable Scan Networks* (RSNs) were introduced. They allow to dynamically split a single chain into segments, each including one or more registers and include/exclude them selectively depending on the need [1].

Following this trend, IEEE published the IEEE 1687 standard [2], which in some way extends the popular IEEE 1149.1. The IEEE 1687 standard allows to split the scan chains accessible through the JTAG's Test Access Port (TAP), and to program their configuration; in this way, the designer can flexibly choose the best trade-off between different parameters, such as area or access time. The newest version of the IEEE 1149.1 standard [3] also describes ways to design RSNs within a circuit.

Typical RSNs are composed of chains of flip flops interleaved with special modules (called *Segment Insertion Bits* (SIBs) and *ScanMuxes* (SMs) by the IEEE 1687 standard), allowing to dynamically split the whole chain into segments that may be connected in series or in parallel, and to support a faster and more efficient access to the resources; the user first configures the network, selecting the subset of instruments to be accessed and shifting-in a proper sequence of bits to program the SIBs and SMs, then uses the network to serially read and write the flip-flops belonging to the currently active segments. CAD tools already exist, automating the introduction and the usage of RSNs [4]. When a circuit includes a RSN, the issue of testing the related hardware must clearly be considered, checking for possible defects affecting it. Failing to effectively solve this issue may lead to completely false results when using the RSN itself.

Some works faced the issue of testing the test circuitry mandated by the IEEE 1149.1 standard [5], while other works focused on the test of possible permanent faults affecting a standard scan chain, e.g., by shifting into the chain a sequence of alternated 0s and 1s, and checking that the same sequence appears at the other extreme of the chain [6]–[8].

However, to test an RSN is a more complex task with respect to the standard scan chain test, since examining the ability of flip-flops comprising the scan chain to shift is not sufficient to guarantee correct functionality and expected performance. In addition, testing should check whether the network can be moved from one configuration to another and if it operates correctly after enforcing whichever legal configuration. Although testing an RSN clearly shares some similarities with the task of design validation [9], time required to perform test, i.e. test stimuli duration is considered to be more important with more strict limitations.

The authors in [10] proposed a DfT modification to increase observability of TDR update cells. Different methods to perform structure-oriented test on RSNs are presented targeting specifically stuck-at and flip-flop internal and bridge faults.

In [11] we proposed a general approach to automatically generate a test sequence for an IEEE 1687 network with respect to permanent faults. For each type of programmable module we first introduced a high-level fault model, and then provided techniques for their test, and finally described how to combine them into a single comprehensive test. No modification of the hardware implementing the network is necessary to use this technique. Moreover, the test is applicable regardless of the specific implementation of the network modules. Test generation can start directly from the network structure (as described by the ICL file mandated by the IEEE 1687 standard). The test generation algorithms proposed in [11] are based on heuristics that can easily run even on relatively large RSNs.

In [12] we refined that approach to minimize the duration of the resulting test sequence: the faced problem was properly modeled according to the graph theory, and an optimal algorithm able to generate the minimum-duration test sequence was described. Unfortunately, such an approach can only work on relatively small RSNs, and sub-optimal solutions must be adopted when dealing with real cases.

Since the design of large and complex SoCs may cause the appearance of very complex RSNs, generating an effective test even for large RSNs may turn into a computationally complex problem, and methods able to effectively scale are increasingly important. In [13] we proposed a new method facing this problem, based on evolutionary computation. Its main advantage lies in its ability to always produce a solution, no matter the RSN complexity, while the quality of the produced result (i.e., the duration of the resulting test) is never too far from the optimum, when the latter can be computed, and it is often lower than the one produced by the method described in [11]. In [14] we first modeled the RSN as a Finite State Automaton (FSA), and then we proposed a semi-formal method able to deal with larger and more complex circuits producing a test sequence able to detect any permanent fault affecting the reconfigurable modules, but whose duration is lower than the one of the test sequences previously generated by the heuristic solutions.

In this paper we further extended the approach of [14] by rewriting the test generator. The new algorithm minimizes the number of costly operations, postponing and compacting them, while it still guarantees to reach complete fault coverage. At the same time, the new algorithm is almost always faster than its predecessor. Experimental results on a set of benchmarks [15] demonstrate that the approach is able to generate test sequences orders of magnitude shorter than those reported in [14], [11] and [13], while always keeping the computational cost under control.

The paper is organized as follows. In Section II we summarize the key characteristics of the IEEE 1687 networks. In Section III we propose the proposed technique for generating an optimized test sequence for a RSN. Section IV reports some experimental results gathered on the standard set of IEEE 1687 networks, and Section V finally draws some conclusions.

## II. Background

In this Section we will first briefly overview the key characteristics of an IEEE 1687 network, then explain how their test can be performed according to the approach first introduced in [11], and finally summarize why minimizing the test duration may turn into a computationally complex task.

### A. Overview of Reconfigurable Scan Networks

As was mentioned in Section I, a key feature of RSNs is the possibility to partition the set of instruments into segments, and then dynamically decide which segments are currently accessible and which are bypassed.

Communicating with the instruments is performed through TDRs. A TDR is composed out of one or multiple Shift-Capture-Update (SCU) scan cells, depending on the need. Based on the type of required access, these registers may be used as Read-Only or with an additional update stage as Write-Only and Read-Write. IEEE 1687 introduced reconfigurable module controlled by a Segment Insertion Bit (SIB). SIB is a single bit register with an update stage on a scan chain that allows bypassable segment to be included into, or excluded from the active path. Active path is represented by serially connected shift cells between TDI and TDO. A segment is composed out of one or multiple TDRs or out of various sub-network constructs with TDRs and other programmable components. By using SIBs, it is possible to create a hierarchical structure of the network.

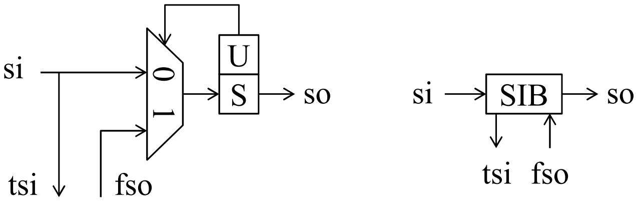

Fig. 1. Segment Insertion Bit (SIB) module: Simplified schematic (left) and symbol (right)

Fig. 1(left) shows the simplified schematic of a possible implementation of a SIB, which is based on a two-input multiplexer and a one-bit shift-update register. SIBs can be programmed by shifting a bit into their S flip-flop and latching that bit into the parallel U latch. If the latched bit is 0, the SIB is de-asserted, and the scan-path is from the `si` terminal, to the `so` terminal via the S flip-flop, bypassing the segment between the `tsi` and `fso` terminals. If, on the other hand, the latched bit is a 1, the SIB is asserted, and the scan-path includes the segment connected between the `tsi` and `fso` terminals of the SIB. In this paper, the symbol shown in Fig. 1(right) is used to represent a SIB. When de-asserted, shortened length corresponds to one bit, while when asserted expands adding the length of the associated segment. Moreover, a design guidance supports different SIB module implementations. Depending on the position of the control bit with respect to the multiplexer from Fig. 1(left), a SIB can be either *post*- or *pre*-. If the control bit is located in the same segment as the multiplexer, the SIB is considered to be in-line (adjacent or distant); otherwise it is remotely controlled.

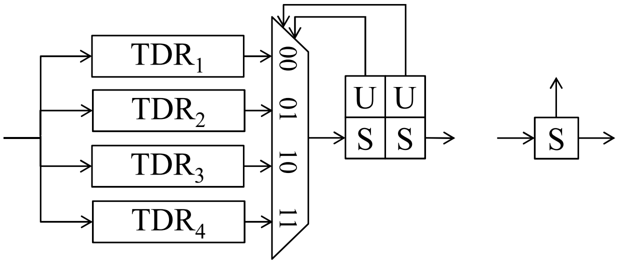

Fig. 2. ScanMux (SM) module: Simplified schematic (left) and symbol (right)

Other ad hoc RSNs can be constructed by the use of SMs and shift-update registers. As an example, consider the network shown in Fig. 2(left) in which a two-bit shift-update register is used to select among four inputs of a 4-to-1 SM. Here again, the configuration of the ScanMux can be performed by shifting the required values into the S shift flip-flops of the control register and then latching the shifted bits into the U latches. In the rest of this paper, the symbol shown in Fig. 2(right) will be used to represent the shift-update register that controls an SM.

The IEEE 1687 standard does not specify a type of external interface that has to be used to access the network. However, the one that is most widely used is IEEE 1149.1 TAP. The TAP finite state machine is responsible for asserting control signals to configure the IEEE 1687 network and access the instruments through it.

An example of a simple RSN is given in Fig. 3, representing a circuit that includes 5 instruments. A Test Data Register (from TDR$_1$ to TDR$_5$) is asociated to each of the instruments, which can be accessed by performing read or write operations through the TAP port. Although all TDRs may be connected into a single scan-chain like in IEEE 1149.1-compliant circuits, the designer may adopt a different strategy in order to reduce instrument access time: an IEEE 1687 network with three SIBs and one SM, as shown in the Fig. 3; depending on the status of the 4 configurable modules, a different subset of TDRs (and the corresponding instruments) is included in the *active path* i.e., the path connected between the scan input and scan output pins of the reconfigurable scan chain at a given time [16]. A list of 16 possible configurations, i.e. 8 different active paths supported by this network is provided in Table I. In the same table, “A” means asserted, “D” means de-asserted, while 0 and 1 correspond to the two possible positions of the SM.

### *B. Test of Reconfigurable Scan Networks*

Testing a standard (non-reconfigurable) scan chain for permanent faults can be performed by shifting a suitable sequence of 0s and 1s through the scan chain. RSNs are however, far more complicated to test. Flip-flops composing the TDRs have to be tested to check if they can correctly shift values. Additionally, reconfigurable modules have to be tested to check whether they are able to move the network to the corresponding configurations.

In this work we use the high-level fault model introduced in [11]. The faults affecting the reconfigurable modules, such as SMs, are modeled such that a different configuration is selected rather than the expected one, and this could lead to a path with different length. For example, in Fig. 3 the multiplexer (MUX) may be affected by a permanent fault whose effect is that the segment connected to the input 0 is always selected, no matter the value in the selection cell. The same may arise for the generic SIB$_i$, which can be affected by faults, which are named stuck-at asserted/de-asserted, or SIB$_i$-s@A, SIB$_i$-s@D. The stuck-at faults in the scan bits of the selection cells are considered as detected by implication by testing such high-level faults, which cover also the faults affecting the update logic of the reconfigurable modules. Moreover, such faults cover some faults affecting the reset logic, whose effect is that the module is stuck at the reset value. The other reset faults (i.e., those that make the reset ineffective) are not considered but can be targeted by resorting to the techniques described in [17].

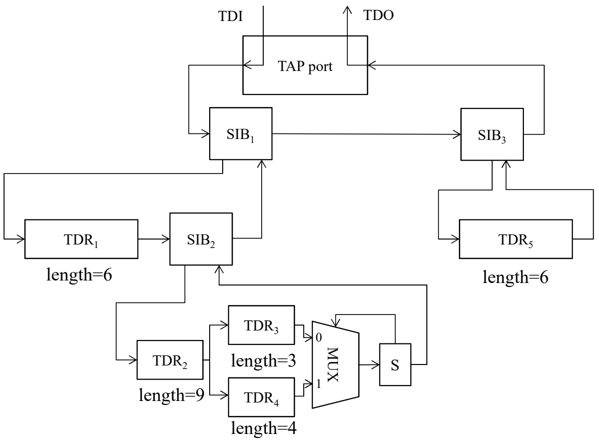

Fig. 3. Example of an IEEE 1687 RSN.

Considering this high-level fault model, a test of an RSN can be performed by first configuring the RSN to excite the target fault, and then comparing the length of the activated path against the length of the expected path. As an example, the high-level fault affecting the SM of Fig. 3, which always selects the segment connected to the input 1, can be excited by a configuration which selects the input 0; configurations $C_{12}$ and $C_{14}$ fulfill this requirement. Once one of them is activated, one can measure the length of the active path by shifting a given sequence (called test vector) in TDI and checking when

TABLE I
SET OF POSSIBLE CONFIGURATIONS OF THE RSN IN FIG. 3.

| Config. | SIB$_1$ | SIB$_2$ | SIB$_3$ | SM | Active path | Len. |
|---|---|---|---|---|---|---|
| C$_0$ | D | D | D | 0 | - | 2 |
| C$_1$ | | | | 1 | | |
| C$_4$ | | A | | 0 | | |
| C$_5$ | | | | 1 | | |
| C$_2$ | D | D | A | 0 | TDR$_5$ | 8 |
| C$_3$ | | | | 1 | | |
| C$_6$ | | A | | 0 | | |
| C$_7$ | | | | 1 | | |
| C$_8$ | A | D | D | 0 | TDR$_1$ | 9 |
| C$_9$ | | | | 1 | | |
| C$_{10}$ | A | D | A | 0 | TDR$_1$, TDR$_5$ | 15 |
| C$_{11}$ | | | | 1 | | |
| C$_{12}$ | A | A | D | 0 | TDR$_1$, TDR$_2$, TDR$_3$ | 22 |
| C$_{13}$ | A | A | D | 1 | TDR$_1$, TDR$_2$, TDR$_4$ | 23 |
| C$_{14}$ | A | A | A | 0 | TDR$_1$, TDR$_2$, TDR$_3$, TDR$_5$ | 28 |
| C$_{15}$ | A | A | A | 1 | TDR$_1$, TDR$_2$, TDR$_4$, TDR$_5$ | 29 |

it will appear on TDO. Any fault modifying the length of the active path can be detected in this way.

A proper test sequence consists of an alternating bits sequence 0101..., as long as the active path length followed by a *sequence terminator*, such as two consecutive 0s or 1s. For example, if the network in Fig. 3 is configured to C$_8$ (see Table I), a proper test vector is 01010101011, that is, 9 bits of alternated 0s and 1s followed by the sequence terminator. Faults affecting the network may corrupt the network by changing the active path, which will cause the sequence terminator to be observed on the scan output in an unexpected clock cycle. For example, if a stuck-at fault affects the selection of the module SIB$_1$ (which is supposed to be asserted in the fault-free scenario), then the network may exclude the SIB$_1$'s controlled segment, as in the SIB$_1$'s de-asserted case. Thus, the active path selected in such a faulty scenario would be the same of the configuration C$_0$ of Table I. In the faulty scenario, the path length is 2, meaning that the sequence terminator is observed earlier than expected on the scan output pin.

In order to test all configurable modules in an RSN, a test sequence can be organized as a set of *sessions*: in each session we first configure the network via one or more configuration vectors (so that each SIB and each SM is switched into a given position), and then check whether the expected path has been inserted between TDI and TDO via a test vector, i.e., whether the right segments can be accessed. Since the number of possible configurations of a network grows exponentially with the number of configurable modules, the problem of identifying a sequence of sessions which guarantees that 1) all the configurations modules are fully tested, and 2) the total test duration is minimized, is not trivial.

## III. PROPOSED APPROACH

In the proposed approach, the RSN of IEEE 1687 is modeled as a finite state automaton as in [14]. Each state corresponds to a *configuration*, that is, a determinate state of SIBs and SMs in the network; the input alphabet corresponds to reconfiguration operations; the output symbols are the lengths of the active paths, as this is an easily observable characteristic [11]. The high-level model is deliberately not complete, that is, the FSA's states encode only a subset of the possible configurations. As not all transitions are possible in all states, either due to the physical configuration of the RSN or to missing states in the FSA, whether an input does not correspond to a transition, the FSA is brought to a special *sink state* ($\Omega$) with no output transitions and a *null* output symbol.

Faults taken into consideration are high-level stuck-at faults affecting SIBs and SMs. Such faults are mapped to multiple transition faults on the high-level automaton, as the same configuration operations may result in different network statuses on faulty circuits, and the goal of the automatic test program generation is to devise a sequence of inputs able to discriminate between the faulty automata and the good one.

The proposed algorithm is based on a greedy search. While the simulation of the automaton is exact, the method is approximate because it does not consider all possible states nor all possible input symbols, and, consequently, not all possible transitions. Nevertheless, the approximation is conservative with respect to testability, as any missing state or transition will cause the automaton to reach the *sink state*, that by construction cannot be further distinguished from any other state.

The complexity of the proposed approach is linear on the number of states $n_{\mathrm{s}}$ times the size of the input alphabet $\mathbf{A}_{\mathrm{in}}$, that is $\mathcal{O}(n_{\mathrm{s}} \cdot |\mathbf{A}_{\mathrm{in}}|)$. In most circuits, both terms depend linearly on the number of configuration bits $n_{\mathrm{cb}}$; when states with a hamming distance of 2 are also required, then $n_{\mathrm{s}} = \mathcal{O}(n_{\mathrm{cb}}^2)$. In all cases, however, the complexity is definitely smaller than the A* algorithm presented in [12], where the search space was $\mathcal{O}(2^{n_{\mathrm{cb}}})$.

### A. Finite State Automaton

The FSA is built incrementally. The FSA is initially composed of only of a state with no output transition and a *null* output symbol. Such *sink state* can not be distinguished from any other state, and, once entered, the FSA is not able to leave it. It is used to denote a pathological condition, where the algorithm is not able to provide reliable results due to the approximation of the model. Next, the *reset state*, when all configuration bits are set to the initial value, is added to the automaton. Then, for each SIB$_i$, two states are created: one with the SIB asserted and one with the SIB de-asserted. For each SM, one state is created for each possible output configuration. Such a straightforward approach, however, is not always sufficient. Scan segments may be nested, and a resource accessible only when its parent SIB is asserted. The procedure for building the FSA detects such situations, and creates the necessary states to handle them. The transitions from the *reset state* to all these states are eventually added.

For instance, SIB$_2$ in Fig. 4 is only accessible when SIB$_1$ is asserted. Therefore, the FSA would include the reset state $(\overline{\text{SIB}_1}, \overline{\text{SIB}_2}, \overline{\text{SIB}_3})$; then the three states with only one SIB asserted { (SIB$_1$, $\overline{\text{SIB}_2}$, $\overline{\text{SIB}_3}$), ($\overline{\text{SIB}_1}$, SIB$_2$, $\overline{\text{SIB}_3}$), ($\overline{\text{SIB}_1}$, $\overline{\text{SIB}_2}$, SIB$_3$) }; finally, the state (SIB$_1$, SIB$_2$, $\overline{\text{SIB}_3}$), as asserting SIB$_1$ is necessary to test SIB$_2$.

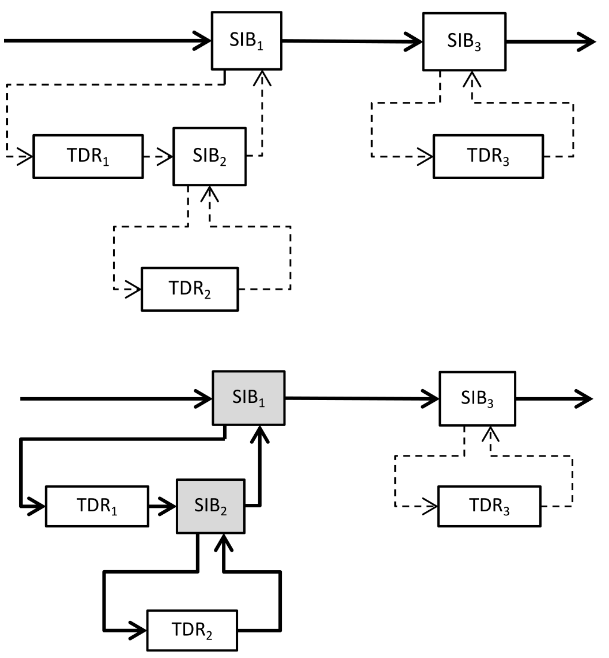

Fig. 4. Example of generating input symbols for SIB RSN.

Then, for each transition in the good automaton, the possible faulty transitions are added, and whether the faulty transition would bring the automaton in a configuration not already encoded as a state, that specific state is added to the FSA. All missing transitions between existing states are also added to the automaton. Eventually, all possible faulty transitions from all existing states are also added, but if one would bring the automaton in a configuration not encoded as a state, its destination is set to the *sink state*, meaning that the FSA is unable to model such situation.

As almost only the states with a hamming distance of 1 from the reset state are added to the FSA, the size of the automaton is linear in the number of configuration bits. It is possible to define an automaton with more states: for instance, at some point of the creation, all complementary states may be added as well; or all states at a hamming distance of 2 from the reset state can be considered. It is important to remember that the size of the automaton influences both the quality of the results and the performance of the algorithm.

Some heuristics are considered in order to match configurations which may reduce the cost. For example, states representing configurations in which accessible SIBs that provide access to the deepest hierarchical level are not asserted may increase the number of required sessions and therefore the cost. Additionally, configurations in which a SIB is still asserted while already being fully tested together with its sub-hierarchical modules increase the cost. Not setting minimal path length configuration of a ScanMuxes that is fully tested together with its sub-hierarchical modules increases the cost.

Experimental evaluations indicate that such extensions are not quite beneficial, but the designers may explicitly add relevant states to this state or provide any additional heuristic.

### *B. Search Algorithm*

The search algorithm builds a test sequence as a sequence of *transition* and *observation* steps. During a *transition*, a sequence of bits is fed into the scan chain, bringing the RSN in a given configuration; such operation corresponds to one or more input symbols in the FSA. During an *observation*, the length of the scan chain is measured; the operation does not affect the FSA.

In more practical terms, the goal of the test generation is to find a short and effective sequence that brings the good circuit and the faulty ones in states where the scan chain is of different lengths; then, to observe the length and detect the faults. Indeed, not all transitions and not all observations require the same number of clock cycles to be performed. The search algorithm aims at minimizing the total length of the test sequence with respect to the number of actual clock cycles required to execute all transitions and observations.

Let $x$ be an input symbol for the FSA. The reset operation is denoted with $\mathbf{reset}$, and it requires a single clock cycle to be performed; the measurement of the length of the scan chain is denoted with $\mathbf{observe}$, it requires several clock cycles and does not affect the state of the FSA. Both appending a symbol to an input sequence and concatenating two sequences are expressed as additions, as no ambiguities are possible. The symbol $\varnothing$ denotes an empty input and has no effect on a sequence, e.g., $\mathbf{t} = \mathbf{t} + \varnothing$. A sequence $\mathbf{t}$ of inputs starting with a $\mathbf{reset}$, i.e., $\mathbf{t} = (\mathbf{reset}, i_0, i_1, ..., i_i)$, unequivocally defines the state of the FSA.

Two states that have indistinguishable output symbols are equivalent and are denoted with $s' \cong s''$. Conversely, non equivalent states have distinguishable output symbols and are denoted with $s' \ncong s''$. By definition, the sink state is equivalent to any other states $\forall s : s \cong \Omega$.

Let $\bar{s}_\mathbf{t}$ be the state of the FSA representing the fault-free circuit after the application of the input sequence $\mathbf{t}$, while $s^i_\mathbf{t}$ be the state of the FSA representing the circuit when fault $i$ is present after the application of the same input sequence. If their output symbols are distinguishable, that is, $\bar{s}_\mathbf{t} \ncong s^i_\mathbf{t}$, then an additional $\mathbf{observe}$ input symbol would allow to mark the fault $i$ as detected, and the fault is said to be *active*. The number of active faults may increase as well as decrease at each step of the input sequence.

Let $\mathcal{D}(\mathbf{t})$ be the set of all faults detected by the sequence $\mathbf{t}$. Indeed, $\mathcal{D}(\mathbf{t}) = \varnothing$ if $\mathbf{t}$ contains no $\mathbf{observe}$ symbols; and all input symbols after the last $\mathbf{observe}$ do not alter the results. Let $\mathcal{D}^*(\mathbf{t})$ be the set of all faults *potentially detected* by the sequence $\mathbf{t}$, that is, all faults either already detected or active after the application of the sequence $\mathbf{t}$, that is, the set of all faults that would be detected by appending an $\mathbf{observe}$ to the input sequence: $\mathcal{D}^*(\mathbf{t}) = \mathcal{D}(\mathbf{t} + \mathbf{observe})$.

The search algorithm incrementally builds a test sequence through a greedy search. Explicit observations, that is $\mathbf{observe}$ symbols, are not included in the test sequence unless required. The function GREEDY returns the most useful input symbol to be added (Algorithm 1), neglecting observations: given an input sequence $\mathbf{t}$, it identifies the symbol $s$ that maximizes $|\mathcal{D}^*(\mathbf{t}+s)|$. If adding a single symbol cannot activate any new fault, the function returns an empty symbol.

**Algorithm 1** Identify most useful input symbol, neglecting observations.

```
function GREEDY($\mathbf{t}$)
    $best \leftarrow \varnothing$                                      ▷ Empty symbol
    for $x \in \{\text{valid input symbols in } \bar{s}_{\mathbf{t}}\}$ do
        if $|\mathcal{D}^*(\mathbf{t}+x)| > |\mathcal{D}^*(\mathbf{t}+best)|$ then
            $best \leftarrow x$
    return $best$                                       ▷ Most useful symbol
```

The search algorithm incrementally builds the test sequence $\mathbf{t}$ calling the function GREEDY iteratively (Algorithm 2). In every step, the most useful symbol is appended to the test sequence, trying to increase the number of active faults. Only when a new symbol $s$ would cause the loss of a previously activated fault, an **observe** symbol is inserted before $s$.

When it is not possible to activate new faults by adding a single symbol, an **observe** symbol is appended and the FSA is rolled back to a previous state where useful input symbols may still be found and the search restarted. Such a state is chosen among the previously traversed ones, and it is the closest one in term of configuration clock cycles. The procedure terminates when all detectable faults have been detected.

**Algorithm 2** Test Sequence Generation

```
procedure TPG
    $\mathbf{t} \leftarrow (\mathbf{reset})$                          ▷ Initial test sequence
    $\mathbf{H} \leftarrow \{\mathbf{t}\}$                                        ▷ History
    while $\mathcal{D}^*(\mathbf{t}) \neq \{\text{all detectable faults}\}$ do
        $s \leftarrow \text{Greedy}(\mathbf{t})$
        if $s \neq \varnothing$ then                        ▷ The greedy succeeded
            if $\mathcal{D}^*(\mathbf{t}) \nsubseteq \mathcal{D}^*(\mathbf{t}+s)$ then
                $\mathbf{t} \leftarrow \mathbf{t} + \mathbf{observe}$                     ▷ Required
            $\mathbf{t} \leftarrow \mathbf{t} + s$                              ▷ Add symbol
            $\mathbf{H} \leftarrow \mathbf{H} \cup \{\mathbf{t}\}$                       ▷ Save sequence
        else                                             ▷ The greedy failed
            $\mathbf{t} \leftarrow \mathbf{t} + \mathbf{observe}$
            $\mathbf{r} \leftarrow \text{shortest}(\{\mathbf{a} \in \mathbf{H} : \text{Greedy}(\mathbf{t}+\mathbf{a}) \neq \varnothing\})$
            $\mathbf{t} \leftarrow \mathbf{t} + \mathbf{r}$                               ▷ Start over
    $\mathbf{t} \leftarrow \mathbf{t} + \mathbf{observe}$                     ▷ Final observation
```

To demonstrate the difference in the Test Sequence Generation procedure between this approach and [14], we can use the network shown in Fig. 2. Table II and Table III show the phases of test sequence generation using procedure described in [14] and the present one, respectively. The first column (*input*) shows input symbols that were chosen and applied. The second column refers to the fault-free network and its states. Columns 3-10 show the state of the faulty circuits, each one for one particular fault. As it can be seen from Table II, in the previous approach each configuration phase is followed by an observation phase (marked in bold). In this way a set of active faults is added to the set of detected faults, which is increasing after each session. However, in Table III one can see that each configuration phase is not necessarily followed by an observation phase. Although the number of configuration steps is higher, the number of observation steps, which can be extermely costly, is lower. In total, the number of clock cycles needed to apply the test sequence given in Table II is 283 clock cycles, while on the other hand, applying the test sequence from Table III requires 220 clock cycles, only.

In comparison with [14], the length of the configuration vector may not be equal to the value of the output symbol of the fault-free circuit's current state, $\bar{s}_{\mathbf{t}}$. The length of the configuration vector is equal to $\max(\bar{s}_{\mathbf{t}}, s^i_{\mathbf{t}}), (\forall i)(i \in \{0, 1, \ldots, n-1\} \wedge (\text{fault } i \text{ not detected}))$, where $n$ represents the total number of faults. The length of the configuration vector is included in the configuration cost. Positions of certain configuration bits in the chain that is defined by the state $s^i_{\mathbf{t}}$ or $s^j_{\mathbf{t}}$, $i \neq j$, may not correspond to any position of configuration bits in the chain determined by the state $\bar{s}_{\mathbf{t}}$. In this case, 0 bits are placed on these positions (Fig. 5). Additionally, on the same position (in chains defined by different states), one may find configuration bits belonging to different modules. This is all taken into account when assembling and then applying configuration vector corresponding to the chosen input symbol.

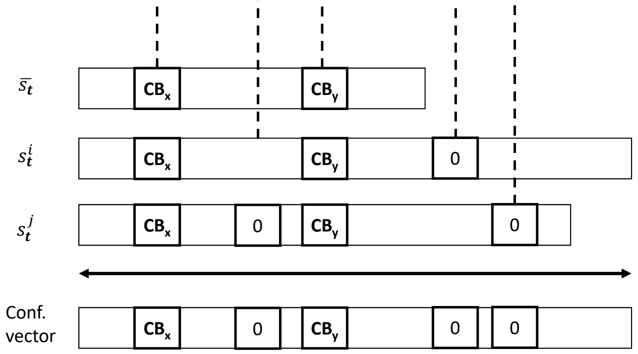

Fig. 5. Configuration vector

## IV. Experimental Results

The effectiveness of the proposed algorithm has been evaluated on a sub-set of the ITC16 suite of benchmark reconfigurable scan networks. Some networks included in the benchmarks have not been considered since they include some constructs that are not currently supported by our environment. The algorithm proposed in this paper has been compared against three alternative approaches. The first approach, to which we refer to as FSA, has been proposed in [14]. The second approach is derived from [11] and is referred to as *depth-first* in this paper. The approach is based on the

TABLE II
TEST PROCEDURE GENERATED BY APPROACH [15]

| input | good | SIB$_1$ s@D | SIB$_1$ s@A | SIB$_2$ s@D | SIB$_2$ s@A | SM br.0 | SM br.1 | SIB$_3$ s@D | SIB$_3$ s@A |
|---|---|---|---|---|---|---|---|---|---|
| reset observation | DD0D (2) | DD0D (2) | **AD0D (9)** | DD0D (2) | DA0D (2) | DD0D (2) | DD1D (2) | DD0D (2) | **DD0A (8)** |
| AxxA observation | AD0A (15) | **DD0A (8)** | **AD0D (9)** | AD0A (15) | **AA0A (28)** | AD0A (15) | AD1A (15) | **AD0D (9)** | **DD0A (8)** |
| AAxD observation | AA0D (22) | **DD0A (8)** | **AD0D (9)** | **AD0D (9)** | **AA0A (28)** | AA0D (22) | **AA1D (23)** | **AD0D (9)** | **DD0A (8)** |
| AA1D observation | AA1D (23) | **DD0A (8)** | **AD0D (9)** | **AD0D (9)** | **AA0A (28)** | **AA0D (22)** | **AA1D (23)** | **AD0D (9)** | **DD0A (8)** |

TABLE III
TEST PROCEDURE GENERATED BY PRESENT APPROACH

| input | good | SIB$_1$ s@D | SIB$_1$ s@A | SIB$_2$ s@D | SIB$_2$ s@A | SM br.0 | SM br.1 | SIB$_3$ s@D | SIB$_3$ s@A |
|---|---|---|---|---|---|---|---|---|---|
| reset | DD0D (2) | DD0D (2) | AD0D (9) | DD0D (2) | DA0D (2) | DD0D (2) | DD1D (2) | DD0D (2) | DD0A (8) |
| AxxA | AD0A (15) | DD0A (8) | AD0D (9) | AD0A (15) | AA0A (28) | AD0A (15) | AD1A (15) | AD0D (9) | AD0A (15) |
| AAxA | AA0A (28) | DD0A (8) | AA0D (22) | AD0A (15) | DA0D (2) | AA0A (28) | AA1A (29) | AA0D (22) | AA0A (28) |
| AA1A observation | **AA1A (29)** | **DD0D (2)** | **AA1D (23)** | **DD0D (2)** | **DA0D (2)** | **AA0A (28)** | **DA1D (2)** | **AA1D (23)** | AA1A (29) |
| DD0D observation | **DD0D (2)** | **DD0D (2)** | **AA1D (23)** | **DD0D (2)** | **DA0D (2)** | **AA0A (28)** | **DA1D (2)** | **AA1D (23)** | **DD0A (8)** |

exploration of the network topology graph performing a depth-first traversal of this graph. The third approach has been proposed in [13] and is referred to as *evolutionary* in this paper. The approach makes use of an evolutionary framework to generate a test sequence possibly able to minimize the test time.

Table IV reports some basic information about the networks used to perform evaluation. In column 2 and 3, the table reports for each network the number of SIBs and SMs, respectively. The number of configuration bits of SIBs and SMs is given in the fourth column. The column *Max depth* indicates the maximum hierarchical depth of each network (for SIB-based networks this value equals to the maximum number of nested SIBs, according to [18]). The column *Max path* reports the length of the longest path in the network, and the rightmost column the number of bits in all the TDRs.

Experiments were run using a tool written in Java. The tool supports network structure extraction from files in different formats including ICL. Moreover, the tool is able to distinguish all faults that are undetectable, due to the inability to produce any difference in the path length. For example, faults affecting SIB modules that do not have any register or any other module on their branch are considered to be undetectable. Additionally, faults affecting ScanMux modules that have registers of equal lengths on their branches are also considered as undetectable, again taking into account the faut model that was used in this approach. However, there is only a small number of undetectable faults in the set of benchmark networks that were used to evaluate the algorithm,for which we provide details in Table Table V.

TABLE IV
ITC'16 BENCHMARK NETWORKS LIST

| Network | SIB | SM | Tot. bits | Max depth | Max path | Scan cells |
|---|---|---|---|---|---|---|
| Mingle | 10 | 3 | 13 | 4 | 171 | 270 |
| TreeBalanced | 43 | 3 | 48 | 7 | 5,219 | 5,581 |
| TreeFlat_Ex | 57 | 3 | 62 | 5 | 5,100 | 5,195 |
| TreeUnbalanced | 28 | – | 28 | 11 | 42,630 | 42,630 |
| a586710 | – | 32 | 32 | 4 | 42,381 | 42,410 |
| p22810 | 270 | – | 270 | 2 | 30,356 | 30,356 |
| p34392 | – | 96 | 96 | 4 | 27,899 | 27,990 |
| p93791 | – | 596 | 596 | 4 | 100,709 | 101,291 |
| q12710 | 27 | – | 27 | 2 | 26,185 | 26,185 |
| t512505 | 159 | – | 159 | 2 | 77,005 | 77,005 |
| N132D4 | 39 | 40 | 79 | 5 | 2,555 | 2,991 |
| N17D3 | 7 | 8 | 15 | 4 | 372 | 462 |
| N32D6 | 13 | 10 | 23 | 4 | 84,039 | 95,158 |
| N73D14 | 29 | 17 | 46 | 12 | 190,526 | 218,869 |
| NE1200P430 | 381 | 430 | 811 | 127 | 88,471 | 108,148 |
| NE600P150 | 207 | 194 | 401 | 78 | 23,423 | 28,250 |

TABLE V
UNDETECTABLE FAULTS

| Network | Number | Comment |
|---|---|---|
| q12710 | 4 | 2 SIBs with the register length equal to 0 |
| N132D4 | 4 | 2 ScanMuxes eq. branch registers (11, 18) |
| NE600P150 | 4 | 2 ScanMuxes eq. branch registers (45, 11) |
| NE1200P430 | 4 | 2 ScanMuxes eq. branch registers (115, 29) |

A laptop equipped with an Intel i5-480M processor was used to run experiments. Table VI summarizes the experimental results. The table shows the number of configuration vectors $cv$ (column 2) and test vectors $tv$ (column 3) generated by the tool. Futhermore, the number of clock cycles required to configure the network is given in column 4, while the number of clock cycles needed to apply test vectors is given in column 5.

TABLE VI
IEEE 1687 TEST ALGORITHM EXPERIMENTAL RESULTS

| Network | $cv$ | $tv$ | Conf. time [cc] | Test time [cc] |
|---|---|---|---|---|
| Mingle | 6 | 7 | 628 | 811 |
| TreeBal. | 7 | 1 | 8,569 | 12,646 |
| TreeFlat_Ex | 6 | 3 | 7,750 | 16,267 |
| TreeUnbal. | 11 | 1 | 105,197 | 77,121 |
| a586710 | 4 | 5 | 46,575 | 170,257 |
| p22810 | 2 | 1 | 2,698 | 90,537 |
| p34392 | 6 | 3 | 29,357 | 111,911 |
| p93791 | 6 | 3 | 103,525 | 403,532 |
| q12710 | 2 | 1 | 8,311 | 78,562 |
| t512505 | 2 | 1 | 8,891 | 230,438 |
| N132D4 | 7 | 2 | 9,387 | 7,682 |
| N17D3 | 5 | 2 | 1,159 | 1,151 |
| N32D6 | 5 | 2 | 230,390 | 282,236 |
| N73D14 | 13 | 2 | 1,073,954 | 537,833 |
| NE1200P430 | 128 | 2 | 1,638,849 | 200,258 |
| NE600P150 | 79 | 2 | 347,629 | 55,098 |

TABLE VII

EXPERIMENTAL COMPARISON OF THE PROPOSED ALGORITHM (FSA2) AGAINST THE PREVIOUS VERSION [14], A DEPTH-FIRST ALGORITHM [11], AND AN EVOLUTIONARY APPROACH [13]. COLUMNS ENDING WITH "VS." SHOW THE COMPARISON AGAINST THE CURRENT RESULT; PERCENTAGES QUANTIFY HOW MUCH THE RESULTS DELIVERED BY THE PREVIOUS APPROACHES ARE WORSE.

| Network | Total test time [clock cycles] | | | | | | | Runtime (wall clock) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | FSA2 | [14] | [14] vs. | [11] | [11] vs. | [13] | [13] vs. | FSA2 | [14] | [11] | [13] |
| Mingle | 1,439 | 2,014 | 39.96% | 2,282 | 58.58% | 2,078 | 44.41% | 49s | 26s | 1s | 8h |
| TreeBalanced | 21,215 | 63,843 | 200.93% | 69,369 | 226.98% | 69,369 | 226.98% | 1m | 48s | 1s | 19h |
| TreeFlat Ex | 24,017 | 41,883 | 74.39% | 71,341 | 197.04% | 55,776 | 132.24% | 1m | 71s | 1s | 8h |
| TreeUnbalanced | 182,318 | 719,375 | 294.57% | 1,071,799 | 487.87% | 1,042,450 | 471.78% | 1m | 34s | 1s | 5h |
| a586710 | 216,832 | 296,796 | 36.88% | 299,624 | 38.18% | 298,241 | 37.54% | 30s | 39s | 1s | 8h |
| p22810 | 93,235 | 152,399 | 63.46% | 152,937 | 64.03% | 152,937 | 64.03% | 44s | 39s | 1s | 9h |
| p34392 | 141,268 | 195,554 | 38.43% | 196,702 | 39.24% | 196,505 | 39.10% | 36s | 1m | 1s | 7h |
| p93791 | 507,057 | 706,242 | 39.28% | 708,878 | 39.80% | 708,878 | 39.80% | 10m | 2m | 1s | 27h |
| q12710 | 86,873 | 131,022 | 50.82% | 131,022 | 50.82% | 131,022 | 50.82% | 39s | 46s | 1s | 5h |
| t512505 | 239,329 | 385,440 | 61.05% | 386,024 | 61.29% | 386,024 | 61.29% | 45s | 50s | 1s | 8h |
| N132D4 | 17,069 | 31,995 | 87.45% | 38,731 | 126.91% | 37,257 | 118.27% | 4m | 2s | 1s | 3h |
| N17D3 | 2,310 | 3,765 | 62.99% | 4,143 | 79.35% | 3,851 | 66.71% | 1s | 1s | 1s | 5h |
| N32D6 | 512,626 | 816,634 | 59.30% | 942,470 | 83.85% | 893,017 | 74.20% | 1s | 6s | 1s | 5h |
| N73D14 | 1,611,787 | 4,377,449 | 171.59% | 5,978,047 | 270.90% | 5,967,137 | 270.22% | 16s | 97s | 3s | 3h |
| NE1200P430 | 1,839,107 | 14,794,857 | 704.46% | 21,515,705 | 1,069.90% | 21,515,705 | 1,069.90% | 19m | 1h | 3s | 50h |
| NE600P150 | 402,727 | 2,694,672 | 569.11% | 3,726,726 | 825.37% | 3,726,726 | 825.37% | 3m | 4m | 3s | 12h |

The cost of every configuration phase expressed in clock cycles has been increased by five (JTAG overhead) [13]. In addition, the same overhead has been taken into account for calculating the cost of a test phase. This cost consists of the length of the longest path and the length of the currently active path increased by two (test pattern termination symbols).

All modeled, detectable faults were detected in each of the experiments, thus reaching full coverage.

An experimental comparison of the proposed approach against the FSA, depth-first and evolutionary approaches is shown in Table VII. Data concerning the FSA approach are taken from [14]. Reported data related to the evolutionary approach are taken from [13]. For the depth-first approach, data have been newly generated on the ITC16 benchmarks by running the tool implementing the same algorithm as in [11]. For each algorithm, Table VII reports the duration in clock cycles of the generated test sequence (referred to as *Test Application Time*) and the CPU time required to apply the algorithm (referred to as *Generation Time*).

Remarkably, results in Table VII show a clear improvement regarding the total test time, since the results delivered by the previous approaches were worse up to 705% for depth-first, up to 1,070% for depth-first and evolutionary method. Moreover, the test sequence generated by the proposed approach is shorter than the sequences obtained by the other algorithms in all networks.

Concerning the runtime, as Java's non-determinism prevents an accurate timing, only the total time is reported for all programs (wall-clock). The proposed algorithm completes in the order of seconds, while 19 minutes was required only for one network (NE1200P430). The depth-first algorithm is very fast to execute, even for large networks. The evolutionary approach, on the other hand, requires hours. Moreover, the results reported in [13] for the evolutionary approach were gathered on a multi-core server, exploiting parallelism, while a simple laptop has been used to run the proposed approach.

## V. CONCLUSIONS

The paper describes an efficient technique for generating sequences for testing IEEE 1687 RSNs. The approach can be defined as semi-formal because the FSA that models the circuit is exact, but incomplete, and the search procedure is based on a greedy algorithm. Experimental results on the ITC'16 benchmark suite clearly demonstrate the effectiveness of the approach: the proposed technique is able to achieve better results with less computation effort than previous methods. The technique may be easily extended to handle different fault models and more complex scenarios, and experts' knowledge could be exploited by tweaking the FSA states and input alphabet. Currently, additional experiments are in progress to better understand the current limitations and possible improvements of the proposed method.

## ACKNOWLEDGMENT

This is an extension of the paper published in ITC Asia 2018, which is selected as one of the top three papers from ITC Asia 2018 to appear in ITC special session on ITC Asia.

This work has been partially supported by the European Commission through the Horizon 2020 RESCUE-ETN project under the Marie Skodowska-Curie grant agreement No.722325.

## REFERENCES

[1] S. Narayanan and M. A. Breuer, "Reconfigurable scan chains: A novel approach to reduce test application time," in *Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design*. IEEE Computer Society Press, 1993, pp. 710–715.

[2] "IEEE standard for access and control of instrumentation embedded within a semiconductor device," *IEEE Std 1687-2014*, pp. 1–283, Dec 2014.

[3] "IEEE standard for test access port and boundary-scan architecture," *IEEE Std 1149.1-2013 (Revision of IEEE Std 1149.1-2001)*, pp. 1–444, May 2013.

[4] F. G. Zadegan, U. Ingelsson, G. Carlsson, and E. Larsson, "Design automation for IEEE p1687," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*. IEEE, 2011, pp. 1–6.

[5] A. T. Dahbura, M. U. Uyar, and C. W. Yau, "An optimal test sequence for the jtag/ieee p1149. 1 test access port controller," in *Test Conference, 1989. Proceedings. Meeting the Tests of Time., International*. IEEE, 1989, pp. 55–62.

[6] K.-J. Lee and M. A. Breuer, "A universal test sequence for cmos scan registers," in *Custom Integrated Circuits Conference, 1990., Proceedings of the IEEE 1990*. IEEE, 1990, pp. 28–5.

[7] S. Maka and E. J. McCluskey, "Atpg for scan chain latches and flip-flops," in *VLSI Test Symposium, 1997., 15th IEEE*. IEEE, 1997, pp. 364–369.

[8] F. Yang, S. Chakravarty, N. Devta-Prasanna, S. M. Reddy, and I. Pomeranz, "On the detectability of scan chain internal faults an industrial case study," in *VLSI Test Symposium, 2008. VTS 2008. 26th IEEE*. IEEE, 2008, pp. 79–84.

[9] R. Baranowski, M. A. Kochte, and H.-J. Wunderlich, "Reconfigurable scan networks: Modeling, verification, and optimal pattern generation," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 20, no. 2, p. 30, 2015.

[10] D. Ull, M. Kochte, and H. J. Wunderlich, "Structure-oriented test of reconfigurable scan networks," in *2017 IEEE 26th Asian Test Symposium (ATS)*, Nov 2017, pp. 127–132.

[11] R. Cantoro, M. Montazeri, M. Sonza Reorda, F. G. Zadegan, and E. Larsson, "On the testability of IEEE 1687 networks," in *Test Symposium (ATS), 2015 IEEE 24th Asian*. IEEE, 2015, pp. 211–216.

[12] R. Cantoro, M. Palena, P. Pasini, and M. Sonza Reorda, "Test time minimization in reconfigurable scan networks," in *Asian Test Symposium (ATS), 2016 IEEE 25th*. IEEE, 2016, pp. 119–124.

[13] R. Cantoro, L. San Paolo, M. Sonza Reorda, and G. Squillero, "New techniques for reducing the duration of reconfigurable scan network test," in *Design and Diagnostics of Electronic Circuits & Systems (DDECS), 2018 IEEE 21th International Symposium on*. IEEE, 2018 (to appear).

[14] R. Cantoro, A. Damljanovic, M. Sonza Reorda, and G. Squillero, "A semi-formal technique to generate effective test sequences for reconfigurable scan networks," in *Test Conference in Asia (ITC-Asia), 2017 International*. IEEE, 2018 (to appear).

[15] A. Tšertov, A. Jutman, S. Devadze, M. S. Reorda, E. Larsson, F. G. Zadegan, R. Cantoro, M. Montazeri, and R. Krenz-Baath, "A suite of IEEE 1687 benchmark networks," in *Test Conference (ITC), 2016 IEEE International*. IEEE, 2016, pp. 1–10.

[16] R. Cantoro, F. G. Zadegan, M. Palena, P. Pasini, E. Larsson, and M. S. Reorda, "Test of reconfigurable modules in scan networks," *IEEE Transactions on Computers*, 2018.

[17] M. A. Kochte, R. Baranowski, M. Sauer, B. Becker, and H.-J. Wunderlich, "Formal verification of secure reconfigurable scan network infrastructure," in *Test Symposium (ETS), 2016 21th IEEE European*. IEEE, 2016, pp. 1–6.

[18] A. Tšertov, A. Jutman, S. Devadze, M. Sonza Reorda, E. Larsson, F. G. Zadegan, R. Cantoro, M. Montazeri, and R. Krenz-Baath, "A suite of IEEE 1687 benchmark networks," in *Test Conference (ITC), 2016 IEEE International*. IEEE, 2016, pp. 1–10.