A Semi-Formal Technique to Generate Effective Test Sequences for Reconfigurable Scan Networks

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

***Abstract*—The broad need to efficiently access all the instrumentation embedded within a semiconductor device called for a standardization, and the reconfigurable scan networks proposed in IEEE 1687 have been demonstrated effective in handling complex infrastructures. At the same time, different techniques have been proposed to test the new circuitry required; however, most of the automatic approaches are either too computationally demanding to be applied in complex cases, or too approximate to yield high-quality tests. This paper models the state of a reconfigurable scan network with a finite state automaton, using the length of the active path as the output alphabet and the configurations as input symbols. Permanent faults are represented as incorrect transitions, and a greedy algorithm is used to generate a functional test sequence able to detect all these multiple state-transition faults. The automaton's state set and the input alphabet are small subsets of the possible ones, and are carefully chosen. Experimental results on ITC'16 benchmarks demonstrate that the proposed approach is broadly applicable; the test sequences are more efficient than the ones previously generated by search heuristics.**

I. Introduction

Most of the Integrated Circuits (ICs) designed in the last years include several resources whose purpose is not to support the circuit functionality, but are rather related to ancillary aspects 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 (e.g., BIST ones) also require registers to activate/initialize the test and retrieve results. In order to effectively access all these registers (also called instruments, or Test Data Registers), companies first included all these registers into a single chain, often accessed through the standard IEEE 1149.1 interface. Due to the continuous growth in the length of such a chain, they then started devising solutions based on the so called *Reconfigurable Scan Networks* (RSNs) [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.

The typical RSNs are 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 an 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]. Testing an RSN is more complex, as testing must also check whether the network can be properly configured and whether it works as expected after enforcing whichever legal configuration.

In [9] 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 provided techniques for their test, and then described how to combine them into a single comprehensive test. This test is independent on the specific implementation of the network elements and does not require any change in the hardware implementing the network. Test generation can start directly from the network structure. The test generation algorithms proposed in [9] were based on heuristics that could easily run even on relatively large RSNs.

In [10] 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 [11] 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.

In this paper we make one step forward, and propose a new method which is able to deal with larger and more complex RSNs and can produce 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 generated by the heuristic solutions proposed in [9], with a more limited computational effort with respect to the method in [11]. Moreover, the approach proposed in this paper is also able to take into account the effects of faults on the RSN configuration phase. The proposed method is based on a semi-formal approach: a Finite State Automaton is dynamically built and used to generate the minimum length test sequence able to detect permanent faults in the target RSN. Experimental results are reported on the set of benchmark networks proposed in [12], which practically demonstrate that the proposed approach is able to identify significantly shorter test sequences than those produced by the methods described in [9] and [11], 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 method first introduced in [9], 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. This is done by incorporating programmable components into the network structure. Configuration of these components 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. One such programmable component is the SIB module introduced by IEEE 1687, which allows for bypassing a segment of a network. Therefore, it is possible to create a hierarchical network with the use of SIBs.

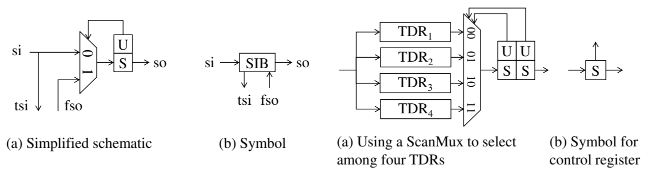

Fig. 1. Segment Insertion Bit (SIB) and ScanMux (SM) modules.

Fig. 1(a) 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. 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(b) is used to represent a SIB.

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. 1(c) in which a two-bit shift-update register is used to select among four inputs of a 4-to-1 SM. In the rest of this paper, the symbol shown in Fig. 1(d) will be used to represent the shift-update register that controls an SM.

To operate an IEEE 1687 network from outside the chip, the IEEE 1149.1 TAP can be used. The TAP finite state machine provides the control signals needed to configure the IEEE 1687 network and access the instruments through it.

Let us now consider a simple RSN example, corresponding to a circuit which includes five instruments: the user can access them through the TAP port, reading or writing from/to the associated Test Data Registers (TDR_1 to TDR_5). In order to save time when accessing to the instruments, the designer, instead of connecting all TDRs into a single chain, like in 1149.1-complaint circuits, may decide to adopt an IEEE 1687 network including three SIBs and one SM, as shown in Fig. 2; each of these four configuration modules can be configured to allow access to a given subset of TDRs (and the associated instruments). Table I reports the eight possible configurations supported by this network, which depend on how the SIBs and the SM have been configured. In Table I, "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 [9]. The faults affecting the reconfigurable modules, such as

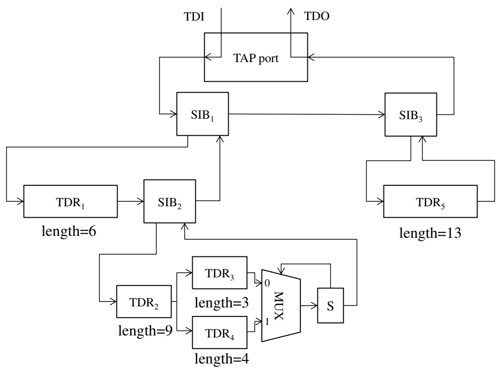

Fig. 2. Example of an IEEE 1687 RSN.

TABLE I
SET OF POSSIBLE CONFIGURATIONS OF THE RSN IN FIG. 2.

Config.	SIB_1	SIB_2	SIB_3	SM	Active path	Len.
C_0	D	D	D	0	-	2
C_1	D	D	D	1	-	2
C_4	D	A	D	0	-	2
C_5	D	A	D	1	-	2
C_2	D	D	A	0	TDR_5	15
C_3	D	D	A	1	TDR_5	15
C_6	D	A	A	0	TDR_5	15
C_7	D	A	A	1	TDR_5	15
C_8	A	D	D	0	TDR_1	9
C_9	A	D	D	1	TDR_1	9
C_{10}	A	D	A	0	TDR_1, TDR_5	14
C_{11}	A	D	A	1	TDR_1, TDR_5	14
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	35
C_{15}	A	A	A	1	TDR_1, TDR_2, TDR_4, TDR_5	36

SMs, are modeled such that a different configuration may be selected rather than the expected one, and could lead to a path with different length. For example, in Fig. 2 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 [13].

According to this high-level fault model, one can test an RSN by first configuring the RSN so that the target fault is excited, and then comparing the length of the activated path against the length of the expected path. As an example, the high-level fault stucking the SM of Fig. 2 to 1, 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 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. 2 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 test vector in such a way that the sequence terminator is 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.

III. PROPOSED APPROACH

In the proposed approach, the RSN of IEEE 1687 is modeled as a finite state automaton (FSA). 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 network, as this is an easily observable characteristic [9]. 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* 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 fault 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_s times the size of the input alphabet $\mathbf{A}_\mathrm{in}$, that is $\mathcal{O}(n_\mathrm{s} \cdot |\mathbf{A}_\mathrm{in}|)$. As both terms depend linearly on the number of configuration bits n_cb, the complexity is definitely smaller than the A* algorithm presented in [10], 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 zero, 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 chains 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. 3 is only accessible when SIB_1 is asserted. Therefore, the FSA would include the reset state $(\overline{\mathrm{SIB}_1}, \overline{\mathrm{SIB}_2}, \overline{\mathrm{SIB}_3})$; the three states with only one SIB asserted { $(\mathrm{SIB}_1, \overline{\mathrm{SIB}_2}, \overline{\mathrm{SIB}_3})$, $(\overline{\mathrm{SIB}_1}, \mathrm{SIB}_2, \overline{\mathrm{SIB}_3})$, $(\overline{\mathrm{SIB}_1}, \overline{\mathrm{SIB}_2}, \mathrm{SIB}_3)$ }; and the state $(\mathrm{SIB}_1, \mathrm{SIB}_2, \overline{\mathrm{SIB}_3})$, as asserting SIB_1 is necessary to test SIB_2.

Then, for each transition in the good automaton, the possible faulty transition 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 transition 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. Experimental evaluations indicate that such extensions are not quite beneficial, but the designers may explicitly add relevant states to this state or provide an additional heuristic.

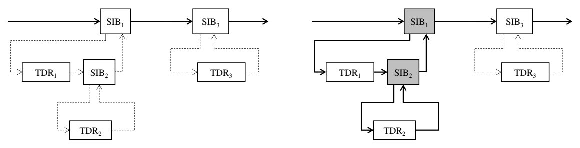

Fig. 3. Example of generating input symbols for SIB RSN.

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 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. A sequence $\mathbf{t}$ of inputs $\mathbf{t} = (\mathbf{reset}, x_0, x_1, ..., x_i)$ unequivocally defines the state of the FSA. Let $\bar{s}_{\mathbf{t}}$ be the state of the FSA representing the fault-free circuit after the application of the input sequence $\mathbf{t}$, and let $\mathbf{S_t} = \{s^0_{\mathbf{t}}, s^1_{\mathbf{t}}, ..., s^{n-1}_{\mathbf{t}}\}$ be the set of the states of the FSA representing the n faulty circuits. $\mathbf{S_t}$ depends on the full sequence $\mathbf{t}$, and some faulty circuits may be in the correct state, thus $\bar{s}_{\mathbf{t}} \in \mathbf{S_t}$. It is possible that a fault also effects the reset state, while such a possibility is easily tractable by the proposed methodology, it was not considered in this work.

Let $\mathrm{DF}(\bar{s}, \mathbf{F})$ be the set of potentially detectable faults when the good machine is in state $\bar{s}$ and the faulty ones in $\mathbf{F} = (s^0, s^1, ..., s^f)$, that is, the set of all faults that caused the faulty machine to be in a state s^i with an output symbol different from $\bar{s}$. If an observation is performed, measuring the actual length of the RSN, any difference would be observed and all such faults, detected.

Given a sequence of inputs $\mathbf{t}$, the function GREEDY extends it with the most promising input symbol (Fig. 4). That is, it appends the input symbol that brings the FSA where the highest number of faults could be detected. If no new fault can be detected by adding a single transition, the function returns an empty input sequence.

The search algorithm incrementally builds the test sequence $\mathbf{t}$ calling the function GREEDY iteratively (Fig. 5). In every

Fig. 4. Greedy step

```
function GREEDY($\mathbf{t}$)
    $\mathbf{m} \leftarrow (\ )$                              ▷ Empty sequence of inputs
    for $x \in$ {valid input symbols in $\bar{s}_{\mathbf{t}}$} do
        $\mathbf{u} \leftarrow \mathbf{t}$
        Append $x$ to $\mathbf{u}$
        if $|\mathrm{DF}(\bar{s}_{\mathbf{u}}, \mathbf{S_u})| > |\mathrm{DF}(\bar{s}_{\mathbf{m}}, \mathbf{S_m})|$ then
            $\mathbf{m} \leftarrow \mathbf{u}$
    return $\mathbf{m}$                            ▷ Most promising sequence
```

Fig. 5. Test Sequence Generation

```
procedure TPG
    $\mathbf{t} \leftarrow (\mathbf{reset})$                         ▷ Initial test sequence
    $\mathbf{H} \leftarrow \{\mathbf{t}\}$                                  ▷ History
    $\mathbf{F} \leftarrow \{\text{all detectable faults}\}$             ▷ Active faults
    while $|\mathbf{F}| \neq 0$ do
        $\mathbf{g} \leftarrow \text{Greedy}(\mathbf{t})$
        if $\text{empty}(\mathbf{g})$ then                    ▷ The greedy failed
            Append $\mathbf{reset}$ to $\mathbf{t}$                        ▷ Start over
            for $\mathbf{t}' \in \mathbf{H}$ do
                $\mathbf{g}' \leftarrow \text{Greedy}(\mathbf{t}')$
                if $|\text{DF}(\bar{s}_{\mathbf{g}'}, \mathbf{S}_{\mathbf{g}'})| > |\text{DF}(\bar{s}_{\mathbf{g}}, \mathbf{S}_{\mathbf{g}})|$ then
                    $\mathbf{g} \leftarrow \mathbf{g}'$                 ▷ Alternative sequence
        Append $\mathbf{g}$ to $\mathbf{t}$
        Append $\mathbf{observe}$ to $\mathbf{t}$
        $\mathbf{H} \leftarrow \mathbf{H} \cup \{\mathbf{g}\}$                       ▷ Save sequence
        Remove $\text{DF}(\bar{s}_{\mathbf{g}}, \mathbf{S}_{\mathbf{g}})$ from $\mathbf{F}$
```

step, the most useful symbol is appended to the test sequence; however, if it is not possible to detect new faults by adding a single symbol, the FSA is rolled back to a previous state where a useful input symbol may be found.

The symbol **observe** is used to denote an observation operation in the test sequence. Although it has no effect on the FSA, its cost in terms of clock cycles needs to be considered.

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 two alternative approaches. The first approach has been derived from [9] and is referred to as *depth-first* in this paper. The approach is based on the exploration of the network topology performing a depth-first search of active faults in the network. The second approach has been proposed in [11] 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.

Basic information about the networks used for evaluation are reported in Table II. For each network, the table reports the number of SIBs (column 2) and SMs (column 3). The fourth column refers to the number of configuration bits of SIBs and SMs. 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 [14]).

Experiments were run using a tool written in Java. The tool supports network structure extraction from files in different formats including ICL. The experiments were run on a laptop equipped with an Intel i5-480M processor. Experimental results are reported in Table III. The table shows the number of configuration vectors cv (column 2) and test vectors tv (column 3) generated by the tool. Moreover, the table reports the number of clock cycles required to configure the network (column 4) and the number of clock cycles needed to apply test vectors (column 5).

The number of clock cycles required for every configuration phase, which includes the length of the currently active path, has been increased by five (JTAG overhead) [11]. In addition, the same overhead has been taken into account for calculating the cost of 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.

Experimental comparison of the proposed approach against the depth-first and evolutionary approaches is shown in Table IV. Data concerning evolutionary approach are taken from [11]. For the depth-first approach, data have been newly generated on the ITC16 benchmarks by running the tool implementing the same algorithm as in [9]. For each algorithm, Table IV reports the duration in clock cycles of the generated

TABLE II
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	23	4	88,471	108,148
NE600P150	207	194	46	12	23,423	28,250

TABLE III
IEEE 1687 TEST ALGORITHM EXPERIMENTAL RESULTS

Network	cv	tv	Conf. time [cc]	Test time [cc]
Mingle	6	7	330	1,684
TreeBal.	7	8	7,325	56,518
TreeFlat_Ex	5	6	5,398	36,485
TreeUnbal.	11	12	86,651	632,724
a586710	4	5	764	296,032
p22810	2	3	573	151,826
p34392	4	5	410	195,144
p93791	4	5	1,291	704,951
q12710	2	3	43	130,979
t512505	2	3	494	384,946
N132D4	5	6	7,170	24,825
N17D3	4	5	758	3,007
N32D6	4	5	147,162	669,472
N73D14	12	13	868,652	3,508,797
NE1200P430	127	128	1,710,412	13,084,445
NE600P150	78	79	413,697	2,280,975

TABLE IV
EXPERIMENTAL RESULTS COMPARISON

Network	Total test time [cc]					Runtime (wall clock)		
	FSA	Depth-first	FSA vs. d-f	Evolutionary	FSA vs. evol.	FSA	Depth-first	Evolutionary
Mingle	2,014	2,282	13.31%	2,078	3.18%	26s	1s	8h
TreeBalanced	63,843	69,369	8.66%	69,369	8.66%	48s	1s	19h
TreeFlat Ex	41,883	71,341	70.33%	55,776	33.17%	71s	1s	8h
TreeUnbalanced	719,375	1,071,799	48.99%	1,042,450	44.91%	34s	1s	5h
a586710	296,796	299,624	0.95%	298,241	0.49%	39s	1s	8h
p22810	152,399	152,937	0.35%	152,937	0.35%	39s	1s	9h
p34392	195,554	196,702	0.59%	196,505	0.49%	1m	1s	7h
p93791	706,242	708,878	0.37%	708,878	0.37%	2m	1s	27h
q12710	131,022	131,022	0.00%	131,022	0.00%	46s	1s	5h
t512505	385,440	386,024	0.15%	386,024	0.15%	50s	1s	8h
N132D4	31,995	38,731	21.05%	37,257	16.45%	2s	1s	3h
N17D3	3,765	4,143	10.04%	3,851	2.28%	1s	1s	5h
N32D6	816,634	942,470	15.41%	893,017	9.35%	6s	1s	5h
N73D14	4,377,449	5,978,047	36.56%	5,967,137	36.32%	97s	3s	3h
NE1200P430	14,794,857	21,515,705	45.43%	21,515,705	45.43%	1h	3s	50h
NE600P150	2,694,672	3,726,726	38.30%	3,726,726	38.30%	4m	3s	12h

test sequence (referred to as *Total test time*) and the CPU time required to apply the algorithm (referred to as *Runtime*).

Remarkably, results in Table IV show an improvement in terms of test time reduction up to 70% in comparison to depth-first, and up to 45% with respect to the evolutionary method. Moreover, the test sequence generated by the proposed approach is shorter than the sequences obtained by the other algorithms in all networks but one (i.e., q12710); in such a network, the three algorithms produce test sequences having the same duration.

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 one hour 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 evolutionary approach was run 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 the IEEE 1687 RSNs. The approach can be defined as a 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 heuristics. 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. We are currently performing further experiments to better understand the limitations of the proposed method.

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. 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.

[10] 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.

[11] 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 (submitted).

[12] 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.

[13] 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.

[14] 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.