Test of Reconfigurable Modules in Scan Networks

Test of Reconfigurable Modules in Scan Networks

Riccardo Cantoro, *Member, IEEE,* Farrokh Ghani Zadegan, *Member, IEEE,* Marco Palena, *Member, IEEE,* Paolo Pasini, *Member, IEEE,* Erik Larsson, *Senior Member, IEEE,* and Matteo Sonza Reorda, *Fellow, IEEE*

Abstract—Modern devices often include several embedded instruments, such as BIST interfaces, sensors, calibration facilities. New standards, such as IEEE Std 1687, provide vehicles to access these instruments. In approaches based on reconfigurable scan networks (RSNs), instruments are coupled with scan registers, connected into chains and interleaved with reconfigurable modules. Such modules embed reconfigurable multiplexers that permit a selective access to different parts of the chain. A similar scenario is also supported by IEEE Std 1149.1-2013. The test of permanent faults affecting an RSN requires to shift test vectors throughout a certain number of network configurations. This paper presents some methodologies to select the list of configurations that perform the complete test of the reconfigurable modules of the RSN. In particular, one method is presented that, by construction, can be proved to be able to apply the test in the minimum amount of clock cycles. Other methods are sub-optimal in terms of test application time (TAT), but scale well on large circuits. In order to provide a comparison between the proposed methods, experimental results on some benchmark RSNs are provided.

Index Terms—Test, Reconfigurable Scan Networks, IEEE Std 1687, A^* Algorithm.

1 Introduction

Due to the complexity of new electronic devices, several features are embedded in such systems beside the core functional logic. Examples of such features are Built-In Self-Test (BIST), included for test and diagnostic sake, interfaces to core testing (e.g., based on the IEEE Std 1500), analog components (e.g., temperature sensors used to monitor the device behavior, oscillators) accessed during the chip calibration, or debug related components (e.g., trace buffers). These features are hereinafter called *instruments*. Creating a mechanism to access instruments has led to many different legacy solutions, facing the complex task of integrating all of them in the system, especially when they come from different designers. In order to mitigate these issues, new standards have been created.

IEEE Std 1149.1-2013 and IEEE Std 1687 have standardized the concept of *reconfigurable scan chains*, which were proposed for various purposes in several previous papers, such as [1]. This kind of chains are segmented in several parts, hereinafter referred to as *segments*, which are interleaved with special elements, hereinafter referred to as *reconfigurable modules*. Each segment can include one or more instruments. The interface with an instrument is the *test data register* (TDR), which can include capture logic (in case of reading capability) and update logic (in the case when writing is allowed). According to the configuration of reconfigurable modules, certain segments are connected together in the so called *active path*, i.e., the path connected between the scan input and scan output pins of the reconfigurable scan chain at a given time. Since the complexity of these reconfigurable scan chains can be high (i.e., many possible active paths may exist), the standards refer to them as *networks*.

A reconfigurable scan network (RSN) can be configured to access a certain instrument. A proper test must check whether the RSN can be moved in all legal configurations and whether it works as expected in any of them (i.e., when a certain set of segments is made part of the active path). The test of reconfigurable modules must be complemented with the test of the remaining components of the RSN, which can resort to both traditional scan test methodologies (e.g., [2], [3], [4]) and new techniques targeting specific defective behaviors (e.g., bridging faults may introduce interaction between instruments, TDRs, and reconfigurable modules).

This paper deals with the test of permanent faults affecting reconfigurable modules, which are not considered during the test of standard scan chains. Other parts of the RSN are not considered in this article, such as the interface with instruments (i.e., the capture and update logic of TDRs), and TDR flip-flops. We assume that faults affecting those parts can be detected by applying suitable functional stimuli to the instruments at the end of the proposed test flow. The main focus of this article is not to improve state-of-the-art scan cell testing techniques but to support minimum-length test of reconfigurable modules. More specifically, the paper focuses first on proposing algorithms to automatically generate a test sequence able to detect possible faults affecting the reconfigurable modules. Since the cost for the test heavily depends on its duration, the paper analyses the length of the solutions generated by the proposed algorithms. Not surprisingly, generating the minimum-length solution

- *R. Cantoro, M. Palena, P. Pasini, and M. Sonza Reorda are with the Department of Control and Computer Engineering, Politecnico di Torino, Turin, Italy.*
  E-mail: riccardo.cantoro@polito.it, marco.palena@polito.it, paolo.pasini@polito.it, matteo.sonzareorda@polito.it
- *F. Ghani Zadegan and E. Larsson are with the Department of Electrical and Information Technology, Lund University, Lund, Sweden.*
  E-mail: farrokh.ghani_zadegan@eit.lth.se, erik.larsson@eit.lth.se

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2018.2834915, IEEE Transactions on Computers

requires very computationally expensive algorithms. Hence, we propose solutions trading-off the length of the generated test sequence with the computation cost for generating it. The basic idea presented by the authors in [5] and [6] is extended in this article with a detailed description of the proposed algorithms and more extensively supported by experimental results.

Since it is common in the industrial practice to start the test generation activities early in the design process, when high-level descriptions are available only, we decided to develop solutions targeting a high-level fault model for the reconfigurable modules. Moreover, in this way we can simplify the test generation process and make the algorithms more general and independent on the specific implementation, while we demonstrate that by using the proposed high-level fault model we can detect the majority of faults detectable by lower-level fault models. To summarize, given an RSN to test, the application of the proposed method results in a list of configurations that cover all targeted faults while minimizing the *test application time* (TAT).

The proposed methodology has been applied to a subset of the ITC'16 benchmark networks [7] and evaluated with respect to the duration of the test. Moreover, the proposed approaches are compared also on a set of synthetic networks, showing that the minimum TAT can be reached on small networks, while test sets for large networks can be generated with sub-optimal algorithms that are very efficient in terms of CPU time and memory requirements.

The rest of the paper is organized as follows. Section 2 presents the background of the research work on RSNs. The basic notions about network testability, fault model, and test vectors are presented in Section 3. In the proposed work, the problem is represented using graphs, as shown in Section 4. Test approaches using these graphs are presented in Section 5. Some experimental results are reported in Section 6. Finally, Section 7 draws some conclusions.

2 Background

This section introduces the main elements composing RSNs, such as TDRs and standardized reconfigurable elements.

TDRs have been introduced by IEEE Std 1149.1 and are composed of a number of bits, each bit containing a capture/scan cell (C) and an optional update cell (U). By acting on the *test access port* (TAP) controller, TDRs can be selected. In the latest revision of the standard (i.e., IEEE Std 1149-2013 [8]), each TDR can be constructed as a chain of multiple segments, some of which are always scanned while others, called *excludable segments* (see Fig. 1) and *selectable segments* (see Fig. 2), are scanned only in particular situations. Each of those segments is controlled by a configuration module composed of one or more bits (i.e., a C cell and a U cell), which act on a selection circuit (e.g., a multiplexer).

The new IEEE Std 1687 [9] tackles the same problem of accessing instruments in different ways, including RSNs. The stardard defines the so called *Segment Insertion Bit* (SIB), which is similar in concept to an excludable segment. Selectable segments can be implemented by means of scan multiplexer (ScanMux) modules. When a SIB is said to be *asserted*, the segment it controls is included in the active path; otherwise, it is said to be *de-asserted*. Each segment

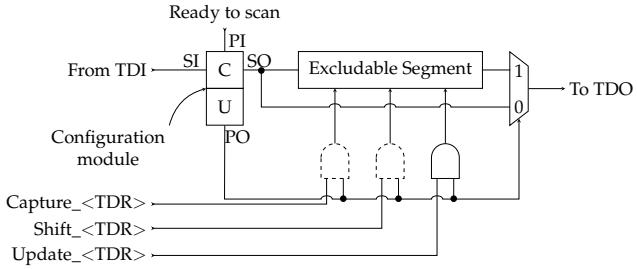


Fig. 1: Excludable TDR segment (IEEE Std 1149.1-2013).

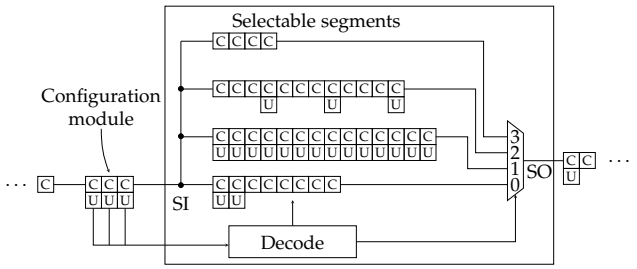


Fig. 2: Selectable TDR segments (IEEE Std 1149.1-2013).

controlled by a SIB or a ScanMux can be a complex network itself.

In order to bring an RSN into a certain configuration, vectors have to be shifted through the scan input port. Then, an update operation moves the vector from the shift flip-flops (C cells) to the update latches (U cells) of the configuration module. This operation changes the active path of the network. Since an RSN can have a hierarchical structure, the operation of making an instrument, placed deep into the network, part of the active path may require multiple configuration phases.

As an example, let us consider a simple IEEE Std 1687 RSN reported in Fig. 3. The network is accessed through an IEEE Std 1149 TAP interface and is composed of two selectable segments: the first one with a single TDR, and the other one with two TDRs, each one controlled by a SIB. In Fig. 3, we also report the bit length of each TDR. The SIB modules and the multiplexer (ScanMux) are associated each one with a configuration bit (cb_1, cb_2, and cb_3) and are highlighted in grey. Depending on the configuration (i.e., the value of the configuration bits of SIBs and ScanMux), the network presents one of five possible active paths, each one including different subsets of TDRs, as listed in Table 1. In this table, 'A' means the SIB is in the asserted position, 'D' means de-asserted, 0 and 1 correspond to the two possible positions of the ScanMux, and 'X' appears when a module belongs to an inaccessible segment (i.e., don't care value). During the system reset, a known configuration is selected. The status of the reconfigurable modules upon reset determines the network *reset configuration*. In the example network, we assume the reset configuration is 0,D,D (using the same module ordering of Table 1).

Testing a standard (non-reconfigurable) scan chain for permanent faults has been a widely studied subject for years. Several techniques exists, e.g., shifting a suitable sequence of 0s and 1s through the scan chain, such as the sequence "00110011" that applies all possible transitions in two cycles [2]. In order to cover scan cells internal defects

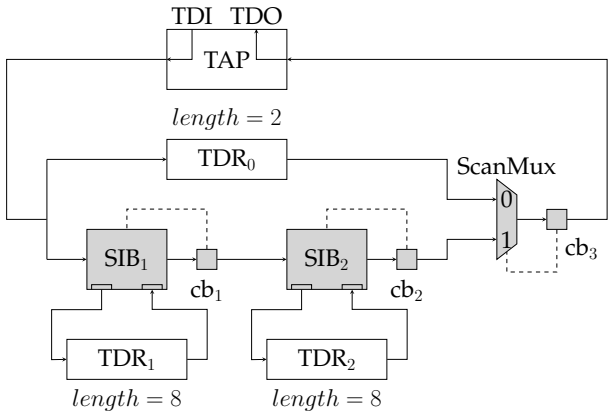


Fig. 3: Example of IEEE Std 1687 RSN.

TABLE 1: Possible configurations for the network in Fig. 3.

ScanMux	SIB1	SIB2	Scan length	Active path
0	X	X	3	TDI→TDR$_0$→cb$_3$→TDO
1	D	D	3	TDI→cb$_1$→cb$_2$→cb$_3$→TDO
1	D	A	11	TDI→cb$_1$→TDR$_2$→cb$_2$→cb$_3$→TDO
1	A	D	11	TDI→TDR$_1$→cb$_1$→cb$_2$→cb$_3$→TDO
1	A	A	19	TDI→TDR$_1$→cb$_1$→ TDR$_2$→cb$_2$→cb$_3$→TDO

(see [10]), the previous sequence is not enough, but has to be enhanced with additional tests, e.g, for stuck-open faults [11], [12], or for bridging faults [13]. Diagnosis of intermittent faults in scan-chains has been discussed in [14].

Due to the recent adoption of the IEEE Std 1687 by commercial tools, several issues have arisen regarding design (e.g., in [15], [16], [17]), validation (e.g., in [18], [19]), and test (e.g., in [20], [21]) of RSNs, as well as their usage in the field. The works in [20], [21] have tested structural faults and investigated the test quality of different test strategies. The combination of the evaluated test strategies achieves high fault coverage even in synthetic difficult to test circuits. The works in [20], [21] aim at maximizing the structural fault coverage but do not target the minimization of the TAT, while our article evaluates the TAT of algorithms aiming for high fault coverage, according to a high-level fault model.

3 Terminology and Fault Model

In this section, the basic terminology introduced in this article is presented. Moreover, in order to generalize as much as possible the problem of testing the reconfigurable modules in an RSN, a high-level fault model is introduced and specified for each type of reconfigurable modules.

3.1 Configurations, Vectors, and Test Application Time

A generic reconfigurable module, hereinafter indicated with M_i, is able to control (or *select*) parts of the network defined as *segments*, according to the values stored in its configuration bits. The segment $s_{i,k}$ is defined as the sub-network attached to the k-th input pin of M_i. In Fig. 3, each SIB controls a segment that includes a TDR, while ScanMux controls two segments: one including TDR$_0$, and the other including the two SIBs and their selectable segments. In this article, we associate each element of the network to the most specific segment possible. For example, TDR$_1$ lies in the segment controlled by SIB$_1$, while cb$_1$ is in the segment controlled by ScanMux. The *depth* of a certain element of the network is defined as the hierarchical level of its segment. In Fig. 3, ScanMux and cb$_3$ have depth 1 (they are placed on the top-level segment), TDR$_0$, SIB$_1$, SIB$_2$, cb$_1$ and cb$_2$ have depth 2, and TDR$_1$ and TDR$_2$ have depth 3. The length of a segment is equal to the number of bits it includes in the selected path. Hence, if the segment includes reconfigurable modules, their configuration affects the length of the enclosing segment.

A generic configuration of the network (i.e., the value of all configuration bits) is referred to as σ_i. The term σ_0 indicates the reset configuration. Each σ_i can be associated to an active path length and a list of possible faults (each referred to as F_i) affecting the network, that can be detected by performing test operations while the network is configured with σ_i.

Such test operations use *test vectors* to verify whether the expected path has been inserted between the scan input and scan output pins, i.e., whether the right instruments can be accessed during the normal operation. The test operations associated to a generic test vector tv_i correspond to:

1) a suitable sequence (as long as the active path length) is shifted in, forcing it to travel along the active path and to appear on its other end;
2) scan output pins (e.g., TDO) are monitored: the sequence previously loaded is expected to come out; based on the fact that the observed sequence matches the expected one or not, possible faults can be detected.

A network *transition* is defined as a change in the configuration, by means of one or more *configuration vectors*. The operations associated to a generic configuration vector cv_i correspond to:

1) as many shift operations as the active path length, so that the next configuration is stored in the C flip-flops of the reconfigurable modules' configuration bits, while the other bits are don't care ('X' in this article);
2) an update operation, so that the next configuration is applied to the network and the active path changes.

If transitioning from the configuration σ_i to σ_j requires a single configuration vector, then σ_j is a *neighbor* configuration of σ_i. In this case, the *transition cost* in terms of clock cycles is equal to the active path length of σ_i plus one (the update operation). We denote by Σ_i the list of neighbors of σ_i. Please note that the neighborhood relation is not reversible. For example, the RSN in Fig. 3 can be moved from $\sigma_1 = \{1, A, A\}$ (see Table 1) to $\sigma_2 = \{0, D, D\}$ by shifting a single vector, while two vectors are needed to reach σ_1 from σ_2, passing through $\sigma_3 = \{1, D, D\}$.

Configuration and test vectors are used by the proposed test techniques and organized in *sessions*. A generic session, referred to as S_i, is composed of two phases:

1) a *configuration phase* (*Cfg*), corresponding to a network transition, in which a certain number of configuration vectors are applied, until the target configuration is reached;
2) a *test phase* (*Tst*), in which test vectors are applied.

The sequence of test vectors to be used in the test phase depends on the kind of defects to be tested.

Considering the generic session S_i, we denote by t_i^c the duration (in clock cycles) of the configuration phase Cfg_i and by t_i^t the duration of the test phase Tst_i. Each configuration vector of the session requires a certain time to be shifted in, plus a few clock cycles (their exact number is implementation dependent) to update it into the U cells of the corresponding path (this time is denoted as *JTAG protocol overhead* in [22]). The active path changes after each update operation, thus each vector may have a different length. The duration of the test phase (t_i^t) depends on the active path length l of the target configuration (i.e., after the last configuration vector). The test application time for a network that needs N sessions to cover each testable fault is thus given by:

$$TAT = T_c + T_t = \sum_{i=1}^{N} t_i^c + \sum_{i=1}^{N} t_i^t \quad (1)$$

where T_c is the sum of clock cycles of each Cnf_i and T_t is the sum of the clock cycles of each Tst_i. Hereinafter, the terms T_c, T_t, and TAT are all measured in clock cycles. Such terms should be clearly distinguished from the CPU time (expressed in seconds) required to generate the stimuli to be used in all N sessions.

During test generation, a *fault list* is used, which is composed of all possible faults affecting the network, according to the proposed fault model. The fault list includes an indication for each fault, hereinafter indicated with F_i, about whether F_i is tested, still untested, or untestable in any possible network configuration.

3.2 Fault Model for Reconfigurable Modules

The fault model used in this article has been derived from the analysis of possible stuck-at faults affecting the structural implementation of reconfigurable modules. The effect of such faults, observed in simulation, is that the scan output pin of the reconfigurable module produces:

1) a sequence of known values (i.e., all 0s or all 1s), due to stuck-at faults affecting the input pins of a multiplexer;
2) a sequence of unknown values, due to unconnected pins;
3) values from a segment not selected by the current configuration, due to stuck-at faults affecting the shift flip-flops (C cells) of the configuration bits, the selection logic of a multiplexer, or the update logic.

We consider the first two cases as easy-to-test, since they drastically affect the output values, while the last case has been better analyzed and a high-level fault model has been derived. We have verified with fault simulation experiments that the high-level fault model matches well with the stuck-at fault model on the synthesized design, when suitable test vectors are used. Faults affecting reset and enable logic have not been taken into account (the work in [21] provides solutions to this limitation).

A proper test for high-level faults of a reconfigurable module is composed of the following operations:

1) The network is configured so that the faulty element becomes part of the active path. This operation is needed to excite the fault, i.e., to create a difference with respect to the fault-free scenario. In the case of reconfigurable modules, this difference is the length of the active path.
2) A proper sequence is shifted into the network, while the expected path length is compared against the length of the faulty path. This comparison is performed by looking at the number of clock cycles required by the input sequence to appear on the scan output pin.

TABLE 2: Effect of the high-level fault on the ScanMux of Fig. 3, which always selects the input 1, when selecting different active paths.

ScanMux	SIB1	SIB2	Path length (faulty/active)	Faulty path
0	D	D	3/3	TDI→cb_1→cb_2→cb_3→TDO
0	D	A	11/3	TDI→cb_1→TDR_2→cb_2→cb_3→TDO
0	A	D	11/3	TDI→TDR_1→cb_1→cb_2→cb_3→TDO
0	A	A	19/3	TDI→TDR_1→cb_1→ TDR_2→cb_2→cb_3→TDO

As an example, the high-level fault on the module ScanMux of Fig. 3, which always selects the segment connected to the input 1, can be excited by a configuration which selects the input 0; all such possible configurations are listed in Table 2, which also reports the length of the selected faulty paths. In the table, the first configuration is not able to detect the fault, due to the fact that the faulty path length is equal to the active path length. Thus, one of the remaining three configurations can be selected for the test.

Some situations may exist, in which all faulty paths have the same length of the active path, i.e., no configuration able to test the fault exists. In this case, the fault is *untestable*.

In the following subsections, the basic concepts presented above are applied in a test procedure for SIB modules and ScanMuxes. Configuration and test vectors are applied in the test procedure. Since each test vector aims at checking whether the active path is as long as expected, we include an *initialization vector* to the test phase, which forces the scan paths to a known value. A sequence of 0s can be used for this purpose, whose length is equal to the longest path in the network. In this case, each t_i^t contribution in Eq. (1) includes the length of the longest path. In the example of Fig. 3, an initialization vector composed of 19 0s can be used before each test phase.

3.2.1 SIBs

Given a SIB, the test procedure for testing both the SIB stuck-at asserted (stuck-at-A) and de-asserted (stuck-at-D) faults is the following:

1) configure the network, so that the target SIB becomes part of the active path;
2) shift in an initialization vector whose length is equal to the one of the longest path in the network;
3) shift in a test vector as long as the expected path length;
4) check whether the expected sequence appears on the output of the path;
5) reconfigure the network, so that the SIB is part of the active path and at the opposite configuration;
6) shift in an initialization vector whose length is equal to the one of the longest path in the network;
7) shift in a test vector whose length is equal to the one of the expected path length;
8) check whether the expected sequence appears on the output of the path.

As an example, the test of SIB_1 in Fig. 3 is presented, assuming that in the reset configuration the SIBs are de-

asserted and the ScanMux selects the input 0. As test vectors, a sequence of alternated 0s and 1s is used, followed by two consecutive 1s, which are used as sequence terminator. The output pin is monitored until the sequence terminator is shifted out: this permits to calculate the active path length and to compare it against the expected one. In details:

1) Reset – active path (AP): TDI→TDR_0 →cb_3→TDO
2) Apply configuration vector 1:
   a) Shift in `001` (lenght = 3)
   b) Update – AP: TDI→cb_1 →cb_2 →cb_3→TDO
3) Apply initialization vector 1:
   a) Shift in `00000000000000000000` (lenght = 19)
4) Apply test vector 1:
   a) Shift in `010` (lenght = 3)
   b) Shift in `11` (lenght = 2)
5) Check test vector 1, while applying config. vector 2:
   a) Shift in `101` (lenght = 3)
   b) Update – AP: TDI→TDR_1 →cb_1 →cb_2 →cb_3→TDO
6) Apply initialization vector 2:
   a) Shift in `00000000000000000000` (lenght = 19)
7) Apply test vector 2:
   a) Shift in `01010101010` (lenght = 11)
   b) Shift in `11` (lenght = 2)
8) Check test vector 2.

In the last check (step 8), extra bits are shifted in until the last sequence terminator comes out from the output pin (at maximum, as long as the longest path plus the length of the sequence terminator, i.e., $19 + 2 = 21$ in the example).

3.2.2 Scan Multiplexers

The same test procedure can be extended to scan multiplexers. The basic idea is once again to first configure the network so that the ScanMux is switched to a given configuration, thus making a given path accessible. The difference with respect to the SIB is that the faulty path of a SIB is always longer (or shorter) than the active path. On the contrary, that is not the case of faulty paths of ScanMuxes (see Table 2). In that case, the length of the faulty path may even vary depending on the configuration of other modules in the active path. Moreover, the faulty paths can be more than one (e.g., in a 4-to-1 multiplexer, each configuration has 3 faulty paths). In order for the fault to be testable, the length of each faulty path has to be different than the active path. In details, the test procedure for a testable scan multiplexer fault is the following:

1) apply a certain number of configuration vectors, until:
   a) the multiplexer is part of the active path and set at a certain configuration, and
   b) the other modules are configured such that the faulty paths have different length than the active path;
2) shift in an initialization vector as long as the longest path in the network;
3) shift in a test vector as long as the expected path length;
4) check whether the expected sequence appears on the output of the path;
5) repeat the previous steps for all the multiplexer's configurations.

4 Network Representation

RSNs can be represented in different formats, such as the *Instrument Connectivity Language* (ICL) for IEEE Std 1687 networks. Structural representations are hard to handle and extracting the main properties of a network requires to pass through internal representations. In order to abstract the main functions of an RSN to be tested, we propose two formal graphs representations.

These representations are traversed by the proposed test algorithms, as discussed in the following section, in order to generate a sequence of configuration and test vectors for the target network.

The first graph, named *Topology Graph*, is based on the network topology and consists of all possible scan paths of the network. This graph is a topological view of the network. The second graph, named *Configuration Graph*, is based on the list of possible network configurations. A path in this graph represents a sequence of configurations in which the network can be sequentially placed.

4.1 Topology Graph

The topology graph is a simplified representation of the RSN providing a topological view of it. The elements of the RSN (TDRs, SIBs, ScanMuxes, configuration bits) are associated with vertices, each one annotated with its hierarchical depth, while the connections between elements are represented by edges, which are eventually annotated with a configuration value. The set of vertices also includes those associated to the input and output pins of the RSN, e.g., TDI and TDO. In case of a single pair of input/output pins, the graph has a single source vertex (e.g., TDI) and a single sink vertex (e.g., TDO).

The topology graph of the RSN of Fig. 3 is shown in Fig. 4. In such a graph, the reconfigurable modules' vertices are highlighted in grey and their outgoing edges are annotated the configuration value. For each vertex, the depth k is also reported in Fig. 4 using the notation $d = k$. In details, the elements ScanMux and cb_3 are placed at the top-level ($depth = 1$), i.e., they are always part of the active path. The elements TDR_0, SIB_1, cb_1, SIB_2, and cb_2 are placed in the segments controlled by ScanMux, thus their vertices are annotated with $depth = 2$. Finally, the vertices TDR_1 and TDR_2 are annotated with $depth = 3$, since they are placed in the segments controlled by SIB_1 and SIB_2, respectively.

Every possible path in the RSN is represented by a path in the topology graph from source to sink vertices. When the network is in a certain configuration, each reconfigurable module selects a given segment. Similarly, one *active edge* comes out from the related vertex in the topology graph. Moreover, each source vertex is connected to a sink vertex by means of an active path.

4.2 Configuration graph

The topology graph offers a view of the interconnections between modules of the network. Such a representation, however, does not include information about the time (in terms of scan clock cycles) required to move the network from one configuration to another. Thus, an alternative representation is needed.

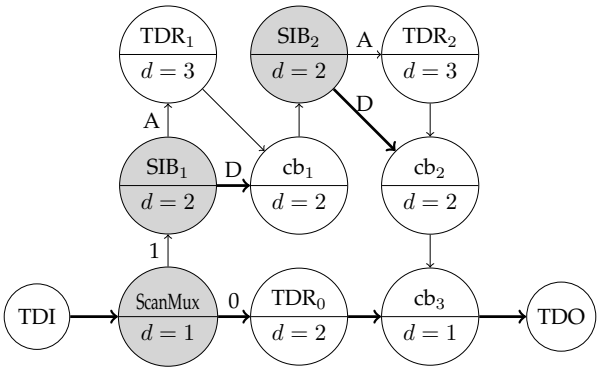


Fig. 4: Topology graph of the example network in Fig. 3.

The list of configurations and the neighborhood relation are used to build a directed graph $G = (V, E)$. Each vertex V_i corresponds to a network configuration σ_i. The reset state V_0 is used to refer to σ_0. Each edge (V_i, V_j) represents a transition from σ_i to σ_j with σ_j neighbor of σ_i and it is labeled with its transition cost, equal to the active path length of σ_i (possibly incremented by one, corresponding to the extra clock cycle for the update operation). The active path of a vertex can be obtained by applying the corresponding configuration to the topology graph.

The configuration graph can be built by applying the following procedure. As many vertices are created as the number of possible network configurations. In details, for each configuration σ_i, the following steps are performed:

1) a vertex V_i is created, if not existing;
2) the neighborhood Σ_i is identified;
3) for each configuration $\sigma_j \in \Sigma_i$, a vertex V_j is created, if not existing, and an edge (V_i, Vj) is created and labeled with the active path length of σ_i.

The process can be implemented as a recursive procedure that starts from the reset configuration and returns when all neighbor configurations are extracted from the neighborhood set.

As an example, the procedure has been applied on the network represented in Fig. 3. The adjacency matrix of the resulting configuration graph is shown in Table 3. Each row of the matrix reports the transition cost of the outgoing edges from a vertex to other vertices, or '–' when the two vertices are not connected by an edge. It can be noticed that the first four configurations have only one outgoing edge each. In such configurations, in fact, the ScanMux is configured to the value 0, thus the other configuration bits are not part of the active path. On the contrary, all the other configurations can reach all the other vertices of the graph. Moreover, all outgoing edges from a certain vertex are labeled with the same transition cost, equal to the active path of the vertex plus one.

5 Network Test

The complete test of a reconfigurable network must pass through a certain number of configurations, each one able to include in the active path a subset of the registers and the reconfigurable modules. Once each target configuration is reached, the active path of the network is tested and the response is observed by monitoring the scan output values. In Section 3, the concepts of configuration and test vectors and sessions have been introduced.

TABLE 3: Adjacency matrix of the configuration graph built on network in Fig. 3.

	0,D,D	0,D,A	0,A,D	0,A,A	1,D,D	1,D,A	1,A,D	1,A,A
0,D,D	–	–	–	–	4	–	–	–
0,D,A	–	–	–	–	–	4	–	–
0,A,D	–	–	–	–	–	–	4	–
0,A,A	–	–	–	–	–	–	–	4
1,D,D	4	4	4	4	–	4	4	4
1,D,A	12	12	12	12	12	–	12	12
1,A,D	12	12	12	12	12	12	–	12
1,A,A	20	20	20	20	20	20	20	–

After the system reset, the network is set to its initial configuration (which is known). The overall test procedure requires a certain amount of sessions. After each session, the network target configuration is changed and the target configuration of the previous session becomes the starting configuration.

During the test phase, the active path includes a certain number of reconfigurable modules to be tested. The test vectors to be applied in this phase depend on the specific defects under analysis. For simplicity, each Tst_i simply consists of an initialization vector composed of as many 0s as the longest path length, followed by a test vector composed of an alternate sequence of 0s and 1s. Clearly, more complex sets of vectors can be used in this phase.

Since the amount of possible configurations of a network grows exponentially with the number of reconfigurable modules, the problem of identifying a sequence of sessions which guarantees the full network test coverage while minimizing the TAT is not trivial. This article achieves the intended goal with a tractability limitation of the approach on large circuits, due to the size of the search space. In the following, an approach implementing a minimum cost search on the configuration graph is presented. The proposed optimal approach is also useful for evaluating the effectiveness, in terms of TAT, of alternative solutions to the same problem, e.g., based on heuristics or optimization techniques. Later on in this section, a pair of sub-optimal approaches based on the topology graph traversal are presented. The main advantage of these solutions is that they are easy to implement and successfully applicable even to very large circuits.

5.1 Optimal Approach

The approach proposed in this section formulates the problem as a graph search at the minimum cost. The cost to be minimized is the TAT, as expressed in Eq. (1). The configuration graph, enriched with test information, is traversed using a modified version of the A^* algorithm [23]. A^* is an informed search algorithm, which operates by searching among all possible paths to the solutions (goals) for the one that incurs the smallest cost. In a labeled graph, vertices are the problem states, while each edge represents the transition cost to move from a state to another. In the classic A^*, the goal is typically identified by a certain vertex.

The problem at hand is different from the classic A^* formulation in which there is not a predetermined goal

state. The goal is to reach the full test coverage of the network, thus it cannot be associated to a certain state of the configuration graph. Instead, whether or not a given state is the goal depends on the path followed to reach it. In other words, the algorithm is looking for a goal path instead of a goal state.

The proposed formulation of the A^* algorithm minimizes the following function:

$$f(n,p) = g(n,\mathscr{F}) + h(p) \quad (2)$$

where p is a path, n is the last vertex on p, and $\mathscr{F}$ is the list of faults that are detected by p. Moreover, a goal function $G(p)$ is introduced, that states whether or not p is a goal path (i.e., if it covers the whole list of testable faults). This means that the estimates of the cost from a state to the goal must be a function of the current path as well. The heuristic function that computes those estimates is therefore denoted as $h(p)$ in Eq. (2) as well. Details about the heuristic function are given later in this section. In classic A^* we keep a frontier of open vertices that corresponds to a tree of partial paths rooted in the initial state. Each of these vertices is labeled with information that is used to keep track of the current best path (in terms of actual cost plus estimated cost to the goal) from the initial state to such a vertex. Every time a path to a vertex with a lower actual cost is found, such information (predecessor vertex) is updated. The classic A^* approach is sound because the estimated cost for the vertices does not change. Therefore, a lower actual cost makes the new path always preferable. In our case, depending on the path we follow to reach a given vertex, the set of faults that are covered by the path may vary. Therefore, we need to keep multiple instances of each vertex open at any given time, in order to take into account the different paths to reach the vertex along with their different set of detected faults. The proposed solution is to keep a frontier of open vertices like in classic A^*; however each of these vertices maintains a hash table of paths with keys equal to the set of detected faults. When a new path to a vertex is found, if it has the same set of detected faults of a previously found path, its actual cost is checked and, if lower that the previous path, the latter is updated. This contribution is indicated with $g(n,\mathscr{F})$ in Eq. (2).

In other words, each vertex keeps track of a set of paths, each with a different set of detected faults. A new path overwrites a previously stored one if it detects the same set of faults but has a lower actual cost. For each vertex we also keep track of the current best path to reach it, that is the currently open path to the vertex that has the lower combined cost (i.e., actual cost plus estimated cost to the goal). The estimated cost to the goal is computed by means of an admissible heuristic function.

The pseudo-code of the algorithm is reported in Fig. 5, where the f, g, and h functions are referred to Eq. (2), G is the goal function, key corresponds to $\mathscr{F}$, *OpenQueue* is the frontier, and *ClosedPaths* contains the list of paths that are proved to be non-optimal (i.e., they either do not detect all faults or they do so, but with a higher TAT than the current best path). Briefly, the algorithm iteratively extracts the current best open path from the frontier, visits its neighborhood by updating/adding open paths until the goal is reached or no more open paths remain. When a best path to a node is extracted from the frontier, so that the goal function is satisfied, it represents the optimal solution (the algorithm is exhaustive in its search).

The performance in terms of CPU time needed to reach the goal highly depends on the heuristic function. The proposed heuristic uses the length of the segments connected to each configurable element to estimate the cost of the remaining tests required to fully cover the network faults. Given a reconfigurable module M with k selectable segments, a fault F that forces M to select the segment s_i can be detected by configuring M so that a segment s_j, with j different than i, is included in the active path, and then shifting a test vector into the network. Such a test vector is at least as long as the length of s_j. According to this reasoning, the contribution of F to the heuristic function is comparable to the length of the shortest segment other than s_i, plus the number of configuration bits of the $F-$related module; if such a segment includes other reconfigurable modules, such modules and the segments they control are not counted; this permits not to take a segment into consideration multiple times in the heuristic function computation. For example, the fault that forces the ScanMux module of Fig. 3 to the value 1 is detected by a configuration in which ScanMux is set to 0 and selects the segment that includes TDR_0. The contribution to the heuristic function of such a fault is the TDR_0 length plus the configuration bit (i.e., $2 + 1 = 3$). The cost of the opposite ScanMux fault is estimated as the length of a modified version of the other segment (i.e., the one which includes the two SIBs), in which the inner SIBs have been removed; thus, the cost is zero for the segment plus the configuration bit (i.e., $0 + 1 = 1$).

The heuristic function value is computed while considering each of the remaining untested faults of the fault list. When no such faults exist anymore, the goal is reached (i.e., the path is a test sequence). The application of the algorithm on the example network in Fig. 3 produces the following test sequence after reset:

- Session 1
  1) Configuration 1,D,D
  2) Test: SIB_1 stuck-at-A, SIB_2 stuck-at-A
- Session 2
  1) Configuration 1,A,A
  2) Test: SIB_1 stuck-at-D, SIB_2 stuck-at-D, ScanMux stuck-at-0
- Session 3
  1) Configuration 0,A,A
  2) Test: ScanMux stuck-at-1

5.2 Sub-Optimal Approaches

For large networks, the optimal approach based on A^* is hardly applicable due to the excessive search space size. In such situations, a scalable approach is preferred, even if the TAT obtainable is sub-optimal. In the following, two alternative approaches are shown, both based on topology graph traversal: a depth-first approach and a breadth-first approach.

The proposed strategies apply a sequence of test sessions by traversing the topology graph. At each step, a vertex associated to a reconfigurable module M_i is found

```
OpenQueue ← ∅;
ClosedPaths ← ∅;
insert reset configuration state V_0 into OpenQueue;
while OpenQueue ≠ ∅ do
    extract V_i with the lowest f(V_i) from OpenQueue;
    p ← best path to V_i;
    put p into ClosedPaths;
    if G(p) is true then
        return p;
    end
    N ← neighbors of V_i;
    foreach V_j ∈ N do
        q ← path p connected to V_j;
        if q ∉ ClosedPaths then
            key ← faults covered by q;
            cost_p ← actual cost of path p;
            cost_ij ← transition cost from V_i to V_j;
            if cost_p + cost_ij < g(V_j, key) then
                update g(V_j, key);
                update f(V_j, q) with h(q);
            end
            put V_j into OpenQueue;
        end
    end
end
```

Fig. 5: Pseudo-code of the optimal approach based on the A^* algorithm.

in the graph. In order to change the configuration of M_i, a configuration phase is performed, which consists of one or more configuration vectors. After the configuration phase, a different outgoing edge of vertex M_i becomes active (as defined in Section 4.1), while the previously selected edge becomes inactive. As a result, the active path of the network changes. Then, a test phase is performed and the fault list is updated. By carefully identifying target edges on the graph, the process alternates these two phases, until the full test coverage is reached. The two strategies differ in the way these edges are selected.

5.2.1 Depth-First

The topology graph is traversed by following a depth-first approach. At each step of the graph traversal, a subset of reconfigurable modules is selected, that are part of the current active path. In the set of selected modules, the configuration is changed only for each module that is able to excite some of the untested faults and lies at the maximum depth. The depth of each module is found as an annotation of the topology graph vertices (see Section 4.1). All new configurations are applied together by means of a single configuration vector, in case all the configuration bits are part of the active path (i.e., reconfigurable modules with local control), otherwise multiple configuration vectors are needed. A configuration in which all excited faults become observable is reached (i.e., in which the selected modules are part of the active path), then a test vector is applied. The process is repeated until all faults are covered. The pseudo-code of the depth-first approach is reported in Fig. 6.

As an example, the application of the depth-first strategy on the graph in Fig. 4 produces the following test sequence after reset:

- Session 1
  1) Configuration 1,D,D
  2) Test: SIB_1 stuck-at-A, SIB_2 stuck-at-A
- Session 2
  1) Configuration 1,A,A
  2) Test: SIB_1 stuck-at-D, SIB_2 stuck-at-D, SMux stuck-at-0
- Session 3
  1) Configuration 0,A,A
  2) Test: SMux stuck-at-1

In the case of the example, the algorithm is able to produce the same test sequence of A^*.

```
FL ← all testable faults;
while FL ≠ ∅ do
    M ← reconfigurable modules in the active path;
    d ← maximum depth of m ∈ M which is able to
     excite a fault in any configuration;
    foreach m ∈ M do
        if depth(m) = d and m is able to excite a fault
         then
            activate an outgoing edge from m that
             excites a fault;
        end
    end
    foreach just configured m ∈ M do
        reach a configuration in which m is part of the
         active path;
    end
    apply a test vector;
    remove tested faults from the fault list;
    if all m ∈ M are fully tested then
        reach a configuration in which untested
         modules are part of the active path;
    end
end
```

Fig. 6: Pseudo-code of the sub-optimal approach based on the depth-first algorithm.

5.2.2 Breadth-First

The topology graph is traversed by following a breadth-first approach. The algorithm groups reconfigurable modules into levels, according to their hierarchical depth. Starting from the top-level, modules are tested level by level. At each iteration, the network is configured such that one or more modules of the target level are part of the active path and new faults can be excited. Then, a test vector is applied and new faults that are tested are removed from the fault list. Once all reconfigurable modules of the target level have been fully tested, the next level is considered, until the maximum depth of the network has been reached or all faults have been detected. The pseudo-code of the breadth-first approach is reported in Fig. 7.

As an example, the application of the breadth-first strategy on the graph in Fig. 4 produces the following test sequence after reset:

- Session 1
  1) Configuration 1,D,D
  2) Configuration 0,A,D
  3) Test: $SMux_1$ stuck-at-1
- Session 2
  1) Configuration 1,A,D
  2) Test: SIB_1 stuck-at-D, SIB_2 stuck-at-A, SMux stuck-at-0
- Session 3
  1) Configuration 1,D,A
  2) Test: SIB_1 stuck-at-A, SIB_2 stuck-at-D

```
FL ← all testable faults;
d ← 1;
while FL ≠ ∅ do
    M ← reconfigurable modules of level d;
    while all m ∈ M are not fully tested do
        foreach m ∈ M do
            if m is able to excite a fault then
                activate an outgoing edge from m that
                 excites a fault;
            end
        end
        repeat
            reach a configuration in which one or more
             m are part of the active path;
            apply a test vector;
            remove tested faults from the fault list;
        until all previously configured m ∈ M have been
         considered;
    end
    d ← d + 1;
end
```

Fig. 7: Pseudo-code of the sub-optimal approach based on the breadth-first algorithm.

6 Experimental results

The effectiveness in terms of test duration of the proposed algorithms has been evaluated with an in-house tool on a sub-set of RSNs of ITC'16 benchmarks [7]. Moreover, additional networks have been created, which show the main differences between the proposed algorithms. The A^* algorithm has been compared against the depth-first search and the breadth-first search algorithms.

We wrote a Java tool able first of all to read the network topology described in different formats. The *ICL Tools Software library* from [24] has been included in the developed tool. The tool also allows to compute the cost parameters t_i^c and t_i^t presented in Section 3.1.

The experiments were run on a server equipped with a dual Intel Xeon CPU E5-2680 v3 and 256 GB of RAM. Each benchmark network has been tested using A^* and both sub-optimal approaches (depth-first and breadth-first). Each experiment used a maximum heap size of 64 GB.

To assess the effectiveness of the proposed high-level fault model, we synthesized some selected benchmark networks using the NanGate 45nm Open Cell Library. We then performed some fault simulation experiments using Synopsys TetraMAX to compute the stuck-at fault coverage that can be achieved by running the test sequences generated by our method. The fault simulation results showed that in general a high or complete stuck-at fault coverage is achieved. The few missed faults are related to MUXes having more than 2 data inputs, especially when they are located deep in the hierarchy (and thus are less frequently excited). Finally, the stuck-at faults affecting the update logic and the flip-flops are either covered or they propagate long sequences of Xs in the circuit. Since the proposed test sessions demand for a precise sequence of 0s and 1s to be observed on scan output ports, faults that propagate sequences of Xs can be safely marked as covered.

6.1 Experiments with ITC'16 Benchmarks

The key characteristics of the ITC'16 benchmark networks are detailed in Table 4. For each network, the table reports first the number of SIBs and ScanMuxes. The fourth column refers to the number of configuration bits of SIBs and ScanMuxes. 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 [7]). Finally, the column *Longest path* reports the maximum possible number of scan cells on active path, while *Total scan cells* is the sum of the lengths of all scan registers in each network.

In the experiments, the cost for a configuration vector has been set to the active path length plus the JTAG protocol overhead (to move from shift to update, see [22]). The cost for a test vector has been set to the sum of the following contributions:

1) the JTAG protocol overhead (to move from update to shift), which has been set to 5;
2) the longest path length (initialization vector, see Section 3.2);
3) the active path length plus two (a sequence of alternated 0s and 1s as long as the active path followed by two consecutive 1s).

Due to tractability limitations (given the size of the search space), A^* resulted in out-of-memory failures for most of the benchmarks, while the sub-optimal algorithms were occupying few memory resources even for very large networks (no out-of-memory has been experienced by reducing the heap size up to 1GB). CPU time required for A^* was in the order of minutes, while few seconds were required to run sub-optimal algorithms. Experimental results on ITC'16 benchmarks are shown in Table 5. For each algorithm, the table reports the number of sessions (column 3), each one composed of one or more configuration vectors and a test vector. The total number of configuration vectors is also reported (column 4). The table also indicates the number of clock cycles required by configuration vectors (column 6) and test vectors (colum 7), as well as their sum (column 8). Finally, for sub-optimal approaches, the ratio of the TAT over the A^* TAT is reported (column 9), when A^* succeeded, i.e., only for the networks *Mingle* and *N17D3*. All modeled faults have been covered in each experiment (i.e., test coverage is 100%).

An analysis of Table 5 shows that the two sub-optimal approaches produced the same results (i.e., the TAT) for most of the benchmark networks. In order to understand the

TABLE 4: Characteristics of the ITC'16 benchmark networks

Network	SIB	ScanMux	Config. bits	Max depth	Longest path	Total 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
TreeUnbalanc.	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	96,158
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

reasons, we have analyzed the topology of the benchmarks and figured out that all networks but *TreeBalanced* and *TreeFlat_Ex* contain SIBs and 2-to-1 ScanMuxes (i.e., with one configuration bit). Moreover, in all such networks that have only 2-to-1 ScanMuxes, one of the two segments controlled by a ScanMux does not include any other nested SIB or ScanMux, with the only exception of the network *Mingle*. We will refer to ScanMuxes of this kind as *Unbalanced ScanMuxes*, while ScanMuxes that have nested reconfigurable modules on both segments (more than one segment, in case of ScanMuxes larger than 2-to-1) are referred to as *Balanced ScanMuxes*. Balanced ScanMuxes are one category of the modules that determine a different result between depth-first and bread-first, as for the network Mingle. The other factor is the presence of ScanMuxes with more than one configuration bits (i.e., larger than 2-to-1) not placed at the maximum hierarchical depth (level) of the network. In the benchmarks, the networks TreeBalanced and TreeFlat_Ex both have one ScanMux with 3 configuration bits. The only difference between them is that the ScanMux of network TreeBalanced is placed in the bottom hierarchical level, while in TreeFlat_Ex it is placed in an intermediate level. For this reason, only the network TreeFlat_Ex presents different results between depth-first and breadth-first.

6.2 Experiments with Synthetic Benchmarks

The effectiveness of the proposed algorithms has been also evaluated on new synthetic networks. A tool able to generate random networks (constrained with some parameters that affect the topology shape) has been purposely devised. The tool has been used in an evaluation experiment, where around 20k networks have been generated. Networks include both unbalanced and balanced ScanMuxes, also wider than 2-to-1, and are manageable with the A^* approach. The characteristics of the generated networks are the following:

- number of ScanMuxes between 2 and 16;
- number of configuration bits between 2 and 19;
- longest path between 22 and 55,428 scan cells;
- cumulative path between 22 and 70,088 scan cells;
- maximum depth between 2 and 9 levels.

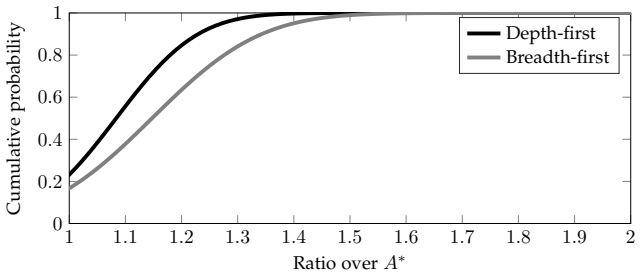


Fig. 8: Normal cumulative distribution function (CDF) of the ratio between sub-optimal approaches (depth-first in black, breadth-first in gray) and A^* on the randomly generated networks.

The CPU time required by A^* ranged between few seconds and half an hour, while it was negligible for sub-optimal algorithms.

For each network, the TATs of the test sequences developed by depth-first and breadth-first approaches have been divided by the TAT of the test sequence generated by A^* (as was the case for the column *Ratio over* A^* of Table 5). The normal distribution of the results of the depth-first approach has a mean of 1.08 and a standard deviation of 0.11, while the breadth-first has mean equal to 1.15 and standard deviation equal to 0.15. The maximum values are 1.99 for depth-first and 2.15 for breadth-first. The cumulative distribution functions (CDFs) of the two algorithms are reported in Fig. 8. The figure shows that depth-first and breadth-first approaches have been able to find the optimal solution (ratio over A^* equal to 1) in 23% and 17% of the cases, respectively, while in 90% of the cases the test sequence produced by the two algorithms is long 1.23 and 1.34 times with respect to A^* or shorter. Fig. 8 shows that depth-first performs better than breadth-first on the synthetic networks. Moreover, both algorithms produce test sequences that compare well with the optimal ones in terms of TAT.

The analytic inspection of the results shows that in 0.6% of the cases, breadth-first performed better than depth-first (which performs better in 50.9% of the cases). Since the complexity of the two algorithms is comparable, we recommend to implement the depth-first strategy for large networks.

7 Conclusions

The article presented several methods for the test of reconfigurable modules of an RSN. The proposed methodology represents the network topology and the possible configurations as graphs. An optimal test sequence in terms of TAT can be generated by applying the A^* search algorithm on the configuration graph. Sub-optimal approaches traverse the topology graph and are based on depth-first and breadth-first algorithms. Such approaches scale well on large networks, when the optimal approach is not applicable, while still producing a test set whose duration compares well with the optimal one. Experimental results on a large set of synthetic RSNs showed that depth-first approach slightly outperforms breadth-first. We are working on a systematic way to improve the generated test sequences, thus achieving a full detection of the few structural faults which are currently

TABLE 5: Experimental results on the ITC'16 benchmark networks

Network	Algorithm	Sessions	cv	tv	T_c [clock cycles]	T_t [clock cycles]	TAT [clock cycles]	Ratio over A^*
Mingle	A^*	7	7	7	337	1,684	2,021	–
	Depth-first	7	6	7	362	1,920	2,282	1.13
	Breadth-first	8	8	8	453	2,173	2,626	1.30
TreeBalanced	Depth-first	8	7	8	8,580	60,789	69,369	–
	Breadth-first	8	7	8	8,580	60,789	69,369	–
TreeFlat_Ex	Depth-first	6	5	6	15,263	56,078	71,341	–
	Breadth-first	7	8	7	30,250	66,177	96,427	–
TreeUnbalanced	Depth-first	12	11	12	237,475	834,324	1,071,799	–
	Breadth-first	12	11	12	237,475	834,324	1,071,799	–
a586710	Depth-first	5	4	5	1471	298,153	299,624	–
	Breadth-first	5	4	5	1,471	298,153	299,624	–
p22810	Depth-first	3	2	3	573	152,364	152,937	–
	Breadth-first	3	2	3	573	152,364	152,937	–
p34392	Depth-first	5	4	5	697	196,005	196,702	–
	Breadth-first	5	4	5	697	196,005	196,702	–
p93791	Depth-first	5	4	5	1,950	706,928	708,878	–
	Breadth-first	5	4	5	1,950	706,928	708,878	–
q12710	Depth-first	3	2	3	43	130,979	131,022	–
	Breadth-first	3	2	3	43	130,979	131,022	–
t512505	Depth-first	3	2	3	494	385,530	386,024	–
	Breadth-first	3	2	3	494	385,530	386,024	–
N132D4	Depth-first	6	5	6	9,332	29,399	38,731	–
	Breadth-first	6	5	6	9,332	29,399	38,731	–
N17D3	A^*	5	5	5	900	3,007	3,907	–
	Depth-first	5	4	5	802	3,341	4,143	1.06
	Breadth-first	5	4	5	802	3,341	4,143	1.06
N32D6	Depth-first	5	4	5	183,439	759,031	942,470	–
	Breadth-first	5	4	5	183,439	759,031	942,470	–
N73D14	Depth-first	13	12	13	1,577,674	4,400,373	5,978,047	–
	Breadth-first	13	12	13	1,577,674	4,400,373	5,978,047	–
NE1200P430	Depth-first	128	127	128	5,014,931	16,500,774	21,515,705	–
	Breadth-first	128	127	128	5,014,931	16,500,774	21,515,705	–
NE600P150	Depth-first	79	78	79	916,829	2,809,897	3,726,726	–
	Breadth-first	79	78	79	916,829	2,809,897	3,726,726	–

not detected. Moreover, we are working on techniques for detection of more sophisticated defect categories, as well as on the extension of the proposed methods to support diagnosis of RSNs.

ACKNOWLEDGMENTS

We acknowledge Anton Tšertov for the precious support on the ITC'16 networks and Mehrdad Montazeri for the contribution in previous conference papers, which this article is based on.

REFERENCES

[1] S. Narayanan and M. A. Breuer, "Reconfigurable scan chains: A novel approach to reduce test application time," in *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)*, Nov 1993, pp. 710–715.

[2] K. J. Lee and M. A. Breuer, "A universal test sequence for CMOS scan registers," in *IEEE Custom Integrated Circuits Conference*, May 1990, pp. 28.5/1–28.5/4.

[3] S. R. Makar and E. J. McCluskey, "ATPG for scan chain latches and flip-flops," in *IEEE VLSI Test Symposium*, Apr 1997, pp. 364–369.

[4] 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 *IEEE VLSI Test Symposium*, Apr 2008, pp. 79–84.

[5] R. Cantoro, M. Montazeri, M. Sonza Reorda, M. Ghani Zadegan, and E. Larsson, "On the testability of IEEE 1687 networks," in *IEEE Asian Test Symposium*, Nov 2015, pp. 211–216.

[6] R. Cantoro, M. Palena, P. Pasini, and M. Sonza Reorda, "Test time minimization in reconfigurable scan networks," in *IEEE Asian Test Symposium*, Nov 2016, pp. 119–124.

[7] A. Tsertov, A. Jutman, S. Devadze, M. Sonza Reorda, E. Larsson, F. Ghani Zadegan, R. Cantoro, M. Montazeri, and R. Krenz-Baath, "A suite of IEEE 1687 benchmark networks," in *IEEE International Test Conference*, Nov 2016, pp. 1–10.

[8] "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.

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

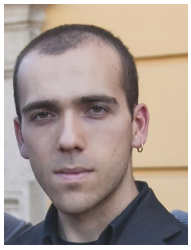
[10] R. Guo, L. Lai, H. Yu, and W. T. Cheng, "Detection and diagnosis of

static scan cell internal defect," in *IEEE International Test Conference*, Oct 2008, pp. 1–10.
[11] M. K. Reddy and S. M. Reddy, "Detecting FET stuck-open faults in CMOS latches and flip-flops," *IEEE Design & Test of Computers*, vol. 3, no. 5, pp. 17–26, Oct 1986.
[12] F. Yang, S. Chakravarty, N. Devta-Prasanna, S. M. Reddy, and I. Pomeranz, "Detection of internal stuck-open faults in scan chains," in *IEEE International Test Conference*, Oct 2008, pp. 1–10.
[13] ——, "Detectability of internal bridging faults in scan chains," in *Asia and South Pacific Design Automation Conference*, Jan 2009, pp. 678–683.
[14] D. Adolfsson, J. Siew, E. J. Marinissen, and E. Larsson, "On scan chain diagnosis for intermittent faults," in *IEEE Asian Test Symposium*, Nov 2009, pp. 47–54.
[15] F. Ghani Zadegan, U. Ingelsson, G. Carlsson, and E. Larsson, "Access time analysis for IEEE P1687," *IEEE Transactions on Computers*, vol. 61, no. 10, pp. 1459–1472, Oct 2012.
[16] ——, "Design automation for IEEE P1687," in *Design, Automation and Test in Europe*, Mar 2011, pp. 1–6.
[17] F. Ghani Zadegan, E. Larsson, A. Jutman, S. Devadze, and R. Krenz-Baath, "Design, verification, and application of IEEE 1687," in *IEEE Asian Test Symposium*, Nov 2014, pp. 93–100.
[18] R. Baranowski, M. A. Kochte, and H. J. Wunderlich, "Modeling, verification and pattern generation for reconfigurable scan networks," in *IEEE International Test Conference*, Nov 2012, pp. 1–9.
[19] ——, "Reconfigurable scan networks: Modeling, verification, and optimal pattern generation," *ACM Transactions on Design Automation of Electronic Systems*, vol. 20, no. 2, pp. 30:1–30:27, Mar 2015.
[20] M. A. Kochte, R. Baranowski, M. Schaal, and H. J. Wunderlich, "Test strategies for reconfigurable scan networks," in *IEEE Asian Test Symposium*, Nov 2016, pp. 113–118.
[21] 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.
[22] F. Ghani Zadegan, U. Ingelsson, G. Asani, G. Carlsson, and E. Larsson, "Test scheduling in an IEEE P1687 environment with resource and power constraints," in *IEEE Asian Test Symposium*, Nov 2011, pp. 525–531.
[23] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, Jul 1968.
[24] (2014) BASTION web-site. [Online]. Available: http://fp7-bastion.eu/

Riccardo Cantoro received the MS degree in Computer Engineering from Politecnico di Torino, Italy, in 2013, and his PhD degree in Computer and Control Engineering in 2017 from the same university. He is a currently a post-doc researcher in the Department of Computer Engineering, Politecnico di Torino. His research interests include software-based functional test of microprocessor based systems, and reconfigurable scan networks.

Farrokh Ghani Zadegan received his BS degree in electrical engineering from Ferdowsi University of Mashhad, Iran, in 2001, his MS degree in electrical engineering from Linköping University, Sweden, in 2010, and his PhD degree in electrical engineering from Lund University, Sweden, in 2017. His research interests include optimized design and operation of reconfigurable on-chip instrument access networks. He received the best paper award at the IEEE European Test Symposium 2016.

Marco Palena received his MS degree in Computer Engineering in 2012 from Politecnico di Torino. In 2017, he received his PhD degree in Computer and Control Engineering from the same university. He currently is a Post-doctoral Researcher at the Dept. of Control and Computer Engineering of Politecnico di Torino. His research interests include SAT-solving and applied formal methods.



Paolo Pasini received his MS in Computer Engineering (2012) and his PhD in Computer and Control Engineering (2017), both from Politecnico di Torino. He is currently a post-doc researcher at the Dept. of Control and Computer Engineering of the same University. He is active in the field of Formal Verification, with a specific focus on Hardware Model Checking.

Erik Larsson is Associate Professor at the Department of Electrical and Information Technology at Lund University (LU). He received his M.Sc., Tech. Lic and Ph.D from Linkping University in 1994, 1998, 2000, respectively. He did his Post Doc (2001-2002) at Nara Institute of Science and Technology (NAIST) and a sabbatical at NXP Semiconductors, The Netherlands (2008-2010). From 2002 until 2012 he was with Linköping University, as an Assistant Professor (2002-2005) and as Associate Professor (2006-2012). His current research interests include test planning for manufacturing test, test during operation (in-situ), scan-chain diagnosis, silicon debug and validation, IJTAG/SJTAG, stacked 3D chip test, fault-tolerance for MPSoCs (Multi-Processor System-on-Chip), and property checking in distributed systems (MPSOcS with Network-on-Chip (NoC)). He has more than 150 publications in these areas. He received the Institution of Engineering and Technology (IET) Premium Award, 2009, and the best paper award at IEEE Asian Test Symposium (ATS)(2002) and at IEEE European Test Symposium (ETS)(2016). Erik Larsson is an Associate Editor of Transactions on VLSI, member of a number of committees, and is Senior member of IEEE.

Matteo Sonza Reorda received his MS degree in Electronics (1986) and PhD degree in Computer Engineering (1990), both from Politecnico di Torino. He currently is a Full Professor at the Dept. of Control and Computer Engineering of the same University. He is an IEEE Fellow. His research interests include test of SoCs and fault tolerant electronic system design.