Testability-Enhancing Resynthesis of Reconfigurable Scan Networks

Lylina, Natalia; Wang, Chih-Hao; Wunderlich, Hans-Joachim

Proceedings of the IEEE International Test Conference (ITC'21), Virtual,
10 - 15 October 2021, pp. 1–10

doi: http://dx.doi.org/10.1109/ITC50571.2021.00009

Abstract: Reconfigurable Scan Networks (RSNs) have to be tested before they can be used for post-silicon validation, diagnosis or online reliability management. Even a single stuck-at fault in the switch logic of an RSN can corrupt the scan paths and make instruments inaccessible. Testing the switch logic of an RSN is a complex sequential test problem. The existing test schemes for RSNs rely on the assumption that a fault in the switch logic will be detected by the altered length of the erroneously activated scan path. However, often this assumption does not hold and faults in the switch logic remain undetected.In this paper, an automated testability-enhancing resynthesis is presented. First, the testability of the initial RSN is accurately analyzed. If any single fault in the switch logic is undetectable by the altered path length, a small number of scan cells is inserted into the RSN. The presented scheme is applicable to arbitrary RSN designs and is compliant with state-of-the-art test methods and the applicable standards. The experimental results show the efficacy, the efficiency and the scalability of the approach.

Preprint

General Copyright Notice

This article may be used for research, teaching and private study purposes. Any substantial or systematic reproduction, re-distribution, re-selling, loan or sub-licensing, systematic supply or distribution in any form to anyone is expressly forbidden.

This is the author's "personal copy" of the final, accepted version of the paper published by IEEE.[1]

[1] **IEEE COPYRIGHT NOTICE**
©2021 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Testability-Enhancing Resynthesis of Reconfigurable Scan Networks

Natalia Lylina, Chih-Hao Wang, Hans-Joachim Wunderlich
ITI, University of Stuttgart, Pfaffenwaldring 47, D-70569 Stuttgart, Germany
{lylina, wangco, wu}@informatik.uni-stuttgart.de

***Abstract*—Reconfigurable Scan Networks (RSNs) have to be tested before they can be used for post-silicon validation, diagnosis or online reliability management. Even a single stuck-at fault in the switch logic of an RSN can corrupt the scan paths and make instruments inaccessible. Testing the switch logic of an RSN is a complex sequential test problem. The existing test schemes for RSNs rely on the assumption that a fault in the switch logic will be detected by the altered length of the erroneously activated scan path. However, often this assumption does not hold and faults in the switch logic remain undetected.**

In this paper, an automated testability-enhancing resynthesis is presented. First, the testability of the initial RSN is accurately analyzed. If any single fault in the switch logic is undetectable by the altered path length, a small number of scan cells is inserted into the RSN. The presented scheme is applicable to arbitrary RSN designs and is compliant with state-of-the-art test methods and the applicable standards. The experimental results show the efficacy, the efficiency and the scalability of the approach.

***Keywords*-Reconfigurable Scan Networks, Testability, Design-for-Test, Online Test, Offline Test, Switch Logic**

I. Introduction

Modern devices integrate complex on-chip instrumentation to enhance fast yield bring-up as well as to ensure its dependable and reliable operation [1]. Numerous instruments, such as Built-In Self-Test registers (BIST), sensors, aging monitors or trace buffers are integrated into the Device-under-Test (DUT). Reconfigurable Scan Networks (RSNs), as standardized by IEEE Std. 1687 [2] and IEEE Std. 1149.1 [3], can be used to collect the evaluation results from the embedded instruments and to control their operation. Thereby, an efficient online reliability management for safety-critical systems can be enabled using RSNs throughout the device lifetime [4, 5].

All these applications rely on the correct operation of the RSNs, which may become a dependability bottleneck of a device. Even a single fault in the RSN could corrupt scan paths, the erroneous data may be fetched by the RSN, make the instruments inaccessible through the RSN, and eventually result in a system failure. To ensure the reliable operation of RSNs, various approaches for test, diagnosis and post-silicon validation of RSNs have been presented [6–12]. The publications [7] and [8] present test methods to address gate-level stuck-at-faults in scan segments. Additional control structures can be added into the RSN to improve the observability of shadow regissters [7]. In [9], the focus is on the testability of reconfiguration modules, such as scan multiplexers. In [11], the problem of post-silicon validation of RSNs is considered to find possible mismatches between the specification and the actual silicon implementation. In [12], a diagnostic procedure is presented for permanent faults in the reconfiguration logic.

The above mentioned test, validation and diagnosis methods rely on the fact that a fault or a mismatch in the switch logic is detected based on the altered scan path length [6–12]. However, if such fault does not alter the length of the activated scan path, it remains undetected. Although, the untestable mismatches can be enumerated using simulation-based techniques as in [11], currently there does not exist any systematic solution to detect them during test.

Sequential test generation could be applied to detect the faults, which remain undetected by the above mentioned test strategies [13]. However, such strategies are costly, typically require a high number of test vectors and are often not feasible due to their test length. In addition, these strategies are not applicable for online and concurrent test of RSNs during lifetime. At the same time, structural resynthesis has been proposed to improve various dependability aspects of RSNs, including trustworthiness [14], fault-tolerance [15] and security compliance with the DUT [16–18].

This paper presents a design-for-test (DfT) approach to improve the offline and online RSN test with negligible hardware costs. First, an accurate testability analysis is performed to find faults, which are not detectable by an altered path length. If such faults are present in the RSN, a light-weight automated resynthesis enhances the testability with respect to single faults in the switch logic of RSNs. The resynthesis changes some scan path lengths and ensures a successful fault detection by the existing test, diagnosis and validation schemes. The only changes consist in adding a small number of scan cells, and leave the RSN still compliant with state-of-the-art standards and algorithms.

The remainder of the paper is organized as follows. The background of RSNs is presented in Section II. In Section III, the details of RSN modeling are given. In Section IV, a testability analysis is presented for so-called series-parallel RSNs. Section V discusses the applicability of the presented algorithm for arbitrary RSN graphs. In Section VI, a method is presented to improve the testability by adding a minor number of scan cells. In Section VII, some implementation details are given. Section VIII provides experimental results, which show the efficacy, the efficiency and the scalability of the presented method. Finally, Section IX concludes the paper.

Regular Paper

II. Reconfigurable Scan Networks

A. Basic Definitions

Reconfigurable Scan Networks, as shown in Fig. 1, are constructed of the basic components, which are commonly referred as scan primitives, and include the following:

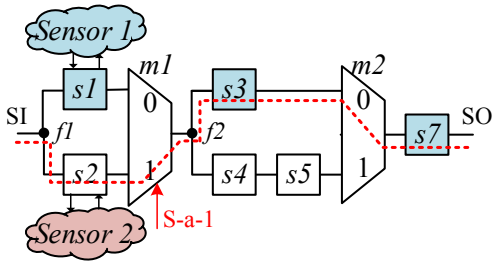

Fig. 1. RSN example

- The *Scan Segments* are the scan primitives, which directly access the instruments and transfer the data from a primary scan-in port (SI) towards a primary scan-out port (SO). Each scan segment consists of a shift register and of an optional shadow register.
- The *Scan Multiplexers* are the control scan primitives, which include specific parts of the RSN into an activated path, depending on the value of the address control signal.
- The *Segment Insertion Bits* (SIBs) are the control scan primitives, which either include or exclude specific part of the RSN into a path.

The control signals, which drive the control scan primitives, can be external to an RSN, or be internal, if they come from the shadow registers of the control scan segments. A *Scan Configuration* is defined by the state of the control scan segments. A non-circular path, which includes currently selected scan primitives, is referred as an *Active Scan Path* (ASP), and considers valid assignments to control signal values. In a valid scan configuration, a single ASP is activated.

Each access to an RSN is modeled as a *Capture-Shift-Update (CSU)-operation*, as presented in [6]. During the capture-phase, the data is read from the instruments to the shift registers of the scan segments. During the shift-phase, this data is shifted through the RSN towards the scan-out port, and new data is being shifted-in through the scan-in port. Finally, during the update-phase, the data is latched in the shadow registers and can be used to drive the internal control signals or to control the instrument's operation. More details on RSNs can be found in [6].

B. Test of RSNs

Test and debug infrastructure occupy a significant area on a chip. For example, an industrial report describes applications where 20% of functional bugs occur in the debug circuitry [19]. Faults in RSNs may arise in various fault locations, which include the scan segments, the control scan primitives and the interconnects. In contrast to testing conventional scan chains [20, 21], specific fault effects in RSNs are observable only for certain scan configurations. To test a particular fault in a scan primitive, the scan primitive must be included into an active scan path at least once.

Stuck-at-faults in scan segments can be addressed as in [7, 8, 22]. Single high-level stuck-at fault in the switch logic, which may occur e.g. in a scan multiplexer or in a SIB, can corrupt the data flow in the RSN. If a scan multiplexer is affected by a *"stuck-at-id"* fault, it always selects a specific scan-input regardless of the logic value, which drives its address control port. A SIB is *"stuck-at-asserted"*, if access to the underlying segment is always provided. In the opposite case, the SIB is *"stuck-at-deasserted"*.

As mentioned above, state-of-the-art test and validation schemes rely on the assumption that a fault in the switch logic changes the length of the activated scan path and can be detected this way [6–12]. If this assumption is not valid, the target fault remains undetected.

Definition 1: Any single fault f in the switch logic of the RSN is categorized as *"detectable by a path length (DT-PL)"*, if under the same scan configuration, the path length of the fault-free RSN is different compared to the faulty path, which is erroneously activated due to the fault f. Otherwise, a fault is categorized as *"undetectable by a path length (UDT-PL)"*.

Example: *In Fig. 1, an initial ASP traverses the scan segments $s1$, $s3$ and $s7$. If the scan multiplexer $m1$ is affected by a stuck-at-1 fault, the actual scan path is corrupted and now traverses the scan segment $s2$ as shown with a dashed red line. In this case, the data from the sensor 2 is captured instead of the data from the sensor 1. If the scan segments $s1$ and $s2$ have the same length, the fault is undetectable by a path length and Silent Data Corruption occurs.*

C. Resynthesis Goals

The presented testability-enhancing automated resynthesis of RSNs has the following goals:

- *Exact testability analysis:* The testability of RSNs should be accurately analyzed as shown in Section IV. If even a single fault is undetectable by a path length, the RSN is untestable. Otherwise, it is proven to be testable.
- *Enhanced fault detection for single faults:* For any initially untestable RSN, an automated resynthesis should ensure that all single faults in the switch logic are detectable by a path length, as discussed in Section VI.
- *Completeness and scalability:* The method should be applied within an acceptable runtime to large RSN designs with an arbitrary structure. For the resulting RSN, the testability analysis should validate that all target faults are now detectable by a path length.
- *Minimized hardware overhead:* A number of scan cells, which are added into the RSN to ensure unique path lengths, should be minimized.
- *Compatibility with the existing test schemes:* The RSN topology rules should not be modified. Thereby, the efforts of changing the existing test and access patterns are minimized, and also the security compliance with the DUT [17, 23] is not affected. The method should be applicable in addition to the existing online or offline techniques for RSNs [6–12].

III. RSN Modeling

A. Flat RSN Graphs

An RSN is modeled as a directed graph $G := (V, E)$, where V is a vertex set and E is an edge set of the graph. The sources I are the vertices, which only have outgoing edges; the sinks O only have incoming edges. A graph representation of the RSN from Fig. 1 is shown in Fig 2.

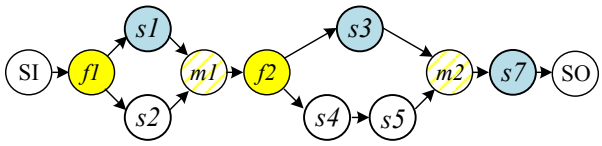

Fig. 2. Graph representation

Each vertex models a single scan primitive, a fan-out or corresponds to a primary scan-in or to a primary scan-out port. The edges represent the direct structural connectivities between the vertices. All control scan primitives, such as SIBs and multiplexers with an arbitrary number of scan inputs, are modeled as a combination of 2-input scan multiplexers and scan segments.

The considered fault set includes all the single high-level "stuck-at" faults in the switch logic.

B. Series-Parallel RSN Graphs

A so-called series-parallel graph simplifies the testability analysis presented in Section IV and the synthesis presented in Section VI. A general algorithm is proposed in Section V.

Definition 2: A directed acyclic *Series-Parallel* RSN graph (SP-RSN) $G := (V, E)$ with a vertex set V and an edge set E has a single source vertex and a single sink vertex, and is defined as:

- A graph, which consists of two vertices, which are connected via a single edge;
- Or a composition of two series-parallel graphs $G_1 := (V_1, E_1)$ and $G_2 := (V_2, E_2)$:
  1) A *parallel* composition: The source of G_1 is identified with the source of G_2, and the sink of G_1 is identified with the sink of G_2.
  2) A *series* composition: The sink of G_1 is identified with the source of G_2.

Fig. 3.a and b show a parallel and a series composition of two sub-RSNs. Any directed graph, which does not fulfill the conditions above is referred as a *non-series-parallel graph*.

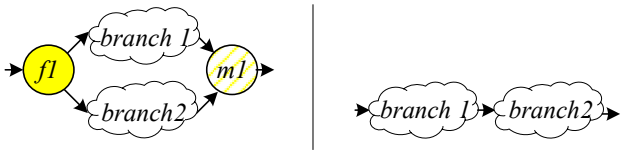

Fig. 3. RSN segments connected in a) parallel b) series

C. Hierarchical RSN Structures

To process RSNs efficiently, possible hierarchical relations between the scan primitives are formalized, as shown in Fig. 4.

A gate mn is reachable from another gate mi if at least one valid path can be activated from mi to mn in the RSN. For each reconvergent fan-out stem fi, a gate mi is its reconvergence gate [24], if there exist at least two disjoint paths from fi to mi. Due to the RSN structure, only scan multiplexers can serve as reconvergence gates. A closing reconvergence of the stem region is such a reconvergence gate, which does not reach any other reconvergence gate of the corresponding fan-out stem. The stem regions of the reconvergent fan-out stems are composed of all such scan primitives which are reachable from a given fan-out stem and also that a path from any of these primitives to the closing reconvergence exists. The stem region of fi is shown with grey color. The following hierarchical relations are considered:

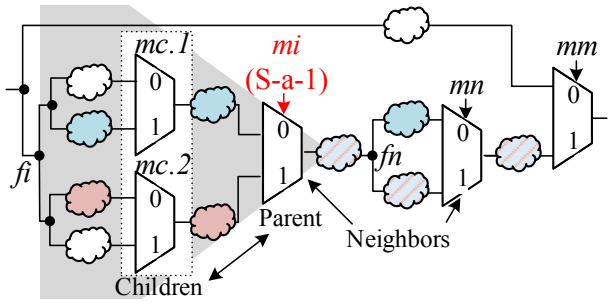

Fig. 4. Hierarchical relations

- All the paths through a scan primitive mi traverse another scan primitive mn and mn is reachable from mi. We say that mn *dominates* mi.
- A scan primitive mi dominates $mc.1$, and $mc.1$ belongs to such a stem region, where mi is a closing reconvergence. We say that mi is a *parent* of $mc.1$.
- A scan primitive $mc.1$ reaches a scan-input "0" of its parental scan mux mi, it is referred as an *0-child*.
- A scan primitive mn dominates mi, but is not its parent. We say that mn is a *neighbor* of mi.

Example: *The RSN graph from Fig. 2 is a series-parallel RSN. The multiplexer $m2$ is a reconvergence multiplexer of the fanout stem $f2$ and is its closing reconvergence.*

IV. Testability Analysis

Testability analysis has to identify all the single faults at the control signals which cannot be detected by observing the length of the activated path, in other words, the faults, which are *undetectable by a path length*. This section describes an algorithm to be applied to series-parallel RSN graphs. Series-parallel graphs allow to process the data hierarchically by a divide-and-conquer approach [24, 25]. To keep the algorithm complete, in Section V, some details on the additional steps are provided, which are required to transform an arbitrary non-series-parallel RSNs into a series-parallel representation.

In Section IV-A, a construction of binary decomposition trees for series-parallel RSNs is described. This information is further used to relax the testability constraints given a single fault assumption, as discussed in Section IV-B. Finally, in Section IV-C, a divide-and-conquer graph-based testability analysis is presented for single faults, which uses the binary decomposition tree to split a larger analysis problem into a number of smaller problems.

A. Binary Decomposition Tree Construction

To check, whether an RSN graph is series-parallel, an algorithm based on [26] is applied. The reachability of all scan primitives within the RSN is computed as in [23]. If the initial RSN graph has multiple sources or sinks, auxiliary source and sink vertices are added into its graph representation. A depth-first-search routine validates, whether a given RSN graph is acyclic. If the RSN graph contains cycles, an acyclic representation is built by removing a few edges. Although a graph, which contains cycles is non-series-parallel by definition, we proceed with the scheme below.

In order to identify the hierarchical relations between the scan primitives, which are used for a further analysis, a sequence of series and parallel reductions is applied. Fig. 5 shows an example for the left part of the RSN from Fig. 1.

- A *series reduction* (S) is applied, if two series-parallel subgraphs are connected in series, and the sink of the first subgraph is the source of the second subgraph. In the resulting graph, the source is taken from the first subgraph, and the sink – from the second subgraph.
- A *parallel reduction* (P) is applied, if the source of the first sub-graph is also the source of the second sub-graph; the same applies to the sinks.

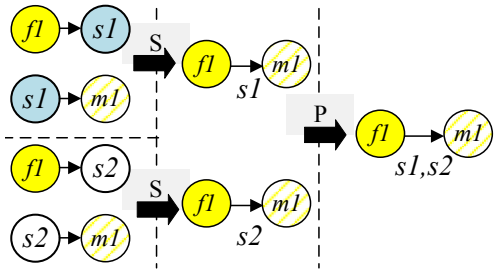

Fig. 5. Graph reduction: two edge pairs are reduced by a series reduction. Then, the resulting subgraphs are reduced in parallel, and a subgraph on the right hand side is obtained

At each step, the information about the performed reduction is saved into a binary decomposition tree. It includes the reduction type and the pointers to two initial subgraphs, each with a list of included scan primitives. If a parallel reduction is applied, some multiplexer mi becomes the sink of the resulting graph. The multiplexers, which belong to the initial subgraphs, are identified as *children* of the multiplexer mi, and mi is identified as their *parent*. If a series reduction is applied, all the scan multiplexers of the first subgraph are identified as *neighbors* of the multiplexers of the second subgraph, and vice versa.

The reductions are applied, until no more reductions are possible, in a top-down manner. A binary decomposition tree for the RSN from Fig. 1 is shown in Fig. 6. If an RSN graph is series-parallel, its decomposition tree can be decomposed to a single vertex. The vertices of the decomposition tree define, whether the subgraphs are connected in parallel, as shown with a green "P" vertex, or in series, as shown with a blue "S" vertex.

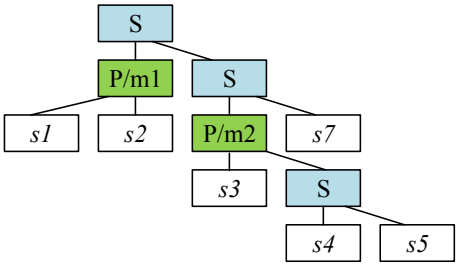

Fig. 6. Binary decomposition tree

Thanks to the Church-Rosser property [27] of the reduction system, any sequence of such reductions, which are applied to an RSN graph, would lead to a graph, which consists of a single edge. In the other case, the graph is not series-parallel and has to be handled as detailed in Section V.

B. Single Fault Reachable Paths Identification

The presented analysis is restricted to a realistic case of single faults and can be incrementally extended for multiple faults. The method presented in this subsection allows to relax the testability conditions under the single fault assumption. More specifically, it allows to identify for a pair of paths, whether the first path can be erroneously activated instead of the second path, due to a single fault f in the switch logic, or vice versa. If it is not the case, the detection of the fault f does not depend on the difference between the lengths of these paths.

Definition 3: An ASP p_l is called to be *"single fault reachable"* from the intended ASP p_k if and only if, due to a single fault in the switch logic, instead of an intended path p_k a path p_l is activated.

To check if the paths are single fault reachable, their activation conditions are analyzed. First, the activation conditions are computed for all vertices in the RSN graph. This step is only performed once for a given (sub-)RSN. For each vertex, the possible partial paths, which start at a (pseudo-) primary scan-in and reach the vertex, are collected into a map. A path length serves as a value, and arrays of possible conditions, which are required to activate a certain path, serve as a key, as shown in Table I for the RSN from Fig. 1. Each condition array includes the assignments to the multiplexers, which are required to include a given primitive into an ASP. Each bit $conds(mi)$ of the configuration array $conds$ specifies the required value of the address control signal of the scan multiplexer mi.

To determine, whether a certain partial path p_1 is "single fault reachable" from another partial path p_0 given a fault in the multiplexer m_i (as shown in Fig. 4) in a series-parallel

TABLE I
POSSIBLE PATHS

Segment	Length	Conditions m1	Conditions m2
$s1$	$\|s1\|$	0	X
$s7$	$\{\|s1\|or\|s2\|\} + \|s3\| + \|s7\|$	X	0
	$\{\|s1\|or\|s2\|\} + \|s4\| + \|s5\| + \|s7\|$	X	1

RSN, the conditions arrays of the last elements of the paths are compared bit by bit. The comparison continues either until both arrays are processed or until at least one contradicting value is found. If all the conditions below are fulfilled the paths are single fault reachable, and otherwise they are not single fault reachable:

- **Faulty multiplexer mi:** The conditions arrays should differ at the bit $conds(mi)$ to reflect the fault.
- **Children of mi:** The conditions arrays can take arbitrary values for the bits, which represent the children of mi. If the multiplexer mi, due to a fault, always selects a wrong scan-input, e.g. it is stuck-at-1, the multiplexer $mc.1$ is no longer included into an ASP, and can take an arbitrary value. The scan multiplexer $mc.2$, on the contrary, is now included into a path. If its value was not restricted in the initial path p_0 (X-value), it can also take an arbitrary value. A similar logic is applied, if the multiplexer mi is stuck-at-0.
- **Neighbors and parents of mi:** The conditions should not differ for any of the bits, which correspond to the neighbors or to the parents of a faulty multiplexer. In Fig. 4, mn is a neighbor, mm is a parent of the multiplexer mi. Since these multiplexers do not belong to a stem region, where a mi is a closing reconvergence, a fault in the multiplexer mi does not affect the data propagation through the multiplexers mn or mm.

Example: *In Fig. 1, if the multiplexer $m1$ is faulty, the partial ASP, which traverses the RSN from the fan-out stem $f2$ to the scan-out port is not affected. The faulty ASP, which is highlighted with a dashed red line, is "single fault-reachable" from the intended ASP, which is shown with a blue color. The paths through $s7$ in Table I are single fault reachable from one another, given a fault in the multiplexer $m2$.*

C. Testability Property Analysis

Section IV-C1 identifies whether there are *single fault reachable* paths with the same length in a simple flat RSN graph. This approach is extended in a divide-and-conquer manner, using the binary decomposition tree, as shown in Section IV-C2.

1) Analysis of Elementary Flat RSNs : The detectability of the faults, which affect a multiplexer m, is analyzed as shown in Algorithm 1. The multiplexer is annotated with a list of possible path length differences between the different scan-inputs of this multiplexer. All the combinations of partial paths, which end at the 0-input $paths_0$, and the 1-input $paths_1$ (Line 1-2) of a multiplexer m, are analyzed. The values in the conditions arrays of the last vertices in the partial paths are compared (Line 9) as described in Section IV-B.

If the activation conditions ($conds_0$ and $conds_1$) for these paths fulfill the single fault reachability property, the differences between the path lengths ($pathLen$) are added into the $differences$ set (Line 12). If any single fault reachable pair of paths is indistinguishable by a path length, a fault is undetectable UDT-PL (Lines 15-16).

Algorithm 1: $Node :: analyze$

```
   Input: The initial scan multiplexer m_i
   Output: List of problematic spots problems
1  paths_0 ← m_i.child0.paths;
2  paths_1 ← m_i.child1.paths;
3  for path_0 ∈ paths_0 do
4      pathLen_0 ← path_0.key();
5      conds_0 ← path_0.value();
6      for path_1 ∈ paths_1 do
7          pathLen_1 ← path_1.key();
8          conds_1 ← path_1.value();
9          if (1FReachable(conds_0, conds_1, m_i)) then
10             dif ← [pathLen_0 − pathLen_1];
11             differences.append(dif);
12         end
13     end
14 end
15 if differences.contains(0) then
16     problems.add(m_i, differences)
17 end
```

2) Hierarchical Analysis: The testability analysis is applied hierarchically and follows the order of the reverse polish notation of the decomposition tree from Section IV-A. The smallest considered subgraph is limited by a first traversed parallel composition vertex of the tree. The sub-graph is processed as described in Section IV-C1, and the detectability of the faults, which affect the control logic in the sub-RSN, is analyzed. The whole subgraph is abstracted to a single vertex, and the possible partial path lengths in this subgraph are used for annotation of this vertex. As soon as the low-level subgraphs are processed, the computation proceeds to a higher level, until the whole RSN is validated. If all the target faults are detectable by a path length, then the RSN is testable and the computation converges. Otherwise, the information about the control primitives, which are affected by undetectable faults, as well as the possible differences of the partial path lengths through different scan-inputs of the affected primitive is used to ensure fault detection during the resynthesis.

Example: *Given the decomposition tree from Fig. 6, the computation starts at the scan segment $s1$. The tree is traversed upwards until the first parallel composition is found. The smallest considered network includes the nodes $s1$, $s2$ and $P/m1$. The Algorithm 1 is performed on the subgraph. After the path lengths are validated in this sub-RSN, it is represented as a single node, as shown in Fig 7.*

At this step, not all the children of the top level series composition are processed. The algorithm backtracks to the node $s3$, and then to the nodes $s4$ and $s5$. The considered subgraph now includes the nodes $s3$, $s4$, $s5$, S and $P/m2$, and its testability is analyzed. The computation continues, until the whole RSN is processed.

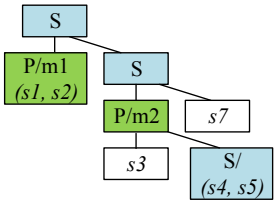

Fig. 7. Tree with merged nodes

V. Non-Series-Parallel RSNs

Although most RSNs can be directly modeled as series-parallel graphs, for some RSNs additional steps might be required to transform them into a series-parallel representation. To transform a non-series-parallel graph into a series-parallel form, either additional vertices or edges might be added into the initial graph [28], or removed from the graph [29].

In this section, for each non-series-parallel RSN region, a minimized number of additional virtual vertices is added into the initial RSN graph to build its series-parallel representation. The resulting series-parallel RSN graph representation is used for a testability analysis and, if needed, for a resynthesis of the RSN. Since the virtual changes will be reverted in the resynthesis phase, additional hardware overhead is not needed to transform the RSN graph into a series-parallel representation.

In the RSN graph, such fan-out stems are identified, which prevent the RSN graph from being series-parallel. Any fan-out stem f_{viol}, which is located in the stem region of another fan-out stem f_{init}, and which has the same closing reconvergence gate, is referred as a violation spot. The vertices, which are located between the fan-out stem f_{init} and the violation spot f_{viol}, are duplicated and are placed after the violation stem in the graph representation.

Example: *The initial RSN example from Fig. 1 is modified by adding an additional scan segment $s6$, as shown in Fig. 8.a. If its graph representation, which is shown in Fig. 8.b, would be reduced as much as possible, a single vertex representation will not be achieved. The decomposition tree for the resulting structure would include two sub-RSNs connected in series, as shown in Fig. 9.a. The first one consists of the scan segments $s1$ and $s2$, which are connected in parallel, and of the non-series-parallel subgraph.*

In Fig. 10.a, the fanout-stem $f2$ is a violation spot. In Fig. 10.b, the branch 1 is duplicated; its copies are placed after the violation fan-out stem. The violation spots are resolved sequentially, until a series-parallel representation of the RSN graph is obtained. The violation spots and their relative order of processing, are selected in a topological order of the RSN graph, which starts at the scan-in port. The fan-out stems, which are located closer to a primary scan-in vertex, are processed first, followed by the fan-out stems in their stem region, each time either going deeper in the hierarchy, or moving forward to a neighboring fan-out stem.

Example: *In the subgraph on the right hand side of Fig. 8, the fanout vertex $f3$ is located between the vertices $f2$ and $m3$. Since the multiplexer $m3$ is the closing reconvergence of the fanouts $f2$ and $f3$, it prevents the stem region of $f2$ from being series-parallel. In Fig. 10.c, an NSP region is transformed into a series-parallel form by duplicating the vertex $s4$. The resulting hierarchical binary decomposition tree, as shown in Fig. 9.b, only contains parallel and series compositions, as well as the leaf nodes, which correspond to the individual scan segments.*

VI. Automated Resynthesis

At this point, either the initial RSN is series-parallel, or a small number of virtual vertices is added to obtain a series-parallel representation. The automated resynthesis is performed on series-parallel RSNs only if any faults are identified as undetectable by the altered path length. The resynthesis method discussed in Section VI-A ensures the detectability of a single fault in an elementary RSN and in Section VI-B the detectability of all single faults in the switch logic of a flat RSN is discussed. Section VI-C presents a divide-and-conquer approach, which uses the previously constructed binary decomposition tree and makes the resynthesis approach scalable for large RSNs.

A. Enhancing the Testability for a Single Multiplexer

In order to guarantee the detectability of the "stuck-at" faults, which affect a multiplexer m, it should be ensured that all the "single fault reachable" paths through the multiplexer have a unique path length. Therefore, a small number of cells is added at the scan-inputs of this multiplexer.

The possible differences between the lengths of the partial single fault reachable paths, which end at the specific scan-inputs of the multiplexer, is obtained from the testability analysis as the set $differences$. A minimized length difference $addedCells$ is calculated as a smallest by the absolute value number, which is missing in the $differences$ set.

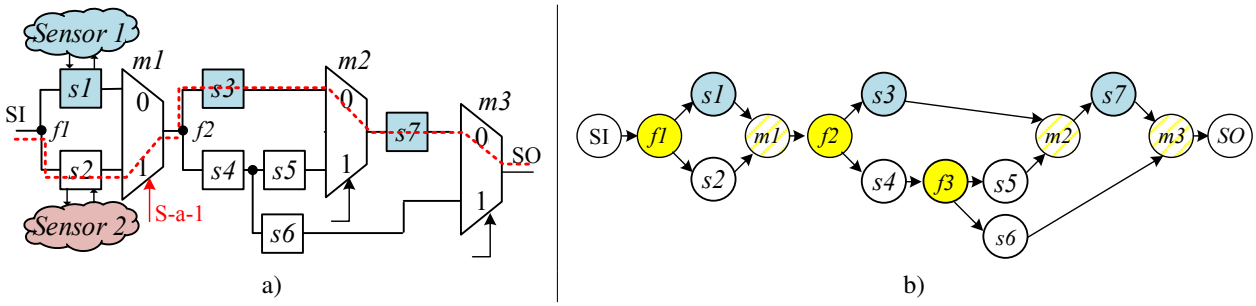

Fig. 8. a) Non series-parallel RSN b) Its graph

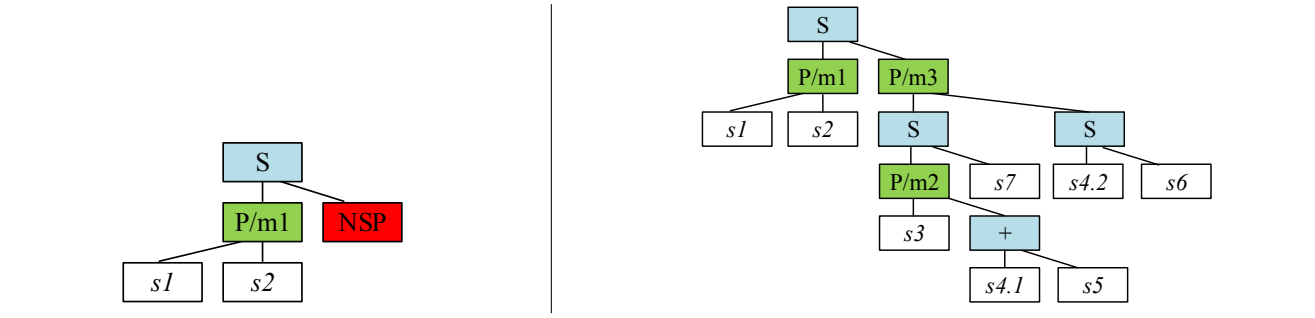

Fig. 9. Binary decomposition tree for a) non-series-parallel RSN from Fig. 8 b) its series-parallel representation

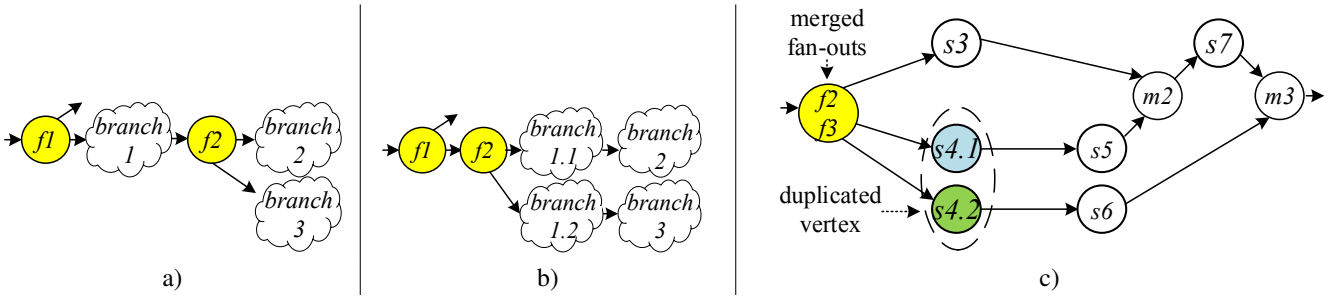

Fig. 10. a) Initial prohibited subgraph b) Transformed subgraph b) Transformed sub-graph of an RSN from Fig. 8

If the minimum length difference is a positive number, a number of scan cells equal to the absolute value of $addedCells$ is inserted at the 1-scan-input of the multiplexer. The cells are added at the 0-scan-input otherwise.

Example: *If the set of the path length differences contains the values* $\{-3, -1, 0, 1, 2, 3\}$*, a value* "-2" *is taken and two scan cells are added at the zero scan-input.*

B. Enhancing the Testability in a Flat RSN

The testability in a flat RSN graph is enhanced by using Algorithm 2, which ensures that the *single fault reachable* paths are distinguishable by path lengths. The vertices V_{in} of the graph serve as an input. The algorithm provides the vertex set V_{out} annotated with the updated lengths as an output. The following steps are performed:

- The topological order is computed, starting at the pseudo-primary scan-input (Line 1).
- Each node is annotated with a weight, which is equal to the length of a scan primitive (Line 2). The lengths of the partial paths from a (pseudo-)primary scan-input to a given vertex and their activation conditions are obtained from the analysis, which is described in Section IV-B.
- If a given node is a scan multiplexer (Line 6), the information about its children is obtained. After the children are processed, it is ensured that the paths through the 0-input of the multiplexer, are distinguishable from the paths through the 1-input by the altered path length, as discussed in Section VI-A (Line 9).
- Additional scan cells are inserted at one of the scan-inputs of the multiplexer, and the path lengths of the children nodes are updated considering the added scan cells (Line 10-11). The possible paths through the multiplexer are computed as a union of the paths through the scan-in branches of the multiplexer.

Algorithm 2: $uniqueLength$

Input: The initial vertex set V_{in}
Output: The resulting vertex set with updated lengths V_{out}

```
1  vertexOrder ← TopologicalOrder(V_in, SI);
2  assignWeights(V_in);
3  V_out ← V_in;
4  for node ∈ vertexOrder do
5      node.processed := true ;
6      if (node.type = scanMux) then
7          child_0 ← node.child0;
8          child_1 ← node.child1;
9          if (child_0.processed ∧ child_1.processed) then
10             ⟨cells_0, cells_1⟩ ← node.ensureLen2Ins();
11             child_0.updateLen(cells_0);
12             child_1.updateLen(cells_1);
13             V_out.updateValues(child_0, child_1);
14             node.paths ← [child_0.paths ∪ child_1.paths];
15         else
16             node.ensureChildrenProcessed();
17         end
18     end
19 end
20 V_out ← updatedLengths();
21 return V_out
```

- The output vertex set V_{out} is updated with the new lengths of the scan primitives (Line 20).

C. Hierarchical Resynthesis

The resynthesis of Section VI-B is applied in a divide-and-conquer manner, using the *binary decomposition tree*, as shown in Algorithm 3. The binary decomposition tree T of the series-parallel RSN serves as an input, and the output is the vertex set V_{out}^{RSN} with the updated scan segments lengths. In the resulting RSN, all the single faults in the control logic are detectable by the altered path length. The computation starts from the left-most leaf node of the tree, and continues in the order of the reverse polish notation (Line 1).

Algorithm 3: $uniqueLength\_hierarchical$

Input: The binary decomposition tree T of a series-parallel graph representation G_{SP}^{RSN}
Output: The vertex set with updated scan segments lengths V_{out}^{RSN}

1 $vertexOrder \leftarrow Order(T, Rev - Polish)$;
2 **for** $node \in vertexOrder$ **do**
3 $child_0 \leftarrow node.treeChild0$;
4 $child_1 \leftarrow node.treeChild1$;
5 **if** $(child_0.processed \wedge child_1.processed)$ **then**
6 $node.ensureChildrenProcessed()$;
7 **end**
8 **if** $\overline{node.processed}$ **then**
9 $V_{sub}.append(node)$;
10 $node.processed \leftarrow true$;
11 **end**
12 **if** $node.treetype =' P'$ **then**
13 $V_{sub}^{new} \leftarrow uniqueLength(V_{sub})$;
14 $V_{out}^{RSN}.updateValues(V_{sub}^{new})$;
15 $V_{sub} \leftarrow \emptyset$;
16 **end**
17 **end**
18 **return** V_{out}^{RSN}

The order $vertexOrder$ implies that the stem regions of the neighboring fan-out stems are processed sequentially, starting from a top-level fan-out stem, which is located closer to a primary scan-in port. Each stem region is processed in a bottom-up manner, starting from the leaves.

For each vertex, it is ensured that its children are already processed (Lines 5-6). A subnetwork is constructed by adding each further unprocessed vertex into a subset V_{sub} (Line 7-9). The graph traversal continues until the first parallel composition vertex is met (Line 12). A low-level sub-network, which includes the vertices V_{sub}, is processed as in Algorithm 2 (Line 13), and the output vertex set is updated with the values from the vertex subset V_{sub}^{new}, which considers the updated segment lengths. (Line 14).

Example: *For RSN in Fig. 1, the resynthesis follows the same order, as the hierarchical validation in Section IV-C2.*

VII. Implementation

An initial RSN is given e.g. in the Instrument Connectivity Language (ICL). The information about the virtual vertices, which have been added, as discussed in Section V, to build a series-parallel RSN representation, is only required to speed up the analysis and the resynthesis process and is not needed for the output ICL generation.

The information about the new lengths of the scan segments, which have been updated to enhance the testability of a given RSN, is used to regenerate the ICL description. The ICL code lines, which provide the information about the length of a scan segment located at a particular scan input of a scan multiplexer, are updated with a new value. If, in the original RSN, there is no scan segment at the scan input of the multiplexer, an additional scan segment is added to the RSN structure and also to its ICL description. Since only some buffer scan cells are added into the initial RSN, the presented approach does not affect the functional connectivity and thereby also the security compliance [23] of the RSN. In the resulting RSN, any fault in the switch logic would result in the altered ASP length.

VIII. Experimental Results

The automated resynthesis flow has been implemented in the *eda1687* framework of [6]. The experiments have been conducted on Intel(R) Xeon(R) W-2125 CPU at 4.00GHz with 132 GB of main memory. The RSN benchmarks have been taken from the ITC'2016 [30] and the DATE'2019 [16] benchmark sets.

For the considered benchmarks the number of scan multiplexers (Column 2), SIBs (Column 3), scan segments (Column 4), scan cells (Column 5) and the highest hierarchy level (Column 6) are given in Table II.

TABLE II
Characteristics of benchmarks

(1)**Design**	(2) #muxes	(3) #sibs	(4) #segs	(5) #cells	(6) #lvl
BasicSCB	10	-	21	176	4
Mingle	13	10	22	270	3
TreeFlat	24	12	24	101	2
TreeUnbalanced	28	28	63	41,887	11
TreeBalanced	46	43	90	5,581	7
TreeFlat_Ex	60	57	123	5,194	5
q12710	25	25	47	26,183	2
a586710	47	-	79	41,682	3
p34392	142	-	245	23,261	3
t512505	160	-	288	77,006	2
p22810	283	283	537	30,111	3
p93791	653	-	1,241	98,637	3
MBIST_1_5_5	15	8	113	548	4
MBIST_1_5_20	15	8	338	1,523	4
MBIST_1_20_20	45	23	4,179	6,068	4
MBIST_2_5_5	28	16	224	1,091	4
MBIST_2_5_20	28	16	674	3,041	4
MBIST_2_20_20	88	46	2,624	12,131	4
MBIST_5_5_5	67	40	557	2,720	4
MBIST_5_20_20	217	115	6,557	30,320	4
MBIST_5_100_20	1,017	515	32,557	151,520	4
MBIST_5_100_100	1,017	515	151,135	671,520	4
MBIST_20_20_20	862	460	26,222	121,265	4
MBIST_55_20_5	2,367	1,265	22,607	118,970	4
MBIST_100_20_5	8,102	2,300	41,102	216,305	4
MBIST_100_100_5	20,102	10,300	172,700	1,080,305	4

A. Initial Benchmark Set

The presented analysis method has checked the existence of faults in the switch logic, which remain undetected due to the unaltered path length. If any faults are undetectable, additional scan cells are inserted into the RSN. The testability analysis has been performed within an acceptable runtime.

1) Problems with Testability: For the benchmark designs *TreeBalanced, Mingle, BasicSCB*, the presented testability analysis approach identified single faults in the switch logic, which are undetectable by a path length. A minor number of scan cells, which ranges between 4 scan cells for *Mingle* and 8 scan cells for *BasicSCB*, has been added into the initial RSN structure to ensure fault detection by the existing methods. A negligible runtime (below 3 seconds) has been required to perform both the analysis and the resynthesis.

2) Proof of Testability: As expected, for a major part of all widely-approved benchmarks, any fault in the RSN switch logic is detectable by an altered path length. Using the presented approach, it has been algorithmically proven that the remaining considered benchmarks are testable with respect to single faults. The ability to prove this property for any given RSN design with an arbitrary structure is one of the major contributions of this paper.

B. Specifics of Processing

1) ITC'16 Benchmarks [30]: The ITC'16 benchmarks are processed with the presented DfT method. As expected, most of the RSN benchmarks can be modeled as series-parallel graphs, and a transformation into a series-parallel form is not required. Only the *"TreeFlat"* benchmark graph is not series-parallel. It is transformed into a series-parallel form, and virtual vertices are added to perform the transformation.

2) DATE'19 Benchmarks [16]: A hierarchical structure of benchmarks from the DATE'19 benchmark set is shown in Fig. 11. A top-level chip TAP controller is used to access N cores, such that each of the cores accesses the memory via M controllers. Each core is accessed via a separate Segment Insertion Bit (SIB core).

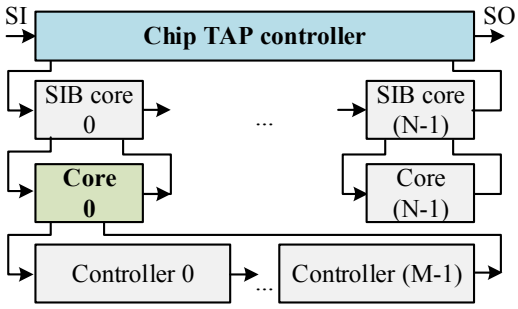

Fig. 11. MBIST benchmarks structure

The automated testability-enhancing resynthesis is applied to the DATE'19 benchmarks hierarchically. First, each sub-RSN, which corresponds to a single core (shown with green in Fig. 11), is modeled as a series-parallel graph. The resynthesis approach is first applied on a core-level to ensure single fault detection. A top-level chip TAP controller graph representation, is constructed next. Here, each core is modeled with a single vertex, and the information about the possible path lengths in a core is used for vertex annotation. A top-level representation is transformed from an arbitrary non-series-parallel graph into a series-parallel graph by adding just two virtual vertices for each benchmark. The detection of faults in a top-level representation is analyzed with the presented approach, and, if needed, a minimized number of cells is added.

C. Artificial Benchmarks

Since for the most initial benchmarks, the testability property has been proven, the applicability of the structural resynthesis up to this point has been only validated on a limited number of the initial benchmarks. For the third-party RSN designs the testability property is not guaranteed, and the scalability and efficiency of the resynthesis must be validated as well. To create a representative benchmark set, which contains a high number of large RSN designs, additional bypass registers with a controllable length are implemented. The testability of the modified benchmarks has been analyzed and the benchmarks have been resynthesized to ensure the unique path lengths within a single fault assumption in the switch logic. The output of the automated resynthesis method is the list of scan segments with their modified length. This list is used to modify the initial RSN to obtain a testable RSN.

The results are presented in Table III. Column 2 shows a number of additional scan cells, which have been added into the RSN. In average, one additional scan cell has been required for each modified scan segment. Compared with the total number of scan segments in RSNs (Column 4, Table II), and especially with the number of scan cells (Column 5, Table II), the presented method requires a negligible hardware overhead, and the increase of the instruments access latency through the resynthesized RSN is negligible as well. The number of virtual changes, which have been performed to transform an arbitrary graph into a series-parallel form, is given in Column 3 for all the benchmarks.

The number of faults in the control scan primitives is given in Column 4 for all the benchmarks, and includes all single stuck-at faults, affecting control primitives. In Column 5, the number of undetected faults, whose detection relies on ASP length, $UDT - PL$ is given for the initial RSNs. For the resynthesized RSNs, after the presented method is applied, each single fault in the control logic is detectable by an altered path length, as validated by the repeated testability analysis. The runtime of the approach is acceptable and requires less than 25 minutes for the largest benchmarks, and just few seconds for the most of the benchmarks (Column 6).

IX. Conclusion

This paper presents an automated resynthesis approach, which allows to test the control signals of RSNs. The testability of RSNs with respect to high-level stuck-at faults in the switch logic and control signals is accurately analyzed. If at least one fault in the switch logic is not detectable by the changed path length in the initial RSN, a light-weight resynthesis is applied to the RSN and adds a minimized number of scan cells. The resulting RSN is validated again using the presented testability analysis method, to check whether all the faults are now detectable by a path length. The presented method can be applied in addition to the known test, diagnosis and post-silicon validation methods for RSNs. The results show the efficiency and the scalability of the presented method for large RSNs designs with an arbitrary structure.

Acknowledgments

This work was supported by the German Research Foundation (DFG) under grant WU 245/17-2 (ACCESS) and partially supported by Advantest as part of the Graduate School "Intelligent Methods for Test and Reliability" (GS-IMTR) at the University of Stuttgart.

Bibliography

[1] M. A. Kochte and H.-J. Wunderlich, "Dependable on-chip infrastructure for dependable MPSOCs," in *Proc. Latin-American Test Symp. (LATS)*, 2016, pp. 109–114.

[2] "IEEE Standard for Access and Control of Instrumentation Embedded within a Semiconductor Device," *IEEE Std 1687-2014*, pp. 1–283, Dec. 2014.

TABLE III
TESTABILITY-ORIENTED RESYNTHESIS TO DETECT ALL UDT-PL

(1) **Design**	(2) $\#cells_{added}$	(3) $\#virtual_{added}$	(4) $\#faults_{total}$	(5) *UDT-PL*	(6) $runtime$[s]
BasicSCB	8	0	40	16	1.0
Mingle	4	0	52	8	1.2
TreeFlat	2	2,047	48	4	25.5
TreeUnbalanced	9	0	112	18	2.1
TreeBalanced	9	0	200	22	2.4
TreeFlat_Ex	28	0	256	56	3.5
q12710	23	0	108	46	1.0
a586710	22	0	128	44	1.3
p34392	73	0	388	146	2.2
t512505	107	0	636	214	8.6
p22810	224	0	1,080	460	3.2
p93791	550	0	2,384	1,100	10.1
MBIST_1_5_5	4	2	30	10	1.1
MBIST_1_5_20	4	2	30	8	1.6
MBIST_1_20_20	15	2	90	30	3.3
MBIST_2_5_5	7	2	56	14	2.2
MBIST_2_5_20	9	2	56	18	0.8
MBIST_2_20_20	32	2	176	64	9.1
MBIST_5_5_5	20	2	134	40	1.1
MBIST_5_20_20	75	2	434	150	26.1
MBIST_5_100_20	365	2	2,034	730	120.7
MBIST_5_100_100	375	2	2,034	750	1453.0
MBIST_20_20_20	300	2	1,724	600	21.4
MBIST_55_20_5	866	2	4,734	1,732	356.2
MBIST_100_20_5	1,586	2	8,604	3,172	168.1
MBIST_100_100_5	7,600	2	40,204	15,200	280.3

[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] A. Tsertov, A. Jutman, K. Shibin, and S. Devadze, "IEEE 1687 Compliant Ecosystem for Embedded Instrumentation Access and In-Field Health Monitoring," in *Proc. IEEE AUTOTESTCON*, Nov. 2018, pp. 1–9.

[5] A. M. Y. Ibrahim and H. G. Kerkhoff, "An On-chip IEEE 1687 Network Controller for Reliability and Functional Safety Management of System-on-Chips," in *Proc. Int'l. Test Conf. in Asia (ITC-Asia)*, Sep. 2019, pp. 109–114.

[6] R. Baranowski, M. A. Kochte, and H.-J. Wunderlich, "Reconfigurable Scan Networks: Modeling, Verification, and Optimal Pattern Generation," *ACM Trans. Design Automation of Electronic Systems (TODAES)*, vol. 20, no. 2, pp. 30:1–30:27, 2015.

[7] D. Ull, M. Kochte, and H.-J. Wunderlich, "Structure-Oriented Test of Reconfigurable Scan Networks," in *Proc. IEEE Asian Test Symp. (ATS)*, Nov. 2017, pp. 127–132.

[8] M. A. Kochte, R. Baranowski, M. Schaal, and H.-J. Wunderlich, "Test Strategies for Reconfigurable Scan Networks," in *Proc. IEEE Asian Test Symp. (ATS)*, Nov. 2016, pp. 113–118.

[9] R. Cantoro, F. G. Zadegan, M. Palena, P. Pasini, E. Larsson, and M. S. Reorda, "Test of Reconfigurable Modules in Scan Networks," *IEEE Trans. on Computers*, vol. 67, no. 12, pp. 1806–1817, 2018.

[10] R. Cantoro, A. Damljanovic, M. S. Reorda, and G. Squillero, "A New Technique to Generate Test Sequences for Reconfigurable Scan Networks," in *Proc. IEEE Int'l Test Conf. (ITC)*, 2018, pp. 1–9.

[11] A. Damljanovic, A. Jutman, G. Squillero, and A. Tsertov, "Post-Silicon Validation of IEEE 1687 Reconfigurable Scan Networks," in *Proc. IEEE European Test Symp. (ETS)*, May 2019, pp. 1–6.

[12] R. Cantoro, A. Damljanovic, M. S. Reorda, and G. Squillero, "A Novel Sequence Generation Approach to Diagnose Faults in Reconfigurable Scan Networks," *IEEE Trans. on Computers*, vol. 69, no. 1, pp. 87–98, 2020.

[13] I. Hamzaoglu and J. H. Patel, "Deterministic test pattern generation techniques for sequential circuits," in *IEEE/ACM Int'l Conf. on Computer Aided Design (ICCAD)*, Nov. 2000, pp. 538–543.

[14] M. A. Kochte, R. Baranowski, and H.-J. Wunderlich, "Trustworthy Reconfigurable Access to On-Chip Infrastructure," in *Proc. IEEE Int'l Test Conf. in Asia (ITC-Asia)*, Sep. 2017, pp. 119–124.

[15] S. Brandhofer, M. A. Kochte, and H. Wunderlich, "Synthesis of Fault-Tolerant Reconfigurable Scan Networks," in *Proc. Conf. on Design, Automation Test in Europe (DATE)*, Mar. 2020, pp. 798–803.

[16] P. Raiola, B. Thiemann, J. Burchard, A. Atteya, N. Lylina, H.-J. Wunderlich, B. Becker, and M. Sauer, "On Secure Data Flow in Reconfigurable Scan Networks," in *Proc. Conf. on Design, Automation Test in Europe (DATE)*, Mar. 2019, pp. 1–6.

[17] N. Lylina, A. Atteya, C.-H. Wang, and H.-J. Wunderlich, "Security Preserving Integration and Resynthesis of Reconfigurable Scan Networks," in *Proc. IEEE Int'l Test Conf. (ITC)*, Nov. 2020, pp. 1–10.

[18] N. Lylina, A. Atteya, and H.-J. Wunderlich, "A Hybrid Protection Scheme for Reconfigurable Scan Networks," in *To appear in Proc. of the IEEE VLSI Test Symp. (VTS'21)*, Apr. 2021, pp. 1–7.

[19] S. Kumar, "Industrial Challenges of Bugs and Defects," in *NSF/SRC/DFG Joint Workshop on Bugs and Defects in Electronic Systems: The Next Frontier*, Apr. 2013, Schloss Dagstuhl, Germany.

[20] F. Yang, S. Chakravarty, N. Devta-Prasanna, S. M. Reddy, and I. Pomeranz, "Detection of Internal Stuck-open Faults in Scan Chains," in *Proc. Int'l. Test Conf. (ITC)*, Dec. 2008, pp. 1–10.

[21] S. R. Maka and E. J. McCluskey, "ATPG for Scan Chain Latches and Flip-Flops," in *Proc. VLSI Test Symp. (VTS)*, Apr. 1997, pp. 364–369.

[22] R. Cantoro, M. Montazeri, M. S. Reorda, F. G. Zadegan, and E. Larsson, "On the Testability of IEEE 1687 Networks," in *Proc. Asian Test Symp. (ATS)*, 2015, pp. 211–216.

[23] N. Lylina, A. Atteya, P. Raiola, M. Sauer, B. Becker, and H.-J. Wunderlich, "Security Compliance Analysis of Reconfigurable Scan Networks," in *Proc. IEEE Int'l Test Conf. (ITC)*, Nov. 2019, pp. 1–9.

[24] F. Maamari and J. Rajski, "A method of fault simulation based on stem regions," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 9, no. 2, pp. 212–220, 1990.

[25] H. Fujiwara and T. Shimono, "On the Acceleration of Test Generation Algorithms," *IEEE Trans. on Computers*, vol. C-32, no. 12, pp. 1137–1144, 1983.

[26] J. Valdes, R. E. Tarjan, and E. L. Lawler, "The Recognition of Series Parallel Digraphs," in *Proc. of the Annual ACM Symp. on Theory of Computing*, 1979, p. 1–12.

[27] A. Church and J. B. Rosser, "Some properties of conversion," vol. 1, no. 2, p. 472–482., 1936.

[28] J. Keller and R. Gerhards, "PEELSCHED: a Simple and Parallel Scheduling Algorithm for Static Taskgraphs," *PARS: Parallel-Algorithmen, -Rechnerstrukturen und -Systemsoftware*, vol. 28, Oct. 2014.

[29] M. Mitchell, "Creating minimal vertex series parallel graphs from directed acyclic graphs," in *Proc. of the Australasian Symp. on Information Visualisation (invis.au'04)*, 2004, p. 133–139.

[30] A. Tsertov, 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 *Proc. IEEE Int'l Test Conf. (ITC)*, Nov. 2016, pp. 1–10.