Security Preserving Integration and Resynthesis of Reconfigurable Scan Networks

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

Proceedings of the IEEE International Test Conference (ITC'20), Washington DC, USA, 2020, pp. 1–6

doi: https://doi.org/10.1109/ITC44778.2020.9325227

Abstract: Reliable operation, test, debug and diagnosis of complex integrated systems are ensured by embedded instruments, such as sensors, aging monitors or Built-In Self-Test (BIST) registers. Reconfigurable Scan Networks (RSNs) offer a flexible and efficient way to access such test instruments throughout the whole life-cycle. However, improper RSN integration might introduce additional connectivity properties to the device under test (DUT), which can be exploited to perform unauthorized access or cause information leakage. The existence of such additional connectivity through the RSN can compromise the security of the DUT and is considered as a security threat.In this paper, a method is presented to resolve all such security compliance violations. The problem is formulated in terms of Integer Linear Programming (ILP) as a minimum cut problem in multicommodity flow. An efficient heuristic is presented, which, to our knowledge, for the first time allows to consider the whole set of violations simultaneously and thereby to find a minimized number of changes to the RSN structure in order to make it compliant with the initial security requirements of the DUT and prevent the information leakage through the scan chain.

Preprint

Security Preserving Integration and Resynthesis of Reconfigurable Scan Networks

Natalia Lylina[1], Ahmed Atteya[1], Chih-Hao Wang[2], Hans-Joachim Wunderlich[1]
[1]ITI, University of Stuttgart, Pfaffenwaldring 47, D-70569 Stuttgart, Germany
[2]Department of Electrical Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan
{lylina, atteyaad, wangco,wu}@informatik.uni-stuttgart.de

***Abstract*—Reliable operation, test, debug and diagnosis of complex integrated systems are ensured by embedded instruments, such as sensors, aging monitors or Built-In Self-Test (BIST) registers. Reconfigurable Scan Networks (RSNs) offer a flexible and efficient way to access such test instruments throughout the whole life-cycle. However, improper RSN integration might introduce additional connectivity properties to the device under test (DUT), which can be exploited to perform unauthorized access or cause information leakage. The existence of such additional connectivity through the RSN can compromise the security of the DUT and is considered as a security threat.**

In this paper, a method is presented to resolve all such security compliance violations. The problem is formulated in terms of Integer Linear Programming (ILP) as a minimum cut problem in multicommodity flow. An efficient heuristic is presented, which, to our knowledge, for the first time allows to consider the whole set of violations simultaneously and thereby to find a minimized number of changes to the RSN structure in order to make it compliant with the initial security requirements of the DUT and prevent the information leakage through the scan chain.

***Keywords*-Reconfigurable Scan Networks, Hardware Security, Secure DFT, Integer Linear Programming, Graph Partitioning**

I. Introduction

A great variety of instrumentation is being used to facilitate fast yield ramp-up through test and diagnosis as well as to ensure the reliable operation throughout the whole life-cycle by continuous monitoring and maintenance. Efficient access to the instruments can be provided by Reconfigurable Scan Networks (RSNs), which have been standardized by IEEE Std. 1687 [1] and IEEE Std. 1149.1 [2].

In order to guarantee the secure operation of a Device under Test (DUT), a system designer might develop the connections inside the DUT in a way that prevents the information leakage. A test engineer may not be fully aware of the designer's intentions and may therefore extend the connectivity properties of the DUT, while integrating the RSN, and thereby compromise the security of the design. An attacker might exploit the RSN, e.g., to gain unauthorized access to protected data or to alternate the system behavior [3–6]. Since the RSN must be available online throughout the whole life-time of a DUT to ensure its reliable and fault-tolerant operation, denying access to the RSN during the functional time is not an option. To secure the RSNs, authentication can be accomplished as in [7], and fine-grained protection schemes, such as Locking Segment Insertion Bits (Locking SIBs) [8, 9] and Secure SIBs [10], as well as schemes using linear feedback shift registers [11] or light weight encryption [12] have been proposed. Such schemes can make unauthorized access extremely difficult, but do not generally guarantee that the RSN will not extend the allowed information flow.

In [13], an automated algorithm is presented to accurately analyze the security compliance of a given RSN with the security properties of the original DUT and identify all the connectivities inside the RSN, which extend the allowed information flow of the DUT, as security compliance violations. To resolve these violations and prevent the information leakage through the scan infrastructure, all functional paths causing security violations must be cut physically, by reconnecting certain scan segments, or logically, by only performing security preserving accesses to the RSN. The logical filter-based approaches [14, 15] can efficiently handle the security requirements and allow only security preserving accesses to be performed. However, fulfilling some security requirements, e.g. restricting access to two subsequent scan segments, can make the corresponding instruments inaccessible through the RSN and can potentially lead even to a system failure. The existing physical approaches [16, 17] are able to handle any compliance violations, but resolve the violations in a local way, each time considering a single violation independently. Such strategy might require many changes to the RSN structure, especially if the number of violations is large as in [13], and affect the complexity of the retargeting mechanism, especially in case of homogeneous SIB-based RSN structures [18, 19].

In this work, we present an automated resynthesis approach for RSNs, which is applicable even to a high number of security compliance violations. The violations due to the RSN integration are considered all together, which allows to dramatically reduce the number of changes in the RSN structure compared to [16, 17]. The problem is formulated in terms of Integer Linear Programming (ILP) as a minimum cut problem in multicommodity flow [20, 21], which is known to be NP-complete. The presented algorithm utilizes a divide-and-conquer heuristic to find an appropriate solution and preserves an acceptable run-time even for the largest benchmarks. The major contributions of the presented approach are fourfold:

- Firstly, the integration of the resulting RSN does not extend the allowed connectivity properties of the DUT and is compliant with its initial security properties, specified by a system designer.
- Secondly, the presented approach preserves the accessibility of the instruments through the RSN.
- Thirdly, the effectiveness of the applied heuristic allows to resolve a high number of violations with a few structural changes in an acceptable time even for the large

978-1-7281-9113-3/20/$31.00

RSN designs.

- Lastly, the resynthesis approach is adjustable for the needs of a test engineer and allows to consider the explicit requirements, e.g. to the maximum access latency for specific safety-critical instruments as well as restrictions for hardware overhead, such as the maximal quantity and complexity of the scan multiplexers.

The remainder of this paper is organized as follows. In Section II the basic terminology on Reconfigurable Scan Networks is given. Section III introduces the modeling of the problem. Section IV provides an overview of the security compliance analysis approach. In Section V the details of a method are given to resolve the security violations and integrate an RSN in a security preserving way. In Section VI an approach to reintroduce the accessibility of all scan segments is presented. An evaluation of experimental results is given in Section VII. Finally, Section VIII concludes the paper.

II. Reconfigurable Scan Networks

Reconfigurable Scan Networks (Fig. 1) are constructed using *Scan Primitives* as the basic components, which comprise the following:

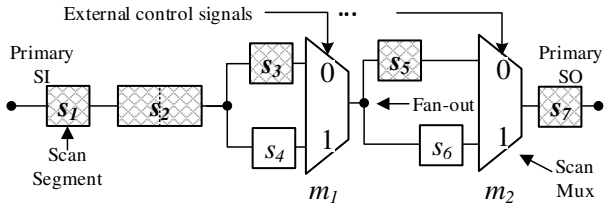

Fig. 1. Reconfigurable Scan Network (RSN)

- *Scan Segments* are the scan primitives, used to shift the data from Scan-In (SI) to Scan-Out (SO) and operated by external control signals: *Capture*, *Shift* and *Update*. A scan segment consists of a *Shift Register* with a defined length n and of an optional *Shadow Register*.
- *Scan Multiplexers* are the scan primitives, used to control an activated path by choosing one of the scan input branches. The address control of a scan multiplexer can be driven by an internal or an external control signal and is used to specify the selected scan input.
- *Segment Insertion Bits* (SIB) are the scan primitives, used to include or exclude the scan segments from a path.
- *Wrapped Instruments* are basic components, which consist of a *Scan Segment* and a corresponding *Instrument*, such as an aging monitor, a sensor or a Built-In Self-Test register, as well as the connections between them.

An acyclic path through selected *Scan Primitives* between primary *SI* and primary *SO* is called an *Active Scan Path* (ASP). In the provided example the initial active scan path is going from SI to SO through the scan segments s_1, s_2, s_3, s_5 and s_7. The state of all sequential elements and external control signals defines the current *Scan Configuration* $c \in C$, where C denotes the set of Scan Configurations. In a valid *Scan Configuration*, only one *ASP* can exist.

An access to an RSN can be divided into three phases, namely *capture*, *shift* and *update* phases, together forming a CSU-operation. During the *capture*-phase test data from the attached instruments is read. Data is being shifted from *SI* to *SO* during the *shift*-phase. Then newly shifted-in data is latched into the *Shadow Registers* of the *Scan Segments* during the *update*-phase. The data in the *Shadow Registers* either can be used to generate control signals or can be written to the connected *Instruments*. Only *Scan Segments* included into an *ASP* can be used to read or write data into the *Instruments*. A sequence of CSU-operations can be used in an RSN to transport the data. The computation of the control patterns for such a sequence is called *Retargeting*.

Definition 1: The transition relation T is defined as the set $T \subset C^2$ that consists of all pairs of scan configurations (c_1, c_2) such that $c_2 \in C$ can be reached from $c_1 \in C$ within a single CSU-operation.

The Capture-Shift-Update (CAM)-accurate model of [22] is used to represent the structural and functional properties of RSNs and is defined as follows.

Definition 2: The CSU-accurate model (CAM) of an RSN is a tuple $M := \{ST, In, C, c_0, T\}$, such that the set ST is used to represent all sequential elements, the set In denotes external control inputs, the set C represents all possible scan configurations, c_0 is the initial scan configuration, and T is used to denote the transition relation.

More details on CAM-accurate modeling can be found in [22].

III. Modeling

The analyzed system consists of two major parts, the device under test (DUT) and the RSN. The DUT includes the set of functional registers R and the set of instruments I. The RSN is used to access the instruments through the set of scan segments S.

Example *In Fig. 2 an example of an analyzed system is shown, which is used throughout the paper to demonstrate the most important concepts. The DUT is depicted in the upper part of the figure, whereas the RSN is drawn in the lower part of the figure.*

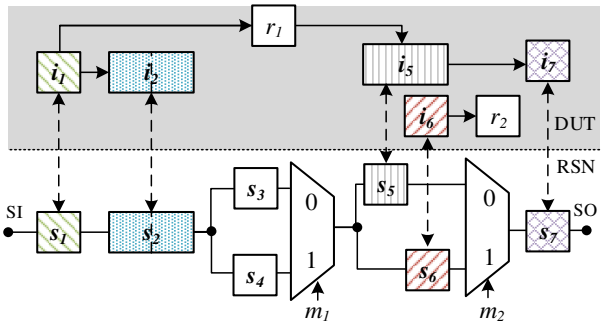

Fig. 2. Running example

The DUT is modeled as a directed graph $G^{DUT} := (V^{DUT}, E^{DUT})$. The vertex set V^{DUT} represents the registers of the instruments $i \in I$, being accessed through the RSN. Edges $e \in E^{DUT}$ correspond to the direct functional dependencies between the vertices $v \in V^{DUT}$ or dependencies through the functional registers.

Functional reachability properties of the DUT are extracted from the structural circuit description and augmented with the explicit security requirements, specified by the system designer as described in [13]. The security specification may explicitly define the allowed connectivities between the instruments and include the information on the trustworthiness of specific IP cores and the data confidentiality as in [23]. The reachability properties together with security specification form the set of security properties of the DUT. In order to consider functional connections of the DUT, false path analysis [24–27] can be applied. The verification of possible glitch propagation through false paths, as well as the security analysis of the DUT, must be accomplished by the system designer and lay out of scope of this work.

A directed graph $G^{RSN} := (V^{RSN}, E^{RSN})$ is used to model an RSN. Each vertex $v \in V^{RSN}$ represents a single scan segment s_j or scan multiplexer m_j, a primary *SI* or *SO*. Each edge $e \in E^{RSN}$ represents a direct connection between the scan primitives.

The combined system is modeled as a directed graph $G := (V, E)$ with vertices V and edges $E \subset V^2$, which is denoted as a *System Graph*. The vertex set V and the edge set E of the *system graph* are defined as follows:

$$V := V^{DUT} \cup V^{RSN} \tag{1}$$

$$E := E^{DUT} \cup E^{RSN} \cup E^{connect}, \tag{2}$$

where $E^{connect}$ defines the set of edges used to access the instruments I through the scan segments S of the RSN.

Example (continued) *The system graph for the running example is depicted in Fig. 3. The DUT graph is represented in the upper part of the figure. The vertex set V^{DUT} consists of the vertices $(i_1, i_2, i_5, i_6, i_7)$. An edge between the vertices i_1 and i_2 represents a direct connection between the corresponding instruments of the DUT. An edge from i_1 to i_5 represents the functional reachability of the instrument i_5 from the instrument i_1 through the functional register r_1. The RSN graph is represented in the lower part of the figure.*

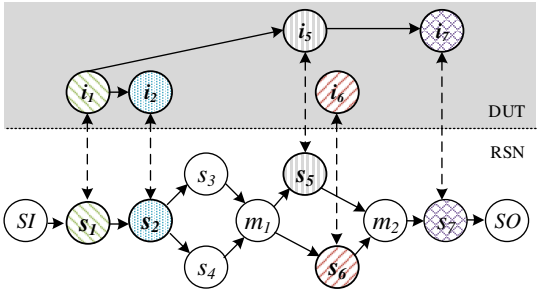

Fig. 3. System graph for the running example

IV. Security Compliance Verification

In order to keep this paper self-contained, this section briefly summarizes the security compliance verification approach of [13], which serves as the basis for the security preserving resynthesis of RSNs below.

A. RSN Reachability Computation

The functional dependencies in the RSN described in [13] consider possible structural cycles in the G^{RSN} as well as the capability of the RSN to transfer the data using multiple scan configurations through retargeting. The reachability analysis of the RSN can be divided into the following steps:

- *Compute structural pairwise connections:* The transitive closure over the RSN graph is computed and the set of all structurally possible connections between the scan primitives is determined.
- *Reduce connections to valid scan dependencies:* Logic signals, used to include specific scan primitives into an ASP and to control the multiplexers, are analyzed and the subset of connections, which belongs to valid scan configurations, is calculated.
- *Determine dependencies between scan configurations:* Reachability dependencies within single scan configurations are combined to an unbounded sequence of scan configurations and the set of functionally possible connections is computed.
- *Extract connections corresponding to the instruments:* Based on the reachability properties of the RSN, the connectivity between instruments is determined.

For all $v_j \in V$, the set of *functional predecessors* in G is defined as $p(v_j, G) \subset V$. For all $v_j \in V, v_k \in p(v_j, G)$ data transfer from v_k to v_j is functionally possible. The set of *direct functional predecessors* of v_j is defined as $pd(v_j, G)$. The set of all *functional successors* $s(v_j, G)$ and the set of *direct functional successors* $sd(v_j, G)$ are also defined for all v_j. A functional path $path_{k,l}$ in G between vertices $v_k, v_l \in V$ is a sequence $v_k ... v_l$ of vertices, such that $\forall v_j, v_{j+1} \in path_{k,l}$: $v_j \in pd(v_{j+1}, G)$, where j, $j+1$ are the indices of the vertices in $path_{k,l}$. A functional path can be represented as a connected sequence of sub-paths of a smaller size, e.g.: $path_{a,c} = path_{a,b} \cup path_{b,c}$.

B. Compliance Verification

The reachability of the instruments in the combined system considers the reachability properties of the RSN and of the DUT as well as hybrid paths, crossing both the DUT and the RSN. Security preserving RSN integration must not provide any new possibilities for an attacker to transfer the data inside the DUT. The compliance of a given RSN with the DUT is checked, and additional functional connectivity properties, which have been introduced into the allowed information flow of the DUT due to the RSN integration, are being identified.

Definition 3: A security violation $viol_{k,l}$ is a pair of vertices (s_k, s_l) in G^{RSN}, such that the functional path between the corresponding scan segments is sensitizable and extends the functional connectivity properties of the DUT due to the RSN integration.

The list of security violation warnings is generated and serves as an input for a security preserving RSN resynthesis. More details on the RSN reachability analysis as well as on the security compliance verification of RSNs are given in [13].

Example (continued) *In Fig. 3 a functional path between the scan segments s_2 and s_6 extends the connectivity properties of*

the DUT by making the instrument i_6 functionally reachable from the instrument i_2 through the RSN. Functional connectivities between s_1 and s_6, s_2 and s_5, s_2 and s_6, s_2 and s_7, s_6 and s_7 are considered as security compliance violations.

V. Security Preserving Resynthesis

In this section, an automated security preserving resynthesis approach for RSNs is described and formulated in terms of ILP. This approach is able to resolve the whole set of the security compliance violations, which has been identified using the verification method presented above.

A. Multicommodity Flow Problem

A single commodity $(vs - vt)$ in a directed flow network graph $G := (V, E)$ with vertex set V and edge set E is defined as a pair of vertices $vs, vt \in V$, such that vs is a source, vt is a destination. A $(vs - vt)$ cut of G is a subset $V_{cut} \subset V$, such that in the resulting graph $G_{cut} := (V \setminus V_{cut}, E)$ the vertex vt is not reachable from vs.

If k multiple commodities $(vs_k - vt_k)$ exist in a graph $G := (V, E)$, a cut in a multicommodity flow network is computed. Subset $V_{cut} \subset V$ is a cut of a multicommodity flow network, if its removal from the vertex set V would preclude the connectivity between the source vs_k and destination vt_k for all the commodities $(vs_k - vt_k)$.

The size of the cut can be minimized by means of Integer Linear Programming (ILP). For each vertex $v_j \in V$ a variable x_j is defined, such that $x_j := 1$, if a vertex v_j is removed from the vertex set and $x_j := 0$ otherwise. The objective function of a 0-1 linear programming algorithm minimizes the number of vertices in the cut V_{cut} and is formulated as follows:

$$minimize(\sum_{j=1}^{n} x_j), n = |V| \tag{3}$$

Subject to the constraints:

$$\sum_{j=1}^{n} (a_j)_l x_j \geq 1, l \in 1...m \tag{4}$$

$$x_j \in \{0, 1\}, j \in 1...n \tag{5}$$

The constraints describe all possible paths between the vertex pairs (vs_k, vt_k), representing the commodities, and m denotes a total number of such paths. Binary coefficients $(a_j)_l$ define if a vertex v_j, represented by x_j, belongs to a path $path_l$. If at least one vertex v_j is removed from the path, the connectivity through this path is prevented.

B. Overview

The problem of security preserving resynthesis of RSNs is formulated as a minimum cut problem in a multicommodity flow. Since the multi-commodity flow problem is NP-complete, it is not feasible to solve it straightforward [20, 21] and in order to achieve acceptable run-time and memory consumption the exact solution can be sacrificed. To overcome the exponential complexity of enumerating all paths and to find an appropriate solution, a heuristic based on the divide-and-conquer approach is presented, and at each step a problem of a smaller size is resolved:

- A set of connectivities SC, $(vs_k, vt_k) \in SC, vs_k, vt_k \in V^{RSN}$, extending the connectivity properties of the DUT, is provided by the security compliance verification as a list of security violation warnings.

  Example (continued) *All the functional paths between vertex pairs s_1 and s_6, s_2 and s_5, s_2 and s_6, s_2 and s_7, s_6 and s_7 extending the connectivity, must be cut.*

- A cut $(vm_1, ...vm_r)$, precluding the connectivities, is computed. For each intermediate vertex $vm_j \in V^{RSN}$, on the path from vs_k to vt_k is to decide, whether:
  1) vm_j belongs to the cut $(vm_1, ...vm_r)$, or
  2) all the paths between vs_k and vm_j are cut, or
  3) all the paths between between vm_j and vt_k are cut.

  In Fig. 4, a single intermediate vertex vm_j can be removed to cut all the paths between vs_k to vt_k.

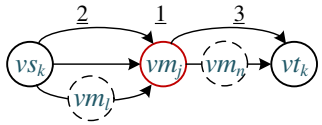

Fig. 4. Node cutting options

- The solution is adjusted recursively and a possibly small vertex cut is chosen. If, for some connectivity (vs_k, vt_k), at some iterative step, vs_k is a direct predecessor of vt_k, the vertex vs_k is put into the cut. Although the applied heuristic does not guarantee an optimal solution, experimental results show, that using the presented approach, a large number of violations can be resolved by applying just a few changes to the RSN structure.
- From the connectivity perspective, the removal of a vertex vs_k is equivalent to the removal of its all outgoing edges. So, instead of the vertices in the cut, all their outgoing edges are removed from the graph, to preserve the vertices, corresponding to the scan elements.
- The accessibility of the affected scan elements is reintroduced as described in Section VI.A. The completeness of the algorithm is discussed in Section VI.B.

C. Divide-and-Conquer Heuristic

To resolve the violations, the RSN is represented as a graph at various abstraction levels, starting from a lower-granular *Level-0 graph*, which is represented in Fig. 5 for the running example, and ending with a high-granular G^{RSN} graph.

The *Level-0 graph* represents all security compliance violations (connectivities from SC), as the pairs of vertices and denotes the existence of violating functional paths in-between as the edges. The sources of security violations are represented by the set VS and the destinations - by the set VT. A scan segment can serve as a source of one security compliance violation and as a destination of another violation, so that, the sets VS and VT are not forced to be disjoint. It is also noteworthy to mention that it is not required to cut all the possible paths between all the combinations of vertices in the sets VS and VT. Only the paths between the pairs of vertices,

which have been considered as the violations and denoted as an edge in the graph representation, must be cut.

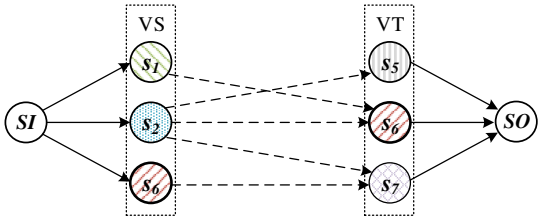

Fig. 5. Level-0 graph

Example (continued) *The sources of violations are the vertices s_1, s_2 and s_6 and the destinations – s_5, s_6 and s_7. The vertex s_6 is a source of one violation and a destination of two other violations. Since the functional paths between the scan segments s_1 and s_5 do not violate the compliance with the DUT, it is not required to cut the functional paths between them, although s_5 is functionally reachable from s_1.*

At each step, starting from *Level-1*, we decide, which functional paths must be cut. At each further step this decision is adjusted to determine, where exactly the functional path must be cut. The first step of the algorithm, corresponding to the construction of the *Level-1* graph (Fig. 6) for the "running example" is described here in detail and all further steps follow the same idea. The number of required steps depends on the size of the graph and the maximal distance between the source and destination of a security compliance violation in terms of the number of vertices in the RSN graph.

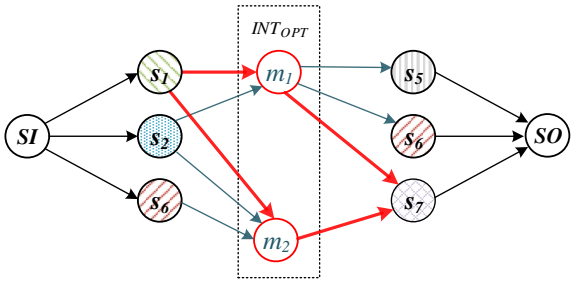

Fig. 6. Level-1 graph

The edge set of the *Level-1 graph* represents all functional paths between the sources VS, intermediate vertices and destinations VT. Let the set of intermediate vertices INT include all the vertices from V^{RSN}, which are located on the functional paths, violating the security compliance with the DUT. If all the vertices from INT would be included for constructing a graph, the run-time of the minimum cut problem even on the *Level-1* graph can be exponential from the whole number of vertices in the RSN graph.

To prevent this, the set of intermediate vertices is optimized during the construction of each *Level-j* graph. E.g., for *Level-1* an optimized set of intermediate vertices $INT_{OPT} \subset V^{RSN}$ is built, which covers all functional paths between all pairs of vertices, which cause security violations at *Level-0* with a small number of vertices.

Example (continued) *Since the violations in the running example are distributed through the whole RSN graph, the set INT would include all vertices from V^{RSN}, except the vertices SI, SO, s_1 and s_7.*

All functional paths for all violations can be intuitively covered by just two vertices, m_1 and m_2, which constitute the set INT_{OPT}. The edges in the graph show the existence of a functional path between the vertices. Any path from s_1 to s_7 includes at least one of the intermediate vertices (depicted by bold line in Fig. 6), whereas all the paths from s_2 to s_5 cross the vertex m_1 only. The formal way to construct INT_{OPT} is shown below.

Since the accuracy of this step only affects the number of recursive steps needed to perform the main algorithm, an efficient polynomial-time heuristic is applied to minimize the set of intermediate vertices and is described below:

- Firstly, for each intermediate vertex $v_j \in INT$ a number of *violation sources* $v_l \in VS$, having at least one functional path to v_j, is used as a global weight $Weight_G(v_j)$. This value considers all the violations simultaneously and determines, how often a certain intermediate vertex belongs to any path, introducing additional connectivity to the DUT.
- Secondly, an optimized set of intermediate vertices INT_k is constructed for each violation $viol_k$. This set covers all functional paths from vs_k to vt_k with a small number of vertices. The sub-graph G_k^{RSN} is defined for each violation and contains the vertices vs_k, vt_k and all vertices from V^{RSN}, which are traversed by at least one functional path between vs_k and vt_k, as well as all the edges from E^{RSN} connecting those vertices. For any intermediate vertex v_j the sets of predecessors $p(v_j, G_k^{RSN})$ and successors $s(v_j, G_k^{RSN})$ in a sub-graph are computed. The set of covered vertices COV_k for $viol_k$ is empty at this step.
  A local weight of a vertex v_j in a sub-graph is defined as a number of predecessors and successors in this sub-graph:
$$Weight_L(v_j, G_k^{RSN}) := |p(v_j, G_k^{RSN})| + |s(v_j, G_k^{RSN})| \quad (6)$$
- If a certain vertex v_j has been included into an optimized set of vertices INT_{k1} for an already considered violation $viol_{k1}$, it is included into an optimized set INT_{k2} for another violation $viol_{k2}$ with a higher probability. This prevents the size of the set INT_{OPT} from increasing. For each intermediate vertex v_j a boolean function $sel(v_j)$ defines, how many times this vertex has already been included into the optimized sets of vertices for the considered violations.
- For each violation $viol_k$ the vertex v_j with a highest value of a cost function, considering both the global and the local weight, and the value of the $sel(v_j)$ function:
$$cost_(v_j, G_k^{RSN}) := f(Weight_G(v_j), Weight_L(v_j, G_k^{RSN}), sel(v_j)) \quad (7)$$
  is selected and added to an optimized set INT_k and to the

set of covered vertices COV_k. The functional successors and predecessors of v_j in G_k^{RSN} are also added to the set of covered vertices. Then the next vertex with the second highest value of a cost function is selected and the procedure is repeated, until all intermediate vertices for a violation are in the set of covered vertices COV_k. The same procedure is repeated for all violations.

Example (continued) *The security violation from s_1 to s_6 is considered first. A sub-graph for this violation is shown in Fig. 7.a).*

- *The vertex f_2 has the global weight of two, since the functional paths from the source vertices, s_1 and s_2 to f_2 exist in the RSN.*
- *All the vertices in the subgraph are either functional successors or functional predecessors of f_2, so that f_2 has the highest local weight in the subgraph.*
- *Since no other violation has been considered, the value of $sel(v_j)$ function is zero for all the vertices in the subgraph.*

The vertex m_1 has the highest value of a cost function. Assume that m_1 is selected and added to the optimized set. The value of $sel(m_1)$ function is incremented. Since all the paths from s_1 to s_6 are covered with a vertex m_1, the computation for this violation converges.
Next, violation from s_2 to s_5, is considered. Since, now m_1 has $sel(m_1) := 1$, it is automatically selected to form the optimized set for the second violation. All other violations, except the violation from s_6 to s_7, shown in Fig. 7.b), are also covered with m_1. The vertex m_2 is selected to cover this violation.

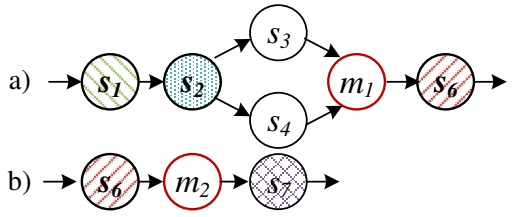

Fig. 7. Subgraph for a violation a) s_1 to s_6 b) s_6 to s_7

- The optimized set of intermediate vertices INT_{OPT} is constructed as a union of the optimized sets for single security violations, K being the total number of violations:

$$INT_{OPT} = \bigcup_{k=1}^{K} INT_k \tag{8}$$

Example (continued) *The optimized set of intermediate vertices consists of the vertices m_1 and m_2, representing the optimized sets for single violations.*

Finally, for each $v_j \in INT_{OPT}$, such that for a certain security violation $viol_k$ a functional path from a vs_k to v_j and a path from v_j to vt_k exists, it is to decide, whether to cut the path before v_j ($path_{s_k,j}$) or after it ($path_{j,t_k}$). To define the minimized set of edges to remove from the graph, a minimum cut problem in a multicommodity flow of a smaller size is solved by means of ILP as described above.

If m certain functional paths $(path_1, ..path_m)$ are cut, at the next step it should be decided with a higher granularity, where exactly are they cut. At each iterative step the solution is incrementally improving and an optimized set of intermediate vertices INT_{OPT} is recomputed considering the paths $(path_1, ..path_m)$, removed at the previous step, as security violations. At each further level the lengths of the paths, which must be cut, are gradually decreasing and the granularity of the RSN graph is increasing until the exact vertices in a high-granular graph G^{RSN}, are found.

Example (continued) *The min-cut in multicommodity flow is run on the Level-1 graph (Fig. 6), which includes the source vertices s_1, s_2 and s_6, intermediate vertices m_1 and m_2, and destination vertices s_5, s_6 and s_7. After applying the algorithm, all violating connectivities are cut, and the all paths before m_1, and after m_2 are removed.*

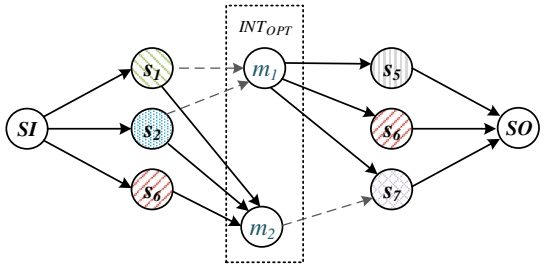

Fig. 8. Level-1 graph with the cut

Next, a Level-2 graph is constructed (Fig. 9). It is necessary now to decide, where exactly to cut. Since m_2 is a direct predecessor of s_7, the vertex m_2 is added to the cut and its outgoing edge to s_7 is removed from the graph representation. All the functional paths from s_1 to m_1 and from s_2 to m_1 are covered by the vertices s_3, s_4 and the optimized set of intermediate vertices INT_{OPT} includes the vertices s_3, s_4. A min-cut algorithm is applied once again and the paths from s_3 to m_1 and from s_4 to m_1 are removed to resolve the remaining violations. Finally, vertices s_3, s_4 and m_2 are in the cut and their outgoing edges are removed.

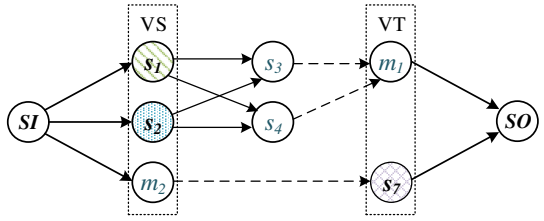

Fig. 9. Level-2 graph with the cut

Fig. 10 demonstrates a resulting graph after resolving all security violations. The edges between the scan segment s_3, s_4 and the scan multiplexer m_1, and between the scan multiplexer m_2 and the scan segment s_7, are removed from the edge set.

VI. Reintroducing the Accessibility

After resolving the violations in the RSN not all the scan segments, represented by the vertices $v \in V_{secure}^{RSN}$, may be accessible anymore through the functional ASPs going from *SI* to *SO* as shown in Fig. 10.

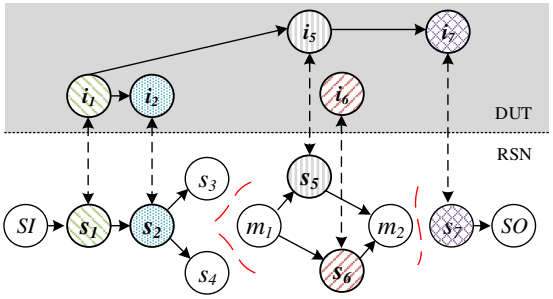

Fig. 10. Graph G^{RSN}_{secure} with no security violations

The accessibility of the corresponding scan segments is recovered in an automated way considering such optimization criteria as access latency or hardware overhead, and the final RSN graph $G^{RSN}_{final} := (V^{RSN}, E^{RSN}_{secure} \cup E^{RSN}_{access})$ is generated. The size of the set of augmenting edges is limited by $|E^{RSN}_{access}| \leq 2 * m$, where m is number of edges, removed at the previous step. The edges are reintroduced sequentially. For each vertex v_j, having no functional successors, the accessibility is reintroduced through the following steps:

- The subset of the possible compliance-preserving successors $SEC(v_j) \subset V^{RSN}$ is computed, such that for any $v_l \in SEC(v_j)$, a direct functional connection from v_j to v_l would be compliant with the security properties of the DUT and a path from v_l to SO exists. The functional connection from v_j to v_l, as well as the connections between all the pairs of the functional predecessors of v_j and the functional successors of v_l must not extend the connectivity of the DUT. For each v_j, at least the vertex, corresponding to SO, is compliance-preserving, and the set $SEC(v_j)$ is not empty.
- A single vertex v_l is selected from $SEC(v_j)$ to become a successor of v_j and an edge $e_{j,l}$ is added to the edge set of the graph G^{RSN}_{secure}. If multiple choices are possible to reintroduce the accessibility of a certain vertex, additional requirements to access latency or acceptable hardware overhead of the RSN can be considered. Time to access specific scan segments could be crucial, especially in safety-critical applications. One can decrease the access latency by constructing more short paths from SI to SO, which would, however, add additional scan multiplexers to the RSN structure, and increase the hardware overhead. The hardware overhead by the RSN integration must be limited as well. Since the number of scan registers will remain the same, the difference in the overhead will be mostly caused by the number of scan multiplexers in the given RSN and their complexity. Additional constraints for the choice of an appropriate successor-vertex, can be used to keep the complexity of the retargeting mechanism rather low, while still obtaining a compliant RSN. E.g., for the SIB-based tree structures, the connections must be adjusted in a way that the resulting RSN still retains a tree structure as far as possible.
- The functional reachability of the vertices v_j and v_l, as well as the reachability of the functional predecessors of v_j and the functional successors of v_l is adjusted, considering the newly introduced edge $e_{j,l}$.

The process is repeated for the next vertex $v_m \in V$, having no functional successor, until all the vertices have at least one successor. The same idea is applied to reintroduce the accessibility of the vertices, which do not have any functional predecessor. The accessibility of the affected vertices can be then verified as in [22].

Example (continued) *The functional reachability properties of the graph (in Fig. 10) are recomputed. Since the vertex m_2 is now not accessible, it is necessary to reintroduce a functional path from m_2 to SO. The connection from vertex m_2 to vertex SO is added to the RSN graph, since this connection is compliant with the requirements of the DUT. The functional reachability of the vertices m_2 and SO, as well as the reachability of the predecessors of m_2 is adjusted corresponding to the newly added edge between m_2 and SO. Since the vertex m_1 represents a scan multiplexer and neither has any incoming edges inside the RSN graph nor is used to transfer data towards the DUT, it is removed from the graph representation. The procedure above is applied again, when the connections from vertex SI to vertices s_5 and to s_6, and from s_3 and s_4 to m_2 are consequently added to the graph G^{RSN}_{secure}. The vertex s_7 is now connected to SO through m_2, since multiple incoming edges are only allowed for the vertices, corresponding to multiplexers. The graph G^{RSN}_{final}, which is shown on Fig. 11, represents the resulting RSN.*

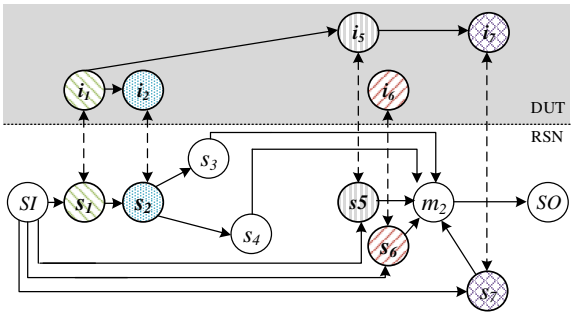

Fig. 11. Accessible graph G^{RSN}_{final} with no security violations

A. Completeness of the Algorithm

Fig. 12 represents a so-called "Verify ↔ Resolve loop", which is used to guarantee that all the existing security violations are resolved using the presented heuristic.

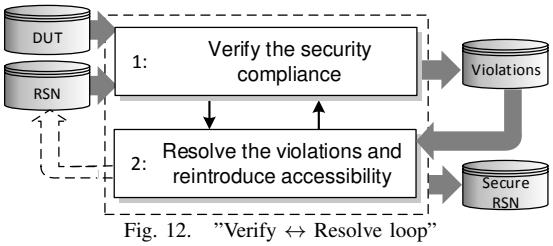

Fig. 12. "Verify ↔ Resolve loop"

At *Step 1*, the verification of the initial RSN is performed and the compliance violations are identified. After resolving the violations and reintroducing the accessibility (*Step 2*), the compliance of the resulting RSN is validated again using the verification approach (*Step 1*). This step shows whether all security compliance violations have been resolved:

- If no additional connectivity is introduced by the RSN integration, the integrated RSN is compliant with the DUT and the computation converges.
- If the RSN integration still extends the connectivity of the DUT, *Steps 2* and *1* are repeated until all the violations are resolved.

It is guaranteed that the process terminates with a compliant RSN. In the worst case, the procedure would end up with a parallel RSN structure, where all the scan segments, accessing the instruments, are located in the individual branches, in order to guarantee the the security compliance with a given DUT. However, for the most of the considered cases, just one iteration of the flow was enough to resolve all the violations. For some cases, after the first iteration, a minor number of violations were still present. For those cases, the whole resynthesis process required a couple of subsequent iterations of the "Verify $\leftrightarrow$ Resolve" loop.

VII. Evaluation

A. Experimental Setup

The effectiveness of the proposed method has been evaluated on the subset of benchmarks from the Bastion benchmark set [19] and on the whole DATE'2019 benchmark set [17]. All experiments have been conducted on Intel(R) Xeon(R) W-2125 CPU at 4.00GHz with 132 GB of main memory, using the *eda1687* framework [28] to model the RSN. The optimization has been performed using the tool *Gurobi* [29] and was aborted if a timeout of 30 hours was reached.

The evaluated benchmarks provide access to boundary and internal scan chains and have a hierarchical structure. The characteristics of the benchmarks are given in Table I. For all benchmarks the number of scan multiplexers (Column 2), SIBs (Column 3), scan cells (Column 4) and the highest hierarchy level (Column 5) are given.

TABLE I
Characteristics of benchmarks

Design(1)	#muxes(2)	#sibs(3)	#scan cells(4)	#level(5)
BasicSCB	10	-	176	4
Mingle	13	10	22	3
TreeFlat	24	12	101	2
TreeUnbalanced	28	28	41,887	11
TreeBalanced	46	43	5,581	7
TreeFlat_Ex	60	57	5,194	5
q12710	25	25	26,183	2
a586710	47	-	41,682	3
p34392	142	-	23,261	3
t512505	160	-	77,006	2
p22810	283	283	30,111	3
p93791	653	-	98,637	3
N17D3	8	7	447	3
N32D6	10	13	96,135	4
N73D14	17	29	218,823	11
N132D4	40	39	2,912	5
MBIST_1_5_5	15	8	548	4
MBIST_1_5_20	15	8	1,523	4
MBIST_1_20_20	45	23	6,068	4
MBIST_2_5_5	28	16	1,091	4
MBIST_2_5_20	28	16	3,041	4
MBIST_2_20_20	88	46	12,131	4
MBIST_5_5_5	67	40	2,720	4
MBIST_5_20_20	217	115	30,320	4
MBIST_5_100_20	1,017	515	151,520	4
MBIST_5_100_100	1,017	515	671,520	4
MBIST_20_20_20	862	460	121,265	4
MBIST_55_20_5	2,367	1,265	118 970	4
MBIST_100_20_5	8,102	2,300	216,305	4
MBIST_100_100_5	20,102	10,300	1,080,305	4

B. Experimental Results

Table II summarizes the experimental results for security preserving resynthesis of RSNs. The reachability analysis has been conducted for all benchmarks as in [13] and functionally possible connections between the scan primitives have been identified. The security compliance analysis as well as the security preserving RSN transformation consider the retargeting capabilities of RSNs and the data propagation through the hybrid paths. Since a precise security analysis of the DUT lays in hands of system designer, the complexity of the DUT benchmarks is not important for the experimental setup and all considered benchmarks have been integrated into the benchmarks from ISCAS'89 [30]. The connectivity properties of the ISCAS benchmarks have been used to represent the connections inside the DUT. Each flip-flop in the ISCAS benchmark represented a single instrument, accessed by the RSN. The connections from a single scan segment to an instrument are assigned in a random manner.

The security compliance violations due to the integration of the given RSN design into a DUT have been identified (Column 2). All security violations have been resolved. A cost function for reintroducing the accessibility was chosen to minimize the latency. The third column represents the quantity of connections removed from the graph representation to resolve the violations. The fourth column shows the number of connections added into the graph representation to reintroduce the accessibility.

The experimental results show that the presented approach allows to resolve a large number of violations by applying a minor number of changes. Let the *relative number of changes* (Column 6), which must be applied to resolve one security compliance violation, be defined as follows:

$$connect_{rel} = connect_{cut}/viol_{correct}, \tag{9}$$

where $connect_{cut}$ defines the number of connections removed from the RSN and $viol_{correct}$ represents the number of corrected violations. The experimental results show that in average one change to the RSN structure was able to resolve 8,6 violations.

The run-time of the algorithm (Column 5) does not only depend on the number of resolved violations, but also on the size of the considered RSN graph. However, thanks to the applied divide-and-conquer-based heuristic, the exponential complexity is mitigated in an average case. The size of the graph affects the number of required recursive steps for the consecutive representation of the RSN graph. Even for the largest benchmarks the run-time does not exceed 10 minutes.

The benchmarks from the DATE'2019 benchmark set [17] have a highly hierarchical structure, which allows to efficiently process them despite their large size. Each benchmark *"MBIST_N_M_O"* (Fig. 13) handles N cores, each with M memory BIST controllers, each controlling O memory blocks, and is accessed through the Test Access Port (TAP) controller. All other control blocks are omitted and control signals are not shown for the better readability. In order to handle the high number of violations, the bigger RSN graph is split

TABLE II
SECURITY COMPLIANCE ANALYSIS AND RESOLVING ALL VIOLATIONS

Design(1)	$\#viol_{detect}$(2)	$\#connect_{cut}$(3)	$\#connect_{added}$(4)	t[m:s](5)	$connect_{rel}$(6)
BasicSCB	148	33	49	00:10	0.22
Mingle	158	27	53	00:15	0.17
TreeFlat	211	41	47	00:05	0.19
TreeUnbalanced	291	28	43	00:06	0.02
TreeBalanced	208	64	47	00:03	0.23
TreeFlat_Ex	458	76	107	00:23	0.16
q12710	316	45	55	00:53	0.14
a586710	1,625	36	52	00:31	0.02
p34392	2,227	63	100	02:33	0.03
t512505	1,005	174	237	00:46	0.17
p22810	2,114	119	275	05:55	0.06
p93791	355	15	4	06:31	0.04
N17D3	189	29	45	00:14	0.15
N32D6	285	33	48	00:02	0.11
N73D14	567	67	98	04:53	0.11
N132D4	932	115	181	09:28	0.12
MBIST_1_5_5	1,096	133	262	01:01	0.12
MBIST_1_5_20	387	53	105	02:14	0.13
MBIST_1_20_20	348	49	66	05:23	0.14
MBIST_2_5_5	1,057	125	20	00:05	0.11
MBIST_2_5_20	530	92	179	00:12	0.17
MBIST_2_20_20	938	182	175	00:59	0.19
MBIST_5_5_5	24,732	4,483	180	00:10	0.18
MBIST_5_20_20	60,519	5,279	18	02:01	0.08
MBIST_5_100_20	53,301	2,270	67	02:17	0.03
MBIST_5_100_100	57,338	734	207	03:15	0.03
MBIST_20_20_20	992	49	29	00:20	0.05
MBIST_55_20_5	678	61	75	01:14	0.09
MBIST_100_20_5	690	43	34	01:10	0.06
MBIST_100_100_5	2,164	64	58	03:01	0.03

into a number of smaller subgraphs and the violations are sorted, according to which subgraph they belong. E.g., the violation between the scan segments SA and SB can handled inside the subgraph, corresponding to the controller 0. The violation between the scan segments SB and SC, located in two different memory controllers of the core 0, can be handled by cutting the interconnection in the core-level graph, which models the memory controllers as single vertices.

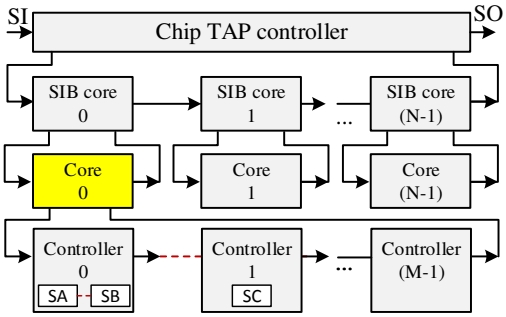

Fig. 13. Structure of MBIST benchmark.

C. Comparison to Previous Work

In order to show the efficiency of the proposed method, an additional experiment has been conducted. To provide a fair comparison to the results, which have been obtained in the prior work [17] in this field, a comparable number of security violations has been used (Table III). The RSN benchmarks have been connected to the generated randomized circuits. The number of such scan registers, where at least one flip-flop causes a security violation with a predecessor, before applying the resynthesis algorithm is computed for the presented approach (Column 2) and for the previous work (Column 5), and is denoted as $\#viol_{reg-pred}$.

A number of changes ($\#changes$), required to resolve all the violations is presented for both methods, respectively, in Columns 3 and 6. We computed the *relative number of changes* $changes_{rel-pred}$ for the presented approach (Column 4) as well as for the previous work (Column 7) as:

$$changes_{rel-pred} = \#changes/\#viol_{reg-pred} \quad (10)$$

A small number of violations to be resolved, can be considered as a worst case scenario for the presented algorithm, since the presented algorithm becomes more and more efficient with the increasing number of violations. Despite this fact, the experimental results show that the presented method requires 3.85 times less changes to the RSN structure in average. Reducing the number of structural changes to the RSN, while still obtaining a secure RSN, does not only require less costs in terms of hardware to apply the changes, but also allows to keep the structure of the resulting RSN similar to the initial RSN and requires less changes for the retargeting mechanism.

VIII. CONCLUSION

This paper presents an approach to automatically resolve all the violations, which have been identified by a previously proposed accurate security compliance analysis approach [13], considering the security properties of the device under test. A minimized set of the structural changes to a given RSN is identified, which allows to perform a security preserving RSN integration and thereby prevent the information leakage through the scan chain. The problem has been formulated in terms of Integer Linear Programming as a minimum cut problem in a multicommodity flow and avoids exponential computational complexity for an average case by applying an efficient heuristic, based on a divide-and-conquer approach, to obtain an approximate solution. The effectiveness and scalability of the algorithm has been evaluated on the set of

TABLE III
COMPARISON TO PREVIOUS WORK [17]

	Presented Approach			Previous Work		
Design(1)	$\#viol_{reg-pred}$(2)	$\#changes$(3)	$changes_{rel-pred}$(4)	$\#viol_{reg-pred}$(5)	$\#changes$(6)	$changes_{rel-pred}$(7)
BasicSCB	3.2	2.0	0.63	1.56	2.0	1.28
Mingle	5.0	3.2	0.64	2.21	2.5	1.13
TreeFlat	6.0	4.2	0.70	3.65	4.7	1.28
TreeFlat_Ex	12.2	8.0	0.66	8.45	12.1	1.43
TreeBalanced	8.8	6.8	0.78	7.22	9.0	1.25
TreeUnbalanced	8.2	6.2	0.75	6.27	7.6	1.21
q12710	5.6	5.2	0.93	5.20	7.1	1.37
a586710	7.2	4.2	0.57	5.89	8.4	1.43
p34392	18.0	12.4	0.67	11.26	21.4	1.90
t512505	12.0	6.0	0.50	12.44	24.9	2.00
p22810	33.4	17.2	0.51	21.75	41.9	1.93
p93791	52.2	23.4	0.44	40.51	79.5	1,96
MBIST_1_5_5	5.0	3.0	0.60	6.64	13.2	1.99
MBIST_1_5_20	8.2	4.8	0.58	9.00	39.5	4.39
MBIST_1_20_20	9.6	5.2	0.55	7.60	40.6	5.34
MBIST_2_5_5	9.2	5.0	0.55	6.18	11.7	1.89
MBIST_2_5_20	8.4	4.4	0.52	8.88	43.6	4.90
MBIST_2_20_20	9.8	6.2	0.63	2.45	2.6	1,06
MBIST_5_5_5	6.6	2.2	0.33	9.54	21.7	2.27
MBIST_5_20_20	8.2	4.0	0.48	4.56	12.9	2.82
MBIST_20_20_20	19.0	8.4	0.44	19.62	104.8	5.34

industrial benchmarks. The obtained run-time is acceptable. In average, for a large number of violations as in Table II, only one structural change is needed to resolve eight violations.

ACKNOWLEDGMENTS

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

BIBLIOGRAPHY

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

[2] "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.

[3] R. Elnaggar, R. Karri, and K. Chakrabarty, "Securing IJTAG against data-integrity attacks," in *Proc. IEEE VLSI Test Symp. (VTS)*, Apr. 2018, pp. 1–6.

[4] S. Kan and J. Dworak, "IJTAG Integrity Checking with Chained Hashing," in *Proc. IEEE Int'l Test Conf. (ITC)*, Oct. 2018, pp. 1–10.

[5] X. Ren, R. D. S. Blanton, and V. G. Tavares, "Detection of IJTAG attacks using LDPC-based feature reduction and machine learning," in *Proc. IEEE European Test Symp. (ETS)*, May 2018, pp. 1–6.

[6] 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.

[7] M. Portolan, V. Reynaud *et al.*, "Dynamic Authentication-Based Secure Access to Test Infrastructure," in *Proc. IEEE European Test Symp. (ETS)*, May 2020, pp. 1–6.

[8] A. Zygmontowicz, J. Dworak *et al.*, "Making It Harder to Unlock an LSIB: Honeytraps and Misdirection in a P1687 Network," in *Proc. Conf. Design, Automation & Test in Europe (DATE)*, Mar. 2014, pp. 195:1–195:6.

[9] J. Dworak, A. Crouch *et al.*, "Don't forget to lock your SIB: hiding instruments using P1687," in *Proc. IEEE Int'l Test Conf. (ITC)*, Sept. 2013, pp. 1–10.

[10] R. Baranowski, M. A. Kochte, and H. Wunderlich, "Fine-grained access management in reconfigurable scan networks," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 34, no. 6, pp. 937–946, June 2015.

[11] H. Liu and V. D. Agrawal, "Securing IEEE 1687-2014 Standard Instrumentation Access by LFSR Key," in *Proc. EEE Asian Test Symp. (ATS)*, 2015, pp. 91–96.

[12] B. Thiemann, L. Feiten *et al.*, "On Integrating Lightweight Encryption in Reconfigurable Scan Networks," in *Proc. IEEE European Test Symp. (ETS)*, 2019, pp. 1–6.

[13] N. Lylina, A. Atteya *et al.*, "Security Compliance Analysis of Reconfigurable Scan Networks," in *Proc. IEEE Int'l Test Conf. (ITC)*, Nov. 2019, pp. 1–9.

[14] R. Baranowski, M. A. Kochte, and H.-J. Wunderlich, "Access Port Protection for Reconfigurable Scan Networks," *Journal of Electronic Testing: Theory and Applic. (JETTA)*, vol. 30, no. 6, pp. 711–723, 2014.

[15] A. Atteya, M. A. Kochte *et al.*, "Online Prevention of Security Violations in Reconfigurable Scan Networks," in *Proc. IEEE European Test Symp. (ETS)*, May 2018, pp. 1–6.

[16] P. Raiola, M. A. Kochte *et al.*, "Detecting and Resolving Security Violations in Reconfigurable Scan Networks," in *Proc. IEEE Int'l Symp. on On-Line Testing And Robust System Design (IOLTS)*, Jul. 2018, pp. 91–96.

[17] P. Raiola, B. Thiemann *et al.*, "On Secure Data Flow in Reconfigurable Scan Networks," in *Proc. Conf. Design, Automation & Test in Europe (DATE)*, Mar. 2019, pp. 1016–1021.

[18] A. Ibrahim and H. G. Kerkhoff, "Analysis and design of an on-chip retargeting engine for IEEE 1687 networks," in *Proc. IEEE European Test Symp. (ETS)*, May 2016, pp. 1–6.

[19] A. Tsertov, A. Jutman *et al.*, "A suite of IEEE 1687 benchmark networks," in *Proc. IEEE Int'l Test Conf. (ITC)*, Nov. 2016, pp. 1–10.

[20] L. R. Ford and D. R. Fulkerson, "A Suggested Computation for Maximal Multi-Commodity Network Flows," *Management Science*, vol. 5, pp. 97–101, 1958.

[21] A. Hall, S. Hippler, and M. Skutella, "Multicommodity Flows Over Time: Efficient Algorithms and Complexity," *Theoretical Computer Science*, vol. 379, no. 3, pp. 387 – 404, 2007.

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

[23] M. A. Kochte, M. Sauer *et al.*, "Specification and Verification of Security in Reconfigurable Scan Networks," in *Proc. IEEE European Test Symp. (ETS)*, 2017, pp. 1–6.

[24] K. Nakamura, K. Takagi *et al.*, "Waiting false path analysis of sequential logic circuits for performance optimization," in *IEEE/ACM Int'l Conf. on Computer-Aided Design. Digest of Technical Papers (ICCAD)*, Nov 1998, pp. 392–395.

[25] Z. Hanna and V. M. Purri. (2013, Apr.) Verifying Security Aspects of SoC Designs with Jasper App. [Online]. Available: https://www.edn.com/

[26] A. Nahiyan, M. Sadi *et al.*, "Hardware trojan detection through information flow security verification," in *Proc. IEEE Int'l Test Conf. (ITC)*, Oct. 2017, pp. 1–10.

[27] M. Soeken, P. Raiola *et al.*, "SAT-Based Combinational and Sequential Dependency Computation," in *Proc. Int'l Haifa Verification Conference (HVC)*. Springer, Nov. 2016, pp. 1–17.

[28] R. Baranowski, M. A. Kochte, and H.-J. Wunderlich, "Modeling, Verification and Pattern Generation for Reconfigurable Scan Networks," in *Proc. Int'l Test Conf. (ITC)*, Nov. 2012, pp. 1–9.

[29] "Gurobi Optimization, LLC", "Gurobi Optimizer Reference Manual," 2019. [Online]. Available: http://www.gurobi.com

[30] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *Proc. Int'l Symp. on Circuits and Systems (ISCAS)*, May 1989, pp. 1929–1934 vol.3.