Post-Silicon Validation of IEEE 1687 Reconfigurable Scan Networks

Aleksa Damljanovic*, Artur Jutman†, Giovanni Squillero‡, Anton Tsertov§
*‡*Politecnico di Torino, Torino, Italy*; †§*Testonica Lab, Tallinn, Estonia*
*aleksa.damljanovic@polito.it †artur@testonica.com ‡giovanni.squillero@polito.it §anton@testonica.com

***Abstract*—The increasing number of embedded instruments used to perform test, monitoring, calibration and debug within a semiconductor device has called for a brand new standard—the IEEE 1687. Such a standard resorts to a *reconfigurable scan network* to provide efficient and flexible access to instruments and to handle complex structures. As it has to deliver reliable service, many approaches, both formal and simulation-based, have been proposed in the literature to perform test, diagnosis and verification of such networks. This paper focuses on the problem of post-silicon validation of a network, a problem that has not been adequately addressed, yet. We analyze the mismatches between the specification and its silicon implementation, and we propose a methodology to detect a subset of them by applying functional patterns and observing the length of the active scan path. Experimental results on ITC2016 benchmarks demonstrate that the proposed approach is broadly applicable, and able to generate very effective sequences. We also classify mismatches that cannot be targeted relying exclusively on the active scan path length information.**

***Index Terms*—Validation, Pattern Generation, Reconfigurable Scan Networks, IEEE 1687**

I. Introduction

Nowadays, the standard IEEE 1687-2014 [1], also known as IJTAG, is being increasingly adopted by the semiconductor industry for arranging access to internal scan chains during scan-based test as well as for accessing dedicated embedded instrumentation such as SerDes conditioning and error monitoring circuits, memory BIST, temperature and voltage sensors, and many others alike. Since both production test and in-field monitoring processes majorly depend on such embedded instruments and other features, it is of utmost importance to ensure their correct operation. Therefore, many approaches, both formal and functional, have been proposed in the literature to perform test [2]–[7], diagnosis [8] and verification [9]–[11] of IEEE 1687 networks. However, the correct operation of IJTAG-compliant infrastructure is a product of many aspects and components including the actual hardware on the chip, the respective standard descriptions, such as ICL (Instrument Connectivity Language) and PDL (Procedure Description Language) files as well as the software used to import the descriptions and control the hardware.

The importance of the problem is being escalated by previous experience of the electronics industry, which suffered from the inconsistency between description files and actual hardware implementation of an earlier similar standard: IEEE 1149.1. In surprisingly numerous cases, the BSDL (Boundary Scan Description Language) descriptions did not match the actual implementation of JTAG features in silicon. Most of those mismatches were caused by simply non-matching revisions of the silicon and the BSDL, but of course a certain number of problems were related to bugs and design errors in hardware. Even if the error-checking is performed before tape-out, it does not necessarily imply that the silicon will work or that the ICL matches the actual silicon implementation. Independent of particular reasons causing such mismatches, the task of proving full compliance between the silicon and the documentation is not trivial. Taking into account the fact that the infrastructure described by IEEE 1687 is certainly more complex than classical Boundary Scan, ensuring its correct operation is an important research topic.

This paper is focused on the problem of checking the equivalence between the silicon implementation of IEEE 1687 RSNs and their respective ICL descriptions. The method we describe assumes that the former is a black box and the latter plays the role of specification, while no other information about the target system is available. Although observability of signals in simulation (for pre-silicon verification) is exceptional, this is unfortunately not the case when accessing the read device through its interface, i.e. we can only apply stimuli and observe responses through scan input and output ports.

Previous work that addresses this problem is very limited. The problem's general definition along with a trivial algorithm for simple RSNs was first proposed in EU FP7 BASTION project report [12]. An important contribution of that work was in defining three levels of validation thoroughness with respect to required test access and effort. At the base level ("Level 0") the RSN infrastructure is validated by checking scan chain length and capture values in various configurations ensuring that every instrument is correctly accessible.

The main contribution of this paper is twofold. First, we introduce a comprehensive fault model defined as a set of mismatches between the ICL description and the silicon implementation. Second, for a subset of mismatches falling into Level 0 category, we propose a universal method and a tool of their detection based on observing the length of the active scan path. In addition, we categorize the mismatches with respect to the level of detection difficulty providing a list of those undetectable by our method. Experimental results based on the set of ITC2016 RSN benchmarks [13] demonstrate that the proposed approach is broadly applicable as well as that the test tool is able to generate the sequences for detecting all target mismatches. The proposed validation tool is a part of

an ecosystem of IEEE 1687 benchmarks and tools [14], which can be freely used in connection with the benchmarks[1].

The following section provides necessary terminology and background information. Section III describes the fault model, the detection method and the respective tool. Experimental results are given in Section IV. Section V concludes the paper.

II. Background and Terminology

The RSN is an instrument access network residing between device interface and instrument interface. Through dynamic reconfiguration of the RSN, instruments placed on different segments can either be selected or bypassed by controlling RSN's configurable components. This mechanism enables flexibility and facilitates the embedded instrumentation access from the chip boundary. The RSN infrastructure consists of scan registers, multiplexers, control signals and combinational logic.

Active scan path includes all scan cells (flip-flops) connected between SI and SO for a given state of the network. *Test Data Register* (TDR) is an instrument interfacing register. Often it is represented as a separate register field (referred to as a *ScanRegister*) for interfacing individual instrument. It is composed of 1149.1 Scan Cells. The scan cell is a two-stage cell with two flip-flops. First stage is responsible for capture-shift (CS) operations and is a part of scan chain, while the second one is a ripple-free update (U) stage. Therefore, depending on the need, TDRs can be Read-Only, Write-Only or Read-Write. Most commonly used access interface is IEEE 1149.1 TAP controller. Its state machine is responsible for generating control signals to operate the network (*select*, *capture*, *shift*, *update* and *reset*).

Two main architectural constructs, i.e, programmable modules are Segment Insertion Bit (SIB) and Scan Multiplexer (ScanMux). After shifting desired values into the scan chain and performing update, values are latched from the (C)S stage to the U stage of each active, selected cell. The outputs of the U stages of the control bits are used to drive the multiplexers' control inputs. Capture stage here is not necessarily present.

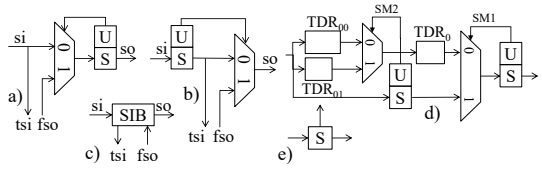

Fig. 1. Simplified schematics of SIB and ScanMux reconfigurable modules

A SIB corresponds to an in-line 2-to-1 ScanMux whose (C)SU control bit is placed next to the multiplexer. SIB allows the scan chain to expand by including nested segment between terminals *tsi* and *fso* into the active path. When in this state, SIB is referred to as *asserted* (opened). Otherwise, SIB is said to be in a *de-asserted* (closed) state, bypassing the *tsi–fso* segment with only one bit between *si* and *so*. Two types of SIBs are defined depending on the control bit position with respect to the multiplexer: the pre-SIB (Fig. 1a), if a multiplexer precedes the control bit and the post-SIB (Fig. 1b) if a multiplexer is placed after the control bit. Adopted symbolic representation of a SIB is given in Fig. 1c.

Apart from inserting, i.e., bypassing certain segments, ScanMuxes can enable the existence of mutually exclusive, selectable scan chains. They can be used in *in-line* and *remote* configurations. In the latter case, source of a multiplexer's control input (control bit, decoded TAP instruction) is located outside the scan segment that contains element being controlled. A simplified schematic containing one in-line (*SM1*) and one remotely controlled (*SM2*) 2-to-1 ScanMux with their corresponding (SU) control bits is given in Fig. 1d. The symbol shown in the same figure (Fig. 1e) is adopted as a representation of a ScanMux control bit throughout the paper.

III. Methodology

The proposed methodology is based on previous work focused on generating efficient patterns to perform end-of-manufacturing test for RSNs [3]. The detection mechanism introduced first in [4] and then adopted in several works [5], [6] has been modified to reduce significant overhead and has been made more suitable for addressing the presented problem.

A. Mismatch model

Well-established metrics exist for post-manufacturing tests (single-stuck-at coverage, transition fault coverage) and experimental results have demonstrated the effectiveness of such metrics. Although pre-silicon verification metrics are less standardized (syntactic (code coverage) and semantic (covering assertion goals)), metrics for post-silicon validation are still the subject of research. The list of considered mismatches was created after analyzing the literature and taking into account that the source of a mismatch is usually confined, such as a typo in the specification, or a localized hardware bug. It contains following items:

- A missing register
- An added register
- A wrong register of different length
- A wrong register with the modified functionality
- Exchanging two or more registers (their position)
- Exchanging a register with a ScanMux or a SIB
- Exchanging two SIBs belonging to the same segment
- Exchanging two ScanMuxes
- Exchanging a ScanMux and a SIB
- Wrong ScanMux configurations
- Exchanging inputs or control lines of the ScanMux
- Wrong SIB type (pre-SIB to post-SIB and vice versa)

For the network given as an example in Fig. 2, Table I provides a set of considered mismatches joined by their type.

Relying on the structural information provided by the ICL is sufficient for detecting a missing register at one of the scan segments, since this type of mismatch directly affects the length of the active scan path. An added register has the similar effect on the scan path length although this time it is reduced with respect to the expected one. A wrong register with

[1]https://gitlab.com/IJTAG

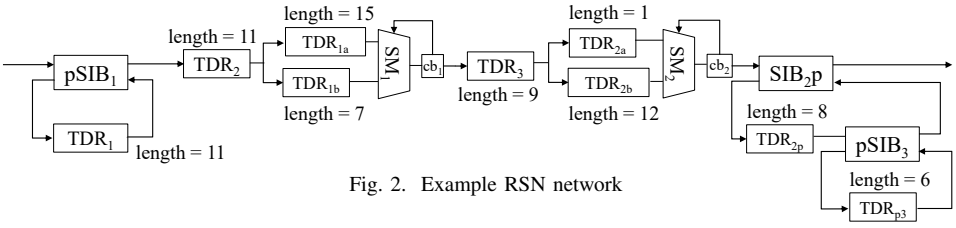

Fig. 2. Example RSN network

TABLE I
LIST OF CONSIDERED AND INJECTED MISMATCHES FOR THE NETWORK FROM FIG. 2

Mismatch Type	N	Mismatch set
SWAP_TDR_SIB	5	$[[pSIB_1, TDR_2]$, $[TDR_2, SIB_{2p}]$, $[TDR_3, SIB2p]$, $[TDR_{2p}, pSIB_3]$, $[pSIB_1, TDR_3]]$
SIB_PRE_POST	3	$[[pSIB_1], [SIB_2p], [pSIB_3]]$
SWAP_CB_CB	1	$[[cb_1, cb_2]]$
ADDED_REGISTER	27	
WRONG_REG_FUNC	9	not considered
WRONG_MUX_CONF	2	$[[SM_1\|0, TDR_{1b}; 1, TDR_{1a}]$, $[SM_2\|0, TDR_{2b}; 1, TDR_{2a}]]$
SWAP_TDR_TDR	1	$[[TDR_2, TDR_3]]$
WRONG_REG_LENGTH	9	all registers
SWAP_TDR_TDR_DD	35	from domain to domain
SWAP_SIB_SM	4	$[[pSIB_1, SM_1], [SM_2, SIB_2p]$, $[SM_1, SIB_2p], [pSIB_1, SM_2]]$
MISSING_REGISTER	9	all registers
SWAP_TDR_SM	4	$[[TDR_2, SM_1], [SM_1, TDR_3]$, $[TDR_3, SM_2], [TDR_2, SM_2]]$
SWAP_SIB_SIB	1	$[[pSIB_1, SIB_2p]]$
SWAP_SM_SM	1	$[[SM_1, SM_2]]$
SWAP_MUX_CONTROL		not modelled explicitly
SWAP_MUX_INPUTS		not modelled explicitly

the different length is equivalent to having a missing and/or added register. Exchanging the registers is being performed not only within the same scan segment but also outside of one domain. This modification can be modelled as having multiple wrong register length mismatches. Since the registers belong to different scan segments, not having both of them on the same path for the first time they are accessed enables immediate detection of this particular mismatch.

Even though a mismatch of exchanged ScanMux control signals (Fig.3a) or inputs (Fig.3b) does not have an effect on the active path length it can still be detected. The configuration of the ScanMux is determined by the value in the control bit. The output signal from the update stage, apart from controlling the multiplexer can also be used to gate *shift*, *update* and *capture*. In that case, if upper input is selected, all data shifted at the input is supposed to go through TDR_1 and appear at the output. However, in case of exchanged control signals, although the segment is chosen according to the configuration, all operations are forbidden on that segment and allowed on the other one. Consequently, shifted values will not propagate to the output, resulting in all 0s or all 1s, depending on the value stored in the last scan cell in the selected input segment. Exchanging input connections is an equivalent mismatch and can be analyzed in a similar way. Guaranteeing that all scan segments are accessed at least once ensures that all mismatches of this type are detected.

Another type of considered mismatch is a ScanMux with configurations incorrectly assigned to its input segments. In Fig. 3c, configurations 00, 01, 10 and 11 result in including registers TDR_0, TDR_1, TDR_2, TDR_3 into the scan path, respectively. In case of a mismatch, chosen registers appear in the different order: TDR_3, TDR_2, TDR_0, TDR_1. If at least one of the segments has a length different than the original one in the same position, the mismatch is detectable comparing the lengths. This type of mismatch also covers the modified order of control bits (cb_0, cb_1-10 with 01).

In case of the wrong SIB module type (pre-/post-), the length and the order of elements on the scan segment remains unchanged when SIB is de-asserted. If there are some control bits in the controlled segment-their order is shifted for one position, while the SIB itself is placed after, i.e., before the elements of the included segment.

In general, mismatches involving the modified order of TDRs, SIBs and ScanMuxes do not affect the length of the active path when the corresponding segment is included into it. However, detecting them remains possible as long as certain configuration bits do not match original positions. Writing into them to set the desired configuration may result in writing into TDRs or some other configuration bits.

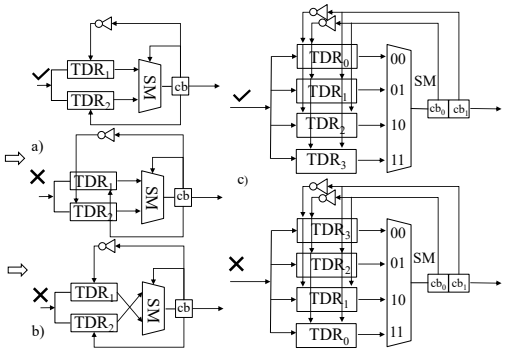

Fig. 3. ScanMux control lines and input mismatch

B. Undetectable mismatches

A mismatch is considered to be undetectable if applying whichever legal configuration results in observing expected length of the scan path.

The modified functionality of the register has been modelled as a permutation of register's bit scan cells and it is equivalent to having permuted connections with the instrument. Since there is no effect on the length of the scan path, this type

of mismatch is undetectable. Exchanged position of registers within the same scan segment may not always be detectable. In particular, this is the case if the registers are adjacent or have the same length. Furthermore, if there is not even a single control bit cell located in between, all configurations are properly applied and no mismatch is observed at the output. In the case of exchanging two or more SIBs within the same scan segment, the mismatch is undetectable if they have completely same structure (the position, length and type of modules), except if they provide access to remote ScanMux control bits. Wrong configuration mismatch is also undetectable when all input segments have the same fixed length (only TDRs).

Potentially, by "Level 1" validation [12], some currently undetectable mismatches could be targeted. Correct reaction of instruments on PDL defined-actions has to be verified upon accessing them through performing read and write operations. Furthermore, the presence of undocumented or specially hidden structures can be targeted by "Level 2" validation (phantom detection) [12].

C. Detection mechanism

The procedure for detecting mismatches is organized as a set of sessions. A session consists of a *configuration* pattern to which an additional sequence of bits is appended. It contains values for defining the state of the network. The appended sequence is used as a key to validate that the expected path is connected between scan input and scan output pins. Configuration sequence of bits has the length of the currently active path. This sequence of bits is shifted into the scan chain, while in parallel the output pin is monitored. If the sequence observed at the output, long as it is the key sequence presented at the input, matches the very same key, it is considered that no potential mismatch could be detected in that session. This is due to the effect of a mismatch which can either corrupt the values of the key loaded into the network (e.g., all 0s or all 1s) or can change its position (postpone it or anticipate it at the output) with respect to the expected one.

Additionally, if a TAP controller is used to control and access the network, its state machine has to traverse *capture* and *update* states, while the *shift* state can be omitted. Therefore, before shifting in the sequence, a capture operation is performed. The values appearing at the output during the shift-in are capture values and can be used to enhance detection capability. The *pause* state, following the *shift* state of the TAP controller can be used to prevent performing update before checking the pattern at the output, thus avoiding undesired effects such as moving the network to an unknown state.

Cost of applying one session is equal to the number of clock cycles (shift operations) as long as the active path increased by the length of a key. It should be mentioned that after performing update to apply wanted configuration, reaching the shift state in a state machine requires certain number of clock cycles (JTAG protocol overhead).

D. Algorithm

The mismatch detection is solved as a problem of discriminating between a set of Finite State Machines (FSMs). One FSM is created for the original network without any mismatch present; for each mismatch, an additional FSM is created. Initial state is appended to each FSM, where the state corresponds to the current configuration of the network. Positions of configuration (control) bits with respect to the network's input are being tracked constantly. After applying the transition (reconfiguration operation), the state of each FSM is updated, while the new length (output symbol) is calculated based on the injected mismatch, if any. Additionally, positions of the configuration bits in a new state are calculated and updated accordingly.

Before generating input symbols, during the construction of the internal network model, every reconfigurable element (ScanMux, SIB) in the network is annotated with auxiliary information. First attribute is the hierarchical level (l_c) of the scan segments in which the module is positioned, while the second one is the highest hierarchical level (l_d) of all modules' levels that are positioned within scan segment(s) attached to the input(s) of the current module. In Fig. 4b a structural representation of the network from Fig. 4a is shown as an example in the form of a tree. A node coincides with the reconfigurable module, while encoding of the segment that is either insertable (SIB) or selectable (ScanMux) is given as a vertex. Nodes n_1, n_2 and n_3 are located at the top level scan segment, while nodes n_{31}, n_{32} and n_{33} are accessible through n_3 (located in its input segment(s)). Nodes $n_{11}(1,1)$ and $n_{13}(1,1)$ provide access only to empty segments and segments that include either TDRs or control bits (registers), which is obviously not the case with the node $n_{12}(1,2)$. This node provides access to the segment at the second level of hierarchy with one node $n_{121}(2,2)$ in it.

The mechanism for generating input symbols is able to determine if the network contains remotely controlled scan multiplexer architecture or all modules are controlled in-line. In the first case, the priority list for accessing nodes is shown in Fig. 4b. The n_i node's position in the list is based on the l_d^i value; the precedence is given to the node with the higher values. However, if the nodes n_i and n_j have equal value of the second parameter ($l_d^i = l_d^j$), the position is decided based on the first parameter value l_c. Finally, in case that $l_c^i = l_c^j$ the nodes whose parent nodes are closer to the input are chosen first e.g., n_{11} has precedence in comparison to n_{31}, while n_{31} is put before n_{33}.

The procedure (Algorithm 1) starts from the position of all configuration bits, taking into account only accessible ones when choosing the element from the priority list (Fig. 4b). When it is a SIB instance, if it is in a de-asserted state, new configuration sets it to assert, while if it is a ScanMux with no children nodes, one configuration is generated for every input segment in order to include it to the active path at least once. For a ScanMux with some reconfigurable nodes in its input segments the decision which configuration to set

is based on the priority list. In that sense n_3 could be a SIB with three serially connected nodes n_{31}, n_{32}, n_{33} located at the same segment, but it could also be a ScanMux with three input segments; in first n_{31}, in second n_{32}, and in third n_{33}. When a segment is included to the active path it is marked as *tested*. If all children nodes for a parent SIB node are marked as *tested*, the next configuration will also de-assert that SIB.

In this work we also considered more complex networks involving remotely controlled scan multiplexer architecture. They are more difficult to manage in terms of module's controllability and observability: corresponding control bits have to be part of the active path in order to set desired configuration, while additional reconfiguration operations are required to include the module itself into the active path. Therefore, an algorithm (Algorithm 2) implemented with two recursive functions CONFIGUREMUX and PUTONPATH provides input symbols for guaranteeing that all scan segments are accessed at least once, while also detecting the full set of considered, detectable mismatches. The order in which multiplexers are provided is obtained using the same rule as in the previous algorithm with the difference that is performed once, globally, taking all multiplexers into consideration (Fig. 4c).

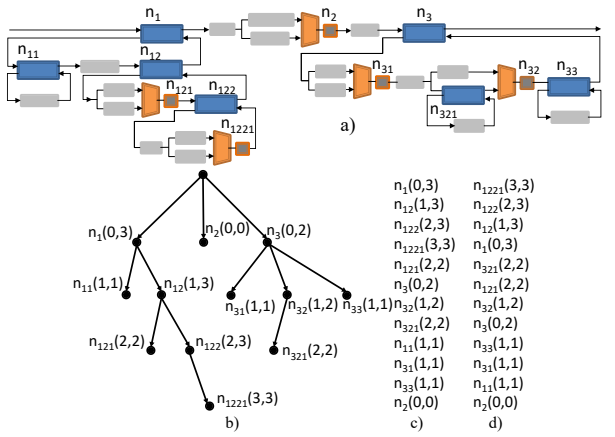

Fig. 4. Hierarchical information on network's nodes

IV. EXPERIMENTS

We developed a prototypical tool, able to build an internal simplified RSN model by reading the ICL description of the network. After generation of selected mismatches, pattern generation is run. Upon completion, a report is generated with the list of applied configurations, set of covered mismatches and those which are considered as undetectable.

Experimental results using the proposed mismatch model and pattern generation algorithms are reported for the subset of ITC2016 benchmark networks, since the tool currently does not support all the constructs. Some synthetic networks from the same set [13] have been translated from an internal XML description to the ICL format and can be found at the ecosystem's website. [2].

[2]https://gitlab.com/IJTAG/benchmarks/tree/master/ICL

Algorithm 1 Deterministic algorithm for in-line RSN architecture

```
function GENERATEINPUTSYMBOL(prevState, state)
    i ← SIZE(bitBuffer)                          ▷ number of control bits
    genState ← ACCESSIBLE(state)                 ▷ visible control bits
    while i ≥ 0 do
        mux ← GETMUX(StateVars, i)               ▷ mux corr. to i
        if ACCESSIBLE(i) then                    ▷ control bit on act. path
            CHECKMUX(0, mux)                     ▷ update mux test status hier.
            if SIB then                          ▷ mux is SIB type
                if bitBuffer[i] A then
                    if HIERTESTED(mux) then
                        bitBuffer[i] ← D         ▷ close SIB
                        SETTESTED(mux)           ▷ mark as tested
        curL ← mux segment hier. level(l_c)      ▷ 1nd parameter
        deepL ← mux deepest hier. level(l_d)     ▷ 2st parameter
        if ScanMux or (SIB and (bitBuffer[i] D
        or (bitBuffer[i] A and ISTESTED(mux))) then
            if deepL ≥ maxD and ¬ISTESTED(mux) then
                if curL ≥ maxC then
                    sMux ← mux                   ▷ save the multiplexer
                    update levels (maxD)
        i ← i − SELECTIONCELLSIZE(mux)
    if ¬ISTESTED(sMux) then                      ▷ decide how to conf. mux
        if ScanMux then                          ▷ traverse ScanMux inputs
            bitBuffer ← apply next encoding
            if last encoding then
                SETTESTED(sMux)                  ▷ mark as tested
        else
            if bitBuffer[i] D then
                bitBuffer[i] ← A
            else
                if (mux, l_c = l_d) then
                    SETTESTED(sMux)              ▷ mark as tested
```

Algorithm 2 Generating configurations for remotely controlled RSN architecture

```
function GENERATECONFIGURATIONS(gen, len, i, d)
    mux ← muxList(0)
    while mux ≠ null do
        currentConfEnc ← mux current configuration
        for encoding all mux encodings do
            if currentConfEnc ≠ encoding then
                CONFIGUREMUX(gen, mux, encoding)
            PUTONPATH(mux)
        index ← index + 1
        mux ← muxList(index)
```

The networks from the evaluation set differ in the number and type of programmable modules. Table II reports all important characteristics for each of the networks listed in column 1 (*Network*). Columns 2 and 3 report the total number of reconfigurable modules - number of SIBs and ScanMuxes, respectively. The total number of control bits is given in column *Conf. bits*. *MaxDepth* column contains maximum hierarchical depth of the network (for SIB-based networks this value equals to the maximum number of nested SIBs, according to [13]). The longest path length is reported in *Longest path* column, while the total number of scan cells in the network can be found in *Scan Cells* column.

TABLE II
BENCHMARK NETWORKS LIST

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

TABLE III
EXPERIMENTAL RESULTS

Network	Num conf.	Total Cost [cc]	Run-time	Total. mism.	Undet. mism
Mingle	32	4,437	1s	124	29
TreeBalanced	62	19,391	9s	1423	47
TreeFlat_Ex	102	16,446	10s	2725	149
TreeUnbalanced	44	150,277	4s	874	50
a586710	106	1,588,162	7s	1,029	186
p22810	518	162,668	8m	36371	851
p34392	310	2,853,219	100s	8,250	1,377
p93791	1,864	76,688,262	11h	203,832	21,342
q12710	50	54,760	4s	491	8
t512505	287	176,506	1m	10,246	557
N132D4	96	100,629	47s	18,140	492
N17D3	17	3,464	1s	553	11
N32D6	29	850,906	5s	1,290	4
N73D14	55	3,361,858	7s	4,836	21
NE600P150	432	1,685,759	37m	319,871	5,818
NE1200P430	854	10,696,840	4h	1,337,343	11,303

The experimental results are given in Table III. For each of the benchmarks in column 1, number of generated configurations is reported in column 2. Column 3 gives the total cost in clock cycles for applying the generated sequence. The key length has been set to 32, while 5 clock cycles are added as JTAG overhead to each of the sessions. Furthermore, the time required by the tool written in Java to apply the algorithm is given in column *Runtime*. Columns 5 and 6 report on total number of considered faults (excluding implicit ones) and the total number of *undetectable* faults, respectively. It is worth noting that in all cases applying generated sequence resulted in achieving full coverage.

V. CONCLUSIONS

Reconfigurable Scan Networks provide flexible instrumentation access through dynamic reconfiguration. To ensure there is no mismatch between prototypical device and initial specifications, product life-cycle requires performing (post-silicon) validation before going into the mass production. Such additional effort prevents rendering the whole infrastructure inoperable and avoids enormous re-design costs. In this paper we proposed a mismatch model for post-silicon validation of RSNs. Furthermore, two algorithms have been developed for generating configuration patterns to detect the set of considered mismatches. The ITC2016 benchmark networks were used to evaluate the proposed methodology. We found that in all cases full detection coverage has been reached. For the detection procedure based on the active path length comparison, the tool is able to generate a list of undetectable mismatches. Furthermore, mismatch model is easily extendable, due to the nature of the problem and internal model of the network extracted by the tool.

ACKNOWLEDGEMENTS

The work has been supported by the European Commission through the Horizon 2020 RESCUE-ITN project under the agreement No. 722325, European Regional Fund, and IUT 19-1 grant of the Estonian Ministry of Education and Research.

REFERENCES

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

[2] M. A. Kochte, R. Baranowski, M. Schaal, and H. Wunderlich, "Test strategies for reconfigurable scan networks," in *2016 IEEE 25th Asian Test Symposium (ATS)*, Nov. 2016, pp. 113–118.

[3] R. Cantoro, A. Damljanovic, M. Sonza Reorda, and G. Squillero, "A new technique to generate effective test sequences for reconfigurable scan networks," in *IEEE 49th International Test Conference (ITC)*, 2018.

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

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

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

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

[8] R. Cantoro, M. Montazeri, M. Sonza Reorda, F. G. Zadegan, and E. Larsson, "Automatic generation of stimuli for fault diagnosis in IEEE 1687 networks," in *IEEE 22nd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2016, pp. 167–172.

[9] F. G. Zadegan, E. Larsson, A. Jutman, S. Devadze, and R. Krenz-Baath, "Design, verification, and application of ieee 1687," in *Proceedings of the 2014 IEEE 23rd Asian Test Symposium*. Washington, DC, USA: IEEE, 2014, pp. 93–100.

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

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

[12] "Report on structural analysis, verification and optimization methodology for icl networks," in *EU FP7 BASTION project report*, 02 2016, pp. 1–42. [Online]. Available: https://cordis.europa.eu/docs/projects/cnect/1/619871/080/deliverables/001-BASTIOND23v204.pdf

[13] A. Tšertov, A. Jutman, S. Devadze, M. Sonza Reorda, E. Larsson, F. G. Zadegan, R. Cantoro, M. Montazeri, and R. Krenz-Baath, "A suite of IEEE 1687 benchmark networks," in *IEEE 47th International Test Conference (ITC)*, 2016, pp. 1–10.

[14] A. Tsertov, A. Jutman, K. Shibin, and S. Devadze, "IEEE 1687 compliant ecosystem for embedded instrumentation access and in-field health monitoring," in *AUTOTESTCON 2018*, 09 2018, pp. 1–9.