POLITECNICO DI TORINO
Repository ISTITUZIONALE

Simulation-based Equivalence Checking between IEEE 1687 ICL and RTL

Original
Simulation-based Equivalence Checking between IEEE 1687 ICL and RTL / Damljanovic, A., Jutman, A., Portolan, M., Ernesto, S., Squillero, G., Tsertov, A.. - ELETTRONICO. - (2019). (50th IEEE International Test Conference, ITC 2019 Washington DC (USA) 12-14 November) [10.1109/ITC44170.2019.9000181].

Availability:
This version is available at: 11583/2751832 since: 2020-04-26T19:38:24Z

Publisher:
IEEE

Published
DOI:10.1109/ITC44170.2019.9000181

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2019 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

17 September 2026

Simulation-based Equivalence Checking between IEEE 1687 ICL and RTL

Aleksa Damljanovic*, Artur Jutman†, Michele Portolan‡, Ernesto Sanchez§, Giovanni Squillero¶, Anton Tsertov‖
*‡§¶*Politecnico di Torino, Italy*; ‡*Universit Grenoble Alpes, France* †‖*Testonica Lab, Estonia*

***Abstract*—A fundamental part of the new IEEE Std 1687 is the Instrument Connectivity Language (ICL), which allows for abstract description of the scan network. The big novelty if compared to legacy solutions like BSDL is the possibility of describing new topology-enabling elements such as the Scan-Muxes in a behavioural way which can be easily and efficiently exploited by Test Generation Tools to retarget instrument-level operations to top-level patterns. This means that for a given design, the Developer will have to write both the RTL and the ICL descriptions: to the author's best knowledge there is no automated tool to make the translation RTL to ICL. This methodology is error-prone due to the human factor, the difference in intent in the two descriptions and the syntactic and semantic complexity of the languages. Incoherence between ICL and RTL will result in retargeting errors, so it is fundamental to validate the equivalence between the two descriptions. This paper presents an automated methodology that starting from the ICL description is able to generate a set of RTL testbenches that can be simulated against the original RTL model to detect discrepancies and incoherence, and provides quantitative metrics in terms of code and functional coverage. Experimental results are reported on the set of ITC2016 set of benchmark networks.**

***Index Terms*—Simulation, RTL, ICL, Code-coverage, Pattern Generation, Reconfigurable Scan Networks, IEEE 1687**

I. Introduction

With the scale and complexity of modern electronic systems, dependability has become a key concern, demanding the deployment of several complementary Design-for-Test (DfT) approaches to reach the disordered quality goals. Traditional scan-based Automated Test Pattern Generation (ATPG) is now coupled with resources embedded within integrated circuits (ICs). These are used for supporting test, such as Built-In Self-Test (BIST) modules, for monitoring internal parameters such as current, temperature or delay sensors and configuration/calibration of different modules through registers. The growing number of these devices made it infeasible to include them into the single scan chain or provide a separate instruction for accessing every single one, relying on the IEEE 1149.1 standard [1]. A solution for simplifying the access to all these resources and to reduce the overhead while allowing complexity scaling has been proposed and published as the IEEE 1687-2014 standard [2]. In this context, the biggest evolution if compared to IEEE 1149.1 is the introduction of variable-length scan chains, thanks to the introduction of dynamic typologies. In order to support this feature, the Standard introduces the Instrument Connectivity Language (ICL), whose role is to describe the new systems in a tool-friendly manner. This means that a given system will have two different descriptions: the RTL, used in the design flow to obtain the final circuit, and the ICL, which will describe the DfT infrastructure. Depending on the company's internal design strategy, ICL and RTL could be developed in parallel starting from the same high-level specifications, ICL could be extracted from the RTL or vice-versa. In all cases, this implies to have two different descriptions of the same system: any incoherence between the two models might cause serious problems. This is especially true for a new standard like IEEE 1687-2014 : ICL can be quite complex and engineers are still learning it, making human error extremely likely.

In recent years, numerous works dealing with different issues related to reconfigurable scan networks (RSNs) have been published. Design automation of optimized 1687 SIB networks is tackled in [3]. In [4]–[7], authors address the issue of testing such structures in the context of test strategies, test-time minimization and structure-oriented test. An approach to diagnose high-level faults in configurable modules has been presented in [8]. Methodology for modelling, verification and pattern retargeting related to RSNs is described by the authors in [9], [10]. Network verification and pattern retargeting are transformed to a Boolean satisfiability (SAT) problem, since the formal model is used to prove if specified structural and functional properties hold. Furthermore, a method for the verification of security properties in RSN designs is proposed in [11]. The methodology is based on formal modelling of the networks and unbounded model checking.

In this paper, we propose an automated and reliable method for verifying equivalence between ICL and RTL descriptions. After giving a short introduction to IEEE-1687 and ICL in Section II, we will introduce our approach in Section III. Section IV will provide experimental results based on the ITC16 benchmark suite [12]. Lastly, Section V will draw conclusions and point out future developments.

II. Background

A. Overview of IEEE 1687 Reconfigurable Scan Networks

In traditional JTAG, all access is done from the Test Access Port (TAP), which allows access to one or more fixed-length Test Data Register (TDR) : all elements connected to the chain are always accessed together. The only way to provide a selection is to use multiple TDRs, individually selected through the Instruction Register (IR). TDR and IR are accessed using a CSU (capture-shift-update) protocol: capture(C) is used to read the data latching the value provided by the instrument to

the scan cell, Shift(S) to shift the data through the scan chain (latching the input of the scan cell) and Update(U) to latch data from the scan cell to the update stage cell of the TDRs. This solution is simple and easy to set-up, but it scales badly: when the number of elements grows, so does the IR and the related selection logic. Moreover the overhead of having to go through the TAP's Finite State Machine each time to change the selected elements rapidly become unbrearable. For the reason, the IEEE 1687 standard enables designers to flexibly trade-off between area, access time and other parameters, because TDRs can now be split and configured thanks to the introduction of programmable modules, i.e., *Segment Insertion Bits* (SIBs) and *ScanMuxes* (SMs). Figure 1 depicts a SIB: a control registers is used to set the configuration of SIB and controls a scan multiplexer module. Configuration is performed by shifting a sequence of values into the chain to match positions of shift(S) flip-flop of the control registers. Finally, by updating, values from the shift(S) cells are latched to the update(U) cells of the control register: as a consequence, the configuration of the Scan Chain is changed for the following access, without needed special operations such an IR access. Therefore, SIB supports variable length of the active scan path and can be used to support construction of a hierarchical structure.

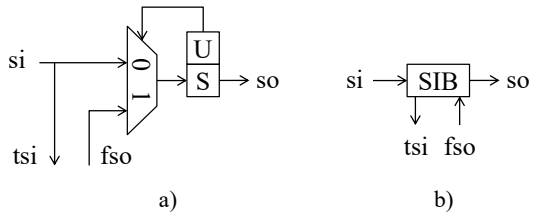

Fig. 1. SIB module with its functionality and symbolic representation

The SIB is just the simplest way of using a 2-way ScanMux: it is also possible to use n-ways scan muxes, as depicted in Figure 2. This configuration is sometimes called a Multiple Insertion Bit, or MIB.

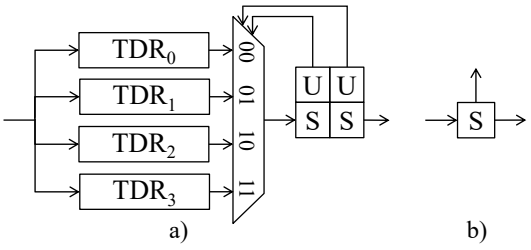

Fig. 2. Scan Multiplexer with 4 inputs and symbol used for conf. bits

Thanks to these new elements, IEEE 1687 has shifted from fixed-length scan chains to what is usually called Reconfigurable Scan Networks (RSNs). Example in Fig.3, shows a network with five instruments. Two TDRs (instruments A and B) are hidden hierarchically behind two SIBs. Two instruments (C and D) are interfaced with two mutually-exclusive TDRs placed in input segments of a Scan Multiplexer. ScanMux control bit is used to include wanted TDR in the active path. One remaining TDR (instrument E) is always included in the scan chain. When SIB_1 is asserted by shifting appropriate value into corresponding shift scan cell and applying update, *Select* signal is asserted to enable operation on the SIB_2 and on the TDR interfacing instrument B. Accessing hierarchically nested SIB (SIB_2) requires opening all the previous SIBs gating its access (SIB_1). For the configuration depicted in Fig. 3, SIB_1 is asserted, including TDR_1 into the active path. SIB_2 is de-asserted, bypassing scan segment containing TDR_2. Additionally, SM is in a such configuration that TDR_4 is selected and a part of the active scan path. Therefore, the total path length is $3 + 1 + 1 + 10 + 1 + 5 = 21$ with instruments A, D and E being accessible.

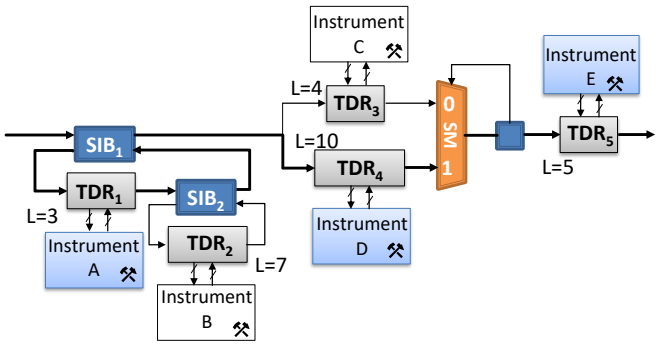

Fig. 3. 1687 RSN example

B. Instrument Connectivity Language

The Standard document states that ICL's purpose is "to describe the elements that comprise the instrument access network as well as their logical (though not necessarily their physical) connections to each other and to the instruments at the endpoints of the network" [2]. To obtain this result, the language takes an approach quite similar to a "light-RTL": registers and instruments can be instantiated and parametrized, and connected through "ports" and "logic signals". Dynamic topologies can be described using "ScanMuxes", whose truth table is used to select the active path. ICL designs can also be split into multiple files for easier maintenance and code reuse. The code base can therefore become quite big, and manual verification cannot be trusted : an automated and quantifiable Equivalence Checking tool is the only viable solution.

A fundamental entity in ICL is called a module. As an example, we provide a description for the pre-SIB module in Figure 4. Here structural description of the module is given together with the definitions of two scan interfaces: host and client interface. The module consists of one *ScanMux* primitive - SIBmux and *ScanRegister* primitive - SR, whose update stage is used to control the multiplexer. This control register (bit) is placed after the multiplexer since its source (*ScanInSource*) is defined as the output of SIBmux. *CaptureSource* and *ResetValue* fields are also specified for the register. SIBmux has

two input segments, first used to bypass (client input - *SI*) and the second one to include the segment (host input - *fromSO*).

A description of the device can contain instantiated modules. In Figures 5, 6 and 7, example network from Fig. 3 is partially described. Parameters are used to define the length of the registers (Fig. 5). It can be seen that not all ports of the instantiated modules are connected. Since ICL is an abstract language rather than a netlist language, they are considered to be connected implicitly. For example, SEL ports of some instances (TDR1 and TDR2) are produced implicitly, directly from the parent modules (SIB1 and SIB2) (Fig. 6). On the other hand, instance TDR3 placed behind the ScanMux has explicitly defined select signal where SM's SEL port is gated with the associated decode of the DO[0:0] signal used to select the active SO signal of the ScanMux (Fig. 7).

III. Methodology

A. Post-silicon validation approach

In [13] we proposed an approach for the purpose of validating silicon implementation of the RSN against respective ICL descriptions. The method itself relies on the ICL as only specification, without any additional information available. Since the observability is reduced compared to gate level or RTL level in simulation, the device itself is considered to be a black-box. Therefore, the proposed approach relies solely on applying the stimuli at the input and observing responses at the output.

Although well-defined and experimentally verified metrics exist for post-manufacturing tests and some less standardized semantic and syntactic for verification (pre-silicon), for post-silicon validation they are still the subject of research. Independent on the specific reasons for the existence of a mismatch between the specification and the implementation we defined a fault model containing a set of mismatches of different type: a missing register, an added register, a wrong register length, exchanged position of two modules (TDR, ScanMux & SIB), wrong configuration, exchanged inputs and control lines of the ScanMux, wrong SIB type.

Additional constraint we set is that the TAP controller is used for controlling the network. Its state machine allows specific order of operations to be executed: either capture shift and then update, in cycles, with the possibility to avoid shift - only capture and then update. Therefore, we organized the procedure for detecting such mismatches as a set of steps. Every step consists of:

- Capture operation
- Shift operation - first a unique key sequence (32/64/128 bits) is inserted into the scan chain. Its purpose is to check the integrity of the scan chain and the length of the active scan path. To continue, a sequence of bits containing the new configuration is inserted while observing the values at the serial output.
- Update operation - to apply the wanted configuration

After analyzing potential mismatches a conclusion has been drawn that their effect is such that the checking sequence is

```
Module SIB_mux_pre {
    ScanInPort SI;
    CaptureEnPort CE;
    ShiftEnPort SE;
    UpdateEnPort UE;
    SelectPort SEL;
    ResetPort RST;
    TCKPort TCK;
    ScanOutPort SO {
        Source SR;
    }
    ScanInterface client {
        Port SI;
        Port CE;
        Port SE;
        Port UE;
        Port SEL;
        Port RST;
        Port TCK;
        Port SO;
    }

    LogicSignal toSel_SR_SEL {
       SR[0] & SEL;
    }
    ScanInPort fromSO;
    ToCaptureEnPort toCE;
    ToShiftEnPort toSE;
    ToUpdateEnPort toUE;
    ToSelectPort toSEL {
       Source toSel_SR_SEL;
    }
    ToResetPort toRST;
    ToTCKPort toTCK;
    ScanOutPort toSI {
        Source SI;
    }
    ScanInterface host {
        Port fromSO;
        Port toCE;
        Port toSE;
        Port toUE;
        Port toSEL;
        Port toRST;
        Port toTCK;
        Port toSI;
    }
    ScanRegister SR {
        ScanInSource SIBmux;
        CaptureSource SR;
        ResetValue 1'b0;
    }
    ScanMux SIBmux SelectedBy SR {
        1'b0 : SI;
        1'b1 : fromSO;
    }
}
```

Fig. 4. Description of the pre-SIB as module in ICL

```
Parameter lenR1= 3;
Parameter lenR2 = 7;
Parameter lenR3 = 4;
Parameter lenR4 = 10;
Parameter lenR5 = 5;
```

Fig. 5. Parameteres in ICL

```
Instance sib1 Of SIB_mux_pre {
    InputPort SI = SI;
    InputPort fromSO = sib2.SO;
}
Instance TDR1 Of WrappedScan {
    InputPort SI = sib1.toSI;
    Parameter dataWidth = $lenR1;
}
Instance sib2 Of SIB_mux_pre {
    InputPort SI = regR1.SO;
    InputPort fromSO = regR2.SO;
}
Instance TDR2 Of WrappedScan {
    InputPort SI = sib2.toSI;
    Parameter dataWidth = $lenR2;
}
```

Fig. 6. SIBs with TDRs in ICL

either corrupted - its values are modified, or shifted in time - values are appearing later or earlier than expected. When a segment that includes modules (TDRs, SIBs and ScanMuxes) whose order is modified is included into the active for the first time, it does not have an effect on the length of the active path. 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.

```
Instance TDR3 Of WrappedScan {
    InputPort SI = sib1.SO;
    InputPort SEL = sel_SR3;
    Parameter dataWidth = $lenR3;
}
ScanMux SM SelectedBy controlSM.DO {
    0: TDR3.SO;
    1: TDR4.SO;
}
Instance controlSM Of SCB {
    InputPort SI = SM;
}
LogicSignal sel_SR3 {
    SEL & (controlSM.DO[0:0] == 1'b0);
}
```

Fig. 7. ScanMux and TDR in ICL

The algorithm for generating configurations is deterministic. It starts from the internal network model and the set of considered mismatches. Some of the mismatches are considered detected implicitly with the condition that each scan segment has to be accessed at least once.

The network is modelled as a Finite State Machine (FSM):

- State is represented as the current configuration of the network.
- Output symbol is the length of currently active scan path.
- Input symbol is the bit stream shifted at the input
- Transitions are reconfiguration operations.

Apart for the original, unmodified network, one FSM is created for every mismatch. Such FSMs contain the set of segments that do not match those in the original network due to the injected mismatch. Additionally, the record of positions for all configuration bits is kept. Initial states are generated and set for all FSMs, while consecutive states are being created dynamically.

Reconfigurable modules are listed based on their position in the hierarchy of the network as well as on the highest hierarchical level they provide access to.

Two algorithms for generating configurations were developed in order to support both, networks with all modules controlled in-line and those incorporating remotely controlled configurable modules as well, where the ScanMux and associated set of control bits are either non-adjacent or do not belong to the same scan segment. The latter are more difficult to manage since for a certain network configuration to be applied, multiple reconfiguration operations may take place to first set the desired value(s) of relevant control bit(s) and then include the module into the active path.

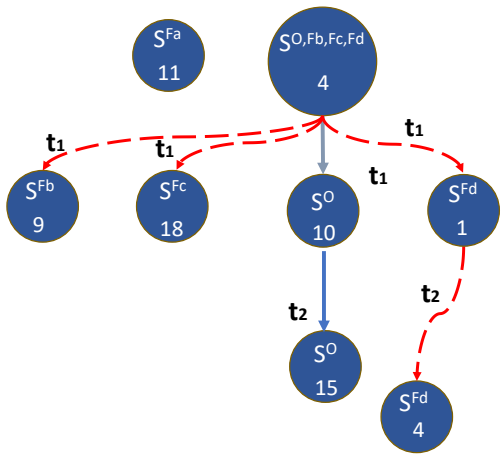

Fig. 8. FSMs example

B. Application to RTL Equivalence

In this paper, we propose to apply the same approach to the problem of RTL and ICL Equivalence: starting from the ICL description of the System Under Test we develop a testbench, which thanks to the properties detailed in the previous subsection, when simulated against the correct RTL description

provides a 100% coverage. In case of non-compliance between the RTL and the ICL, coverage will drop as the hypothesis behind the testbench generation are not true: this condition will therefore allow us detection, as will be detailed in the following Section.

IV. Experimental results

A. Setup

To validate our approach, we devised an experimental setup based on the ITC16 testbenches [12]: for each testbench we follow this experimental procedure:

1) We first parse the ICL provided by the testbenches to obtain an internal network model;
2) From this model, we generate an RTL description of the System-Under-Test, which is correct and coherent with the ICL by construction. It is our golden reference;
3) By applying the method of Section III-A, we obtain a reference benchmark;
4) The testbench is simulated in Questa®SIM, while enabling the calculation of code coverage and functional coverage
5) Starting from the Golden Reference of Point 2, we generate a set of erroneous RTL by mutating the model against a set of possible error
6) For each mutated RTL, we execute the testbench and record coverage: we consider the mutation as detected if there is a coverage drop

In Table I we report some basic information on the subset of ITC2016 benchmark networks [12], together with experimental results. The networks from the evaluation set differ in the number and type of programmable modules. For each network given in column 1 (*Network*) following information is reported:

- Columns 2 (*SIB*) and 3 (*SM*) - the total number of SIB and ScanMux reconfigurable modules, respectively.
- Column 4 (*Conf. bits*) - the total number of configuration bits in the network
- Column 5 (*Max. depth*) - the maximum hierarchical depth of the network (for SIB-based networks this value equals to the maximum number of nested SIBs, according to [12])
- Column 6 (*Long. path*) - the length of the longest scan path in the network
- Column 7 (*Scan Cells*) - the total length of all scan cells existing in the network
- Column 8 (*TB Gen*) - the time needed to generate the testbench for the given network
- Column 9 (*TB Exe*) - the time needed to execute the testbench for the given network

In all of the considered benchmarks, the statement coverage, branch coverage and assertion coverage have reached 100% when simulating the Golden Reference RTL design.

To validate the approach, we selected the following set of possible errors:

- mismatch in one register's length

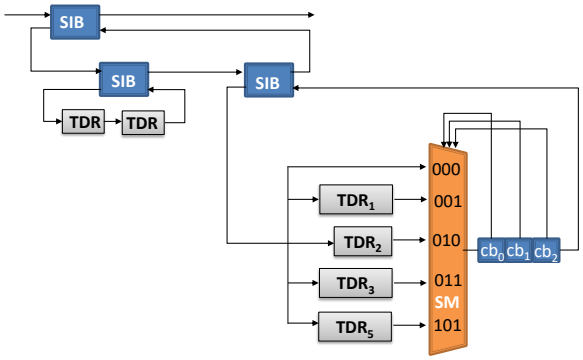

Fig. 9. TreeBalanced ScanMux with equal length registers

- wrong position of the controlling register of a SIB's: pre (i.e. ScanMux precedes the control register) - & post (the ScanMux comes after the control register
- Error in the order of scan segments connected to the inputs of scan multiplexers

This is only a subset of possible errors, but from their experience the authors deem it representative of typical human coding. As this selection only impact the experimental validation and not the benchmark generation itself, it would be extremely easy to verify coverage for other error types.

Fig. 10 depicts the output of an experimental run for a Network being subjected to given error type.

For each mutation, represented on the X-Axis, three coverage metrics are given: statement, branch and assertion. In all 63 cases but 2 (replica 50 and 56) at least one of the three types of coverage is under 100%, which is the mismatch detection condition. For this example, detection rate is therefore 88/90=97.8%.

Instruments are considered to be raw with defined data input and output ports, while the network has been designed with the *feedback* functionality to enable reading out the same values that were previously written. This is the main reason why full toggle coverage cannot be achieved on registers representing TDRs. Additional constraint is that we considered no information is available on which type of instruments are to be integrated or how they are operated.

B. Results

Table II resumes experimental results: for each network in Column 1, ranges for three types of coverage (A-Assertion, B-Branch, S-Statement) for each error type are reported in percentage of hits with respect to the total bin number. Furthermore, for each error type the number of detected errors and number of generated errors is given in the ratio form in Columns Det/Tot. Finally, for each network, the last column gives detection rate taking into account all mutations generated for each error type.

The approach provides extremely good performances: coverage is close to 100% in most cases, and execution times are extremely low, making its usage compatible with the Design

TABLE I
BENCHMARK NETWORKS LIST

Network	SIB	SM	Conf. bits	Max depth	Max path	Scan cells	TB Gen	TB Exe
Mingle	10	3	13	4	171	270	2s	5s
TreeBalanced	43	3	48	7	5,219	5,581	11s	16s
TreeFlat_Ex	57	3	62	5	5,100	5,195	12s	40s
TreeUnbalanced	28	-	28	11	42,630	42,630	6s	2m
a586710	-	32	32	4	42,381	42,410	20s	3m
q12710	27	-	27	2	26,185	26,185	5s	16s
N132D4	39	40	79	5	2,555	2,991	52s	95s
N17D3	7	8	15	4	372	462	2s	2s
N32D6	13	10	23	4	84,039	96,158	7s	4m
N73D14	29	17	46	12	190,526	218,869	9s	10m

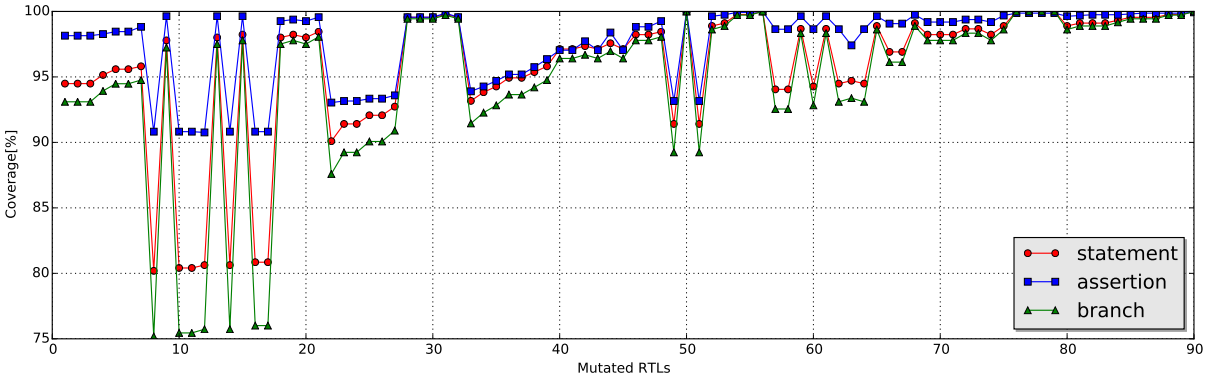

Fig. 10. Coverage for the RTL designs with wrong register lengths

flow without impacting development times. As of now, test-benches are executed until the end to obtain complete coverage metrics, but it is possible to pinpoint the moment the deviation from the Golden Reference occurs: future developments will focus on this aspect to identify the exact difference between ICL and RTL to help debugging. There are some test escapes, which need to be analyzed in more details. In these cases, the mutated circuit cannot be differentiated from the original one because of symmetry inside the network. Take for instance network *TreeBalanced*, whose part is depicted in Fig. 9: the registers TDR_1, TDR_3 and TDR_5 have the same length, so even though the ScanMux has a selection error, it is impossible to tell them apart. Rather than limitations of our chosen detection algorithm, these escapes are pathological networks which result in untestable topologies, and which should be avoided in the final silicon.

V. CONCLUSIONS AND PERSPECTIVES

In this paper we have addressed the problem of detecting inconsistencies between ICL and RTL models of RSNs, resorting to simulation-based verification. Automatically generated test-benches for stimulating the RTL model are based on the patterns used for post-silicon validation of networks. We verified our approach through a series of experimental benchmarks, obtaining extremely high detection coverage with reduced execution time. We were also able to identify test escapes as pathological network configurations. Future work will be based on extending the experimental verification to other error models such as, for instance, ICL connection errors, and to reinforce debug capabilities. Another direction will be the in-depth analysis of test escapes configurations to devise algorithms able to detect potentially untestable networks and warn the designer.

ACKNOWLEDGEMENTS

The work has been partially supported by the European Commission through the Horizon 2020 RESCUE-ITN project under the agreement No. 722325.

REFERENCES

[1] "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.

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

[3] F. G. Zadegan, U. Ingelsson, G. Carlsson, and E. Larsson, "Design automation for IEEE p1687," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*. IEEE, 2011, pp. 1–6.

[4] M. A. Kochte, R. Baranowski, M. Schaal, and H. Wunderlich, "Test strategies for reconfigurable scan networks," in *2016 IEEE 25th Asian Test Symposium(ATS)*, vol. 00, Nov. 2016, pp. 113–118. [Online]. Available: doi.ieeecomputersociety.org/10.1109/ATS.2016.35

[5] R. Cantoro, A. Damljanovic, M. Sonza Reorda, and G. Squillero, "A new technique to generate effective test sequences for reconfigurable scan networks," in *International Test Conference (ITC), 2018*. IEEE, 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 *2017 IEEE 26th 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 *On-Line Testing and Robust System Design (IOLTS), IEEE 22nd International Symposium on*. IEEE, 2016, pp. 167–172.

[9] R. Baranowski, M. A. Kochte, and H. Wunderlich, "Modeling, verification and pattern generation for reconfigurable scan networks," in *2012 IEEE International Test Conference*, Nov 2012, pp. 1–9.

[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, M. Sauer, L. R. Gomez, P. Raiola, B. Becker, and H. Wunderlich, "Specification and verification of security in reconfigurable scan networks," in *2017 22nd IEEE European Test Symposium (ETS)*, May 2017, pp. 1–6.

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

[13] A. Damljanovic, A. Jutman, G. Squillero, and A. Tsertov, "Post-silicon validation of ieee 1687 reconfigurable scan networks," in *24th IEEE European Test Symposium (ETS) (to be published)*. IEEE, 2019.

TABLE II
EXPERIMENTAL RESULTS

.

	SIB type mutations				Register length mutations				ScanMux segments mutations				
	Range [%]				Range [%]				Range [%]				
Network	A	B	S	Det/Tot	A	B	S	Det/Tot	A	B	S	Det/Tot	Detection rate
Mingle	58.5-92.2	89.8-100	92.5-100	10/10	60.2-99.2	89.8-100	92.5-100	9/9	59.7-69.4	84.6-93.2	88.7-94.5	3/3	100%
TreeBalanced	79.03-99.96	65.41-100	73.76-100	43/43	79.47-99.88	66.25-100	74.39-100	44/44	82.24-100	74.58-100	80.71-100	116/121	97.6%
TreeFlat_Ex	96.97-99.98	88.41-100	91.14-100	57/57	97.51-99.98	88.71-100	91.38-100	63/63	98.17-100	93.59-100	95.1-100	116/121	97.9%
TreeUnbalanced	84.6-99.92	68.45-100	75.61-100	28/28	85.31-99.92	70.83-100	77.45-100	35/35	-	-	-	-	100%
a586710	-	-	-	-	100-100	100-100	100-100	0/32	-	67.18-96.48	75-97.32	32/32	50%
q12710	93.89-99.92	85-100	88.72-100	27/27	93.89-99.85	86.5-100	89.84-100	23/23	-	-	-	-	100%
N132D4	97.12-99.97	91.18-100	92.91-100	39/39	95.1-100	79.08-100	83.17-100	167/172	95.73-100	79.08-100	87.17-100	37/40	96.8%
N17D3	79.69-98.49	80.35-99.1	84.43-99.29	7/7	72.43-100	80.35-100	84.43-100	25/27	76.94-88.72	78.57-97.32	83.01-97.87	8/8	95.2%
N32D6	85.64-99.74	79.85-100	83.95-100	13/13	82.56-100	73.13-100	78.6-100	43/44	81.02-97.69	78.73-98.88	83.06-99.1	10/10	98.%
N73D14	93.04-99.93	87.59-100	90.09-100	29/29	90.76-100	75.18-100	80.19-100	88/90	87.12-91.25	72.42-81.98	77.98-85.61	17/17	98.5%