Design, Verification and Application of IEEE 1687

Ghani Zadegan, Farrokh; Larsson, Erik; Jutman, Artur; Devadze, Sergei; Krenz-Baath, René

Published in:
[Host publication title missing]

DOI:
10.1109/ATS.2014.28

2014

Design, Verification and Application of IEEE 1687

Farrokh Ghani Zadegan & Erik Larsson
Lund University, Lund, Sweden

Artur Jutman & Sergei Devadze
Testonica Lab, Tallinn, Estonia

René Krenz-Baath
Hochschule Hamm-Lippstadt, Hamm, Germany

***Abstract*—IEEE 1687 (IJTAG) has been developed to enable flexible and automated access to the increasing number of embedded instruments in today's integrated circuits. These instruments enable efficient post-silicon validation, debugging, wafer sort, package test, burn-in, bring-up and manufacturing test of printed circuit board assemblies, power-on self-test, and in-field test. Current paper presents an overview of challenges as well as selected examples in the following topics around IEEE 1687 networks: (1) design to efficiently access the embedded instruments, (2) verification to ensure correctness, and (3) fault management at functions performed in-field through the product's life time.**

***Keywords*—IEEE 1687 (IJTAG); on-chip instruments; robustness; access time; network design; verification; fault management**

I. Introduction

Integrated circuits (ICs) are equipped with a considerable number of features for test, debugging, configuration, and monitoring. These embedded features are commonly referred to as on-chip instruments. IEEE 1687 (IJTAG) [1] standardizes the access to the instruments, mainly through the JTAG test access port (TAP) [2], by specifying an architecture and a set of description languages.

The aim of IEEE 1687 is to facilitate automation and reuse of on-chip instruments throughout the life cycle of an electronic product from design and characterization to in-field test and monitoring. The idea is to describe the operation of each instrument at its terminals by using a high-level description language, and let a *retargeting* tool translate these instrument-level commands to human- or machine-readable commands at higher levels of the design hierarchy.

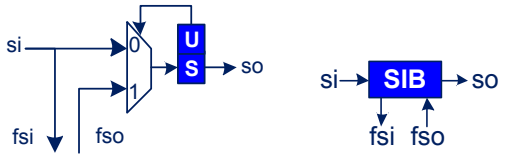
The flexibility proposed by IEEE 1687 makes it a challenge to create an instrument access infrastructure (1687 *network*) which is efficient in terms of access time and hardware overhead. There is a need of EDA tools for design automation, verification, and validation. In this work, we will discuss some of the challenges in the development of 1687-related tools with an emphasis on network design (Section III), verification (Section IV) and fault management (Section V).

II. Background

A. Instrument Access Infrastructure (Network)

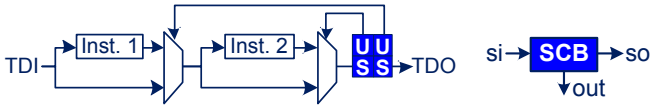
A strong feature in IEEE 1687 networks is the possibility of dynamic reconfiguration, which allows for reduction of instrument access time by varying the length of the scan-path to include only those instruments in the path which are needed for current session. To enable variable-length scan-paths, 1687 introduces two components: a Segment Insertion Bit (SIB) and a *ScanMux* control bit.

SIB is used to include or to exclude a scan-chain from the active scan-path. Fig. 1 shows a simplified schematic of a possible implementation of a SIB, as well as a symbol which



(a) Simplified schematic (b) Symbol

Fig. 1. Segment Insertion Bit (SIB)



(a) A network of two instruments, configured by a two-bit ScanMux control register (b) Symbol

Fig. 2. ScanMux control register (SCB)

we will use through the rest of this paper. Fig. 1(a) shows only as few components and terminals as are needed to explain the operation of a SIB: a one-bit shift-update register, and a mux. A complete schematic would contain more components (such as logic gates for gating control signals, keeper muxes for the registers, and delay elements to avoid race condition) and terminals (such as selection and control signals used to enable shift and update operations). A *ScanMux* control bit is a shift-update register that can be placed anywhere on the scan-path to configure one or more scan multiplexers (ScanMux components). Fig. 2(a) shows a two-bit ScanMux control register used to configure a network of two instruments. In this work, we consider one-bit ScanMux control registers (referred to as SCB in this work) to control two-input muxes which bypass instrument shift registers in, e.g., daisy-chained architectures. We will use the symbol in Fig. 2(b) to represent an SCB in the rest of this paper.

Both SIBs and SCBs must be configured to have correct value every time the scan-path they are on is accessed. Such configuration data results in a time overhead since it is not part of instrument data.

To access the network of instruments from the chip boundary, 1687 specifies the JTAG TAP as the primary interface. Interfacing is performed by connecting the first level (SIBs) of the 1687 network as a custom TDR to the JTAG circuitry. This TDR is referred to as the Gateway. As an example, Fig. 3 illustrates a small 1687 network consisting of three instruments (namely a DFT instrument, a sensor, and a debugging feature) and four SIBs. The instruments are interfaced to the scan-path through shift-registers with parallel I/O. Initially, the SIBs are closed and the scan-path consists of the two SIBs which form the Gateway TDR. To access the instruments, SIBs must be programmed to include corresponding shift-registers into the scan-path. In this paper, *access* is defined as (1) shifting input bits into the instrument's shift-register (shift phase), (2)

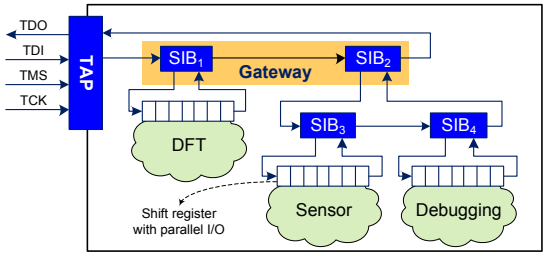


Fig. 3. A 1687 network with three instruments inside a chip

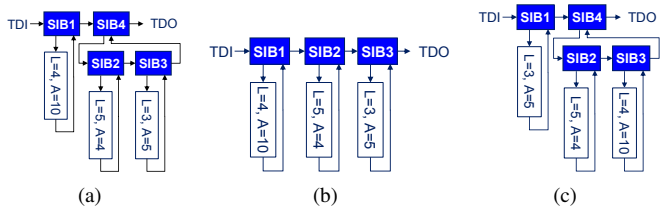


Fig. 4. Three different ways to create a network for the same instruments

latching the contents of the shift-register to be applied to the internal circuitry of the instrument (update phase), (3) capturing the output of the instrument into the shift-register (capture phase), and (4) shifting the captured values out (shift phase). The shifting out of the instrument outputs can overlap in time with shifting in the input bits for the next access. The number of clock cycles it takes to perform the update and capture phases and go back to the shift phase is referred to as Cycle of Update and Capture (CUC) [3].

B. Description Languages and Retargeting

1687 introduces two description languages, namely Instrument Connectivity Language (ICL) and Procedural Description Language (PDL). ICL is used to describe the network, that is, how the instruments are connected to the JTAG TAP. PDL is used to describe the operation of instruments at their terminals. PDL commands allow to perform read/write operations on the instrument shift-registers and configurable components, as well as to wait for an instrument (such as a BIST engine) to finish its operation.

Given the PDL of each instrument, a retargeting tool generates scan vectors to configure the network and transport the required data bits from the JTAG TAP to/from the instruments' shift-registers. A retargeting tool relieves the designer from dealing with network configuration (i.e., writing the PDL to configure SIBs and SCBs directly). For example, assuming that the goal is to read the value from the sensor instrument in Fig. 3, the PDL developer might simply use a write command to activate the sensor, a wait command to wait for the sensor to capture the value, and a read command to read the captured value out. It is then the task of the retargeting tool to generate one scan vector to open SIB_2, one vector to open SIB_3, one vector to write to the enable bit in the sensor's shift-register, a wait cycle of enough length, and finally one vector to scan the captured value out.

III. Analysis and Design

In this section, we discuss how to design a 1687 network that is robust against late changes in how instruments are accessed.

A. The Need for Methodology

The flexibility in 1687 networks brings two types of freedom: freedom to construct the network in a multitude of different ways, and freedom to schedule the access to the instruments in a variety of ways according to the given constraints (e.g., resource conflicts and power budget). As an example (Fig. 4), consider three instruments with the interface shift-register length of four, five and three flip-flops, which are to be accessed ten, four and five times, respectively. Fig. 4 shows three different ways to connect these three instruments as 1687 networks. The network in Fig. 4(b) uses less hardware components (i.e., SIBs in this example) compared to the other two networks. It is interesting to compare these networks w.r.t. the overall access time (OAT), which is the time it takes to transfer instrument data (i.e., perform the required read/write operations on the instruments), plus the network configuration time [3]. Regarding OAT, although the instrument data is the same for all networks, the network in Fig. 4(a) shows less OAT in comparison with the networks in Fig. 4(b) and Fig. 4(c). Moreover, for each of the networks shown in Fig. 4, the access to the instruments can be scheduled in a variety of ways, each way potentially resulting in a different OAT number [4].

The above example shows that there is a need for methods to design optimized 1687 networks (in terms of instrument access time, hardware overhead, etc.). Development of such methods, in turn, requires an exact analysis of different trade-offs (e.g., time overhead vs. hardware overhead). The following works have partially addressed these needs. Access time analysis for SIB-based 1687 networks, under sequential and concurrent access schedules, is presented in [3]. Optimized SIB-based network construction is presented in [5] for concurrent and sequential schedules. The work in [4] presents an access time calculation method for general schedules, as well as optimized power- and resource-constrained test scheduling, for SIB-based 1687 networks. The assumption in the above works is that instruments are always accessed in the same way. In practice, however, instruments might be accessed differently under diverse circumstances (referred to as usage *scenarios*). Therefore, a network optimized for one scenario is not necessarily good for other scenarios. Below, we study robustness of 1687 networks with respect to the changes in access scenario (i.e., change in the number of accesses and the access schedule), and compare seven network design approaches regarding OAT, hardware overhead, and robustness. The complete study can be found in [6].

B. Study of Robustness

Embedded instruments are used to enable post-silicon validation, debugging, wafer sort, package test, burn-in, printed circuit board (PCB) bring-up, PCB assembly manufacturing test, power-on self-test, and in-field test. For each of these scenarios, it is of interest to access some but not all of the instruments [7]. As an example, a memory built-in-self-test (MBIST) instrument might be accessed (1) during yield learning for a new process to choose the most suitable algorithms, (2) during wafer sort and package test to detect defective

Fig. 5. A flat network with dedicated SIBs for instruments

devices and perform repair, (3) in the burn-in process to cause activity in the chip and to detect infant mortality [8], [9], (4) during PCB bring-up [10], (5) during PCB assembly manufacturing test [10], and (6) during power-on self-test and in-field tests. Also, the number of accesses to a given instrument typically varies between different scenarios. For example, during yield learning, an embedded memory might be tested several times by running multiple BIST algorithms. Another example is reading out the memory contents for diagnostic purposes [11]. In both examples many accesses might be needed. In contrast, during manufacturing tests, an embedded memory might be tested only by accessing the associated MBIST engine a few times to setup the algorithm, start the BIST, check for its completion, and read the results.

Furthermore, at design time it is difficult to foresee all needed scenarios and how many times an instrument will be accessed at each of the scenarios. The number of needed scenarios and the number of accesses might be affected by late design changes, adding/excluding tests, or change of constraints, such as power consumption. Some changes may only be known after manufacturing.

Considering the above and that a network optimized for a scenario, might not be optimal when the number of accesses or the access schedule changes, we study the robustness of seven approaches for designing 1687 networks, and examine their efficiency, in respect to OAT and hardware overhead. Intuitively, a robust approach should introduce as little time overhead as possible into OAT regardless of the scenario. That is, considering that OAT consists of both instrument data and overhead (i.e., clock cycles spent on network configuration and CUC), an approach is said to be robust if the ratio of OAT to instrument data does not change dramatically between scenarios. Therefore, we calculate the ratio of OAT to instrument data for each scenario that a given approach is used in, and we consider the standard deviation of the calculated ratios as the metric for robustness of that approach. The smaller the metric value is, the more robust the approach will be.

The studied network design approaches are (1) a flat network, (2) a single hierarchical network, and (3) multiple networks each optimized for a given scenario, as well as a daisy-chained counterpart for each of these three. In addition to the six enumerated approaches, we study one more approach in which two separate JTAG test data registers (TDRs) are used for the instrument access network: one to configure the access network, and one to access the instruments. In the following, the studied design approaches are briefly explained.

1) Flat Network: To construct a flat network, each instrument's shift-register is connected through a SIB (Fig. 5). To access each of the instruments, the corresponding SIB is programmed to include that instrument in the scan-path. The hardware overhead is minimal. For time overhead, since the SIBs are always on the scan-path, they contribute to the overhead for every access.

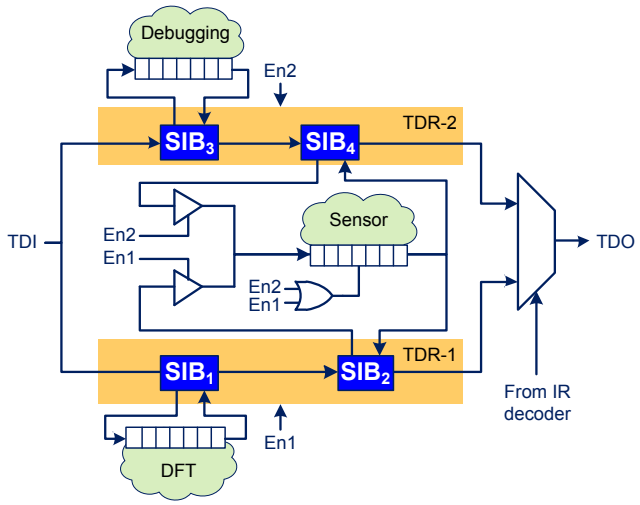


Fig. 6. The sensor instrument is shared by two networks (TDRs)

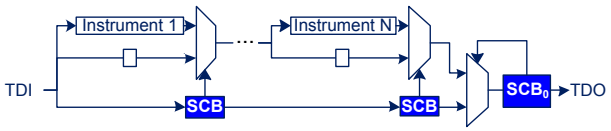


Fig. 7. A flat daisy-chain

2) Hierarchical Network: In a hierarchical network, in addition to the SIBs dedicated to switching the instruments' shift-registers on and off the scan-path, some SIBs are used to switch a network segment (including other SIBs and shift-registers) on and off the scan-path. An example of a hierarchical network is shown in Fig. 3 where the DFT feature is placed in the first level, and the sensor and debugging instruments are placed in the second level. Such hierarchical approach allows for reduction of OAT by excluding the SIBs themselves from the scan-path (when the segment they belong to is not used in the current access). That is, when the sensor and debugging instruments are not needed, their corresponding shift-registers and dedicated SIBs are excluded by programming SIB_2 to be closed. To construct a hierarchical network optimized for multiple scenarios, [6] uses an adaptation of the method in [5] (which was proposed for a single scenario with sequential access schedule).

3) Multiple Networks: In this approach, a dedicated network is designed and optimized for each scenario. Each network is then connected to the JTAG TAP through a dedicated TDR. The instruments whose interface shift-register is to be accessed through multiple scenarios (i.e., multiple TDRs) can be shared among the corresponding networks by using, for example, a scheme similar to the one shown in Fig. 6. In the presented scheme, tristate buffers are used to control to which network the shared instrument shift-register is connected. The enable signals in this scheme (i.e., En1 and En2) are applied from the TAP circuitry. For the design of each network for its given scenario, the algorithms in [5] can be used. For each network and its given scenario, access time is calculated by using the algorithms proposed in [3].

4) Daisy-chained: The daisy-chaining approach for 1687 is illustrated in Fig. 7. To switch the instrument shift-registers on

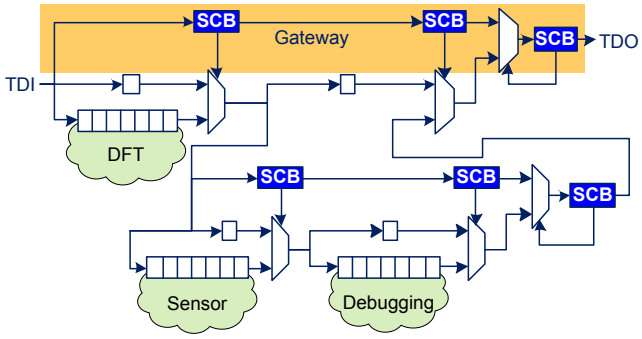


Fig. 8. An example of the use of hierarchy in daisy-chaining

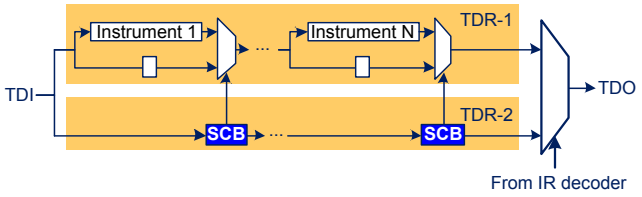


Fig. 9. Using separate TDRs for instruments and SCBs

and off the scan-path, multiplexers are used. These multiplexers are controlled by SCBs placed on a separate branch of the scan-path. To select between the two branches, SCB_0 is used. To avoid long combinational paths, bypass registers are used in place of an excluded shift-register. Fig. 8 shows a hierarchical daisy-chained network, which can be seen as a counterpart for the hierarchical SIB-based network in Fig. 3, in the sense that one instrument is placed at the first level of hierarchy while the other two are placed at the second level. This way, it is possible to create a daisy-chained counterpart for each of the SIB-based flat and hierarchical networks discussed in previous sections. In the same way, for the multiple networks approach, a daisy-chained counterpart can be constructed for each of the networks.

5) Separate Control and Data TDRs: In this approach, there is one TDR for SCBs and one TDR for instruments (Fig. 9). When the scan-path is needed to be reconfigured, the TDR with control bits is accessed (i.e., TDR-2). Then, after the scan-path is reconfigured, the TDR with the instruments (i.e., TDR-1) is selected in order to access the instruments. In this architecture, since SCBs are not on the same scan-path as the instruments, it is possible to pipeline the instrument data through the bypass registers, and therefore effectively reduce the time wasted in the bypass registers.

To compare the studied network design approaches, two sets of experiments were performed. In the first set, 100 instruments and eight scenarios for accessing them were considered, and OAT achieved under each scenario was calculated and summed up for each of the above-mentioned seven approaches. This experiment was performed for three cases where (A) only one scenario (out of eight) is known at chip design time, (B) five out of eight scenarios are known at design time, (C) all scenarios are known at design time. The chart in Fig. 10 presents the results for the first set of experiments. The second set of experiments studied robustness against change of concurrency in a given scenario. Fig. 11 shows the results of this

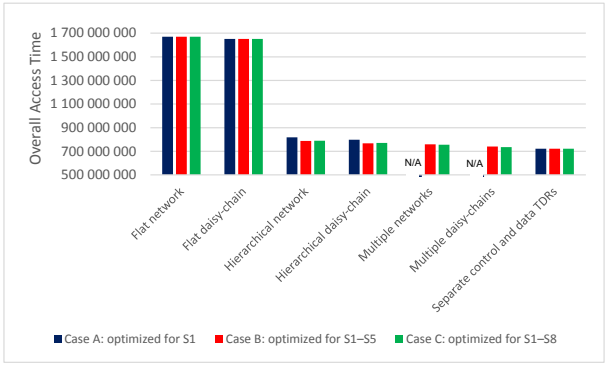


Fig. 10. Summarizing the results for the three cases (N/A signifies that for Case A, "Multiple networks" and "Multiple daisy-chains" are not applicable)

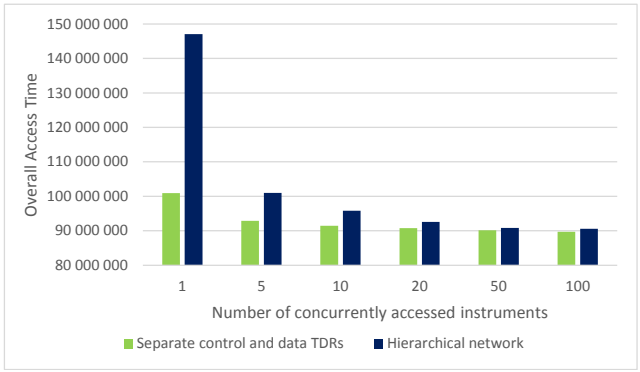


Fig. 11. Change in OAT as concurrency increases

experiment in which for one of the given scenarios, the number of concurrently active instruments is gradually increased from one active instrument to 100 concurrently active instruments.

The experimental results for both sets of experiments showed that the "separate control and data TDR" approach results in the least sum of OAT among the seven considered approaches, has relatively low hardware overhead, and is the most robust among the studied approaches. In this approach, network configuration registers (SCBs) are placed in a TDR separate from the instrument shift-registers TDR (see Fig. 9).

IV. Verification and Validation

Efficient verification and optimization of reconfigurable 1687 networks is one of the key necessities to enable broad application of IEEE 1687. In this section, several examples are presented to show future verification and validation challenges of 1687 networks. Specifically, the challenge of extracting combinational invariants as well as temporal invariants of a given 1687 network are discussed. Extracting these functional invariants is crucial in order to evaluate the possibilities to retarget sets of PDL commands onto a given 1687 network, as well as evaluating 1687 networks with respect to different performance properties.

State-of-the-art approaches offer solutions to solve verification and retargeting challenges. In [12] the authors proposed a first approach to model 1687 networks in order to retarget PDL commands. In [13] the authors extend the earlier proposed method by pseudo-Boolean optimization in order to minimize

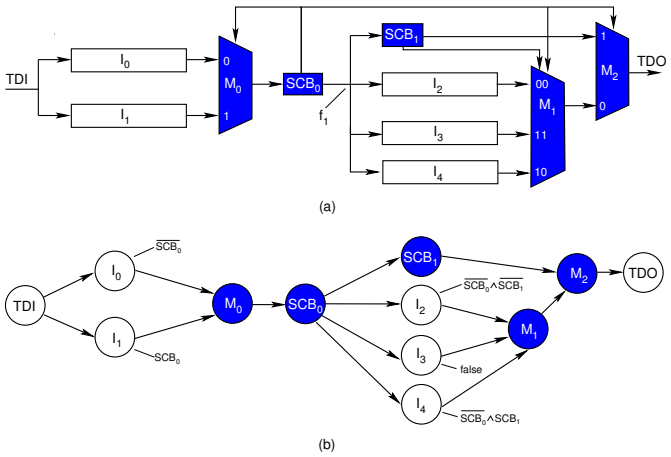


Fig. 12. (a) Example 1687 network; (b) Corresponding graph.

the number of scan access operations and hence the time to perform PDL commands on the 1687 network.

The 1687 network, depicted in Fig. 12(a), consists of three multiplexers $M_0, \ldots, M_2$, five instruments $I_0, \ldots, I_4$ and two SCBs denoted SCB_0 and SCB_1. In order to perform reasoning on a 1687 network, the authors in [12] propose to generate a graph representation $G = (V, E)$, where G is a directed graph, and the sets V and $E \subseteq V \times V$ denote the set of vertices and the set of connecting edges, respectively. A vertex $v \in V$ represents test data inputs (TDIs), test data outputs (TDOs), scan elements, instruments as well as multiplexer or fan-outs within the modeled 1687 network. Every edge $e \in E$ represents a real connection between the components. The graph corresponding to the 1687 network shown in Fig. 12(a) is depicted in part (b) of the same figure. The colored vertices in the graph correspond to control elements of the 1687 network.

To compute a complete scan-path from some TDI to some TDO, every vertex $v \in V$ is activated by a corresponding select signal $sel(v)$, where $sel(v)$ is represented by a predicate function evaluating to *true*, iff the corresponding predicate functions of the predecessor $p \in V$ and the successor $s \in V$ of v evaluate to *true*, $sel(v) = sel(p) \wedge sel(s)$. When there exist several paths $\{s_n, \ldots, s_m\}$ branching out of a vertex towards TDO, e.g., a fan-out $f \in V$, the corresponding predicate function $sel(f)$ is the disjunction of the predicate functions of all successors of f, i.e., s_i, $sel(f) = \bigvee_{\forall_i} sel(s_i)$. Similarly, when multiple paths enter a vertex, e.g., through a multiplexer, the corresponding predicate function is the disjunction of all predicate functions of all predecessors of that multiplexer. To keep the discussion simple, we skip the additional formulas needed to ensure that at most one out of multiple paths from a fan-out (or to a multiplexer) is active at a given time. If the successor of some vertex $v \in V$ is a multiplexer, then the predicate function must fulfill the conditions to activate the corresponding path. The predicate functions of every instrument $I_0, \ldots, I_4$ are depicted next to the corresponding instrument. For example, instrument I_0 in Fig. 12 can only be part a complete scan-path if SCB_0 = 0 , $sel(I_0) = \overline{SCB_0}$.

The necessity to extract combinational invariants on 1687 networks can be easily demonstrated on the given 1687 network. It is easy to see that concurrent access of instruments I_0 and I_1 is not possible since the corresponding predicate functions $sel(I_0)$ and $sel(I_1)$ are mutually exclusive. This property can be discovered by simple structural methods. However, that the activation of instrument I_3 is impossible in the given network, is not solvable by structural approaches. In order to generate a complete scan-path containing I_3 the multiplexers M_1 and M_2 need to be configured appropriately. The path from I_3 to M_1 is activated, if SCB_0 = '1' and SCB_1 = '1' and the path from M_1 to M_2 is activated if SCB_0 = '0'. The conditions for activating both paths are incompatible, $(SCB_0 \wedge SCB_1) \wedge \overline{SCB_0}$ = *false*. Hence there exists no valid scan-path containing instrument I_3 in the described network. Revealing this property is only possible by functional reasoning approaches. In general, these methods are based on Boolean satisfiability [14], [15] or Binary Decision Diagrams (BDDs) [16].

An even more complex problem is the extraction of temporal invariants. For example, if instruments I_2 and I_4 are required to be accessed in two consecutive shift-and-update cycles, then the applied reasoning approach needs to take several time frames into account. Such sequential reasoning approaches, for example BDD-based sequential model checking [17] or SAT-based bounded model checking [18], are significantly more complex than combinational reasoning approaches. Assume that instrument I_2 is part of the currently activated scan-path, hence the logic values in SCB_0 and SCB_1 are '0' during the first shift-and-update cycle. In order to access instrument I_4, which implies that the logic values in SCB_0 and SCB_1 are required to be '0' and '1', respectively, at least one intermediate shift-and-update cycle is required. Since SCB_0 is contained in every valid scan-path obtaining a logic '0' is trivial. However obtaining a logic '1' in SCB_1 requires an additional shift-and-update cycle, where the shift-and-update cycle enables a scan-path containing SCB_1 in order to load the required logic '1'. As mentioned at the beginning of this section, the above described temporal property has a major impact on access time for specific PDL commands.

The third major challenge is to prove the functional equivalence between a 1687 network description provided in ICL and any other system description, for example VHDL or Verilog. Sequential equivalence checking has been one of the most challenging tasks in EDA during the last decades [19], [20]. In order to prove functional equivalence of two systems in a worst-case scenario, it is required to generate and compare a complete representation of the state space of both systems, which is known to be exponential with respect to number of memory elements contained in the systems. Once nonequivalence is proved, the challenge is to analyze and debug functional differences in order to extract relevant design differences and to correct the presumed bug in the design or the corresponding ICL description [21].

Finding the root cause for an unsuccessful retargeting of a set of PDL commands onto some ICL network is similar to the problem of logic debugging [22]. In SAT-based logic debugging, the verified system S is expressed in *Conjunctive Normal Form* (CNF), denoted C_S. CNF is a conjunction of clauses where each clause is a disjunction of a set of literals, and a literal represents a Boolean variable or its negation. A given

CNF is satisfied, if some assignment of the Boolean variables is found, which satisfies each of the contained clauses; a clause is satisfied if it contains at least one literal, which evaluates to one under the current variable assignment. Assuming that system S is designed in order to satisfy a certain property P and this property is also modeled as a Boolean function in form of a CNF, denoted as C_P, then a satisfying variable assignment for the conjunction of C_S and C_P represents one possible scenario under which property P is realized on system S. However, if there exists no satisfying solution for $C_S \wedge C_P$, then system S cannot comply with property P under any circumstance. State-of-the-art debugging methods add an infrastructure to the system representation which enables the injection of arbitrary error candidates or certain modifications of the 1687 network in order to modify the system behavior until the SAT instance becomes satisfiable. In order to finally deduce the actual design error, the above process iteratively injects several error candidates or network modifications into the system model S. The performed analysis is refined by using intermediate results obtained during previous iterations. The final goal is to pin-point to one or several design errors causing the deviation from the initially intended system behavior or network properties which prevent a successful retargeting.

The debugging approach for a non-successful PDL-retargeting could follow the same procedure. Let us re-apply the above example, where the instruments I_2 and I_4 should be accessed in two consecutive shift-and-update cycles. As described earlier, at least one intermediate shift-and-update cycle is needed between the two instrument accesses. The reason for this obstacle is that SCB_1 can only be accessed if SCB_0='1', which is not possible if instrument I_2 is accessed. By applying the discussed debugging concept, for example it would be possible to apply a modification of the 1687 network, where SCB_1 is moved between SCB_0 and the fan-out denoted f_1 in Fig. 12(a). Applying this modification would enable a successful retargeting of the above set of consequtive PDL commands. Hence the position of SCB_1 is recognized as one root cause to the unsuccessful PDL retargeting on the original network.

V. Application in Fault Management

It has been shown that IEEE 1687 standard infrastructure can also be successfully and efficiently used or reused later in the field during the product's life cycle. One such application is fault management [23]—the usage that was hardly foreseen by the IJTAG standard development action group. IEEE 1687 standard allows to create an efficient and regular network for continuously handling fault detection information as well as to manage test and system resources as a system-wide background process during the system operation. The IJTAG framework matches especially well the requirements for supporting graceful degradation of the silicon under pressure of aging, the problem that has been recently reported by the ITRS among a few most important research challenges today [24]. This section describes the way the IEEE 1687 infrastructure can be efficiently extended to support fast emergency signaling for online-fault detection and localization as a background process in a running system. The extension for on-line diagnostics is based on two main components:

1) Hierarchy of status flag registers (IJTAG-compliant);

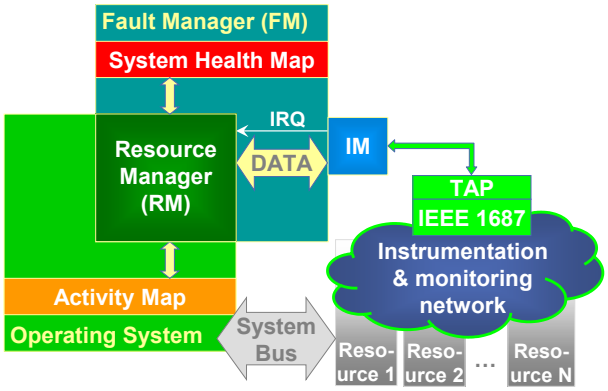


Fig. 13. Failure resilient system with IEEE 1687 fault management network

2) Asynchronous emergency network (non-IJTAG compliant).

A. Aging Failure Resilience and Fault Management

The aging failure resilience framework that we describe is based on fault tolerance and system health monitoring but goes beyond by localizing and classifying faults into different categories that should be handled differently: e.g. transient vs. permanent faults and critical vs. low-priority ones [23]. Aged blocks with permanent faults are then either fully isolated or marked as reduced-capacity ones, which is registered in the system health map. Fig. 13 reflects the top abstraction level of the whole concept. We assume that the fault management framework operates on the same SoC as the target system itself and that the target system contains plenty of heterogeneous or identical resources (or IP cores) capable of fully or partially replacing one another, hence providing a room for graceful degradation by proper task scheduling. The tasks are scheduled by the operating system (OS), which takes into account the System Health Map status provided and maintained by the Fault Manager (FM). FM receives interrupts and collects service/diagnostic information from the IEEE 1687-based Diagnostic Instrumentation Network (DIN) that is implemented on the same SoC as the target system itself. The actual error detection is taking place in-situ by embedded instruments/monitors. The DIN should then immediately pass an emergency signal from the monitor to the OS so that the latter could reschedule the failed task immediately to another available resource. After detecting the fault, the failure has to be diagnosed and analyzed in order to update the system health map and isolate the resource in case of permanent fault detection. IEEE 1687 instrumentation network operates independently from the rest of the system (on the background), which is another very important feature due to the two following reasons. First, the DIN must be functioning even if the system's main communication channels are out of order and second, during maintenance, the service actions should not interfere with the functional tasks performed by the system. In general, the DIN should be simple enough to achieve better levels of reliability than the rest of the system. In context of aging, the service infrastructure should degrade slower than the functional part. This agrees very well with comparatively seldom background on-demand operation of the IEEE 1687 based DIN.

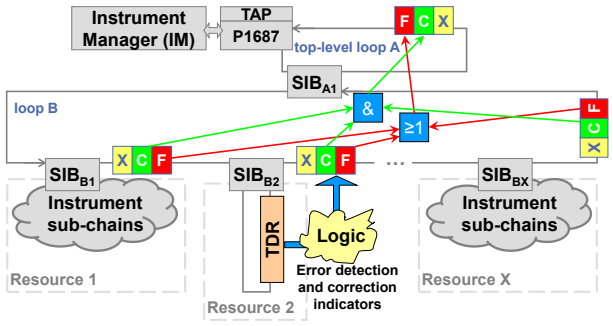


Fig. 14. IJTAG network with asynchronous error indication and propagation system

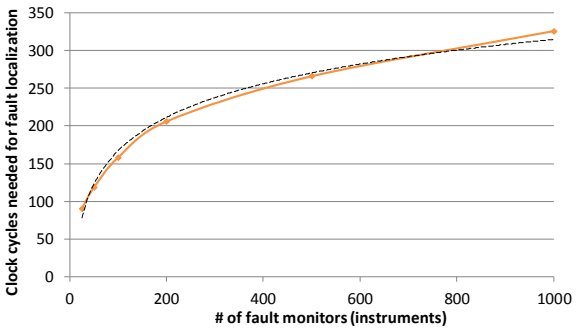


Fig. 15. Performance of the optimized DIN [23]

B. Automation and Scalability Aspects

Given that SoC components, both IP cores and instruments, are usually provided by different vendors, the DIN has to be automatically composed during the system integration phase. Hence, the possibility of design automation, reuse, and sharing provided by a standard-driven approach for infrastructure design is very important. In addition to the normal IEEE 1687 requirements, the DIN needs to be based on unified interfaces to instrument clusters coming from different vendors as a part of IP cores. This fact directly affects the two extensions proposed: status flags and asynchronous signaling. The next subsection details respective requirements (design rules), which enables automatic seamless integration of heterogeneous diagnostic resources into a single homogeneous system-wide DIN. Thanks to these rules and the IEEE 1687 standard, the DIN can be constructed in a way that allows creation of a single very small controller (instrument manager - IM) that handles all faulty resource localization tasks and instrument control for the whole SoC independently of its size [23]. Scalability of IJTAG networks is already good by its nature, but following the concept of hierarchical instrumentation clusters (or segments) allows keeping the active scan chain length within linear or even logarithmic relation with respect to the number of instruments in the SoC [23]. The same hierarchical concept matches the SoC composition principles; hence the IJTAG overhead in general can be kept under a good control.

The main overhead from the adaptation of the IJTAG infrastructure for the fault management purposes is generated by adding status flag registers and the asynchronous signaling. The complexity of both is linear to the number of existing SIBs in the network. The former components could be implemented as a part of SIB, while the asynchronous signals can be easily routed along and together with the main scan chains (no routing overhead). As the result, the extended infrastructure of a relatively modest overhead is capable of fulfilling the first priority requirements for an effective fault management system, such as low error detection latency, high faulty resource localization speed, as well as good reliability and scalability of the service infrastructure itself.

C. Architectural Implementation Details

Fig. 14 demonstrates an example of a DIN implementation in a many-core SoC, which is described in more detail in [25]. The flag-based error reporting system where each block or a sub-module is provided with a dedicated Status Flag Register FCX (F: failure, C: corrected, X: mask), whereas all registers are collectively forming a hierarchical asynchronous error indication and propagation structure tied with IEEE 1687 network has been detailed in [25]. As it was mentioned, error detection appears in instruments. As soon as an error is detected, the corresponding local fault flag F of a particular instrument, resource or core is set to 1. If the error was corrected (e.g. by ECC) the correction flag C is also set to 1. Status flags are propagated to the top-level loop appearing in the top-level FCX register. The 4-bit long top-level loop ($\mathrm{F_{TOP}}$, $\mathrm{C_{TOP}}$, $\mathrm{X_{TOP}}$ and SIB) is being repeatedly polled by the IM, which in general takes 4 TCK cycles (plus cycles required for TAP state transitions). In case of an error, (when $\mathrm{F_{TOP}} = 1$, $\mathrm{C_{TOP}} = 0$ condition met) the IM sends an interrupt request (IRQ) to the FM and traces down the faulty resource by reading the corresponding F flags next to each SIB. At each hierarchical level, the IM activates only the segment marked with active F flag while keeping all other segments inactive. The process continues until the IM reaches the faulty resource and reports its location (or ID) to the FM. IM can also receive commands from the FM and transports requested data between registers/instruments and FM. When FM works with resource-specific registers, IM continues to poll the top-level Status Register. The purpose and functionality of the FM is detailed in [26]. The meaning of the FCX Status Register flags at SIB i is summarized below.

- $F_i = 1$, $C_i = 0$ – error detected but not corrected
- $F_i = 1$, $C_i = 1$ – error detected and corrected
- $F_i = 0$, $C_i = 1$ – default value during normal operation
- $F_i = 0$, $C_i = 0$ – malfunction of the error detection network
- X_i – is a mask bit, which can be set in order to ignore a particular Status Register, e.g. if the corresponding resource is dead or not used.

When $\mathrm{F_{TOP}} = 1$, $\mathrm{C_{TOP}} = 1$ condition met (error detected and corrected), the FM updates the error detection statistics without sending IRQ. The error rate statistics is collected on the background per resource. FM periodically checks error rate statistics and takes actions if needed. As it was mentioned the status bits are being propagated from the instruments towards

the top level loop's status register through the asynchronous network that is essentially a combinational circuit (see Fig. 14) characterized by the following rules:

- $\forall$i, $\mathrm{F_i} = 1 \Rightarrow \mathrm{F_{TOP}} = 1$;
- $\forall$i, $\mathrm{C_i} = 0 \Rightarrow \mathrm{C_{TOP}} = 0$;

Hence the flag propagation delay from the leaves to the root of the network tree, is proportional to the lengths of the corresponding combinational path (logic depths) in this combinational circuit. The latter parameter can be kept of the logarithmic complexity with respect to the number of instruments in the system. The faulty resource (or IP core) localization speed (see Fig. 15) follows the same logarithmic trend. See [23] for details.

D. Summary

The hardware side of the system under test is equipped with embedded test instrumentation. The software side includes FM that maintains the system health map by analyzing diagnostic data from instruments. IM represents a very simple component that takes care of the regular instrumentation network and unified status flag registers FCX, while FM is dealing with core-specific registers/instruments.

Assuming that error detection inside the resources is performed by dedicated monitors, checkers, sensors, etc. collectively called instruments; the new emerging IEEE 1687 standard seems to be a proper choice for the fault management task. The efficiency of the IEEE 1687 instrumentation infrastructure used for fault management has been recently demonstrated in [23] and characterized by a very small error detection latency (practically immediate fault detection) and logarithmic time to localize the faulty resource.

VI. Conclusion

In this work, we briefly surveyed the research done so far on IEEE 1687, with regards to the design and verification of 1687-compatible networks, and presented and discussed some of the challenges to be addressed in these areas. Moreover, we presented an application of IEEE 1687 in a fault-tolerant design.

Acknowledgment

This work has been jointly supported by EU through FP7-2013-ICT-11: 619871 project BASTION and European Regional Development Fund.

References

[1] IEEE Association, "IEEE Draft Standard for Access and Control of Instrumentation Embedded within a Semiconductor Device," workgroup URL: http://grouper.ieee.org/groups/1687.

[2] IEEE Association, "IEEE Std 1149.1-2001, IEEE Standard Test Access Port and Boundary-Scan Architecture," 2001.

[3] F. G. Zadegan et al., "Access Time Analysis for IEEE P1687," *IEEE Transactions on Computers*, vol. 61, no. 10, pp. 1459–1472, Oct. 2012.

[4] F. G. Zadegan et al., "Test Scheduling in an IEEE P1687 Environment with Resource and Power Constraints," in *Proc. Asian Test Symposium (ATS)*, 2011, pp. 525 –531.

[5] F. G. Zadegan et al., "Design automation for IEEE P1687," in *Proc. Design, Automation Test in Europe Conference (DATE)*, March 2011.

[6] F. Ghani Zadegan, G. Carlsson, and E. Larsson, "Robustness of TAP-Based Scan Networks," in *Proc. IEEE Int'l Test Conf. (ITC)*, 2014.

[7] M. Keim et al., "Industrial Application of IEEE P1687 for an Automotive Product," in *Euromicro Conference on Digital System Design (DSD)*, Sept 2013, pp. 453–461.

[8] K. Yamasaki et al., "External Memory BIST for System-in-Package," in *Proc. IEEE Int'l Test Conf. (ITC)*, 2005.

[9] A. Carbine and D. Feltham, "Pentium(R) Pro Processor Design for Test and Debug," in *Proc. IEEE Int'l Test Conf. (ITC)*, 1997, pp. 294–303.

[10] Z. Conroy et al., "Board Assisted-BIST: Long and Short Term Solutions for Testpoint Erosion – Reaching into the DFx Toolbox," in *Proc. IEEE Int'l Test Conf. (ITC)*, 2012.

[11] A. Margulis et al., "Evolution of Graphics Northbridge Test and Debug Architectures Across Four Generations of AMD ASICs," *IEEE Design & Test*, vol. 30, no. 4, pp. 16–25, Aug 2013.

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

[13] R. Baranowski, M. A. Kochte, and H.-J. Wunderlich, "Scan Pattern Retargeting and Merging with Reduced Access Time," in *Proceedings of IEEE European Test Symposium (ETS'13)*. IEEE Computer Society, 2013, pp. 39–45.

[14] M. Davis, G. Logeman, and D. Loveland, "A Machine Program for Theorem Proving," *Communications of the ACM*, vol. 5, pp. 394–397, July 1962.

[15] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an Efficient SAT Solver," in *Proceedings of the 38th ACM/IEEE Design Automation Conference*, Las Vegas, Nevada, June 2001, pp. 530–535.

[16] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, vol. 35, pp. 677–691, August 1986.

[17] J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan, and D. L. Dill, "Symbolic Model Checking for Sequential Circuit Verification," *IEEE Transactions on Computer-Aided Design*, vol. 13, no. 4, pp. 401–424, April 1994.

[18] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic Model Checking without BDDs," vol. 1579, 1999, pp. 193–207.

[19] F. K. A. Kuehlmann, V. Paruthi and M. K. Ganai, "Robust Boolean Reasoning for Equivalence Checking and Functional Property Verification," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 12, pp. 1377–1394, 2002.

[20] F. Pigorsch, C. Scholl, and S. Disch, "Advanced Unbounded Model Checking Based on AIGs, BDD Sweeping, and Quantifier Scheduling," in *Proceedings of the Conference on Formal Methods in Computer Aided Design (FMCAD). IEEE Computer Society Press, Nov 2006*, pp. 89–96.

[21] A. Smith, A. Veneris, M. F. Ali, and A. Viglas, "Fault Diagnosis and Logic Debugging using Boolean Satisfiability," *IEEE TRANS. ON CAD*, vol. 24, pp. 1606–1621, 2005.

[22] A. Sülflow, G. Fey, C. Braunstein, U. Kühne, and R. Drechsler, "Increasing the Accuracy of SAT-based Debugging," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2009, pp. 1326–1331.

[23] A. Jutman, S. Devadze, and K. Shibin, "Effective Scalable IEEE 1687 Instrumentation Network for Fault Management," *IEEE Design & Test*, vol. 30, no. 5, pp. 26–35, Oct 2013.

[24] "Executive summary. International Technology Roadmap for Semiconductors," [Online]. Available: http://www.itrs.net/Links/2009ITRS/2009Chapters_2009Tables/2009_ExecSum.pdf.

[25] K. Shibin, S. Devadze, and A. Jutman, "Asynchronous Fault Detection in IEEE P1687 Instrument Network," in *IEEE 23rd North Atlantic Test Workshop (NATW)*, May 2014, pp. 73–78.

[26] D. Nikolov, M. Väyrynen, U. Ingelsson, E. Larsson, and V. Singh, "Optimizing Fault Tolerance for Multi-Processor System-on-Chip," in *Design and Test Technology for Dependable Systems-on-chip*, R. Ubar, J. Raik, and H. T. Vierhaus, Eds. IGI Global, 2011, pp. 66–91.