Robustness of TAP-based Scan Networks

Ghani Zadegan, Farrokh; Carlsson, Gunnar; Larsson, Erik

Published in:
[Host publication title missing]

DOI:
10.1109/TEST.2014.7035321

2014

Robustness of TAP-Based Scan Networks

Farrokh Ghani Zadegan
Lund University, Lund, Sweden

Gunnar Carlsson
Ericsson AB, Stockholm, Sweden

Erik Larsson
Lund University, Lund, Sweden

***Abstract*—It is common to embed instruments when developing integrated circuits (ICs). These instruments are accessed at post-silicon validation, debugging, wafer sort, package test, burn-in, printed circuit board bring-up, printed circuit board assembly manufacturing test, power-on self-test, and operator-driven in-field test. At any of these scenarios, it is of interest to access some but not all of the instruments. IEEE 1149.1-2013 and IEEE 1687 propose Test Access Port based (TAP-based) mechanisms to design flexible scan networks such that any combination of instruments can be accessed from outside of the IC. Previous works optimize TAP-based scan networks for one scenario with a known number of accesses. However, at design time, it is difficult to foresee all needed scenarios and the exact number of accesses to instruments. Moreover, the number of accesses might change due to late design changes, addition/exclusion of tests, and changes of constraints. In this paper, we analyze and compare seven IEEE 1687 compatible network design approaches in terms of instrument access time, hardware overhead, and robustness. Given the similarities between IEEE 1149.1-2013 and IEEE 1687, the conclusions are also applicable to IEEE 1149.1-2013 networks.**

***Keywords*—IEEE 1687 (IJTAG), IEEE 1149.1-2013, on-chip instruments, robustness, access time, network design**

I. Introduction

It is increasingly common that integrated circuits (ICs) are equipped with embedded instruments 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 operator-driven in-field test. For each of these cases—referred to as usage *scenarios* hereafter—it is of interest to access some but not all of the instruments [1]. 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 devices and perform repair, (3) in the burn-in process to cause activity in the chip and to detect infant mortality [2], [3], (4) during PCB bring-up [4], (5) during PCB assembly manufacturing test [4], and (6) during power-on self-test and operator-driven 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 [5]. 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 (1) difficult to foresee all needed scenarios, and (2) how many times an instrument will be used 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.

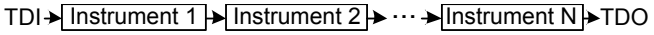


Fig. 1. Chaining instrument shift-registers into a regular scan-chain

The most straightforward approach to create a network to access the instruments in an IC is to chain them in a scan-chain (Fig. 1) which is accessed from the chip boundaries through the JTAG test access port (TAP) [6]. While such approach gives low hardware overhead and fixed time for instrument access, the time overhead for accessing a single instrument (or a few instruments) is high, as many dummy bits are shifted-in/shifted-out [7]. To reduce the time overhead, IEEE 1149.1-2013 [8] and IEEE 1687 (IJTAG) [9] enable flexible access from the chip boundary by using the JTAG TAP. In this work, we use terminology from 1687 and focus on 1687 networks. However, given the similarities between the two standards, the conclusions are also applicable to an 1149.1-2013 context.

Previous works on network design assume one known scenario where the number of accesses is fixed [10]. However, a 1687 network optimized for one scenario, might be not efficient for another one. And, a network optimized for a fixed number of accesses, might not be optimal if the number of accesses changes. Therefore, in this work we study the robustness of seven approaches for designing 1687 networks, and we examine their efficiency, in respect to overall access time (OAT) and hardware overhead. 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. Since there is a need to OAT calculation methods for performing the comparison among the mentioned network design approaches, we present OAT calculation methods for those studied approaches for which such methods are not available from prior work. In the experiments, we compare the considered design approaches for both a number of known scenarios and for cases when additional scenarios are added (unknown at design time).

The rest of this paper is organized as follows: In Section II, the 1687 architecture will be briefly introduced and related previous work will be reviewed. Section III will present the problem formulation, and list the contributions of this work. In Section IV, network design approaches will be presented. Section V will present the performed experiments, and the paper will be concluded in Section VI.

II. Background and Prior Work

In this section we first briefly describe non-reconfigurable and reconfigurable scan networks. We then discuss TAP-based reconfigurable scan networks as described by IEEE 1687 and IEEE 1149.1-2013.

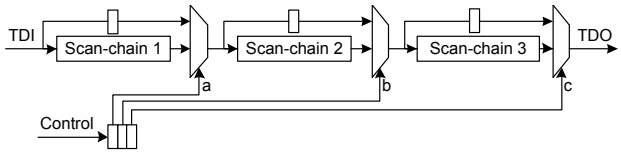


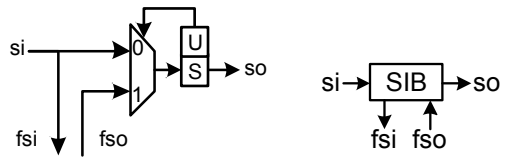
Fig. 2. A daisy-chain implementation with bypass registers

Non-reconfigurable Scan Networks: A regular scan-chain is utilized to access embedded instruments (Fig. 1). The time for accessing any number of instruments in the scan-chain of length l is calculated as $t = (p+1) \cdot l + p \cdot T_a$ [11] where p is the number of times the scan-chain is accessed and T_a is the time it takes to apply a shifted stimuli and capture its response. In [11], T_a is equal to one. The hardware overhead is minimal as all instruments are always on the scan-path. However, if a single instrument is to be accessed, useless dummy bits that contribute to additional time are shifted through for all other instruments.

Reconfigurable Scan Networks: In [11] and [12], it is shown how the use of dynamic reconfiguration for scan-chains lowers the test application time at the cost of extra hardware components. In particular, [11] presents daisy-chaining of scan-chains (Fig. 2) which makes it possible to include only those scan-chains in the scan-path which are needed for current access. To avoid a long combinational path, a bypass register is used for each excluded scan-chain. The control signals for the multiplexers are not provided from the same scan-chain they are reconfiguring, which makes the approaches in [11] and [12] different from TAP-based reconfigurable scan networks where multiplexer control signals are generated by scan elements on the same scan-path. An early example of such a dynamically reconfigurable TAP-based scan network was presented in [13]. As shown in [14], it is however possible to create a daisy-chain-like architecture for 1687, which is discussed in Section IV-D in this work. In [14], the focus is on verification and access vector generation.

IEEE 1687's Hardware: To enable variable-length (flexible) scan-path, 1687 introduces two components:

1) a Segment Insertion Bit (SIB), which is used to include in, or exclude a scan-chain from the active scan-path. Fig. 3 shows a simplified schematic of a possible implementation of a SIB, as well as a symbol which we will use through the rest of this paper. Fig. 3(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. However, a realistic 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).
2) a *ScanMux* control bit, which is a shift-update register that can be placed anywhere on the scan-path to configure one or more scan multiplexers (ScanMux components). Fig. 4 shows a two-bit ScanMux control register used to configure a network of two instruments. In this work, we consider one-bit ScanMux control bits, to control two-input muxes which bypass instrument shift registers in, e.g., daisy-chained architectures.



(a) Simplified schematic (b) Symbol

Fig. 3. Segment Insertion Bit (SIB)

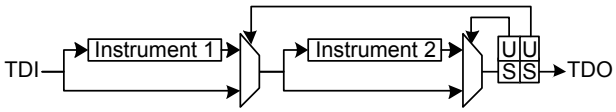


Fig. 4. A network configured by a two-bit ScanMux control register

Both SIBs and ScanMux control bits must be configured to have the correct value every time the scan-path they are on is accessed. Such configuration data is considered as overhead since it is not part of instrument data.

IEEE 1149.1-2013's Hardware: The flexibility in an 1149.1-2013 TDR is achieved by defining segments of that TDR as *selectable*. A selectable segment mux with a one-bit wide control, is similar to the SIB component specified by 1687. Moreover, 1149.1-2013 also allows for controlling a selectable segment mux from another part of the scan-path or from other TDRs. The selectable segments can be nested to create a hierarchical network for accessing instruments, similar to what is achievable by a hierarchical 1687 network.

Although there are differences between 1149.1-2013 and 1687 in implementation details, the corresponding reconfigurable networks described under each of the two standards show the same behavior regarding instrument access time.

The Access: 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. 5 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) 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 CUC [7].

Pattern Description Language (PDL): 1149.1-2013 and 1687 use a similar Pattern Description Language (PDL) for describing the operation of embedded instruments. For example, assuming that the DFT feature in Fig. 5 is a BIST instrument, to operate on this BIST instrument there is a need of PDL commands (read/write) to configure the SIBs such that the

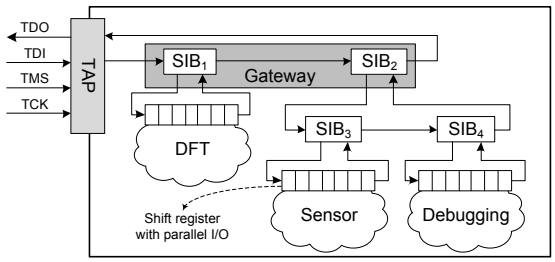


Fig. 5. A 1687 network with three instruments inside a chip

BIST instrument is placed on the scan-path. While the BIST instrument is running, there is no need to access the network for this particular instrument. Hence, the PDL commands can be divided as commands that configure and access the network, such as read/write, and as commands that utilize a given network configuration without requiring any accesses, such as a command used for waiting for a number of clock cycles.

Access time and test time: As some PDL commands are used to configure the network (read/write), the time for accessing instruments is called access time. In access time, the waiting for an instrument (such as a BIST engine) to finish its operation is not included. The waiting time is constant regardless of where in the network the instrument is. The access time, on the other hand, depends on where in the network a particular instrument is placed. The test application time includes both access time and the time each instrument takes to finish its operation (which is captured by wait cycles in PDL). In this paper, we focus on network design; hence, we focus on access time.

Retargeting and Access Schedules: Given the PDL of each instrument, EDA tools generate scan vectors defining which instruments should be active at any time. These scan vectors are applied from the JTAG TAP. The process of generating scan vectors is called *retargeting*. These scan vectors form schedules that determine which instruments should be active at any given time. In general, schedules have to take resource conflicts and power limits into account. A flexible network eases the process of meeting different conflicts and limitations. Interesting to note is that with a flexibility where each instrument individually can be included and excluded, any schedule is possible.

III. Problem Statement and Contributions

In this work we address the problem of designing 1687 networks for multiple scenarios. We use the following notations:

- a set of scenarios, denoted by S, in which for each scenario $s \in S$, an access schedule and a weight W_s are specified. The weight (W_s) is assigned by the designer as a relative metric for the importance of access time reduction for that scenario as compared to the other scenarios, and
- a set I of instruments in which for each instrument $i \in I$ the length of its interface shift-register (L_i) and the number of accesses ($A_{i,s}$) at a scenario s are provided.

The contributions are as follows: First, we compare seven network design approaches in terms of OAT and hardware overhead. The considered approaches are flat network, flat daisy-chained network, a hierarchical network, a hierarchical daisy-chained network, multiple networks (each network is optimized for a given-scenario), multiple daisy-chained network (each network is optimized for a given-scenario), and separate control and data TDRs.

Second, we present OAT calculation for hierarchical daisy-chained networks. It should be noted that the test time calculation formulas presented in [11] cannot be used to calculate OAT for daisy-chaining in 1687, due to (1) the presence of ScanMux control bits on the scan-path, (2) that we consider hierarchical daisy-chaining, as well, and (3) that we consider both sequential and concurrent access whereas in [11] only formulas for the concurrent access are considered. Moreover, the OAT calculation algorithms presented in [7] are for the SIB-based 1687 networks and cannot be used either. Therefore, in Section IV-D, necessary OAT calculation algorithms will be presented for such 1687 daisy-chained architecture.

Third, since it is likely that not all usage scenarios are known at chip design time, we investigate the robustness of the studied approaches toward scenarios not known at chip design time. 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.

In the work, we explore the flexibility features of 1687 networks. To focus on network design, we assume that each instrument can be included in and excluded from the scan-path. Making use of such flexibility, which eases the retargeting process, makes it possible to avoid PDL discussions (discussed in Section II). For the analysis of instrument access, we assume sequential and concurrent schedules. In the sequential schedule, instruments are accessed one at a time and the accesses for each instrument are completed before accessing any other instrument. In the concurrent schedule, accesses to all instruments start at the same time. For both schedules, when there are no more accesses to be performed to a particular instrument, the network is reconfigured to exclude that instrument from the scan-path. In addition, we have performed experiments where we limit the the number of instruments that can be active at the same time.

IV. Design Approaches

In this section, we detail the seven network design approaches, namely the "flat network", "hierarchical network", "multiple networks", "daisy-chained" counterparts for each of these three, as well as "separate control and data TDRs".

For each of the approaches, the network topology is detailed with an example, how to design the network, and the OAT analysis. For the OAT analysis, we assume that the instruments are accessed according to a sequential and a concurrent schedule. For "flat network", "hierarchical network", and "multiple networks" , the algorithms presented in [7] are used. For the daisy-chained counterparts and the "separate control and data TDRs" approach we present OAT calculation algorithms.

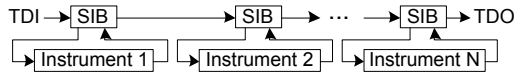


Fig. 6. A flat network with dedicated SIBs for instruments

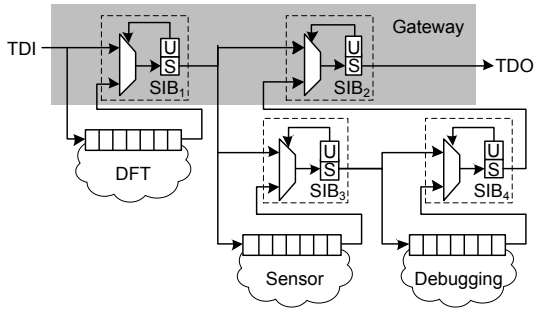


Fig. 7. A more detailed view of the network shown in Fig. 5

A. Flat Network

To construct a flat network, each instrument's shift-register is connected through a SIB (Fig. 6). To access each of the instruments, the corresponding SIB is programmed to include that instrument in the scan-path.

As an example, assume we want to access Instrument 1 (in Fig. 6) A_1 number of times, OAT is calculated as

$$OAT_1 = (T_{CUC} + N) + (T_{CUC} + N + L_1) \cdot (A_1 + 1) \quad (1)$$

where T_{CUC} is the time it takes to perform a CUC (for CUC see Section II) and L_1 is the length of the shift-register for Instrument 1. In (1), the term $(T_{CUC}+N)$ represents the initial configuration of the network (i.e., shifting N bits followed by performing a CUC), the term $(T_{CUC} + N + L_1)$ states that for each access, L_1 bits of instrument data and N bits of SIB programming data should be shifted followed by a CUC, and the term $A_1 + 1$ states that an additional access is required to shift out the final responses.

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.

B. 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. 7 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.

In [10], the hierarchical network design was done with an algorithm inspired by Huffman trees. It was shown that by using hierarchical design approaches it is possible to effectively reduce OAT for both sequential and concurrent schedules.

In this work, we make use of the Huffman tree inspired network construction algorithm. However, instead of using the number of accesses for each instrument as the base for placement of instruments, we assign an attribute, *weighted number of accesses* ($A_{i,w}$), to each instrument. This weighted number of accesses ($A_{i,w}$) captures both the number of accesses for an instrument in each scenario ($A_{i,s}$) and the relative weight of the scenarios (W_s), and is calculated as $A_{i,w} = \sum_{s \in S}(A_{i,s} \times W_s)$. The reason for this assignment is that in the scenario-based design problem, unlike the study in [10], each instrument is associated with more than one number of accesses (one per scenario). The idea is to design a network which performs reasonably well for all the given scenarios, by considering the relative weight assigned to each scenario.

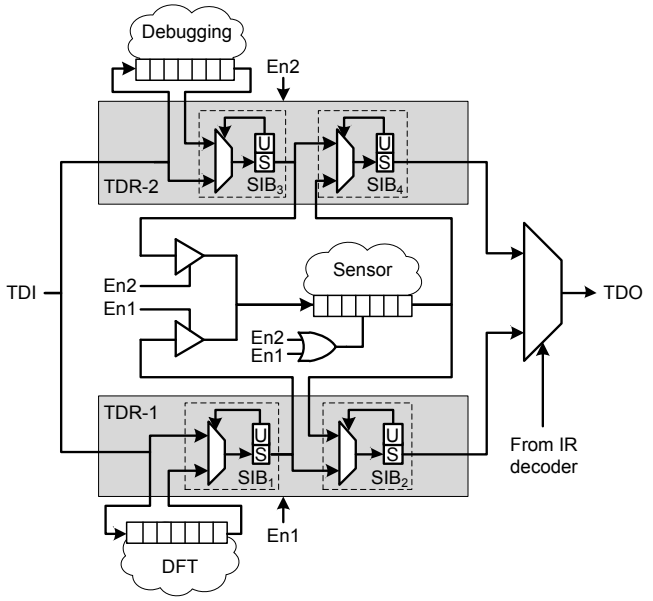


Fig. 8. The sensor instrument is shared by two networks (TDRs)

C. 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. 8. 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. That is, given that no two such TDRs are active at the same time, the same enable signals that are applied to the TDRs, are used to connect the shared instrument shift-registers to the scan-path which belongs to the active TDR. The two networks in Fig. 8 are designed for two scenarios where the Sensor instrument is used in both scenarios, while the DFT and the Debugging instruments are each accessed only in one of the scenarios (hence each accessible only through one of the TDRs). Although the Sensor instrument is shared by both networks, each network dedicates a SIB to it.

For the design of each network for its given scenario, the algorithms in [10] can be used. For each network and its given scenario, access time is calculated by using the algorithms proposed in [7].

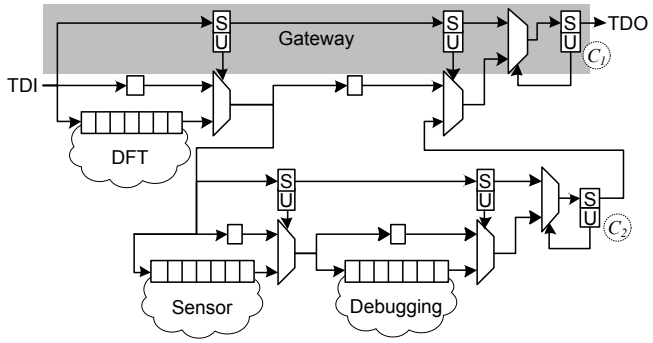


Fig. 9. An example of the use of hierarchy in daisy-chaining

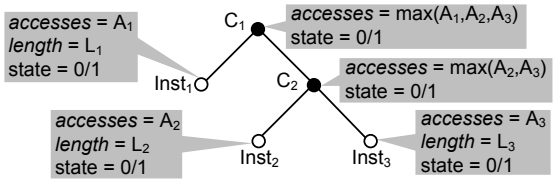


Fig. 10. The tree corresponding to the network in Fig. 9

D. Daisy-chained

The daisy-chaining approach for 1687 is illustrated in Fig. 9. To switch the instrument shift-registers on and off the scan-path, multiplexers are used. These multiplexers are controlled by ScanMux control bits placed on a separate branch of the scan-path. To select between the two branches, other ScanMux control bits denoted by C_1 and C_2 are used which are themselves on the scan-path. To avoid long combinational paths, bypass registers are used in place of an excluded shift-register. In Fig. 9, the sensor and the debugging instruments are placed in a deeper hierarchical level, which allows saving access time by removing their associated bypass registers and ScanMux control bits from the scan-path, when these instruments are not being accessed.

The hierarchical daisy-chained network in Fig. 9 can be seen as a counterpart for the hierarchical SIB-based network in Fig. 7, 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.

In the following, OAT calculation algorithms are presented for the concurrent and sequential schedules. To use these algorithms, we model the given daisy-chained network as a tree in which each internal node corresponds to a C ScanMux control bit (see Fig. 9), and each leaf node corresponds to an instrument. We clarify this with the help of the example tree shown in Fig. 10 which models the network in Fig. 9. Each node in the tree is associated with a *state* attribute which when set to 0, signifies that the node's corresponding instrument/segment is bypassed, and when set to 1 signifies that the corresponding instrument/segment is on the scan-path. Each leaf node has two other attribute/value pairs: $accesses$, marking the number of accesses, and $length$, marking the length of the shift-register for the node's corresponding instrument. Each internal node, has also an $accesses$ attribute whose value is the maximum among the values for $accesses$ found in that node's subtree.

Algorithm 1: For the concurrent schedule

Input: A tree T describing the daisy-chained network
Output: OAT

```
1 while root.accesses > −1 do
2     SL := 0 ;                    // Scan-path length for the current access.
3     root.accesses := TraverseConc(root)
4     OAT := OAT + SL + CUC
```

Below we detail the OAT calculations for concurrent and sequential schedules.

1) Concurrent Schedule: The OAT calculation steps are captured by Algorithm 1 in which each access to the instruments (Lines 2–4) comprises of (1) resetting the variable SL which stores the number of clocks needed to scan data through the scan-path, (2) a call to TraverseConc() (Line 3) which updates SL and returns the number of remaining accesses, and (3) adding the counted number of cycles to OAT (Line 4) which involves shifting SL bits and performing a CUC. The algorithm terminates when there are no more accesses to be performed and the last responses are also shifted out (i.e., $root.accesses \leq -1$).

Function TraverseConc receives a tree $node$ (corresponding to a segment in the daisy-chained network) as input, and by recursively calling itself (1) calculates the number of clocks needed to shift data for the current access (stored in SL), and (2) calculates and updates the remaining number of accesses for each instrument/segment in the $node$'s subtree. If the C ScanMux control bit for the segment represented by $node$ contains a logic zero (Line 3), the multiplexer control path (i.e., the ScanMux control bits path) is selected and should be configured such that the instruments/segments with remaining accesses are placed on the scan-path while the rest are bypassed. This reconfiguration involves (1) shifting one bit per each ScanMux control bit in the segment (Line 2 and Line 4), and (2) updating the $node$'s state to select the instrument path (Line 5). If, however, the instrument path in the current segment is selected (Lines 7–19), for every child node (instrument/segment) on the path which has remaining accesses (Lines 10–17), if the child node corresponds to

- an instrument, the algorithm reduces the remaining number of accesses by one and adds the number of required clocks for shifting the data through the instrument's shift-register to SL (Lines 10–12),
- a segment, the algorithm calls itself recursively (Line 14).

2) Sequential Schedule: The OAT calculation can be performed by traversing the tree and calculating the required number of clocks needed for network configuration and instrument access, at each of the leaf nodes. Such tree traversal is shown in Function TraverseSeq which as input receives an internal tree node and calculates the number of clocks required to sequentially access the instruments in the segment represented by that subtree. When an instrument in a given segment is being accessed, the rest of the instruments/segments in that segment are bypassed which means that their corresponding bypass registers are on the scan-path. This is handled by Line 1 which considers $|node.children| - 1$ bypass registers and one

Function TraverseConc($node$)

```
1  Remaining := −1 ;        // # of remaining accesses in node's subtree
2  SL := SL + 1 ;           // +1 represents the C ScanMux control bit
3  if node.state = 0 then
4      SL := SL + |node.children|
5      node.state := 1
6      Remaining := node.accesses
7  else
8      foreach child ∈ node.children do
9          if child.accesses > −1 then
10             if |child.children| = 0 then
11                 child.accesses = child.accesses − 1
12                 SL = SL + child.length
13             else
14                 child.accesses := TraverseConc (child)
15             if child.accesses < 0 then
16                 node.state := 0
17             Remaining := max{Remaining, child.accesses}
18         else
19             SL := SL + 1 ;          // +1 for the bypass register
20 return Remaining
```

Function TraverseSeq($node$)

```
1  SL := SL + |node.children|
2  foreach child ∈ node.children do
3      OAT := OAT + SL + 1 + CUC
4      if |child.children| > 0 then
5          TraverseSeq (child)
6      else
7          OAT :=
           OAT + (child.length + SL + CUC) · (child.accesses + 1)
8  SL := SL − |node.children|
```

C ScanMux control bit (each internal node corresponds to a C ScanMux control bit) on the path to each of the input node's direct child nodes. For each child node (Line 2), a configuration step is considered (Line 3) to put the node's corresponding instrument/segment on the scan-path and put the other instruments/segments in bypass. If the node is an internal node (Line 4) the function calls itself recursively, otherwise the number of clocks needed to access the instrument corresponding to this leaf node is added to OAT (Line 7). On return, one ScanMux control bit and $|node.children|-1$ bypass registers for $node$'s subtree (i.e., in total $|node.children|$) are reduced from the scan-path length (Line 8). In the OAT calculation (Line 7), it is considered that each instrument i is accessed $A_i + 1$ times ($+1$ for shifting out the last responses), and that for each access $SL + L_i$ bits should shifted followed by performing a CUC. Function TraverseSeq is initially called with the root node as the parameter.

E. Separate Control and Data TDRs

In this approach, there is one TDR for ScanMux control bits and one TDR for instruments (Fig. 11). 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 ScanMux control bits 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. The overhead reduction can be understood by referring to the work in [15] in which it is shown how pipelining of data through bypass registers in a daisy-chained scan-path results in extremely low test time overhead. This is in contrast to the work in [11] which shows that time is wasted in passing the bypass registers. The key difference between [11] and [15] in their assumptions on bypass registers is that in [15] it is assumed that bypass registers are dedicated to DFT, whereas in [11] the bypasses are functional flip-flops converted to scan registers. Since the contents of functional flip-flops change during an execution step (application of stimuli), it is in general not possible to pipeline the test patterns through them. Such wasted time in passing the bypass registers was also present in the daisy-chained approaches discussed in Section IV-D, in spite of assuming dedicated bypass registers. The reason was that the C ScanMux control bits (see Fig.9) were on the same scan-path as the instruments, which required programming them with the correct value for every access. This constraint is, however, not present in the network shown in Fig. 11 since ScanMux control bits are placed on a separate TDR, and therefore, it is possible to reduce the access time overhead by pipelining the instrument data through the bypass registers.

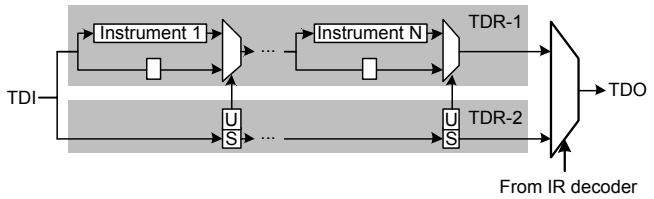


Fig. 11. Using separate TDRs for instruments and ScanMux control bits

The access time calculation for the networks such as the one in Fig. 11 can be done similar to the test application time calculation in [11] for concurrent schedule, and to the test application time calculation in [15] for sequential schedule. However, as mentioned above, the calculations in [11] for the concurrent schedule are done under the assumption that time is wasted while scanning through the bypass registers—which is not the case in the architecture presented here. Moreover, in case of TAP-based networks, we need to take the IR scans (needed to perform the network reconfigurations) into account as well, which makes the access time calculation for TAP-based networks different from the calculations in both [11] and [15]. Therefore, in the following, we present the complete OAT calculations for the separate control and data TDRs approach. We assume that initially TDR-1 is selected (Fig. 11).

Below we detail the OAT calculations for concurrent and sequential schedules.

1) Concurrent Schedule: We start by the concurrent schedule in which accesses to all instruments start at the same time. When there are no more accesses to be performed to a particular instrument, the scan-path is configured such that this instrument is bypassed. OAT (T_{total}) consists of the time it takes to setup the network by configuring the ScanMux control bits (T_{setup}), and the time it takes to perform the required number of accesses (T_{access}):

$$T_{total} = T_{setup} + T_{access}$$

Below we derive the formulas for T_{setup} and T_{access}.

To derive the formula for T_{setup}, assume that there are N instruments, A_i is the number of accesses to be performed on instrument i, where $1 < i < N$, and that the instruments are ordered on the scan-path such that $A_1 > A_2 > \cdots > A_N$. In the concurrent schedule, the network is reconfigured each time the access to an instrument is completed. Hence, there are N reconfigurations needing a setup, and for each reconfiguration, we need to switch to TDR-2, shift in the configuration data into the ScanMux control bits, and switch back to TDR-1. This setup time is captured in the following:

$$T_{setup} = N \cdot (T_{switch} + N + T_{switch}) \quad (2)$$

where the first N represents that the network should be reconfigured N times, T_{switch} represents the time to switch TDRs (taking the TAP FSM from shifting data, to loading an instruction and back to the shifting data state), and the second N represents bits that are shifted in through ScanMux control bits.

We now derive T_{access}. Initially all instruments are included in the scan-path and $A_N + 1$ accesses are performed until the access to instrument N (which has the least number of accesses) is complete and the last responses are shifted out. The time it takes to perform $A_N + 1$ accesses is calculated as follows:

$$T_N = (\sum_{i=1}^{N} L_i + T_{CUC}) \cdot (A_N + 1) - T_{CUC}$$

where L_i is the length of the shift-register for instrument i. The reason that one T_{CUC} is reduced from the calculated time is that this T_{CUC} is included in the time T_{switch} for the next network reconfiguration (i.e., in (2)).

At this point, instrument N should be bypassed, which requires one reconfiguration. Performing the remainder of accesses for instrument $N-1$ takes the following time:

$$T_{N-1} = 1 + (\sum_{i=1}^{N-1} L_i + T_{CUC}) \cdot (A_{N-1} - A_N) - T_{CUC}$$

where 1 represents flushing the pipeline after the last access (i.e., one extra clock is needed to shift the captured responses out completely through the bypass flip-flop for instrument N). In the same manner, we get the following time for performing the remainder of accesses for instrument 1:

$$T_1 = (N-1) + (L_1 + T_{CUC}) \cdot (A_1 - A_2) - T_{CUC}$$

where $(N-1)$ represents flushing the pipeline after the last access through the bypass flip-flops for instruments 2 to N. Finally, we can write T_{access} as:

$$T_{access} = \sum_{j=1}^{N} T_j$$

where T_j is (by assuming $A_{N+1} = -1$):

$$T_j = (N-j) + (\sum_{i=1}^{j} L_i + T_{CUC}) \cdot (A_j - A_{j+1}) - T_{CUC} \quad (3)$$

The above OAT calculations are performed under the assumption $A_1 > A_2 > \cdots > A_N$. But if there are instruments with the same number of accesses, since accessing them starts and ends at the same time, they share the same network reconfiguration step, and also the flushing of the pipeline will be performed once for all of them. Moreover, if instruments do not appear on the scan path in the assumed order, it can happen that two instruments are active with some bypass registers in between them on the scan-path. In this case, instrument data cannot (in general) be pipelined through those bypass registers. The reason is that the captured responses from the instruments at the beginning of the path might break the scan vectors which are pipelined for the instruments further down the scan-path. For these cases, to take the time wasted in the bypass registers—that appear between active (i.e., not bypassed) instruments on the scan-path—into account, (3) should be modified as:

$$T_j = R_b + R_e + (\sum_{i=1}^{j} L_i + R_m + T_{CUC}) \cdot (A_j - A_{j+1}) - T_{CUC} \quad (4)$$

where R_b represents the number of bypass registers on the scan-path preceding the first currently active instrument, R_e represents the number of bypass registers on the scan-path after the last currently active instrument, and R_m represents the number of bypass registers that appear between the currently active instruments. Equation (4) shows that the bypass registers represented by R_m contribute to the access time (as overhead) for every access, whereas those represented by R_b and R_e only increase the time once per reconfiguration (to flush the pipelined data). For the experiments we have performed (Section V), we implemented an algorithm based on the formulas presented in this section, that calculate R_b, R_e, and R_m values based on the placement of the currently active instruments on the scan-path, and therefore takes into account the time wasted passing through the bypass registers.

2) Sequential Schedule: In the sequential schedule, instruments are accessed one at a time and the accesses for each instrument is completed before accessing any other instrument. The order of accesses has no impact on OAT (T_{total}), which can be written again as $T_{total} = T_{setup} + T_{access}$. Regardless of the number of accesses, T_{setup} can be written as:

$$T_{setup} = N \cdot (T_{switch} + N + T_{switch})$$

Assuming that T_i is the time it takes to complete A_i accesses for instrument i, we have:

$$T_{access} = \sum_{i=1}^{N} T_i$$

where

$$T_i = N - 1 + (L_i + T_{CUC}) \cdot (A_i + 1) \quad (5)$$

In (5), $N-1$ represents the bypass flip-flops that should be flushed after the last access to instrument i.

V. Experimental Results

In this section, we present results for the comparison of the presented design approaches, in terms of OAT and hardware overhead. To perform the comparison, a set of on-chip instruments representing those in an advanced system-on-a-chip are needed. We created a benchmark based on the UltraSPARC T2 processor [16], which contains 48 MBIST engines, eight LBIST engines, and about 70 high-speed serial lanes. We assumed that

TABLE I. THE INSTRUMENT TYPES, THE SCENARIOS, AND THE NUMBER OF ACCESSES FOR EACH INSTRUMENT TYPE PER SCENARIO

Instruments		**Scenarios and their assigned weights (in parentheses)**							
Type	**Count**	**S1 (1)** Sequential	**S2 (100)** Concurrent	**S3 (1)** Concurrent	**S4 (1)** Sequential	**S5 (100)** Concurrent	**S6 (1)** Sequential	**S7 (10)** Concurrent	**S8 (10)** Concurrent
1	20	100	10	10	10	10	10	10	10000
2	20	10000	0	0	0	0	0	0	10000
3	10	100	10	10	10	10	10	10	10000
4	40	100000	10	10	100000	100	10	10	10000
5	10	10	100	10	10	10	100000	100000	10000

a BIST technology such as Intel's IBIST [17] is applied to the high-speed serial links, and we ended-up with a list containing more than a hundred on-chip instruments having on average a shift-register length of about 20 flip-flops. Given the above observations, we constructed a benchmark with 100 instruments each having an interface shift-register length of 20 flip-flops.

Moreover, we assumed eight access scenarios for which Table I presents the considered set of instruments and how they are accessed. In Table I, column 1 lists that there are five types of instruments, column 2 lists how many of each type of instrument are considered, and columns 3–10 list the number of accesses for each instrument type for each scenario. In Table I, under the headers for columns 3–10, the access schedules as well as the weights assigned to scenarios (within parentheses), are presented.

In the following, we describe two sets of experiments we have performed to examine (1) the robustness of the studied approaches against new scenarios not known at design time (Section V-A), and (2) the robustness of two of the selected approaches against general access schedules (Section V-B).

A. Robustness Towards New Scenarios

1) Aim: In this set of experiments, we consider three cases: (A) when the networks corresponding to each of the approaches in Section IV are designed and optimized for scenario S1 (i.e., by using the number of accesses and the access schedule for S1), but are later (when the chip is manufactured) used to access the instruments according to scenarios S2–S8 as well, (B) when the networks are designed and optimized for scenarios S1–S5, but are later used to access the instruments according to scenarios S6–S8, and (C) when all scenarios are known at the design time and therefore the networks are designed and optimized for all scenarios.

For the experiments, we calculated OAT for each of the scenarios listed in Table I, by the use of the algorithms proposed in Section IV-D (for the "Daisy-chained" approach), algorithms based on the formulas presented in Section IV-E (for the "Separate control and data TDRs" approach), and the algorithms presented in [7] (for the rest of the approaches). In the access time calculations, CUC is considered to take four test clock cycles, and T_{switch} for the case of "Separate control and data TDRs" is assumed to take 19 test clock cycles.

As baseline approach for the comparison, we use the non-reconfigurable scan network (Section II), and refer to it as "Regular scan-chain" in the presentation of the experimental results. OAT is calculated using the formula presented for the non-reconfigurable scan network (Section II), considering $l = \sum_{i=1}^{N} L_i$ and assuming $T_a = 4$ (TAP overhead, i.e., CUC). When accessing instruments according to the concurrent schedule, p is considered as $\max_{1 \leq i \leq N}\{A_i\}$, where A_i is the number of accesses for instrument i. For sequential access p is considered as $\sum_{i=1}^{N} A_i$.

As the metric for comparison of the networks (i.e., comparing the corresponding considered design approaches), the weighted sum of OATs for each scenario was calculated for each of the approaches, as $Sum = \sum_{s \in S}(OAT_s \times W_s)$. The weights are those assigned to each scenario (Table I).

2) Calculation of Hardware Overhead: Regarding the hardware overhead, the total number of flip-flops and multiplexers used to construct the networks are reported. The aim is not to report an exact component/gate count, but rather a relative metric to enable us to compare the networks. To this end, only the shift and update flip-flops and scan multiplexers are considered. As examples of how the hardware overhead is calculated in the experiments, the hardware overhead calculation for the network in Fig. 7 and its daisy-chained counterpart in Fig. 9, is presented here:

- The network in Fig. 7 is considered to have eight flip-flops (four shift and four update flip-flops), and four muxes.
- The network in Fig. 9 is considered to have 16 flip-flops (six shift, six update, and four bypass flip-flops), and six muxes.

For the "Multiple networks" approach, the reported numbers are the sum over all networks designed for each of the scenarios. Moreover, the number of tristate buffers used to share instruments among these networks are also reported. As mentioned in Section IV-C, if an instrument is shared among multiple networks, each network should separately dedicate hardware resources (e.g., a SIB) to that instrument, which effectively increases the hardware overhead for this approach in comparison with other approaches. As an example, the hardware overhead for the network in Fig. 8 is calculated as eight flip-flops, four muxes, and two tristate buffers. In Fig. 8, if instead of one instrument, all instruments were shared by both networks, the hardware overhead would increase to 12 flip-flops, six muxes and six tristate buffers, as each network would have to dedicate a SIB to each of the three shared instruments.

3) Presentation and Discussion of the Results: Tables II–IV present the results of the experiments in details. The tables have the same layout where the first column lists the examined approaches. Columns 2–4 present the used hardware components for the construction of the network corresponding to each of the approaches. The hardware components are reported as the number of flip-flops (represented by column "F.F."), two-input multiplexers ("Mux"), and tristate buffers ("Buf.") used to share an instrument shift-register among multiple networks (Fig. 8). Columns 5–12 list for each scenario the product of the OAT and the assigned weight for that scenario. Column "Sum" presents the sum of values in columns 5–12, to be

TABLE II. EXPERIMENTAL RESULTS FOR CASE A WHEN THE NETWORKS ARE OPTIMIZED ONLY FOR S1, BUT USED FOR S1–S8

Approach	Hardware overhead*			Weighted OAT ($OAT_s \times W_s$)								Sum
	F.F.	Mux	Buf.	S1	S2	S3	S4	S5	S6	S7	S8	
Regular scan-chain	0	0	0	8,423,014,404	20,240,400	22,044	8,016,803,604	20,240,400	2,005,404,804	2,004,020,040	200,420,040	20,690,165,736
Flat network	200	100	0	521,196,904	4,620,800	18,848	496,059,624	10,020,800	124,096,824	304,158,080	210,422,080	1,670,593,960
Flat daisy-chain	302	101	0	521,207,300	4,463,500	17,980	496,070,020	9,593,500	124,107,220	295,151,350	200,521,100	1,651,131,970
Hierarchical network	304	152	0	147,054,246	4,143,800	19,388	139,024,576	9,687,400	57,032,776	245,169,380	215,625,500	817,757,066
Hierarchical daisy-chain	562	205	0	147,056,424	3,995,000	18,710	139,026,754	9,262,300	57,034,954	236,163,500	205,726,870	798,284,512
Multiple networks	-	-	-	-	-	-	-	-	-	-	-	-
Multiple daisy-chains	-	-	-	-	-	-	-	-	-	-	-	-
Separate control and data TDRs	300	100	0	100,900,100	4,387,000	17,988	96,030,160	9,534,800	24,037,360	286,152,700	200,421,380	721,481,488

*F.F., Mux, and Buf. represent the total number of flip-flops, multiplexers, and tristate buffers, respectively

†This approach is not applicable when designing for one scenario.

TABLE III. EXPERIMENTAL RESULTS FOR CASE B WHEN THE NETWORKS ARE OPTIMIZED FOR S1–S5, BUT USED FOR S1–S8

Approach	Hardware overhead*			Weighted OAT ($OAT_s \times W_s$)								Sum
	F.F.	Mux	Buf.	S1	S2	S3	S4	S5	S6	S7	S8	
Regular scan-chain	0	0	0	8,423,014,404	20,240,400	22,044	8,016,803,604	20,240,400	2,005,404,804	2,004,020,040	200,420,040	20,690,165,736
Flat network	200	100	0	521,196,904	4,620,800	18,848	496,059,624	10,020,800	124,096,824	304,158,080	210,422,080	1,670,593,960
Flat daisy-chain	302	101	0	521,207,300	4,463,500	17,980	496,070,020	9,593,500	124,107,220	295,151,350	200,521,100	1,651,131,970
Hierarchical network	286	143	0	147,621,527	4,014,100	19,190	139,021,159	9,672,800	39,431,119	233,168,410	214,724,180	787,672,485
Hierarchical daisy-chain	517	187	0	147,623,483	3,857,500	18,435	139,023,115	9,243,200	39,433,075	224,161,750	204,824,460	768,185,018
Multiple networks	1172	586	400	147,054,246	3,798,200	18,620	138,621,398	9,468,200	35,082,177	215,164,820	210,522,950	759,730,611
Multiple daisy-chains	1940	677	400	147,056,424	3,632,000	17,752	138,623,769	9,035,000	35,090,463	206,157,200	200,622,840	740,235,448
Separate control and data TDRs	300	100	0	100,900,100	4,387,000	17,988	96,030,160	9,534,800	24,037,360	286,152,700	200,421,380	721,481,488

*F.F., Mux, and Buf. represent the total number of flip-flops, multiplexers, and tristate buffers, respectively

TABLE IV. EXPERIMENTAL RESULTS FOR CASE C WHEN THE NETWORKS ARE OPTIMIZED FOR S1–S8, AND USED FOR S1–S8

Approach	Hardware overhead*			Weighted OAT ($OAT_s \times W_s$)								Sum
	F.F.	Mux	Buf.	S1	S2	S3	S4	S5	S6	S7	S8	
Regular scan-chain	0	0	0	8,423,014,404	20,240,400	22,044	8,016,803,604	20,240,400	2,005,404,804	2,004,020,040	200,420,040	20,690,165,736
Flat network	200	100	0	521,196,904	4,620,800	18,848	496,059,624	10,020,800	124,096,824	304,158,080	210,422,080	1,670,593,960
Flat daisy-chain	302	101	0	521,207,300	4,463,500	17,980	496,070,020	9,593,500	124,107,220	295,151,350	200,521,100	1,651,131,970
Hierarchical network	306	153	0	155,788,222	3,946,600	19,342	147,619,236	9,852,300	33,030,696	224,170,660	215,724,470	790,151,526
Hierarchical daisy-chain	567	207	0	155,790,032	3,785,900	18,546	147,621,046	9,423,000	33,032,506	215,163,590	205,824,850	770,659,470
Multiple networks	1868	934	680	147,054,246	3,798,200	18,620	138,621,398	9,468,200	30,833,900	215,164,820	210,422,080	755,381,464
Multiple daisy-chains	3086	1076	680	147,056,424	3,632,000	17,752	138,623,769	9,035,000	30,836,216	206,157,200	200,521,100	735,879,461
Separate control and data TDRs	300	100	0	100,900,100	4,387,000	17,988	96,030,160	9,534,800	24,037,360	286,152,700	200,421,380	721,481,488

*F.F., Mux, and Buf. represent the total number of flip-flops, multiplexers, and tristate buffers, respectively

used as the comparison metric. For Case A, since the networks were designed and optimized for one scenario, the "Multiple Networks" and "Multiple daisy-chains" approaches are not applicable.

From the "Sum" columns of Tables II–IV it can be seen that Sum for "Regular scan-chain", "Flat network", and "Flat daisy-chain" is at least two times larger than Sum for the rest of the approaches. "Separate control and data TDRs" shows the best Sum among all which makes it specifically interesting given that it has a fixed architecture (which does not change with the usage scenario). Therefore, it can be expected that by adding even more scenarios, more or less the same behavior (in comparison with other approaches in terms of OAT) can be expected from this approach. "Multiple networks" and "Multiple daisy-chains" also show a low Sum as well as good results for the individual scenarios, but at the cost of a very high hardware overhead. In this regard, showing a low Sum for Case C (for which there exists an optimized network for each scenario in "Multiple networks" and "Multiple daisy-chains") is no surprise, but for Case B where S6, S7, and S8 were not initially known at design time, we still observe good results from these two approaches. "Hierarchical network" also shows good results given its relatively low hardware overhead. However, it might be that the low Sum we observe for this approach cannot be expected for additional scenarios, since we already see an increase in Sum when going from Case B to Case C (in which all scenarios are considered at design time) for this approach.

Table V presents the robustness metric (see Section III) for each of the approaches under each of the considered cases. It can be seen that "Separate control and data TDRs" shows the smallest value and therefore is the most robust among the studied approaches. Since this approach has a fixed architecture, its robustness metric is the same for all cases (a similar argument applies to "Flat network" and "Flat daisy-chain", as well). Moreover, since the placement of instruments on the scan-path for the experiments was chosen randomly, and therefore the benefits of pipelining instrument data is only partially exploited (see Section IV-E1), it can be expected that even for future scenarios a similar degree of robustness will be observed. It is notable that under Case A, "Hierarchical network" and "Hierarchical daisy-chain" (which were optimized only for one scenario) show a relatively good degree of robustness. This can be explained by the hierarchical design of these networks: For example, for "Hierarchical network", the average number of SIBs on the scan-path is less than 16 SIBs which should be compared to the average of 100 SIBs for the "Flat network".

TABLE V. ROBUSTNESS OF DIFFERENT APPROACHES

Approach	Robustness metric		
	Case A	Case B	Case C
Flat network	2.42	2.42	2.42
Flat daisy-chain	2.44	2.44	2.44
Hierarchical network	0.57	0.35	0.34
Hierarchical daisy-chain	0.59	0.37	0.36
Multiple networks	-	0.33	0.30
Multiple daisy-chains	-	0.35	0.32
Separate control and data TDRs	0.13	0.13	0.13

To sum up, the "Separate control and data TDRs" and "Hierarchical network" approaches, show good results, in terms of OAT, hardware overhead, and robustness.

B. Robustness Against General Schedules

For the scenarios listed in Table I, we have considered sequential and concurrent access schedules. However, to evaluate the robustness of the networks against general schedules, we

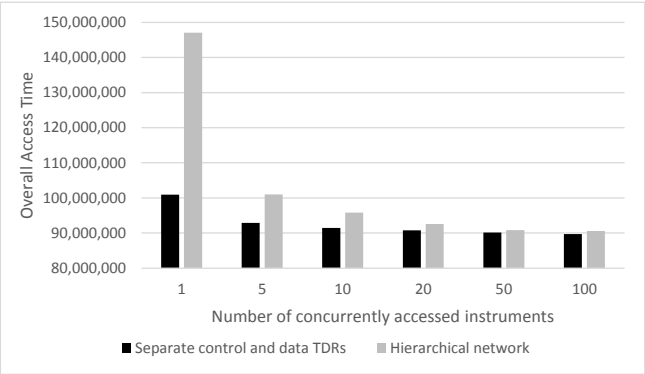


Fig. 12. Change in OAT as concurrency increases

performed an experiment in which the instruments in scenario S1 are accessed according to partially concurrent schedules in groups of 5, 10, 20, 50, and 100 concurrent instruments. As for the approaches, we took "Hierarchical network" (this time only optimized for S1) and "Separate control and data TDRs", which had performed reasonably well in the previous experiment (regarding OAT, hardware overhead, and robustness).

The result of OAT calculation is presented as the chart in Fig. 12. For "Separate control and data TDRs", we assumed a random order of the instruments on the scan-path so that the pipelining of data through bypass registers cannot be fully exploited (see Section IV-E1). A general observation is that the "Separate control and data TDRs" approach still performs better than the "Hierarchical network" approach in all cases. It can also be seen that as concurrency increases, OAT decreases, which is mainly because overhead of CUC (and SIB programming in case of "Hierarchical network") is amortized over multiple concurrent accesses [7]. Moreover, the OAT reduction for "Separate control and data TDRs" from sequential access (denoted by "1" in the chart) to five concurrent instruments is not as significant as the corresponding drop in OAT for "Hierarchical network". The reason is that the positive effect of sharing the CUC overhead is partially countered by a significant increase in overhead from bypass registers (i.e., passing data through those bypass registers that appear between active instruments on the scan-path, preventing pipelining of instrument data through them). This overhead from bypass registers does not decrease much as concurrency increases and therefore, for the case of fully concurrent access (denoted by "100" in the chart), OAT for both approaches become almost similar.

As for robustness, since applying any change in the access schedule (e.g., a change in concurrency, as is the case here) can be regarded as creating a new usage scenario, the same metric defined in Section III can be calculated and used for this experiment, as well. To do so, we calculated the ratio of OAT to instrument data for all the cases presented in Fig. 12 and got 0.05 for "Separate control and data TDRs" and 0.24 for "Hierarchical network'. The lower standard deviation for the former confirms that the "Separate control and data TDRs" approach is robust against changes in the concurrency in the schedule, as well.

VI. Conclusion and Future Work

Integrated circuits contain a high number of embedded instruments which are accessed at several points during the life cycle, from wafer sort to in-field test. At each point, referred to as a *scenario* in this work, some instruments but typically not all are accessed. Different from previous works that only assumed one scenario with a fixed number of accesses, we studied and compared seven 1687 network design approaches in terms of overall access time, hardware overhead, and robustness. We compared the networks against several known scenarios as well as when scenarios not known at design time were added. The results indicate that the approach using two separate JTAG test data registers for the instrument access network, one to configure the access network and one to access the instruments, results in best overall access time and robustness at low hardware cost. While we made use of sequential, concurrent, and general schedules in the experiments, we will as a future work further study the impact of general schedules. We will also explore architectures that make use of broadcasting, which is suitable when instruments of the same type are accessed at the same time. Finally, we will perform a more detailed hardware overhead comparison that also takes routing into account.

References

[1] 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.

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

[3] 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.

[4] 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.

[5] 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.

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

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

[8] "IEEE Standard for Test Access Port and Boundary-Scan Architecture," *IEEE Std 1149.1-2013 (Revision of IEEE Std 1149.1-2001)*, 2013.

[9] IJTAG, "IJTAG - IEEE P1687," Mar. 2012. [Online]. Available: http://grouper.ieee.org/groups/1687

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

[11] J. Aerts and E. J. Marinissen, "Scan Chain Design for Test Time Reduction in Core-Based ICs," in *Proc. IEEE Int'l Test Conf. (ITC)*, 1998, pp. 448–457.

[12] S. Samaranayake et al., "Dynamic Scan: Driving Down the Cost of Test," *Computer*, vol. 35, no. 10, pp. 63–68, 2002.

[13] L. D. Whetsel, "Hierarchical Scan Selection," Oct. 3 1989, US Patent 4,872,169.

[14] 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.

[15] T. Waayers, R. Morren, and R. Grandi, "Definition of a Robust Modular SOC Test Architecture; Resurrection of the Single TAM Daisy-Chain," in *Proc. IEEE Int'l Test Conf. (ITC)*, 2005.

[16] Sun Microsystems, Inc., "UltraSPARC T2 Supplement to the UltraSPARC Architecture 2007," Draft D1.4.3, 19 Sep. 2007.

[17] J. Nejedlo, "IBIST (Interconnect Built-in-Self-Test) Architecture and Methodology for PCI Express," in *Proc. IEEE Int'l Test Conf. (ITC)*, vol. 2, 2003, pp. 114–122.