A Self-Reconfiguring IEEE 1687 Network for Fault Monitoring

Ghani Zadegan, Farrokh; Nikolov, Dimitar; Larsson, Erik

Published in:
21th IEEE European Test Symposium (ETS), 2016

DOI:
10.1109/ETS.2016.7519288

2016

Document Version:
Peer reviewed version (aka post-print)

A Self-Reconfiguring IEEE 1687 Network for Fault Monitoring

Farrokh Ghani Zadegan
Lund University, Lund, Sweden

Dimitar Nikolov
Lund University, Lund, Sweden

Erik Larsson
Lund University, Lund, Sweden

***Abstract*—Efficient handling of faults during operation is highly dependent on the interval (latency) from the time embedded instruments detect errors to the time when the fault manager localizes the errors. In this paper, we propose a self-reconfiguring IEEE 1687 network in which all instruments that have detected errors are automatically included in the scan path. To enable self-reconfiguration, we propose a modified segment insertion bit (SIB) compliant to IEEE 1687. We provide time analyses on error detection and fault localization for single and multiple faults, and we suggest how the self-reconfiguring IEEE 1687 network should be designed such that time for error detection and fault localization is kept low and deterministic. For validation, we implemented and performed post-layout simulations for one self-reconfiguring network. We show that compared to previous schemes, our proposed network significantly reduces the fault localization time.**

Keywords*—IEEE 1687, fault monitoring, self-reconfiguration, time analysis

I. Introduction

While the semiconductor technology development enables integrated circuits (ICs) with increasing transistor count and decreasing features sizes, it has become crucial to address reliability to handle errors that occur when the IC is in operation [1]. One way to address reliability issues is to embed on-chip instruments capable of detecting errors [2]. Such instruments can be connected to a *fault manager* that makes decisions based on the collected error statuses. The fault manager can be implemented in an on-chip or off-chip processor. To avoid that the *fault monitoring network*, which connects the monitoring instruments and the fault manager, becomes the bottleneck, the network must be designed such that the interval (latency) from when instruments detect errors to when the fault manager gets aware of the errors is low and deterministic. It is also important that the fault manager localizes errors in a short and deterministic time (fault *localization* time) in order to launch suitable recovery actions.

A fault monitoring network can be either *stand-alone*, or part of an existing *functional* infrastructure such as network-on-chip or system bus. The advantage of using the functional network is that no additional hardware cost is needed. One drawback is that adding traffic to transport fault monitoring information may impact the performance of the system. It is difficult at design time to know the amount of traffic information that is to be generated from errors as the traffic depends on when the errors occur. Another drawback is that the predictability of the system is reduced, as the traffic on the functional network might also affect the latency of the fault monitoring information. To be on the safe side, the network might be over-designed to ensure that performance is kept high. This is however costly. The advantage with a stand-alone network is twofold: it does not impact the performance of the system, and simplifies achieving a deterministic fault localization time. The downside of using a stand-alone network is adding extra hardware cost, if it is added only for the purpose of fault management. However, most ICs have stand-alone networks, such as IEEE 1149.1 (JTAG) [3] or IEEE 1687 [4], to enable test, diagnosis, configuration, etc., which makes it attractive to reuse such network for fault monitoring and error handling during operation.

There have been a number of works on networks for transporting monitoring data (for transient faults, timing errors, power estimation, etc.) using a dedicated infrastructure [5]–[9]. The works in [7]–[9] stand out as they rely on reusing the existing IEEE 1687 network for monitoring purposes.

In this paper, for the hardware platform, we assume an IC equipped with embedded monitoring instruments that can detect errors and produce error codes accordingly. Additionally, we assume these on-chip monitoring instruments to be interfaced to an IEEE 1687 network. We propose a scheme where the IEEE 1687 network is self-reconfigured (while maintaining standard compliance) to automatically include the instrument registers containing error codes in its scan-path. The proposed scheme enables very fast error detection, and achieves significantly faster fault localization compared to similar IEEE 1687-based fault monitoring approaches.

The main contributions of the paper are as follows:

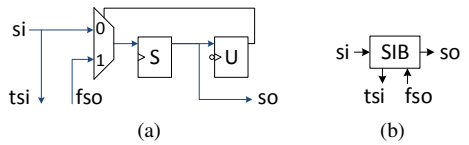
- a *modified* Segment Insertion Bit (SIB) which can be configured not only in the standard way but also by an embedded fault monitoring instrument
- time analysis on the time needed to detect and localize an arbitrary number of errors
- design guidance for the self-reconfiguring IEEE 1687 network such that the error detection and localization time for a single fault is minimized

To validate our proposed self-reconfiguring networks, we implemented one such network and performed post-layout simulations in 65nm technology. To evaluate the efficiency in terms of time we compare the proposed self-reconfiguring scheme against previous IEEE 1687-based fault management schemes.

II. Background and Prior Work

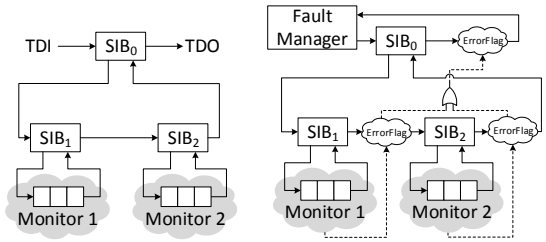
In this section, basics of IEEE 1687 are detailed, and prior work on fault management using IEEE 1687 as the fault monitoring infrastructure is discussed.

The key feature of IEEE 1687 networks is the reconfigurability, i.e., the possibility to switch segments of the network on and off the scan-path. One way to add such reconfigurability is by using SIBs. Fig. 1(a) shows a simplified schematic of a (possible implementation of a) SIB, in which *select* and control signals (namely, *capture*, *shift*, and *update*) are not shown. The SIB has a shift (S) flip-flop, an update (U) flip-flop, and a two-input scan multiplexer. SIBs in the network are programmed by shifting a bit into their S flip-flop and latching that bit into the parallel U flip-flop. If the latched bit is a "0", the SIB is **closed** and the scan-path is from the *si* (scan-in) terminal, to the *so*



(a) (b)

Fig. 1. SIB: (a) simplified schematic, and (b) symbol



(a) (b)

Fig. 2. (a) Example of IEEE 1687 network, and (b) a simplified representation of the basic idea in [7]

(scan-out) terminal via the S flip-flop, bypassing the segment between the *tsi* (to scan-in) and *fso* (from scan-out) terminals. If the latched bit is a "1", the SIB is **opened** and the scan-path includes the segment connected between *tsi* and *fso* terminals of the SIB—referred to as the hierarchical port of a SIB in this paper. The symbol shown in Fig. 1(b) will be used in the rest of this paper to represent a SIB.

Using SIBs makes it possible to design hierarchical IEEE 1687 networks. Fig. 2(a) shows a hierarchical IEEE 1687 network consisting of three SIBs and two 3-bit instrument shift-registers, named Monitor 1 and Monitor 2. To access the network via the JTAG TAP, the top (highest) hierarchical level of the network (SIB_0 in this example) is connected as a custom test data register (TDR) between the test-data-in (TDI) and test-data-out (TDO) terminals of the JTAG TAP. In the network shown in Fig. 2(a), initially, only SIB_0 is on the *active* scan-path. To access, e.g., Monitor 1, SIB_0 should be programmed to be opened, which requires going through Shift and Update states of the JTAG TAP finite state machine (FSM)—referred to as a capture-shift-update (CSU) cycle. Next, SIB_1 should be opened while keeping SIB_0 opened and SIB_2 closed—which requires performing another CSU. At this point, the register for Monitor 1 is placed on the scan-path and can be read from or written to. It can be seen from this example that accessing instruments in an IEEE 1687 network produces two types of overhead: (1) the clock cycles needed to operate the JTAG FSM and (2) the clock cycles needed to shift SIB programming data. It has been shown that the JTAG TAP FSM overhead is negligible compared to the SIB programming overhead [10], and that using a hierarchical design (such as those in Fig. 2) can reduce the SIB programming overhead significantly [11].

Hierarchical IEEE 1687 networks have been used in fault management schemes to connect instruments to a fault manager [7]–[9]. In [7], methods for optimized design and calculation of error localization time are presented for the proposed fault management scheme. The work in [8] extends [7] by elaborating on how the fault manager can react faster to new faults while the instrument access network is in use for other purposes and how multiple faults can be addressed, but presents no time analysis method or experimental results for such cases. In [9], a simulation-based platform for experimenting with fault injection and fault management is elaborated, but no time analysis or network optimization method is presented.

Along with the IEEE 1687 network, [7]–[9] use a fully combinational error propagation network which propagates error flags to the highest hierarchical level of the IEEE 1687 network. A simplified representation of the hierarchical networks used in [7] is shown in Fig. 2(b) where the error propagation network is marked by the dashed lines. The advantage is that by reading the ErrorFlag register in the highest level the fault manager gets informed of any error in the system without checking each and every instrument. To guide fault localization, [7]–[9] added ErrorFlags at every level, resulting in dramatic increase in fault localization time. Also, fault localization in [7]–[9] involves a number of CSUs to open hierarchical levels, each CSU performed over a scan-path longer than the scan-path for the previous CSU, increasing the fault localization time.

To address the fault localization time, we consider a fault management scheme similar to those in [7]–[9] and propose self-reconfiguration. We show that by adding self-reconfiguration it is possible to reduce the fault localization time significantly while keeping conformity to the IEEE 1687 rules.

In this work, we use the terms fault and error interchangeably even though in practice these two concepts are different, i.e., an error is a manifestation of a fault.

III. Self-Reconfiguring Network

In this section, we describe the hardware structure of the self-reconfiguring networks, as well as how to detect and localize errors in the proposed structure.

The basic idea in self-reconfiguration is that when a fault is detected by a fault monitor, the corresponding error code register is automatically included in the active scan-path so that its contents can be readily shifted out and analyzed. Such scheme, improves the speed of fault localization via (1) avoiding to open layers of hierarchy one layer at a time, and (2) using only one single-bit ErrorFlag register instead of placing multiple such registers at each hierarchical level.

A. Hardware Structure

In this work, we assume a hierarchical IEEE 1687 network interfacing all embedded instruments (test, debug, fault monitors, etc.) in a system to a Fault Manager, which has the purpose of detecting and localizing errors that may occur in different components of the system over time, such that it can initiate necessary fault handling actions. The novelty of this work relies on the fact that part of the hierarchical IEEE 1687 has the feature of self-reconfiguration.

Fig. 3(b) shows an example of a self-reconfiguring network. Among all the instruments, we assume that there is a set of fault monitoring instruments. In the top level of the hierarchical network, the fault monitoring instruments are connected through a dedicated SIB, denoted with SIB_0, while all the other instruments (test, debug, etc.) are connected through another SIB, denoted with SIB_{ins}. The top level also includes a one bit shift-register (ErrorFlag) to indicate if any errors are detected by any of the fault monitoring instruments.

We assume that a fault monitoring instrument has a *fault flag* output terminal that is set to logic "1" in case a fault is detected. The *fault flag* stays active until it is acknowledged via a *clear flag* input terminal. The fault flag signal will be used as an input to reconfigure the network, such that an access to the fault monitoring instrument is enabled. Furthermore, the fault flag signal is propagated across the hierarchical levels and is finally captured by the ErrorFlag register in the top level of the hierarchical network.

Additionally, we assume that a fault monitoring instrument produces an error-code which is parallel-loaded during the capture phase into an error-code/mask shift-register (EMR) interfacing the instrument to the IEEE 1687 network. An EMR is assumed to have capture and update features (similar to standard

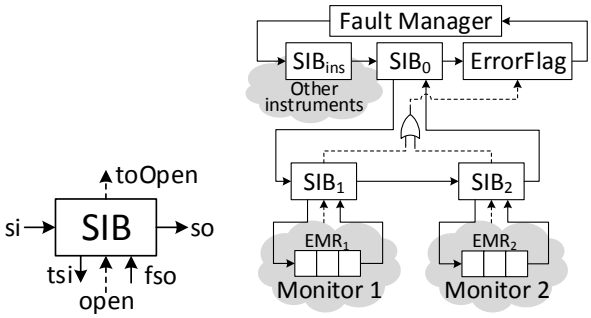


Fig. 3. (a) Symbol for the *modified* SIB, and (b) An example self-reconfiguring network (the dashed line represents the fault propagation network)

JTAG TDRs) and it contains an error-code field (written by the fault monitor) and a mask field (written by the Fault Manager). Error masking is used to stop a permanent fault from constantly raising the fault flag. To be compliant with the IEEE 1687 standard, error masking should be enabled by default at reset to disable self-reconfiguration of the network. When the EMR of a fault monitoring instrument is selected and data is shifted through it, the *clear flag* is asserted to indicate that the fault from the fault monitor has been acknowledged. In Fig. 3(b), the 3-bit registers, namely EMR_1 and EMR_2, are the EMRs associated to Monitor 1 and Monitor 2, respectively.

To enable self-reconfiguration, we propose a *modified* SIB, which is the core component in a self-reconfiguring network. A *modified* SIB, while being IEEE 1687 compliant, can additionally be opened asynchronously via a dedicated terminal. The symbol shown in Fig. 3(a) will be used in the rest of this paper to represent a *modified* SIB. In Section VII, we detail the circuitry of the proposed *modified* SIB.

All fault monitoring instruments in the network are connected to the top-level SIB_0 through a network of *modified* SIBs. The main difference between a regular SIB and a *modified* SIB is the pair of terminals "open" and "toOpen". The "open" terminal of a *modified* SIB is connected either to (1) the fault flag of the monitoring instrument—see SIB_1 and SIB_2 in Fig. 3(b)—or (2) the ORed output of the "toOpen" terminals of all *modified* SIBs attached to it (placed one hierarchical level below). When the "open" terminal is asserted (pulled high), it changes the state of the SIB to opened only if the SIB is not already part of an active-scan path. The signal from the "open" terminal is gated internally using (an inverted copy of) the select signal to make sure that the state of the SIB does not change (from closed to opened) when it is part of an active scan-path (see Fig. 8 for details on the *modified* SIB). The "toOpen" terminal propagates the internally gated signal (from the "open" terminal) via an OR gate either to (1) the *modified* SIB in the hierarchical level above, or (2) the ErrorFlag register in the top level—see Fig. 3(b). Note that when the fault flag has managed to propagate to the ErrorFlag register, all the *modified* SIBs on the path from the fault monitor raising the flag to the top level SIB_0 are properly configured, i.e. the network has self-reconfigured.

A requirement for a *modified* SIB (as well as for SIB_0 and SIB_{ins}) is to have its shift (S) flip-flop placed after the hierarchical mux (similar to what is shown in Fig. 1(a)). Such placement, while being fully standard compliant, ensures that during shifting, the state of the SIB is always shifted out first. This is required by the fault-localization method to determine the current configuration of the network.

B. Fault Detection and Localization Method

In this section, we explain the fault detection and localization method with the help of the example network shown in Fig. 3(b)

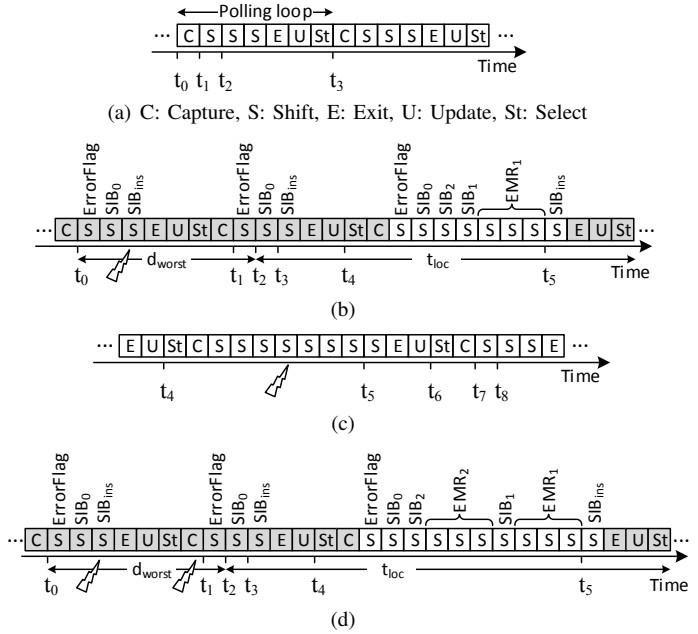


Fig. 4. The detection and localization method: (a) constant polling to detect a fault, (b) an error is detected and localized, (c) another error happens when the previous one is being localized, and (d) when two faults are detected together.

and the timelines shown in Fig. 4. The following scenarios are considered: (1) no error has occurred, (2) an error occurs when the Fault Manager is not localizing another error, (3) an error occurs when the Fault Manager is localizing another error, and (4) two errors occur in a short span of time when the Fault Manager is not localizing another error.

For the first scenario, when no error is reported, the Fault Manager constantly checks the status of the system by polling the value captured by the ErrorFlag register. The Fault Manager does the polling via looping constantly through the *Capture*, *Shift*, *Exit1*, *Update*, and *Select* states in the DR branch of the TAP FSM. Since SIB_0 is closed when no errors are being localized, the polling takes seven test clock cycles (TCK)—the interval between t_0 and t_3 in Fig. 4(a)—as three shifts are required: for SIB_{ins}, SIB_0, and ErrorFlag. The value of the fault flag raised by monitoring instruments is captured at t_1 into ErrorFlag register and can be observed at t_2 (see Fig. 4(a)). The polling continues as long as the shifted out bit corresponding to the ErrorFlag register is a "0". During polling, zeros are shifted in to keep SIB_0 and SIB_{ins} closed.

For the second scenario (see Fig. 4(b)), consider that a fault happens at the interval between t_0 and t_1 and is reported by Monitor 1. The reason we chose this interval is that no matter when in this interval a fault occurs, it will not be captured until t_1 and will therefore not be detected until shifted out at t_2. We refer to the interval between t_0 and t_2 (which is eight TCKs long) as the worst-case fault detection time (when no other error is being localized) and denote it by d_{worst}. When the value shifted out at t_2 (which belongs to the ErrorFlag) is a "1", the localization procedure is launched by shifting a "1" into SIB_0 at t_3 which takes effect at the following Update phase (t_4). Once SIB_0 is open, as the rest of the network is already self-reconfigured, the Fault Manager starts shifting out data from the network (while shifting in zeros to close the SIBs and reset EMRs on the active scan-path) to localize the fault: The first two bits shifted out are the contents of ErrorFlag and SIB_0. The third bit is the contents of SIB_2 for which a value of zero indicates that SIB_2 is closed and the fault is not reported from the network segment connected to the hierarchical port of SIB_2. The next bit is the contents of SIB_1 which is "1" meaning that SIB_1 is open and the fault is

reported from the segment connected to it, i.e., Monitor 1 in this example. The next three bits are the contents of the 3-bit EMR_1 which interfaces Monitor 1 to the IEEE 1687 network. At this point, i.e., at t_5, the error is localized and the error information is retrieved. In this work, however, we include in the localization time (denoted by t_{loc}) the next four TCKs needed to shift-in one more zero for SIB_{ins} and take the TAP FSM back to the capture phase. The worst-case error detection and localization time can then be written as

$$t_{worst} = d_{worst} + t_{loc} \quad (1)$$

In practice, for the above scenario, d_{worst} should be extended to include the time that it takes a fault flag signal to propagate from the fault monitoring instrument to the ErrorFlag. We denote this propagation delay by δ and note that if the fault monitor signals the error later than t_0-δ, it is not captured at t_0. Therefore, d_{worst} should be written as t_2-t_0+δ which is equal to $8/f_{TCK}+\delta$ where f_{TCK} is the maximum frequency that the JTAG TAP can be operated at.

For the third scenario, when an error happens while the Fault Manager is localizing a previous error, consider Fig. 4(c) as continuation of the timeline in Fig. 4(b). As discussed for the second scenario above, at t_4 SIB_0 is opened which puts SIB_1 and SIB_2 on the active scan-path. SIB_0 is closed at t_6 meaning that between t_4 and t_6 SIB_2 is selected (though closed) and, therefore, cannot be opened by a fault flag signal from Monitor 2. That is, any fault reported by Monitor 2 after t_4, is captured at t_7 and detected at t_8. Since SIB_2 is closed, the fault flag from Monitor 2 is not acknowledged and therefore remains active until SIB_2 is opened and the error code from Monitor 2 is captured into EMR_2.

For the last scenario, consider the timeline in Fig. 4(d), where both monitors detect faults in the interval between t_0 and t_1. In this case, both faults are detected at t_2. In comparison to the scenario for one fault (see Fig. 4(b)), the localization procedure takes a longer time as this time SIB_2 is also opened and EMR_2 is also included in the scan-path.

As a final note in this section, we observe from Fig. 4(b) and Fig. 4(d) that the shaded states are traversed no matter how many faults are being detected and localized. We denote this constant overhead of 18 TCKs by J_{OH}, and write Eq. (1) as

$$t_{worst} = J_{OH} + t_s \quad (2)$$

where t_s denotes the number of shift cycles in t_{loc} and varies by the number of faults being localized.

IV. Time Analysis

In this section, we present analyses for the worst-case error detection and localization time (t_{worst}) in a self-reconfiguring network, for two cases: when a single fault occurs, and when multiple faults occur such that they are all detected by the Fault Manager at the same time (see the discussion on Fig. 4(d)).

As shown in Eq. (2), t_{worst} has a constant part J_{OH} and a variable part t_s. For the analyses we focus on calculating t_s.

We present time analyses for balanced tree networks such as the network shown in Fig. 5. In such networks, each SIB, except for the ones in the lowest level, has k SIBs connected to its hierarchical port. The SIBs in the lowest level are interfaced to the instruments. For $N = k^h$ instruments, the network resembles a balanced k-ary tree whose root is SIB_0.

A. In Case of a Single Fault

Given the network in Fig. 5, assuming that only one monitor has raised a fault flag causing all SIBs on its hierarchy to change state to open, there are $s = 2 + k \times h$ SIBs on the path to each

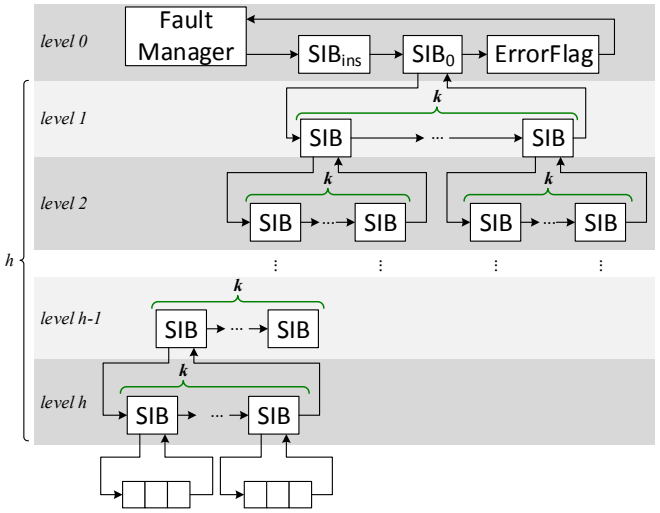


Fig. 5. A balanced tree hierarchical network

instrument, where 2 represents SIB_0 and SIB_{ins}. The total shift time t_s is therefore the sum of s and the length of the EMR (denoted by L) plus one for ErrorFlag:

$$t_s = 2 + k \times h + L + 1 = 3 + k \times \log_k N + L \quad (3)$$

B. In Case of Multiple Faults

Assume that F faults ($F \leq N$) are to be localized at the same time. To calculate t_s, we consider monitors detecting these faults to be spread in the network such they cause maximum possible number of SIBs to be opened (maximizing the length of the active scan-path, thus leading to the longest localization time). As an example, when $F = k$ faults happen in the system monitored via the network in Fig. 5, the localization time is maximized when each of these k faults happen in the subtree of each of the k SIBs in level 1. Another observation is that for localization of $F \geq 1$ faults, the SIB at level 0 is opened, for $F \geq k$, all SIBs in level 1 are opened, for $F \geq k^2$, all SIBs in level 2 are opened, and so on. The number of these SIBs, which are on the scan-path to all F monitors (i.e., shared by all of them), is captured by $\sum_{i=0}^{r} k^i$ where $r = \lceil \log_k F \rceil$ is the number of upper levels in which all the SIBs are open. Next, to calculate the number of SIBs exclusively on the path to each of the F monitors, we note that $h - r$ remaining lower levels are open exclusively for each fault, each having k SIBs—thus $F \times k \times (h - r)$ is the total number of SIBs exclusively opened for the F monitors. To sum up, the total number of SIBs on the scan-path for the F faults is $s = 1 + \sum_{i=0}^{r} k^i + F \times k \times (h - r)$, where 1 is for SIB_{ins}. To calculate t_s, we need to add to this number of SIBs, the total length of EMRs on the scan-path (i.e., $F \times L$) as well as one for the ErrorFlag, as follows:

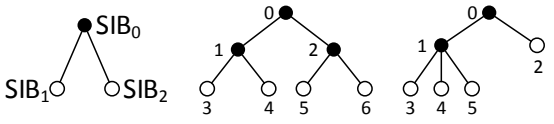
$$t_s = 1 + \sum_{i=0}^{r} k^i + F \times k \times (h - r) + F \times L + 1 \quad (4)$$

V. Network Design

In this section, we describe a design method for the proposed self-reconfiguring networks such that t_{worst} for a single fault is minimized. As t_{worst} has a constant part J_{OH} and a variable part t_s (see Eq. (2)), minimizing t_{worst} reduces to minimizing t_s. For a single fault, t_s is calculated using Eq. (3). Assuming that the number of instruments N is given, to find k which minimizes t_s, we set the first derivative of t_s w.r.t. k to zero:

$$t_s = 3 + \ln N \times \frac{k}{\ln k} + L \;\Rightarrow\; t'_s = \ln N \frac{\ln k - 1}{(\ln k)^2} \quad (5)$$

$$t'_s = 0 \;\Rightarrow\; \ln k = 1 \;\Rightarrow\; k = e \quad (6)$$



(a) (b) (c)

Fig. 6. Alternative representation of networks, where filled circles represent the SIBs which are not directly connected to instruments, empty circles represent SIBs connected to instruments, and edges represent the hierarchical relations: (a) representation of network in Fig. 3(b), (b) and (c) networks for four instruments

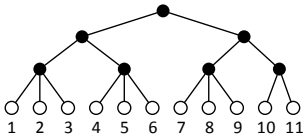


Fig. 7. Representation of a self-reconfiguring network for 11 instruments

Since $\mathrm{e} \approxeq 2.72$, we should either choose $k = 2$ or $k = 3$ to minimize t_s. However, given an arbitrary number of instruments N where N is not a power of two or three, it is not possible to construct a balanced tree. In such cases, a straightforward way to construct the network is to create a balanced tree for $k^{\lceil \log_k N \rceil}$ instruments, where $k = 2$ results in a binary tree and $k = 3$ results in a ternary tree, and prune the tree (after placing the N instruments at the leaf nodes). Pruning can be done by removing the internal nodes to which one or no instrument is connected. After pruning, one can compare the results from the pruned binary and ternary trees and pick the better one. In the following, we present a network construction method that by mixing binary and ternary subtrees yields similar or better results compared with each of the pruned binary and ternary tree alternatives.

Let us now switch to a simpler network representation which is more suitable for the discussion in this section. The tree in Fig. 6(a) captures the hierarchical relation (and not the data connections) between the SIB components in the network shown in Fig. 3(b). The instruments are not shown (as the length of instruments' shift-registers has no effect on SIB shifting overhead) and those SIBs directly connected to instruments are represented by empty circles. In Fig. 6(a), node SIB_0 is parent to sibling leaf nodes SIB_1 and SIB_2. When a parent SIB is opened, its children are on the scan-path no matter if they are opened or closed. In other words, when a node is on the scan-path, all its siblings are also on the scan-path.

As the proposed network construction method is based on bundling instruments in groups of three, we would first like to make an observation for when the remaining number of instruments is one, i.e., when $N \bmod 3 = 1$. Fig. 6(b) and Fig. 6(c) show two networks constructed for four instruments. When in the network in Fig. 6(b) a fault is detected at instrument connected to the SIB at node 3, that SIB (node 3) as well as the SIB at node 1 are opened. This means that in total five SIBs are on the path (namely, nodes 0–4) and it therefore takes five clock cycles to read their status. In this case, as all instruments have the same number of SIBs on their scan-path, the average-case and the worst-case fault localization time is the same for all of them. This, however, is not the case for the network represented in Fig. 6(c) in which for the instrument connected to node 2 three shift clocks are needed while for those connected to nodes 3–5 six shift clocks are needed—averaging to $(3 \times 6 + 1 \times 3)/4 = 5.25$ clock cycles. It can be seen from this example that the network represented by Fig. 6(b) results in both better average-case and worst-case shifting time.

Based on the above observation, we propose the following construction algorithm. For given N instruments, we bundle the instruments into clusters of three instruments each. When N is a multiple of three, we will have $c = N/3$ clusters. If N is not a multiple of three, one or two instruments will remain.

TABLE I. $\mathrm{t}_{\mathrm{worst}}$ FOR A SINGLE FAULT (IN TCKS)

Number of instruments	[7]	Self-reconfigurable networks			
		tree from [11]	binary tree	ternary tree	this work
25	90	35	34	33	33
50	118	37	36	35	35
100	158	39	38	38	37
200	206	42	40	39	39
500	266	52	42	42	42
1000	326	53	44	44	44

Following the observation made for Fig. 6(b), when the number of remaining instruments is one, we make $c = \lfloor N/3 \rfloor - 1$ three-instrument clusters plus two two-instrument clusters. If, however, the number of remaining instruments is two, we make $c = \lfloor N/3 \rfloor$ three-instrument clusters plus one two-instrument cluster. Assuming each cluster to be an instrument, the same procedure described above can be applied to the created clusters, creating clusters of clusters until the network is complete. Fig. 7 shows this procedure for 11 instruments.

VI. COMPARISON WITH SIMILAR APPROACHES

We have compared our proposed self-reconfiguring IEEE 1687 network with the work presented in [7] (which uses a regular IEEE 1687 network for monitoring) with regards to $\mathrm{t}_{\mathrm{worst}}$ (see Eq. (1)). In this section, we present the results of the comparison for the case (1) one fault occurs (discussed in Section IV-A), and the case (2) multiple faults detected by the Fault Manager at the same time (discussed in Section IV-B).

A. For a Single Fault

Table I shows the results of comparison with the approach proposed in [7]. For the construction of the proposed self-reconfiguring network, we compared four alternatives: (1) using the network construction method for regular IEEE 1687 networks presented in [11], (2) a binary tree with pruning, (3) a ternary tree with pruning, and (4) the construction method proposed in Section V. To calculate the number of SIBs on the scan-path for the self-reconfiguring networks, pre-order tree traversal is employed. A fixed number of $J_{\mathrm{OH}}+2 = 18+2$ TCKs is added to the calculated shift time to account for the constant overhead (see Section III-B), ErrorFlag, and $\mathrm{SIB}_{\mathrm{ins}}$. Moreover, another three TCKs are added to account for the length of the fault monitor's EMR (i.e., $L = 3$).

From the results, it can be seen that by using the proposed self-reconfiguration scheme (regardless of the considered network tree construction method), at least 2.6x reduction in localization time is achieved compared to [7]. The reason for this improvement can be attributed to opening many hierarchical levels in a single CSU and having only one single-bit ErrorFlag register directly on the scan-path.

Among the construction methods examined for the self-reconfiguring network, the one described in Section V performs up to 17% better than the method in [11] and results in better or equal $\mathrm{t}_{\mathrm{worst}}$ compared to binary and ternary trees.

B. For Multiple Faults

The work in [7] has not presented analysis and results on multiple faults. On the other hand, our calculations for multiple faults (see Section IV-B) are for balanced k-ary trees, and cannot be directly used for the network structures presented in [7]. Therefore, to perform the comparison, we (1) used constraint programming (by using the constraints formulation in [7]) to get the optimal network structure for the number of instruments suitable for our analysis (see below), and (2) developed time analysis for multiple faults for the networks presented in [7] based on their time analysis for a single fault and our analysis for multiple faults.

TABLE II. t_{worst} FOR MULTIPLE FAULTS (IN TCKS)

# instruments	Number of faults									
	1	2	3	4	5	6	7	8	9	10
27	33	42	51	57	63	69	75	81	87	90
	90	126	162	162	162	162	162	162	162	162
81	36	48	60	69	78	87	96	105	114	120
	146	202	258	314	370	426	426	426	426	426
243	39	54	69	81	93	105	117	129	141	150
	334	538	742	946	1150	1150	1150	1150	1150	1150
729	42	60	78	93	108	123	138	153	168	180
	298	486	674	862	954	1046	1138	1230	1322	1414
2187	45	66	87	105	123	141	159	177	195	210
	394	662	930	1118	1306	1494	1682	1870	2058	2246
Average ratio	6.2	7.1	7.5	7.8	7.8	7.5	7.1	6.7	6.4	6.3

The shaded numbers are calculated based on the approach in [7].

As for the number of instruments, we chose numbers which are powers of three resulting in networks resembling balanced ternary trees. For each of these networks, we calculated t_{worst} for 1 to 10 faults. For each pair of network and number of faults, we calculated t_{worst} using Eq. (2) where $J_{OH} = 18$ and t_s is calculated using Eq. (4). The results are presented in Table II where the shaded rows present the numbers obtained for the network structure type proposed in [7]. The last row, presents the average improvement ratio achieved over [7] in case of multiple concurrent faults which ranges between 6.2 to 7.8 times improvement. As was the case for single faults, the reason for this improvement can be attributed to opening many hierarchical levels at once and having only one ErrorFlag register directly on the scan-path.

VII. Practical Issues

To validate our proposed self-reconfiguring networks, we implemented one such network and performed post-layout simulations (in 65nm technology) for the scenarios discussed in Section III-B. Fig. 8 shows our implementation of the proposed *modified* SIB. Before discussing Fig. 8, we should mention that depending on the available standard cell library, simpler designs with the same functionality are possible, and that our implementation is affected by our ASIC vendor's library.

In Fig. 8, the clock signal is not shown to avoid clutter. The *Reset* signal is the synchronous active-low reset from *Test-Logic-Reset* state in the JTAG TAP FSM. The self-reconfigurability revolves around the U′ flip-flop which is D-type with asynchronous active-high *set*. The *set* input of U′ is connected to a gated copy of the *Open* terminal of the SIB. The *Open* signal is gated via the *Select* signal so that the self-reconfiguration only happens when the SIB is not selected (i.e., not part of the active scan-path). The *Q* output of U′ is used to open the SIB—i.e, include the segment connected between *TSI* and *FSO* terminals in the scan-path. The output of U′ is captured into the S flip-flop (as required by the localization method described in Section III-B) when the TAP FSM goes through the capture phase. U′ is cleared when the TAP FSM goes through the *Update* phase or through the *Test-Logic-Reset* state during initialization.

To give an idea of the propagation delay (denoted earlier by δ) in a large network, we constructed a network with 2187 instruments (which is a balanced ternary tree with seven layers) and performed synthesis and place & route (optimized for a 100MHz TCK) using a 65nm technology. The reported delay between each of the 2187 instrument fault flags and the parallel input of the ErrorFlag register (i.e., through all seven layers) shows a minimum delay of 1.73ns, average delay of 2.1ns and maximum delay of 2.62ns (still shorter than one TCK period).

As an example, assuming an on-chip Fault Monitor which can operate the network at 100MHz, the worst-case localization time (in seconds) for the case of one fault happening in a network with 2187 instruments (see Table II) is calculated as $45 \times \frac{1}{100\times10^6} + 2.62 \times 10^{-9}$ which is 452.62ns.

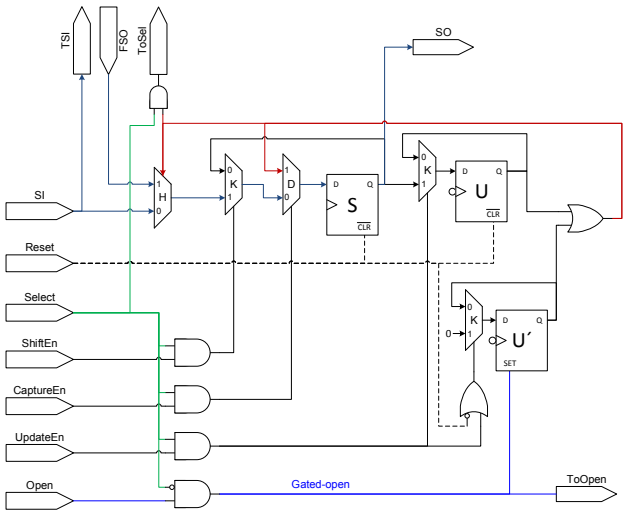


Fig. 8. Schematic of the proposed *modified* SIB

VIII. Conclusion

In this paper, fault localization time is reduced by introducing a segment insertion bit (SIB) that enables self-reconfiguration of IEEE 1687 networks. We presented timing analysis for single and multiple concurrent faults, as well as a construction method for the proposed self-reconfiguring network. We validated the idea of self-reconfiguring networks through post-layout simulations of one such network. We compared the proposed scheme with a previous similar work and observed at least 2.6 times reduction in localization time for a single fault and 6.2 times reduction in case of multiple faults.

Acknowledgments

This work is supported by the European Commission through the FP7 STREP Project no. 619871 (BASTION), as well as by the Royal Physiographic Society of Lund.

References

[1] S. Borkar, "Designing Reliable Systems from Unreliable Components: the Challenges of Transistor Variability and Degradation," *IEEE Micro*, vol. 25, no. 6, pp. 10–16, 2005.

[2] Sun Microsystems, Inc. (2016, Mar.), "UltraSPARC T2™ Supplement to the UltraSPARC Architecture 2007." [Online]. Available: http://www.oracle.com/technetwork/systems/opensparc/t2-14-ust2-uasuppl-draft-hp-ext-1537761.html

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

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

[5] A. Bouajila, A. Lakhtel, J. Zeppenfeld, W. Stechele, and A. Herkersdorf, "A Low-Overhead Monitoring Ring Interconnect for MPSoC Parameter Optimization," in *Proc. IEEE International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, April 2012.

[6] S. Madduri, R. Vadlamani, W. Burleson, and R. Tessier, "A Monitor Interconnect and Support Subsystem for Multicore Processors," in *Proc. Design, Automation & Test in Europe Conference*, April 2009.

[7] 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.

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

[9] K. Petersen, D. Nikolov, U. Ingelsson, G. Carlsson, F. Zadegan, and E. Larsson, "Fault Injection and Fault Handling: An MPSoC Demonstrator using IEEE P1687," in *Proc. IEEE International On-Line Testing Symposium (IOLTS), 2014*, July 2014, pp. 170–175.

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

[11] ——, "Design Automation for IEEE P1687," in *Proc. Design, Automation & Test in Europe Conference (DATE)*, March 2011.