FP7-ICT-2013-11-619871

BASTION

Board and SoC Test Instrumentation for Ageing and No Failure Found

Instrument: Collaborative Project

Thematic Priority: Information and Communication Technologies

Report on methods for IJTAG network adaptation and optimization for error detection and diagnosis (Deliverable D3.2)

Due date of deliverable: June 30, 2016
Ready for submission date: January 4, 2017

Start date of project: January 1, 2014 Duration: 40 months

Organisation name of lead contractor for this deliverable: Hochschule Hamm-Lippstadt
University of Applied Sciences

Revision 1.3

Project co-funded by the European Commission within the Seventh Framework Programme (2014-2016)		
Dissemination Level		
PU	Public	☒
PP	Restricted to other programme participants (including the Commission Services)	☐
RE	Restricted to a group specified by the consortium (including the Commission Services)	☐
CO	Confidential, only for members of the consortium (including the Commission Services)	☐

Notices

For more information, please contact Prof. Dr. Krenz-Baath, e-mail: rene.krenz-baath@hshl.de

This document is intended to fulfil the contractual obligations of the BASTION project concerning deliverable D3.2 described in contract 619871.

Table of Revisions

Version	Date	Description and reason	Author	Affected sections
0.1	May 22, 2016	Initial document created	R. Krenz-Baath	All
0.2	September 02, 2016	Added HSHL contribution	R. Krenz-Baath	Section 5
0.3	September 12, 2016	Added all submitted contributions	R. Krenz-Baath	all
0.4	September 17, 2016, September 27, 2016	Added IFAG contributions Merged and updated contributions	P. Engelke, C. Laudert, R. Krenz-Baath	Section 2.5.1.2, all
0.5	September 27, 2016	Reviewed contributions added introduction	R. Krenz-Baath	Section 1
0.6	October 12, 2016	Section 2 considerably extended	A. Jutman	Section 2
0.7	October 16, 2016	Major Review, extended introduction, added conclusions	R. Krenz-Baath	all
0.9	November 14, 2016	Updated Section 2.5.1.2	P. Engelke	Section 2.5.1.2
0.9a	November 15, 2016	Updated Section 2	K. Shibin	Section 2
1.0	December 23, 2016	Final Review	R. Krenz-Baath	all
1.1	December 23, 2016	Added “Summary” to Section 3	C. Laudert	Section 3
1.1a	December 28, 2016	Preparing for submission	R. Krenz-Baath, A. Jutman	General parts
1.2	December 30, 2016	Update of KPIs	A. Jutman	Conclusions
1.3	January 4, 2016	Update of KPIs	A. Jutman	Conclusions

Author, Beneficiary

René Krenz-Baath, HSHL
Erik Larsson, ULUND
Farrokh Zadegan, ULUND
Artur Jutman, TL
Sergei Devadze, TL
Konstantin Shibin, TL
Piet Engelke, IFAG
Carsten Laudert, IFAG
Matteo Sonza Reorda, PDT

Executive Summary

This document presents BASTION contributions in the area IJTAG network for error detection and diagnosis. In the following for BASTION contributions are discussed in detail. The first contribution focuses on efficient error propagation through IJTAG networks. Secondly aspects and contributions with respect to hierarchical testing using IJTAG networks. The next contribution provides solutions for testing IJTAG networks. After that a contribution to compute upper-bounds in the context of dynamic retargeting techniques is presented. Finally, conclusions are provided, including the discussion on KPIs.

List of Abbreviations

AFPN	Asynchronous Flag Propagation Network
ATPG	Automatic Test Pattern Generation
BIST	Build Inside Test
CPU	Central Processing Unit
CSU	Capture-Shift-Update Cycle
DfT	Design for Testability
DoW	Description of Work
DRC	Design Rule Check
ECC	Error-Correction Code
FP7	European Union's 7th Framework Programme
FM	Fault Management
FMI	Fault Management Infrastructure
FSM	Finite State Machine
ICL	Instrument Connectivity Language
IEEE	Institute of Electrical and Electronics Engineers
IJTAG	Internal JTAG, a short name for IEEE 1687 standard and infrastructure collectively
ILP	Integer Linear Programming
IM	Instrument Manager
IP	Intellectual Property
IST	Information Society Technologies
ITRS	International Technology Roadmap for Semiconductors
JTAG	Joint Test Action Group
LBIST	Logic BIST
LSIB	Locking Segment Insertion Bits
MBIST	Memory BIST
MUX	Multiplexer
PN	Perfect Network
NFF	No Fault Found, also No Failure Found
OAT	Overall Access Time
OS	Operating System
PCB	Printed Circuit Board
PDL	Procedural Description Language
RM	Resource Manager
RSN	Reconfigurable Scan Network
RTL	Register-Transfer-Level
SAT	Boolean Satisfiability
SCB	ScanMux Control Bit
SI	primary Scan Input
SIB	Segment Insertion Bit
SO	primary Scan Output
SoC	System-on-Chip
TAP	Test Access Point
TDI	Test Data Input
TDO	Test Data Output
TDR	Test Data Register
UCC	Upper-bound Computation Core

URL	Uniform Resource Locator

Table of Contents

1 Introduction

In this deliverable, results are reported of the research performed by BASTION partners in the area IJTAG network for error detection and diagnosis. It summarizes and concludes work performed in T3.2 “IJTAG network for error detection and diagnosis”. The contributions are described in separate sections.

The first contribution proposes an extension to IJTAG for system health-monitoring and Fault Management (FM), which has been introduced in BASTON D2.3. Furthermore, issues such as Instrument synchronization, calibration as well as Health Map composition and respective fault classification scheme have been discussed in D2.3 and D3.1. Based on the earlier reported results this deliverable discusses implementation details of the respective hardware.

The next deliverable focuses on the application of reconfigurable scan networks for hierarchical design and test. In particular general requirements for core wrapper isolation are discussed and initial experimental results are presented.

The third contribution in this deliverable discusses aspects of testing IJTAG networks. The described contribution proposes a method to generate sequences of tests which incrementally tests components of SIB-based IJTAG networks. The authors describe in detail the test of SIBs, ScanMuxes and TDRs.

Optimal retargeting of PDL instructions has been discussed in D2.3. The final contribution extends the optimal retargeting by generalizing the computation of the upper-bound of time frames which needs to be considered during the retargeting process. Previously this upper-bound could only be determined for specific structures in IJTAG networks. By applying the new approach, the upper-bound can be determined for large IJTAG networks independent of structural properties. Furthermore, this new concept of modelling IJTAG networks enables future works on verification and validation of such structures.

1.1 The Structure of the Report

This report is structured as follows. Sections 2,3,4,5 contain the descriptions of the contributions in the areas error propagation, hierarchical test and design, test of IJTAG networks and test pattern retargeting, respectively. This report is concluded in Section 6.

2 Fault Detection, Localization, and Propagation through IJTAG Networks

2.1 Introduction

Rapid emergence of embedded instrumentation as an industrial paradigm and adoption of respective IEEE 1687 standard [1] by key players of semiconductor industry opens up new horizons in developing efficient test, debug and health monitoring frameworks. The IEEE Std 1687 also shortly called IJTAG has been initially started as an initiative to standardize access to on-chip embedded instrumentation, like monitors, sensors and checkers as well as DFT (Design-for-Testability) infrastructure, various BIST (Built-In Self-Test) and trace data collection solutions for system and software debug [2]. The IJTAG concept embraces also the paradigm of Reconfigurable Scan Networks (RSN) [3] and has become a very attractive industrial solution for both scan-based manufacturing test and system debug [2], [3], [4], [5].

2.2 Related Work

Expected large-scale adoption of IEEE 1687 standard and respective infrastructure by chip vendors created an important opportunity to reuse IJTAG in the field. An extension to IJTAG for system health-monitoring and Fault Management (FM) has been proposed in [5] and [6] and further elaborated in [7], [8], [9], [10], [11], [12].
Recent works focusing on implementation challenges of on-chip IJTAG retargeting engines [13] and on-the-fly retargeting framework [14] also consider instrumentation reuse in the field. Reliability and fault tolerance of IJTAG networks during online FM operation has been detailed in [15].

2.3 BASTION Contributions

An architectural extension to IJTAG for system health-monitoring and Fault Management (FM) has been presented in BASTION D2.3. Instrument synchronization, calibration and triggering approaches have been introduced in D2.1. The Health Map composition together with respective fault classification scheme, and initial fault handling scenario have been proposed in D3.1. In this deliverable we present details on hardware implementation with main focus made on Instrument Manager (IM).

2.3.1 Instrument Manager

Instrument manager (IM) is a part of the Fault Management Infrastructure (FMI) and is implemented as a hardware block which role is to connect the hardware and software parts of the FMI, as shown in Figure 1. On one side, it is connected to IEEE 1687 IJTAG network as a controller and on the other side it is accessed as a peripheral to the CPU through the FM bus which can be either specialized separate fault-tolerant bus or a normal system bus. Basic commands from FM are write and read requests to particular instruments in the IJTAG network. The interface between IM and FM works on the basis of register (equivalent to instrument in this context) addresses. FM issues a command which contains the register address with data, and IM is responsible to open the access to the target register through the hierarchy of IJTAG network and subsequently write/read the data. The address is a custom interpretation of the instrument's position inside the IJTAG network. A table of addresses is constructed from a network description (ICL) before runtime and is available to both FM and IM. FM uses it to look up the

address of the required instrument from its logical position (e.g. CPU1.FPU.BIST has and

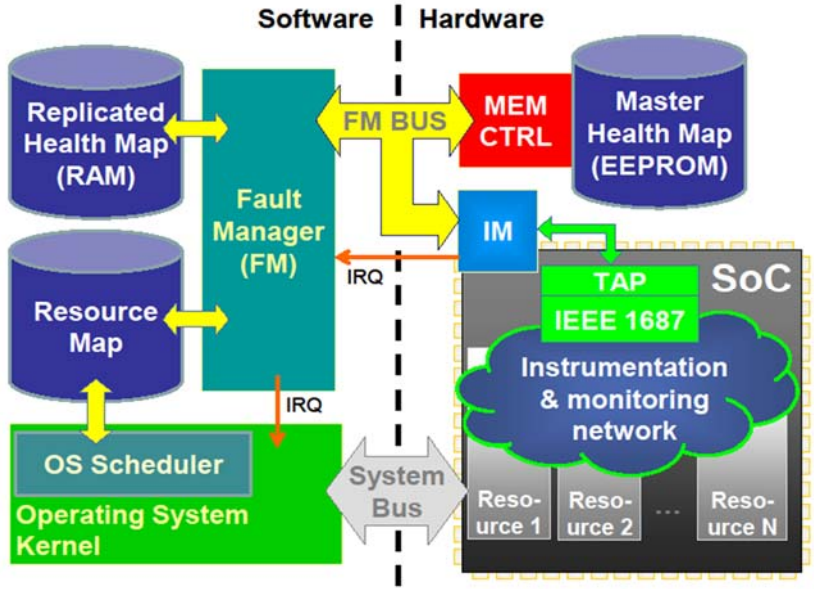


Figure 1. Detailed view of the fault management architecture

address 35). IM has the network map ROM connected directly to the FSM from which it can find out the way to access a register with particular address.

IM is a hardware module which is responsible for the communication with the instruments through the IJTAG network. Whenever FM needs to access the instruments to get the diagnostic information, it gives a read/write command to IM which in turn opens the path to the instrument through the hierarchical IJTAG network and performs the requested operation. Besides instrument access, IM is responsible for reacting to the fault flags set by the instruments and propagated as an asynchronous interrupt signal. IM automatically opens the path to the instrument which raised the fault flag and provides the information about its location to FM or directly to the health map. However, IM can only provide coarse fault location information in this manner and FM should start the diagnostic procedure to find out the fine-grained fault location information and update the health and resource maps.

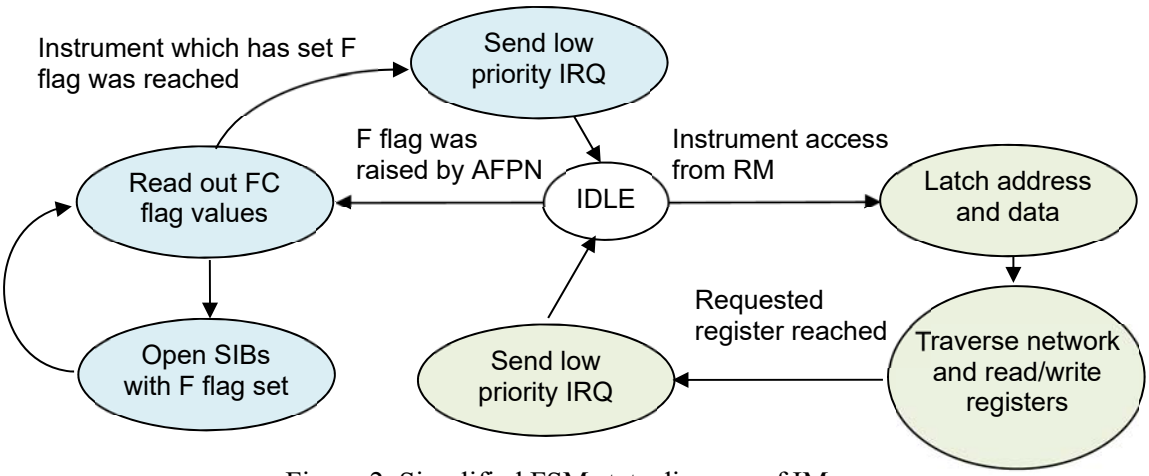


Figure 2. Simplified FSM state diagram of IM

2.3.2 Operation of IM

IM has two main modes of operation:

- Execution of instrument access commands from IM;

- Autonomous fault localization.

When IM is not in either of these modes, it is idle. IM operates as an FSM, the state diagram is shown in Figure 2.

2.3.3 Execution of instrument access commands from FM

In this mode, IM is receiving commands from the software (FM) about performing an operation with IJTAG network. FM can instruct IM to do one of the following operations:

- Read value of TDR of an instrument;
- Write value of TDR of an instrument;
- Open path to TDR of an instrument;
- Update the X bit – used to mask/unmask the flags on a SIB (see details below);
- Close all SIBs – used to restore the initial idle value of the network.

Since in most cases there are many instruments in the IJTAG network, the instrument-related operations need an indication about which instrument is targeted. FM communicates this by means of setting the Instrument Address (IA) in the command register of IM. This address is connected to the position of the instrument in the IJTAG chain and is only used in communication between FM and IM.

2.3.4 Autonomous fault localization

This mode is invoked only when IM is idle and the Asynchronous Flag Propagation Network (AFPN) signals an uncorrected fault condition (F=1, C=0). Apart from issuing the interrupt to notify the upper layers of the system, IM can autonomously start executing the fault localization procedure. This is possible thanks to the fault flags being set at each SIB which has an unmasked fault in the underlying IJTAG network segment. Ideally, by the time FM will react to the incoming fault interrupt, IM will already be ready with the address of the instrument which raised the fault flag, thus speeding up the whole fault handling procedure.

2.3.5 Hardware implementation of Instrument Manager

IM is implemented in hardware (see Figure 3) as a relatively simple IP consisting of:

- FSM;
- ROM for storing the IJTAG network configuration;
- RAM for storing the IJTAG network state (opened and closed SIBs)
- Interrupt generation logic: low and high priority interrupt request signals to CPU.

IM has two interfaces:

- IM-FM interface – used to communicate with CPU which is running the FM, implemented as a memory-mapped peripheral. Allows to receive commands from FM and send/receive data;
- IJTAG network port – used to access the IJTAG-based instrumentation network.

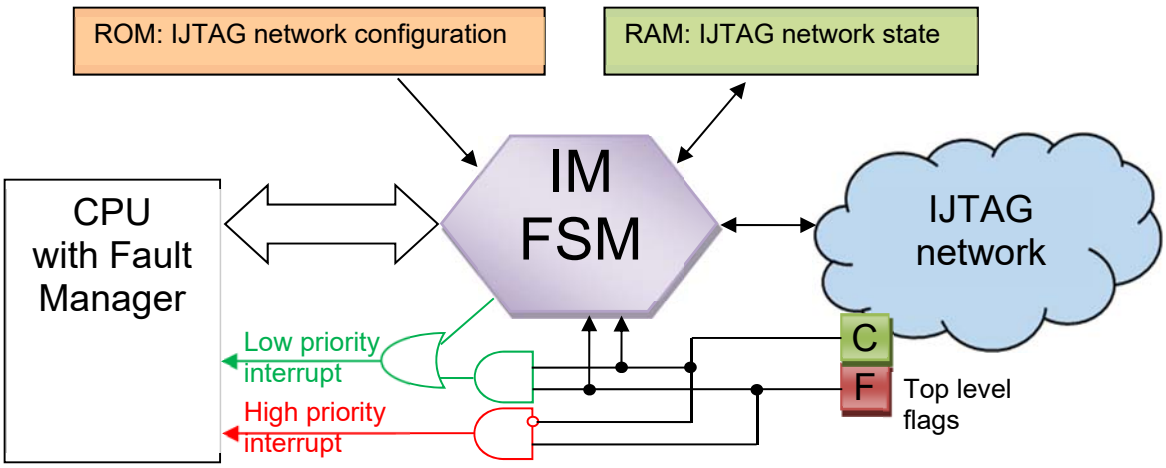


Figure 3. Overview of IM hardware

2.3.6 IM-FM interface

IM uses several signals to communicate with FM on the software side of the fault management architecture:

- IM_CMD – a 32-bit register with command and status bits, bidirectional;
- IM_DATA – a 32-bit register for data transfer, bidirectional;
- Interrupts.

Whenever the AFPN signals that a fault was detected by an instrument or other event that requires reaction from FM occurs, IM immediately sends an interrupt to the CPU(s) which is running the software part of FMI including FM. Since depending on the flags the required urgency of system's reaction is different, there are two interrupt signals:

- High-priority interrupt: only requested in case of an uncorrected fault event (F=1, C=0)
- Low-priority interrupt: all other cases, such as corrected fault event (F=1, C=1), completion of instrument access operation, etc.

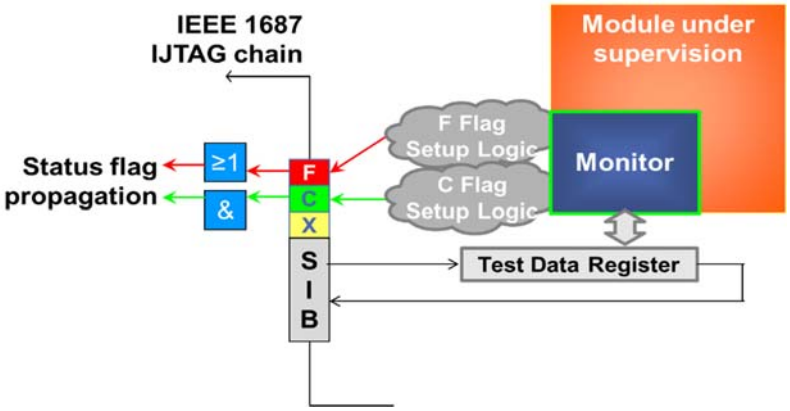


Figure 4. Instrument with FCX emergency flags and TDR

2.3.7 Implementation of SIB with AFPN support

The proposed fault management architecture relies on the asynchronous fault propagation network (AFPN) for fast fault detection. Since AFPN is very closely coupled to the IJTAG network and is also organized in hierarchical fashion (see Figure 5), it is natural that the flags (F, C, X) and the propagation logic is integrated with SIBs. In this section, we describe the modified SIB which is intended to be used with AFPN and includes additional registers to accommodate the flag bits.

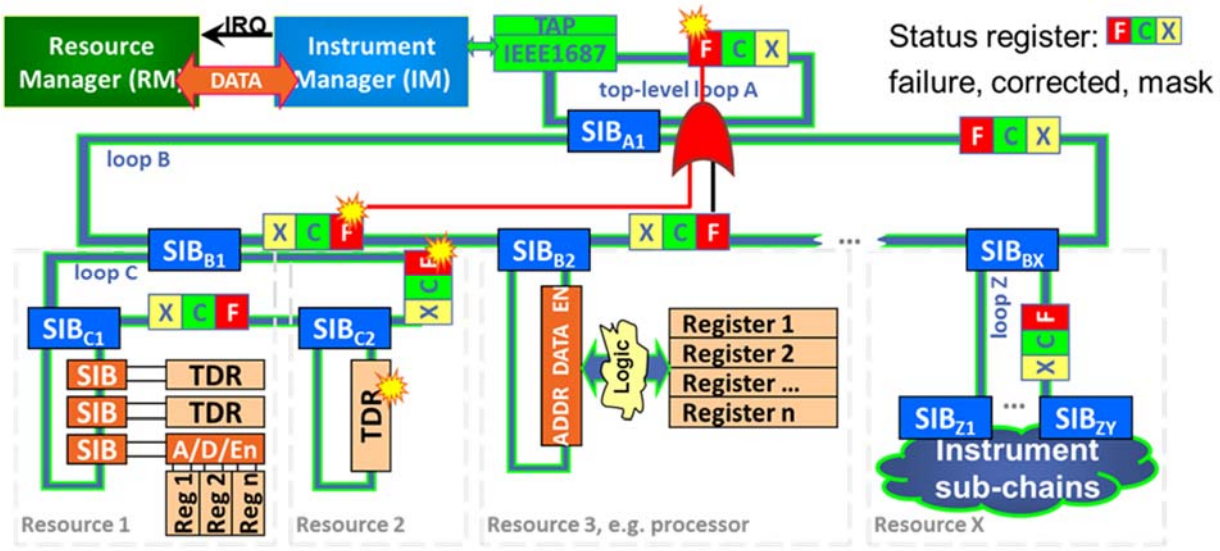


Figure 5. Asynchronous fault propagation and detection in IJTAG network

2.3.8 Requirements for modified SIB

The modified SIB design with AFPN support must include additional registers for capturing and storing the values of the flag bits as well as the circuitry for asynchronous flag propagation. Since F and C flags only need to be captured (read-out) and cannot be written, there is no need to implement the update register part for them. Similarly, the S bit (the one controlling the state of the SIB) and the X flag only need to be written and hence don't need the capture register. Together, the four flags can share the capture and update registers of only two bits:

- F/S bit: when updated, S (SIB state) bit is written; when captured, F flag is read.
- C/X bit: when updated, X (mask) bit is written; when captured, C flag is read.

In this way the extra AFPN flags are added while keeping the impact to communication efficiency minimal: only one extra bit to be shifted in the scan chain (one bit for S is already used by a normal SIB).

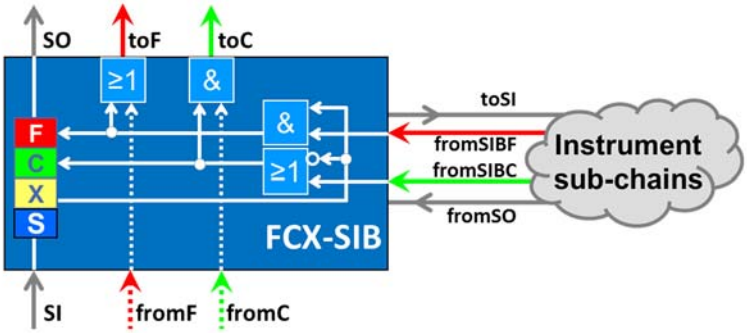


Figure 6. Extended SIB with F&C flag propagation signals (FCX-SIB)

The details of the SIB's hardware implementation are shown in Figure 7, where the logic of the F/S and C/X bits is highlighted.

Besides the capture and update registers and standard SIB logic (generation of gated ShiftEn, CaptureEn and UpdateEn signals), the modified SIB includes the flag propagation logic. This consists of a AFPN logic slice and the signal synchronizers. The latter are composed of two back-to-back flip-flops and are required in case the F and C flag signals are not synchronized to the TCK clock. The AFPN logic can be implemented in several ways, but for the SIB logic it provides an interface with F and C flag outputs and X bit input. Besides that, it must be connected to signals from SIB on lower hierarchical layers and to signals on higher hierarchical layers.

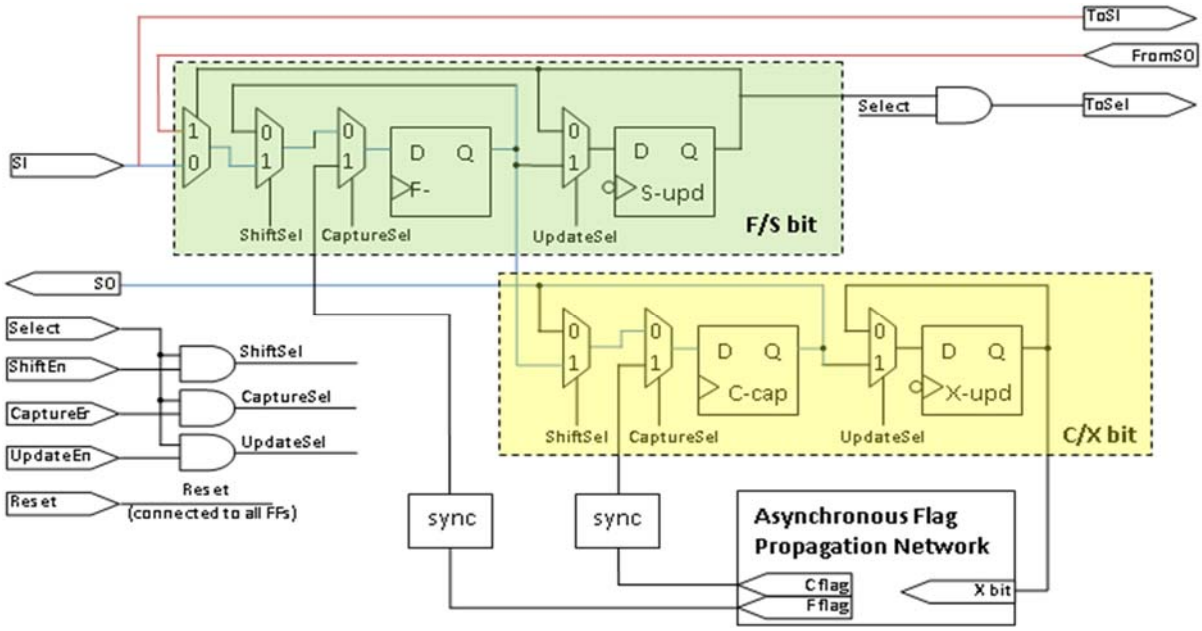


Figure 7. Implementation of SIB with AFPN support

2.4 Handling External Instrument Access Requests (from FM/SW)

When IM receives an access request, it starts shifting in bits by going through ROM and generating the respective values for all active network nodes. When a SIB is encountered, then depending on the state of the SIB, IM either has to shift in only a value for the SIB register (bits S, X, C, F) or shift the data for the underlying network segment as well, a process called dynamic retargeting. The structure of ROM (example of ROM contents in Table 1) facilitates this process by providing the offset value for the case when the SIB is closed. When a SIB is closed, all nodes

of the underlying segments must be skipped and no bits should be shifted in for them. Instead, the bits for the node next to the SIB on the same hierarchical level should be shifted in. The offset is a value which should be added to the current node address to jump the next adjacent node. For instance, let's assume SIB4 is closed in the example network (Fig. 8). Then the following bits shifted in should be those for SIB3. IM implements this by adding the offset value (2) to the current address (1) and jumping to address 3 which corresponds to SIB3. When a SIB as open, then the next word in ROM is used, it corresponds to the last node of the SIB's child network segment. The information about whether each SIB is currently opened or closed is stored in the network state RAM.

When a register is encountered in a ROM word, then its address is compared with the requested address and if they are equal, a value is read or written. If the target register is to be read, then the network's SO values are stored in IM memory and later returned to the host. Otherwise, values of SO are ignored. If the target register is to be written, then the value provided by host is shifted in to the network. When there is no value requested to be written into the current register, all zeroes are shifted in. This limitation will be addressed later by storing the default load and reset values in ROM as described in the IEEE 1687 standard.

The values of SIB F, C, X bits are stored into IM RAM. Value of SIB S bit is generated by IM itself, so there is no need to read it out from the network. The new value for SIB S bit (open or closed) is decided by IM also by using target register address. If the target register address falls in the region between the current SIB address and the word SIB's offset is "jumping" to, then this SIB must be opened since the target register is somewhere inside its underlying network segment. For instance, in the example network, consider register R2 (address 4). In order to decide if SIB2 must be opened, the target address (4) is checked against SIB2 address (0) and where it's offset points to (0+5=5). Since 4 is between 0 and 5, SIB2 must be opened. However, in case of SIB4 and R2, the target address does not fall into the respective region (1 to 3) and hence SIB4 must stay closed.

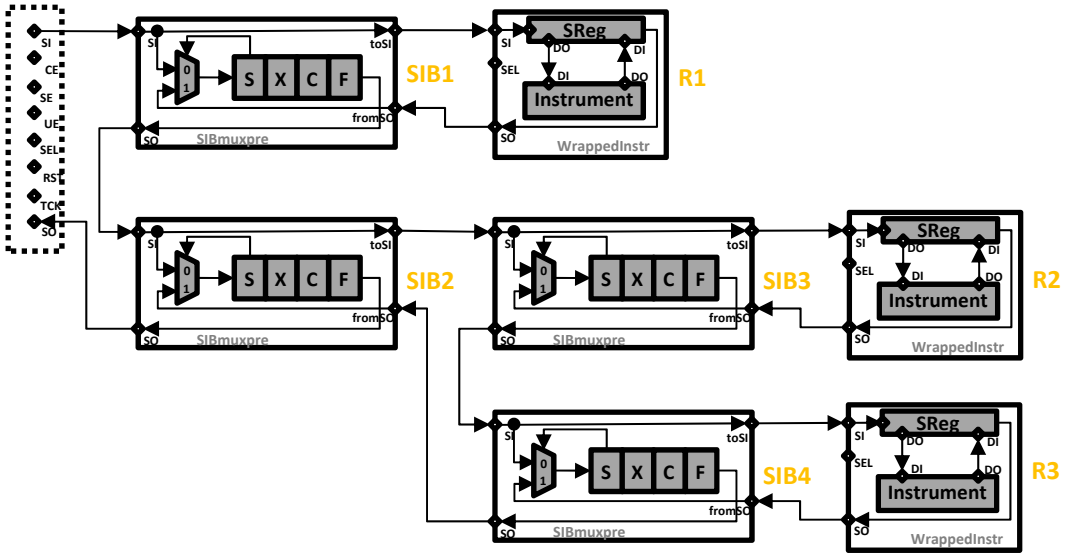


Figure 8. Hierarchical IJTAG network with FMI

2.4.1 IJTAG Network Map ROM Structure

In the network map ROM, each word corresponds to a node in the network (register or SIB). The contents of ROM are generated based on the network description (ICL) starting from the last node (the one connected to SO of the network). When accessing the network, IM can start generating the signal on network SI immediately, as the bits for the last node go in first. By going through the ROM starting from the last node described in the first word, IM can scan in the bits for all nodes of the network.

The host (Fault Manager in the software) requests accesses to registers from IM by using the instrument address in the ROM. The address is equal to the word number in ROM. For instance, in the example network R2 can be accessed by specifying address "4".

Each ROM word has the following fields:

- [1:0] Node type. 00 = SIB, 01 = scan register (instrument), 11 = denotes the end of the map
- [9:2] SIB jump offset/register length. For SIBs, specifies the number of nodes within the underlying network segment which also shows how many ROM words to skip when the SIB is closed. For registers, specifies the length in bits.

Table 1. ROM contents for the example IJTAG network (Fig. 8)

ROM word	Node	SIB offset / reg. length	ROM[9:2] (offset/len)	ROM[1:0] (node type)
0	SIB2	Offset=5 (to 5:SIB1)	101	00
1	SIB4	Offset=2 (to 3:SIB3)	10	00
2	R3	Length=32	100000	01
3	SIB3	Offset=2 (to 5:SIB1)	10	00
4	R2	Length=16	10000	01
5	SIB1	Offset=2 (to 7:END)	10	00
6	R1	Length=32	100000	01
7	END			11

2.4.2 IJTAG Network Status RAM Structure

Each RAM word has the following fields:

- [0] SIB S bit. Shows the state of S bit, i.e. if a SIB is closed (0) or opened (1)
- [1] SIB X bit. Shows the state of X bit (masking)
- [2] SIB C bit. Shows the state of C bit (error corrected)
- [3] SIB F bit. Shows the state of F bit (error detected)

2.4.3 Instrument Manager's FSM Part that handles Access Requests

The core logic of IM is implemented as an FSM with 15 states (see Figure 9). IM starts the operation in IDLE state and upon a request from the host it moves to INIT state. In INIT state it asserts the CaptureEn signal to copy the actual values of the network's nodes to the shift registers. It also resets the address counter since the first node for which the serial scan bits need to be generated is at address 0. Then it moves to HUB state which is a helper state to manage the node address handling. From there, depending on the node type located in ROM at the current address and the requested operation, it will proceed to SIB_F, REG_W, REG_R or UPDATE states. The first one handles SIBs by first shifting in unlock values to F and C bits, respective value for X taken from RAM and then finally the value for the S bit. Its value depends on

whether the SIB need to be opened or closed for the current retargeting goal (as described earlier).
For registers, IM will first shift in the first bit (IM treats the data from host in such a way that LSB is shifted in first) and if there are more bits, it will stay in the loop state while there are bits to shift in or out.
When the network node closest to network's SI is handled, FSM will reach an END entry in ROM. This means that it should assert Update signal and check whether additional Capture-Shift-Update operations are still needed, e.g. if the target register hasn't been reached yet or if the network must be closed after the access is finished. This is decided in the UPDATE state.

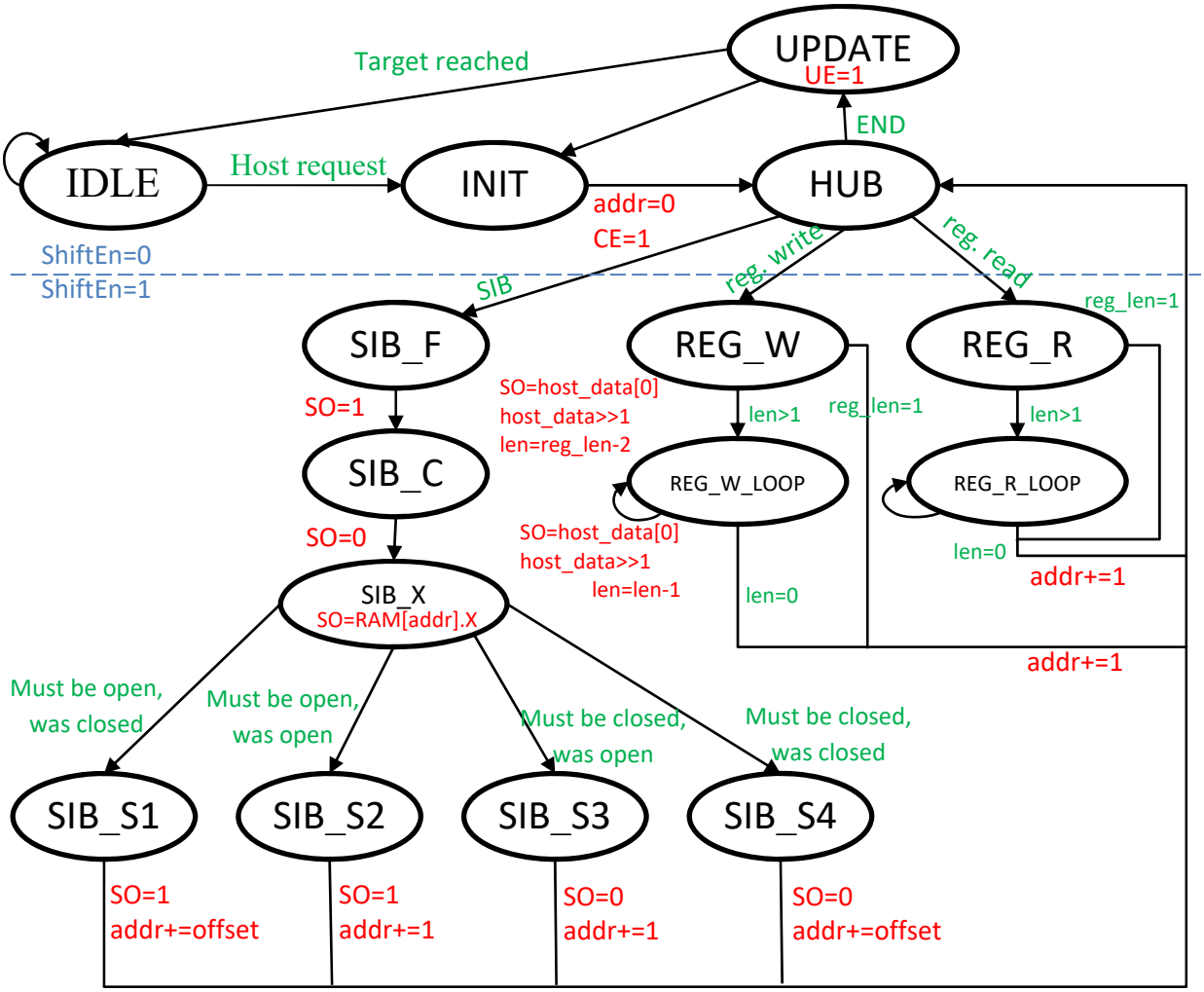


Figure 9. Instrument manager's FSM that handles external requests

2.5 FMI Operation during Fault Detection

2.5.1 Fault Localization and Diagnosis

Here, we detail the way the IEEE 1687 FMI detects and localizes a fault based on a comprehensive example.
First of all, let us define the following sets:

- Set of possible faults F, where fault $f_i \in F$

- Set of fault detection timestamps T, where timestamp $t_i \in T$ corresponds to fault f_i
- Set of fault localization latencies L, where latency $l_i \in L$ corresponds to fault f_i

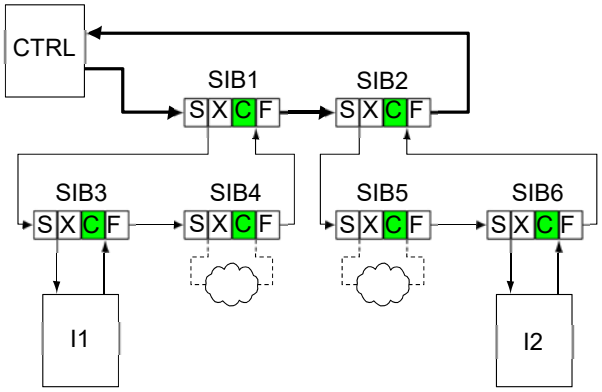


Figure 10. Example of IJTAG instrumentation network with asynchronous fault detection

When a fault occurs, the fault management system will start the localization procedure described in following sections. This will eventually result in identification of the instrument that detected the fault.

Let us now describe in detail the series of events that should follow the detection of a fault in an instrument using an example with small IJTAG instrumentation network. This network in normal state is shown in Figure 10 (omitting the display of SIB internals such as asynchronous signals and their propagation gates). In the following, we depict *FCX-SIBs* as blocks of four 1-bit scan registers (S, X, C, F) together with scan ports of a SIB (SI, SO, toSI and fromSO in Figure 6). Each scan register in a SIB has a binary value and it is shown graphically: register rectangle white fill if value is zero, color fill if value is one. The active scan chain is shown in bold while the inactive parts are shown in thin lines.

In our example, a fault is detected by instrument *I1*. The fault is not automatically corrected and thus needs to be taken care of by network controller. The initial state of the instrumentation network is with all SIBs closed (as in Figure 10).

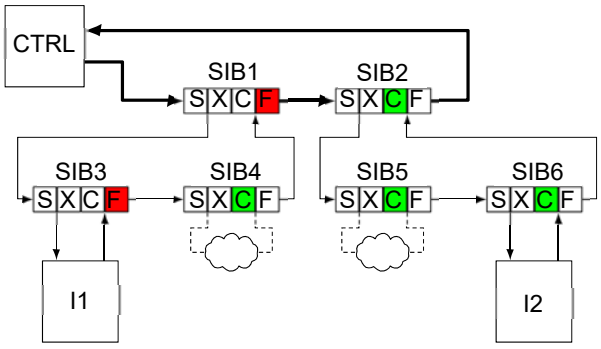


Figure 11. Detection of a single fault by instrument *I1*

2.5.1.1 Fault Detection

At the time of detection, instrument sets its Fault flag and clears Correction flag. These changes are propagated asynchronously to *SIB3* and then to *SIB1*, although *SIB4* is still in normal state. This situation is shown in Figure 11.

2.5.1.2 Fault Localization

When asynchronous fault detection signals reach the network controller, the latter responds by performing localization actions. Let us follow the sequence of events:

- $t = 0$: The network controller receives asynchronous signal about the fault in the network, it starts the localization procedure. It scans the top-level loop and sees Fault flag in *SIB1* (Figure 10) and recognizes that the fault location is somewhere in the child network segment(s) of *SIB1*. This operation takes $E + 4 \bullet b_1 = 6 + 4 \bullet 2 = 14[T_{TCK}]$, where b_1 is number of SIBs in the first level of hierarchy.
- $t = 14$: The controller now makes another shift to update the register of *SIB1* to open it. Since the active scan chain is not changed, it also takes 14 T_{TCK}.
- $t = 28$: The active scan chain now includes the child segment of *SIB1*. Controller makes another shift to find out which SIB asserted the Fault flag. This operation takes $E + 4 \bullet (b_1 + b_2) = 6 + 4 \bullet 4 = 22[T_{TCK}]$
- $t = 50$: The localization procedure is finished; the controller recognizes that the fault is situated in the child segment of *SIB3* which consists only of instrument *I1*.

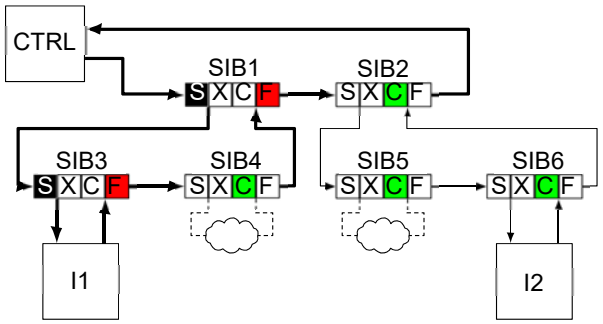


Figure 12. Localization of a single fault up to instrument *I1*

Then the controller will open *SIB3* and read out *I1* status. At this point the fault is localized and diagnosed. The state of the network is shown in Figure 12. Following actions may include writing to *I1* or masking out the fault by Mask bit in *SIB3*.

According to [8], the time of fault localization can be calculated by following formula:

$$t_n = (2 \bullet n - 1) \bullet E + 4 \sum_{i=1}^{n} \left(\left(2(n - i) + 1 \right) b_i \right)$$

where n is the level of hierarchy of the instrument, E is the constant of TAP diagram transitions (equal to 6 test clock cycles [1]) and b_i is the number of SIBs in i-th level of hierarchy. The units of the result of this equation are test clock cycles T_{TCK}.
In the case of currently discussed example, $n = 2$, $b_1 = 2$ and $b_2 = 2$. Hence, the required time to localize I1 is: $t_{I1} = 3E + 4(3 \bullet 2 + 2) = 18 + 32 = 50[T_{TCK}]$. This conforms with manual calculations.

2.5.2 Interruption of Ongoing Access in Event of Fault

The normal work of IJTAG instrument network consists of read and write accesses to instruments, opening and closing the SIBs which are the results of normal requests from higher levels of the system (like OS). Since the IJTAG network can be constructed hierarchically to

optimize access times, it may take considerable time to finish ongoing access and switch to another request.
Since asynchronous fault detection is designed to be as quick as possible to detect and find the source of fault, its speed would be useless if corresponding accesses to faulty re- source were delayed until current normal access is completely finished. Therefore, it is reasonable for IJTAG network with asynchronous fault detection to interrupt ongoing access and service faulty resource localization as soon as possible.
The localization of faulty resource would probably require modifying the configuration of IJTAG network, i.e. closing some SIBs and opening other SIBs. Hence, the current status of IJTAG network must be considered and an optimal modification, i.e. action that leads to quickest access to faulty resource, should be taken.
We foresee two possible solutions that can provide fastest reconfiguration of the IJTAG network in different situations:

- Dynamic retargeting
- Reset of IJTAG instrumentation network

2.5.2.1 Dynamic Retargeting

Let us consider a situation where required time to change the current configuration of instrumentation network to required one is less than the time required to start from reset state and open all levels of hierarchy that are needed to access the faulty resource. In this case it makes sense to halt the ongoing access and close corresponding SIBs, while opening the required SIBs. Since this should be done simultaneously, there must be means to dynamically schedule access requests with different priorities to IJTAG network. This can be done with dynamic retargeting techniques which are described in [16].

2.5.2.2 Instrument Network Reset

In case the reconfiguration from the current state of IJTAG network is too expensive in means of time, it is reasonable to perform a reset of IJTAG network configuration. It means that all SIBs are closed and the active scan chain corresponds to top-level loop of the network. After the reset, normal sequence of SIB openings and register accesses could be performed.

2.5.3 Multiple Fault Scenario

During the lifetime of the system it may happen that multiple faults occur at the same time, or at least, their localization times could overlap. In this case, it is important to make sure that fault management architecture would allow detecting both of them and reacting accordingly.
Whenever several faults occur at the same time or one fault occurs when another is already being localized, the overall status of asynchronous fault detection scheme remains adequate. Firstly, Fault and Correction flags at top level of IJTAG network will always indicate current status of fault detection. This means that if one fault is already addressed and the corresponding instrument Fault flag is cleared or masked, the top-level Fault flag will still indicate that the system is in the fault state. Similarly, if several faults occur at the same time and one of them is automatically corrected and Correction flag is set, while another fault is uncorrected, the top-level flags will indicate that the system is in a state of unhandled fault.
In the following, we will discuss how the fault management system with asynchronous fault detection deals with multiple faults occurring at the same or nearly the same time. Firstly, we will define the multiple fault events, then we will discuss several simple examples of two faults occurring simultaneously.

2.5.3.1 Multiple Simultaneous Faults

In general, we can describe an abstract situation, where two faults occur at the same time, or more precisely, the effects of these faults for fault management architecture (need to localize the fault) overlap in time. In other words, it is a situation, when some of the procedures required to localize these faults should be carried out simultaneously.

Let us consider two faults f_a and f_b , $t_a \leq t_b$. They are considered to happen simultaneously if their localization procedures overlap: $t_b < t_a + l_a$.

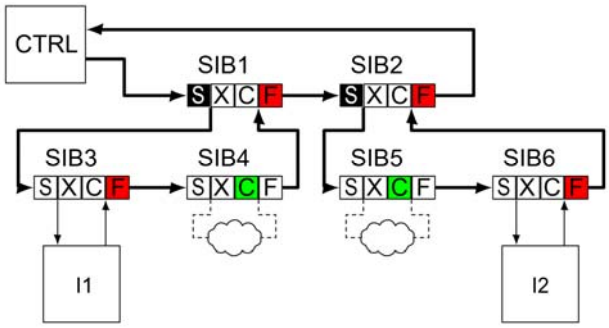


Figure 13. Example of two simultaneous faults

2.5.3.2 Examples with 2 faults

Let us describe the situation where two faults f_1 and f_2 are detected at nearly the same time. For this, let us take the same network as in example for single fault ($t_1 = 0$, section V) and make the instrument I_2 detect another fault f_2 in following cases:

- Case A: $t_2 = 0$. f_2 happens simultaneously with f_1
- Case B: $0 < t_2 \leq 14$: f_2 happens during the first shift in the top loop
- Case C: $14 < t_2 \leq 28$: f_2 happens during the shift that opens SIBs in the top loop
- Case D: $28 < t_2 \leq 50$: f_2 happens during the last shift that localizes the fault in I_1

1) Case A:

- $t = 0$: The same as before, but now controller understands that there is also fault somewhere in the child segment(s) of SIB2. Takes 14 T_{TCK}
- $t = 14$: The same as before, but now controller must also open SIB2. Takes 14 T_{TCK} .
- $t = 28$: Now child segments of both SIB1 and SIB2 are included (see Fig. 13), which makes scan chain longer. Controller should now scan the flags of all SIBs. This operation takes $E + 4 \cdot (b_1 + 2b_2) = 6 + 4 \cdot 6 = 30\ [T_{TCK}]$
- $t = 58$: Both f_1 and f_2 are localized
- $l_1 = 58, l_2 = 58$

2) Case B:

- $t = 0$: The same as before. Takes 14 T_{TCK}
- $t = 14$: The same as before, controller opens SIB1, but now recognizes that there is a fault under SIB2. Takes 14 T_{TCK} .
- $t = 28$: Now the SIB1 is opened and controller also writes 1 to SIB2 to open it. This operation takes $E + 4 \cdot (b_1 + b_2) = 6 + 4 \cdot 4 = 22[T_{TCK}]$.
- $t = 50$: f_1 is localized and now information from I_1 should be read. Since the length of registers of I_1 and exact amount of read/writes is not known, let us now make a simplification by not opening SIB3, but leaving it in the active scan chain. SIB2 is now

opened and its child segment should be scanned. This operation takes $E + 4 \cdot (b_1 + 2 \cdot b_2) = 6 + 4 \cdot 6 = 30[T_{TCK}]$

- $t = 80$: f_2 is now also localized.
- $l_1 = 50,\ 66 \leq l_2 \leq 79$

3) Case C:

- $t = 0$: The same as before. Takes 14 T_{TCK}
- $t = 14$: The same as before, controller opens SIB1. Takes
- 14 T_{TCK} .
- $t = 28$: Now the SIB1 is opened and controller recognizes
- that there is a fault somewhere under SIB2. This operation takes $E + 4 \cdot (b_1 + b_2) = 6 + 4 \cdot 4 = 22\ [T_{TCK}]$.
- $t = 50$: f_1 is localized and now information from I_1 should be read. Controller now writes 1 to SIB2. This operation takes 22 T_{TCK}
- $t = 72$: SIB2 is now opened and its child segment should be scanned. This operation takes $E + 4 \cdot (b_1 + 2 \cdot b_2) = 6 + 4 \cdot 6 = 30\ [T_{TCK}]$
- $t = 102$: f_2 is now also localized.
- $l_1 = 50,\ 74 \leq l_2 \leq 87$

4) Case D:

- $t = 0$: The sequence is the same as for single fault scenario
- $t = 50$: f_1 is localized, controller makes a shift to read out status of I_1 and finds out that there is a fault somewhere under SIB2. This operation takes 22 T_{TCK}
- $t = 72$: Controller writes 1 to SIB2, while SIB1 is still open. This operation takes 22 T_{TCK} .
- $t = 94$: SIB2 is now opened and its child segment should be scanned. This operation takes $E + 4 \cdot (b_1 + 2 \cdot b_2) = 6 + 4 \cdot 6 = 30\ [T_{TCK}]$
- $t = 124$: f_2 is now also localized.
- $l_1 = 50,\ 74 \leq l_2 \leq 95$

2.5.3.3 General Case

From the previous examples it can be seen that:

- Localization time of multiple faults will differ from single fault localization time and it will always be higher
- The main reason for increased localization time is the length of active scan chain
- The best case for two simultaneous faults is when they are detected at the same time
- The worst case for two simultaneous faults is when the second fault is detected when the path to first fault is already fully open

Also, although not described in previous cases, the relative position of the instruments that detected two simultaneous faults has obvious effect on localization time: while the best case of relative position would be instruments under adjacent SIBs, the worst case is when those instruments are situated in the opposite ends of the instrumentation network.

Overall, the worst case of two simultaneous faults localization time would be when the instruments that detect them are in totally different branches of instrumentation network and the second fault is detected when the first one is already localized. This will add considerable, however, constant overhead to each shift cycle of the second fault localization.

2.6 Section Summary

In this section we have discussed the details of asynchronous fault detection scheme for IEEE 1687 instrumentation network. This approach allows for quick detection and localization of faults that are indicated by embedded instruments by means of dedicated signals that are propagated asynchronously. We describe additional hardware needed for these functions and the impact it has on instrumentation network operation.

We also analyze possible scenarios of interruption by fault event and behavior of the system in multiple fault condition. This analysis shows that proposed asynchronous fault detection scheme is maintaining adequate status and allowing the system to cope with several faults that may occur simultaneously.

In case of two simultaneous faults, the fault localization time suffers from a penalty, but even in the worst case the system still remains adequate and scalable.

While this section details low-level fault localization and handling procedures, the high-level scenarios performed by the Operating System based on information delivered by IM and IJTAG instruments are described in deliverable D3.3.

3 Hierarchical Design and Test

3.1 Introduction

Previous work on hierarchical design and test mandates adding isolation logic for the respective core under test [17] [18]. This isolation logic makes it possible to test the functionality of the core, independently of the state the surrounding logic is currently in.

The isolation logic is typically placed close to the physical boundary of a core. It must not disturb the regular operation of a core. Thus in functional mode, it shall transparently pass I/O signals through. In isolation mode, however, it shall capture and/or launch values at its inputs and outputs, respectively. Usually the isolation logic or *core wrapper isolation* is implemented as a chain of *wrapper cells* sometimes referred to as wrapper boundary registers [19]. A chain of wrapper cells, called *wrapper chain* is similar to a regular scan chain. Yet the chain is separated from regular (core) scan chains.

Depending on the application scenario, wrapper chains may be implemented differently. One aspect influencing the wrapper's implementation is the type of core to be isolated. The internals of cores that are delivered as pre-layouted protected IP, so called hard cores [18], cannot be modified. Consequently, in case a core is not already prepared for hierarchical test, wrapper cells may only be added at the physical boundary of that core. That is the isolation has to be placed between the I/O ports and the core's logic. This design style typically adopts the IEEE 1500 [19] paradigm. Cores that are delivered as RTL or as a gate-level netlist (so called soft or firm cores [18]) may be modified when implementing hierarchical test. In particular this means that wrapper cells do not necessarily have to be placed at the core's boundary [20]. In addition to that, we may reuse existing flip-flops as wrapper cells to implement the core's logic isolation. Compared to dedicated wrapper cells, these so called shared wrapper cells have several advantages. In particular, they do not increase the overall area of the core, and have no detrimental impact on timing.

In the following we define general requirements for core wrapper isolation. We describe both dedicated and shared wrapper cells, and compare the two types of cells. Using the result of that comparison we show an improved wrapper cell. Then we combine this input to define an optimized wrapper implementation. It is specifically suitable for in-field test enabling independent test of the core while the surround logic may remain in the functional mode of operation. Finally we present experimental results on the insertion of this optimized wrapper implementation into industrial cores.

3.2 Wrapper Test Modes

As already mentioned, wrapper chains have to be transparent in the core's normal mode of operation. In addition to that at least two test modes have to be supported: A mode enabling the test of the logic situated in between different cores. Typically, that is the glue logic on top-level connecting different cores. Furthermore, a mode is needed that allows us to test the core's internal logic independently of the surrounding logic. Conventionally this means that the state of the surrounding logic must not have any influence on the test of core's internals. For in-field test applications, however, the surrounding logic may be in functional mode while the core is being tested. Consequently, we may want to have an isolation preventing the test of the core's internals from influencing the surrounding logic as well. This may either be implemented as part of the

internal test mode or as an extra test mode. This optional mode is often called safe mode. In the following we describe the modes of operation that may be supported by a wrapper chain.

3.2.1 Inactive or Functional Mode

In this mode the wrapper chain is inactive and transparently passes I/O signals through the wrapper cells. This is the behavior used in functional mode

3.2.2 Inward-facing Mode – INTEST

The *inward-facing* or *INTEST* mode is required to test the core in isolation of the surrounding logic. It is needed to enable hierarchical ATPG, and may also be used for testing a core by embedded instrumentation, e.g. LBIST. Furthermore it may also be extended to support in-field test (see Section 3.2.4).

In this mode input wrapper chains isolate the inputs going into the core from the surrounding logic, see Figure 14. Thus, the input wrapper chains prevent the surrounding logic from influencing the state of the core under test (illustrated by ✖ in the figure). Additionally, the input wrapper chain allows us to control the core's inputs directly, independently of the surrounding logic. Similarly, in INTEST mode output wrapper chains allow us to observe the core's responses. In summary this mode supports an isolated test of the core's logic using the wrapper chains and the core's internal scan chains. Any interface logic on both input and output side remains untested.

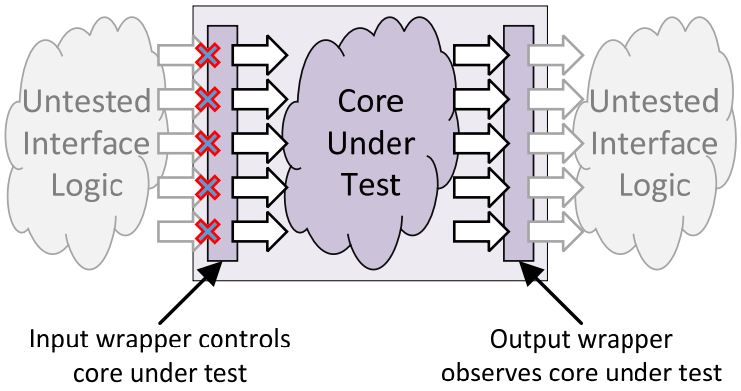


Figure 14: INTEST configuration of the core

3.2.3 Outward-facing Mode – EXTEST

The *outward-facing* or *EXTEST* mode is used to test the logic surrounding the core in isolation from the core itself. This includes any interface logic of the core, see Figure 15. Typically, this mode is required for hierarchical ATPG.

In outward-facing or EXTEST mode the responses of the surrounding logic that drives the inputs of the core are observed by the input wrapper chains. The output wrapper chain enables us to control the surrounding logic that is driven by the core's outputs independently of the core's internal logic. Thus, this wrapper chain is isolated from the core's internal logic (illustrated by ✖ in the figure). The core-internal logic is not tested in this mode.

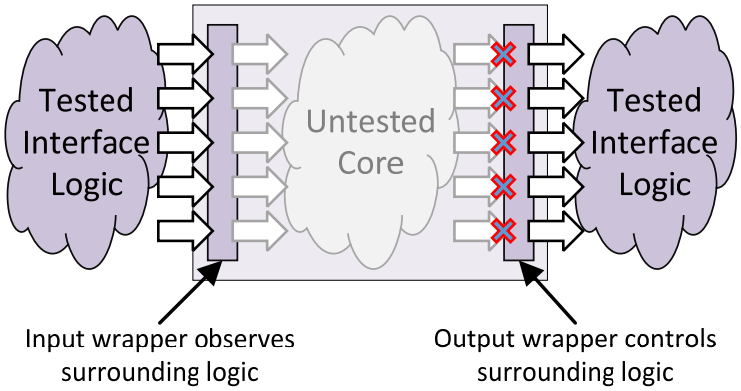


Figure 15: EXTEST configuration of the core

3.2.4 Safe Mode

The optional test mode extends the inward-facing test mode (INTEST) by using special wrapper cells in the output wrapper chain. These wrapper cells contain additional logic preventing data that is being captured in the cell from affecting the surrounding logic (output interface and top-level). This mode is crucial for in-field test applications in which individual cores are tested in isolation while the remaining circuit operates in the functional mode. The safe mode can be implemented as part of the regular inward-facing test mode (INTEST) or as a dedicated test mode.

Note that traditionally the safe mode applies to the outward-facing test mode (EXTEST). In this standard way of implementation data that is being captured by the input wrapper cells is blocked such that it does not affect the core's internal logic.

3.3 Wrapper Cells

Starting from the wrapper test modes introduced in the previous section, we can identify at least three different modes that are to be supported by each wrapper cell. First of all, it shall provide a transparent functional data path linking the core with the surrounding logic. Secondly each cell shall be able to capture response values from its input, and (optionally) isolate the cell's output. Finally, it shall be possible to drive a user selectable value at the output of the wrapper cell, and isolate the input of the wrapper cell. As already mentioned wrapper cells shall have a scan input and a scan output such that individual cells may be stitched to together like a scan chain. Through this scan chain drive values may be loaded into the wrapper cells, e.g. from a primary input. At the same time the scan chain transports observed values that are captured in the wrapper cells to e.g. a primary output. Additionally, a wrapper cell shall have minimal impact on timing and introduce as little logic (area) overhead as possible. In the following we will introduce both dedicated and shared wrapper cells that implement the three test modes.

3.3.1 Dedicated Wrapper Cells

A *dedicated wrapper cell* uses an additional flip-flop to provide controllability, observability, and shift capabilities. It can transparently pass the functional I/O signal through or can capture values at its input and/or launch values at its output. An example from IEEE 1500 standard [19] is depicted in Figure 16. The interface to the wrapper consists of five input and two output signals.

As can be seen, each dedicated wrapper cell adds one flip-flip and two multiplexers to the core. One of these multiplexers is introduced into the functional data path going from “cfi” to “cfo”. Consequently, dedicated wrapper cells have negative impact on both area and functional timing.

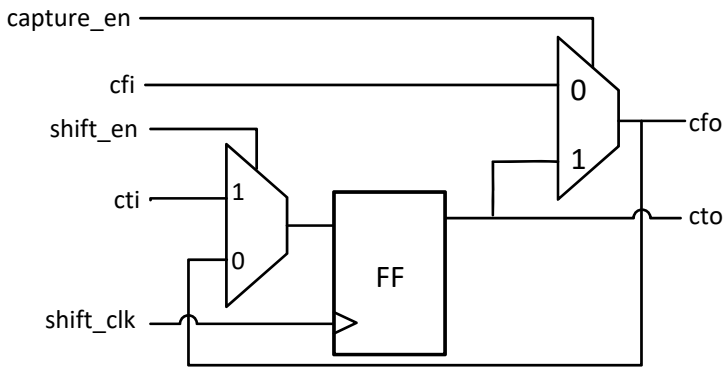


Figure 16: Dedicated wrapper cell

Core Test Input (cti)

This is the test input to the wrapper cell. If this is the first instance in the wrapper chain it will be connected to the source of the test data (e.g. a primary input). Otherwise it is driven by the output signal “cto” of the previous wrapper cell in the chain.

Core Test Output (cto)

This is the test output of the wrapper cell. If this is the last instance in the wrapper chain it is connected to the sink of the test data (e.g. a primary output). Otherwise it drives the input signal “cti” of the next wrapper cell in the chain.

Core Functional Input (cfi)

For input wrappers, this input is fed from the logic surrounding the core. For output wrappers, this input is directly driven by the core’s internal logic.

Core Functional Output (cfo)

For input wrappers, this output drives the core’s internal logic. For output wrappers, this output drives the logic surrounding the core.

Wrapper Clock (shift_clk)

This is usually driven by the test clock available in the core (e.g. “wrp_clock”). It clocks the flip-flops within the wrapper cell.

Shift Enable (shift_en)

This is the scan enable signal for wrapper cells. When the signal is high, the wrapper clock shifts data through the “cti” and “cto” scan data pins. When the signal is low, the wrapper clock captures the functional input value or holds the current state, depending on the value of the “capture_en” signal. The signal “shift_en” is controlled differently depending on the wrapper chain the cell is used in (input or output wrapper chain).

Capture Enable (capture_en)

This signal controls the captured data when the wrapper is not shifting. When the signal is low, the signal "shift_clk" captures the functional input value. When the signal is high, the signal "shift_clk" holds the current state of the wrapper instance. Similarly, it controls the value provided at "cfo". This signal is controlled by a logic created during wrapper insertion.

Safe-State Function (safe_ctrl) – optional

Wrapper instances with a safe-state function contain at least one additional input: "safe_ctrl". This signal is controlling a separate multiplexer at the "cfo" output of the wrapper instance to drive a static logic value. This value can either be provided by an additional input (e.g. "safe_value") or it can be generated internally. This mode is required to support in-field test.

3.3.2 Shared Wrapper Cells

A *shared wrapper cell* replaces an existing flip-flop by a special shared wrapper cell as depicted in Figure 17 (see [21]). This type of cell offers the same functionality as the dedicated wrapper cell introduced in Section 3.3.1. Compared to the latter type of cell, however, shared wrapper cells place the flip-flop into the functional path from "cfi" to "cfo".

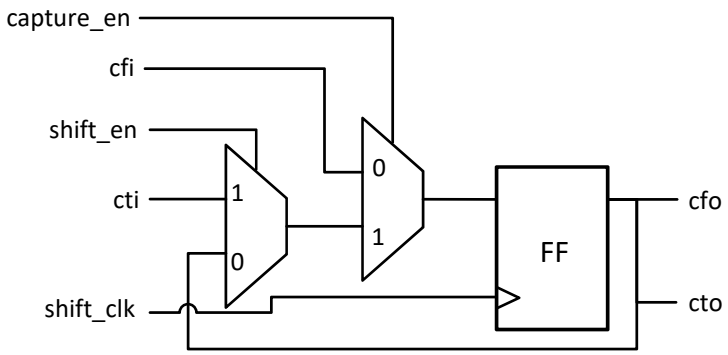


Figure 17: Shared wrapper cell

3.4 Optimized Wrapper Implementation

While the shared wrapper cell of Figure 17 partly addresses the area overhead by reusing existing flip-flops for the isolation wrapper, there is still the area penalty caused by the two additional test multiplexers. In the next sections an optimized wrapper implementation is described which further reduces the area overhead by using "normal" scan flip-flops for isolating the core. This approach has basically no area or timing impact compared to non-isolated scan test architectures.

In addition the next sections introduce a proposal how to combine the core isolation wrappers with on-chip test data compression and LBIST, respectively. Test data compression is typically required for reducing test time and tester memory requirements in production test. LBIST, on the other hand, enables in-field testing of the core's logic.

3.4.1 Optimized Shared Wrapper Cells

As described in Section 3.3.2 shared wrapper cells replace existing flip-flops by a special wrapper cell. Compared to dedicated wrappers cells as depicted in Figure 16 this avoids area overhead by reusing existing flip-flops. To reduce overhead of shared wrapper cells even further, we may exploit the fact that the flip-flop will typically be replaced by a regular scan flip-flop. Additionally we can drop the support of the state-holding mode that is not required in our application scenario. Thus, the only additional change that is necessary affects the scan-enable signal of the scan flip-flop. Compared to the wrapper cells of Section 3.3 it has to be controlled differently. This will be detailed in the following sections.

Figure 18 maps the signal names introduced in Section 3.3.1 to a regular scan-flip-flop (SDFF) as suggested in [19]. Note that since there is no state-holding mode the port "capture_en" can be omitted. In most cases this optimized implementation of a shared wrapper cell will have no impact on either area or timing, as the original flip-flop would be replaced by a scan flip-flip anyway. Only for the rare case in which a non-scan flip-flop is converted into a wrapper cell an additional multiplexer is added to the functional data path going from "cfi" to "cfo". The additional area used by the second multiplexer of the shared wrapper cell implementation of Figure 17 is avoided. In case the optional safe mode described in Section 3.2.4 is added to a wrapper cell, an additional logic is introduced into the functional data path. This is illustrated by the gray OR gate depicted in Figure 18. Furthermore, the additional pin "Safe_ctrl" is required to enable the safe mode.

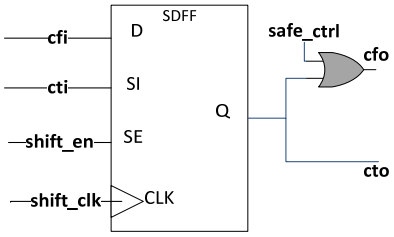


Figure 18: Optimized shared wrapper cell (optional support for safe mode depicted in gray)

3.4.2 Optimized Inward-facing Mode with Test Data Compression

Figure 19 shows an optimized architecture for the INTEST mode introduced in Section 3.2.2. This implementation being tailored to the optimized wrapper cell introduced in Section 3.4.1 removes a substantial part of the overhead involved with the wrapper cells introduced in Section 3.3. Additionally the implementation includes test data compression consisting of two modules "Test Decompressor" and "Test Compactor".

In order to enable the INTEST mode, the control signal "EXTEST_EN" will be set to a constant low value. In this mode the input wrapper chain is supposed to isolate the core form the surrounding logic. This is implemented by keeping the input wrapper chain in shift mode. Consequently, "Scan_EN (I)" is set to a constant high value. The output wrapper chain shall capture responses from the core's functional logic. Thus, its scan enable signal "Scan_EN (O)" is operated in sync with the scan enable signal of the core-internal scan chains. Note that both scan enable signals are included for illustration purposes only. In a typical implementation there will only be one scan enable signal driving going into the wrapped core. Core-internal signals

"Scan_EN (I)" and "Scan_EN (O)" will be derived from this scan enable and the setting of "EXTEST_EN".

In this architecture for INTEST, test data compression is applied to both the wrapper chains and the core-internal scan chains. Compressed test data is supplied from a test data source via the "Channel_in" port(s) to the module "Test Decompressor". There this data is decompressed, and is subsequently supplied to all wrapper chains and the core-internal chains. Responses captured by both types of chains are compacted by the "Test Compactor" module, and routed through the "Channel_out" port(s) to the test data sink. In production test both source and sink of the test data will typically be the ATE.

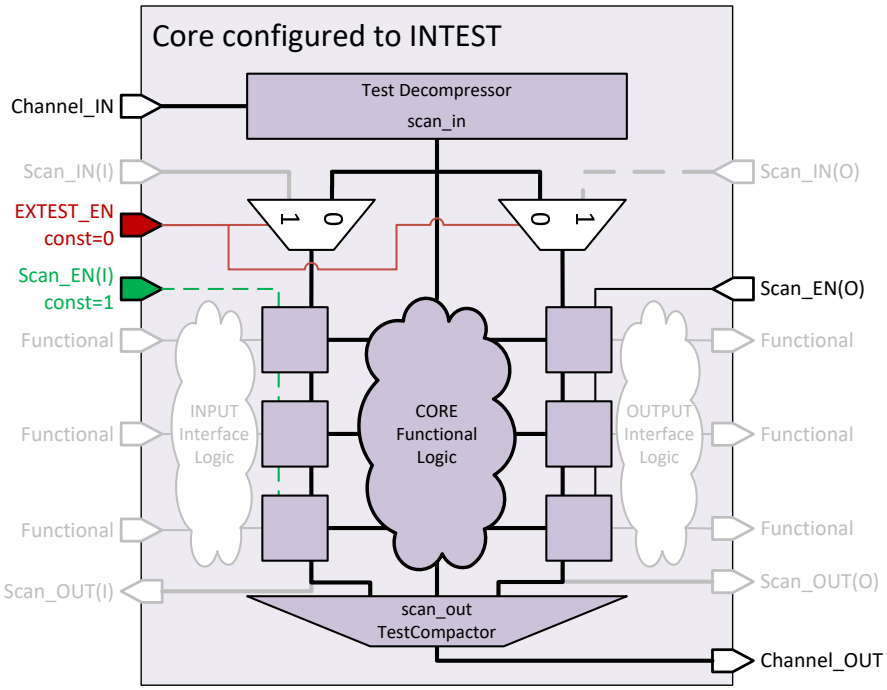


Figure 19: Detailed view of INTEST configuration including test data compression

3.4.3 Optimized Inward-facing Mode with Test Instrumentation

For in-field test the wrapped core may be supplemented by chip-internal test instrumentation. One example for this type of instrumentation is a LBIST module, see Figure 20. This implementation is based on the test architecture for the INTEST mode that is introduced in Section 3.4.2. Here pseudo random test patterns are generated by the "Pseudo Random Pattern Generator" in the upper part of the figure. Depending on the implementation the pattern generator may be shared with the "Test Decompressor" of the test compression hardware (see Figure 19). The test responses are compressed and analyzed by the "Response Analyzer" module shown at the bottom of Figure 20. This could be a MISR creating a signature that is compared to a golden reference signature at the end of the test run. Again some logic may be shared with the test data compactor. The overall LBIST execution is controlled by "LBIST CTRL". This block also contains registers to configure and control the LBIST operation. These registers are accessible via IJTAG. Note that the safe mode capability of the wrapper cells is enabled in the output wrapper chain (illustrated by ✖ in the figure). This prevents any test-related activity of the core-internal logic from disturbing the surrounding logic.

With the addition of the LBIST module the optimized wrapper architecture supports in-field test of the core. Due to the core's wrapper, the LBIST may be executed in isolation from the surrounding logic that may continue operating in the functional mode. Furthermore, the IJTAG interface enables the integration of the test instrument into an existing system health monitoring and error localization infrastructure.

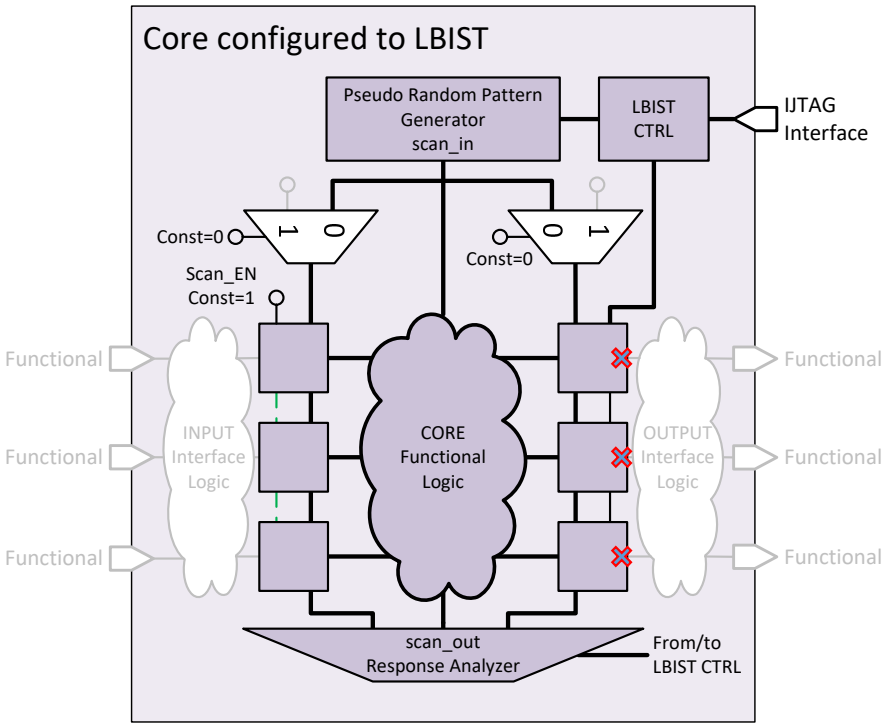


Figure 20: Detailed view of INTEST mode with on-chip instrumentation (only relevant pins are shown)

3.4.4 Optimized Outward-facing Mode

Figure 21 depicts the optimized test architecture introduced in Section 3.4.2 when configured in EXTEST mode. This mode is enabled by setting "EXTEST_EN" to a constant high value. In EXTEST the input wrapper chain is supposed to capture data from the surrounding (input interface) logic. The output wrapper chain, however, shall drive the surrounding (output interface) logic, that is it has to remain in shift mode. Compared to INTEST, the signals "Scan_EN (I)" and "Scan_EN (O)" swap their roles. Signal "Scan_EN (O)" remains constantly at high, while "Scan_EN (I)" is controlled in sync with the scan enable of the surrounding logic. As described for the optimized INTEST mode in Section 3.4.2, the control of both scan enable signals may be derived from a global scan enable and the setting of "EXTEST_EN".

In contrast to the INTEST mode the outward-facing test mode does not make use of the core-internal test data compression. Nevertheless, the wrapper chains may be connected to a test data compression module implemented at a higher level of hierarchy (e.g. the top-level). The input wrapper chain is observed through the "Scan_OUT (I)", while the output wrapper chain is controlled via "Scan_IN (O)". Depending on the top-level scan architecture two ports of each core may not be needed in this mode: "Scan_IN (I)" and "Scan_OUT (O)". To reduce top- level routing, the port "Scan_IN (I)" may be tied to a constant value, while the port "Scan_OUT (O)" can be left open.

When designing the test architecture of the INTEST mode we may optimize the number and lengths of the wrapper chains and the scan chains locally on core-level. This is not possible for the architecture of the EXTEST. Here the number of wrapper chains per core and consequently the length of each chain are dependent on the test architecture of the overall design. On top-level the wrapper chains of all cores and the scan chains running through the logic situated on top-level have to be stitched together. Only if the scan-chains on top-level do not exceed a certain pre-defined length and are well-balanced, test time targets can be met. To ease scan chain balancing on top-level, wrapper chains should be split into several smaller chains. Additionally, input and output wrapper cells may be mixed. At the same time, however, routing effort on top-level is increased if the number of wrapper chains per core is too large. This trade-off has to be considered carefully before deciding on the number of wrapper chains per core.

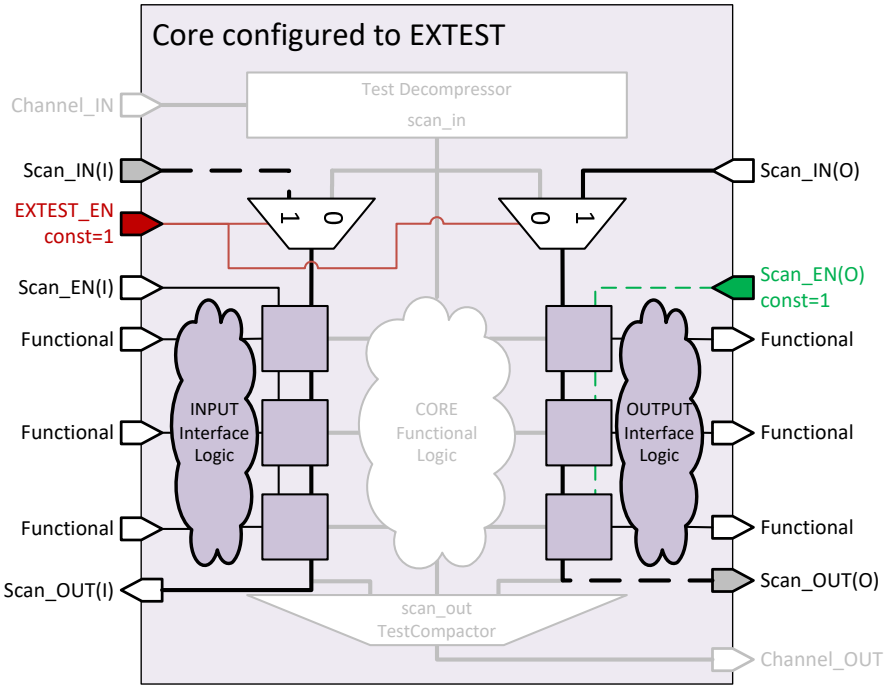


Figure 21: Detailed view of EXTEST configuration

As the core's internal logic will not be required for EXTEST, it can be abstracted and modelled as a so called *gray box*. This model only contains the logic that is sitting outside of the wrapper chains. That is the input interface logic between the input ports and the input wrapper chain(s) (including the ports and the wrapper cells), and the output interface logic between the output wrapper chain(s) and the ports (including wrapper cells and the ports). All purely combinational paths, i.e. connections from input to output through combinational logic only, are also part of the gray box. In EXTEST the ATPG tool may use the gray box instead of the core's full netlist to reduce runtime and memory usage.

3.4.5 IJTAG Interface of the Optimized Wrapper

When looking at a test scenario in which the test data is supplied from outside of the chip (e.g. ATE-based production test) IJTAG may help to substantially reduce test setup effort. In particular for a circuit containing complex cores and/or an intricate top-level test architecture there are numerous core configurations, clocking modes, and other test-related settings. A typical production test suite for this type of circuit will have to change these settings frequently. Using IJTAG this task becomes much easier and less error prone. For the actual test patterns and test

responses, however, IJTAG is typically not used. The high bandwidth requirements for this kind of data mandate a simple (parallel) pin-level interface.

Also for embedded instrumentation, IJTAG is an ideal solution. This is true for both production and in-field test scenarios. An LBIST controller, for example, may have many different settings and read-out values: start seed, number of patterns, masking, signatures, etc. This is best controlled via IJTAG.

In general a core wrapper isolation implemented using IEEE 1500 is compatible with IEEE 1687 (see [22]), i.e. it can be modeled using ICL and PDL. This is also true for the optimized wrapper implementation introduced in the previous sections. In particular this means that all signals required for wrapper configuration and test control are set via IJTAG. This includes the port "EXTEST_EN" described in Sections 3.4.2 and 3.4.4. On top of that for INTEST mode there may be configuration registers for the clock control and the test data compression module (not covered here). Furthermore the LBIST test instrument described in Section 3.4.3 is controlled via IJTAG. Thus, this architecture, that combines core wrapper isolation with on-chip test instrumentation, becomes a building block for a system health monitoring and error localization infrastructure.

3.5 Implementing Hierarchical Design and Test

For a hierarchical test approach, two test scenarios need to be considered: the top-level test and the test of the individual cores. The test of the cores employs the inward-facing mode (INTEST). Each test run may either be executed on-chip by embedded instrumentation or the test data may be supplied from an external ATE. In the latter case for each core test patterns for INTEST mode are generated upfront using ATPG. These core-internal test patterns can be retargeted to the top-level without being regenerated. This, however, is only possible for cores that are isolated properly, and that may be tested independently of the surrounding logic. Similarly, in an in-field test scenario, the core-internal test must not disturb the surrounding logic that may be operating in the functional mode. The test of the cores has to be scheduled for either sequential or (partly) parallel execution. Any interconnect between the cores and the top-level logic that is not part of a core is not covered by the core-internal test, and has to be tested separately. This top-level test requires the outward-facing mode (EXTEST) mode to be enabled in all involved cores. Again the top-level test can be performed by chip-embedded instrumentation or by using pre-generated ATPG test patterns.

In the following we will look into several implementation aspects of the core wrapper isolation. First of all we will review which ports of a core should not be isolated. Afterwards we will discuss the use of shared and dedicated wrapper cells, respectively. Finally we will present some experimental results on core wrapper insertion for two industrial cores.

3.5.1 Ports Not to be Isolated

Naturally the ports supplying the wrapper control signals to the core must not be isolated by a wrapper cell to give us full control over the wrapper's behavior. In addition to that further input and output ports of a core must not be isolated by a wrapper cell:

- Port connects to a functional clock
- Port connects to a test clock
- Port connects to an asynchronous SET
- Port connects to an asynchronous RESET
- Port connects to a scan input
- Port connects to a scan output
- Port connects to a wrapper signal (scan input, scan output, and other wrapper control)
- Port connects to a constant test signal
- Port is involved in pure combinational feedback path

3.5.2 Shared vs. Dedicated Wrapper Cells

For the isolation of an input or an output port of a core there exist two solutions: The first option is to add a dedicated flip-flop to the port used solely for isolation. Typically, this dedicated wrapper cell is inserted automatically by a tool. The second option is to reuse an existing flip-flop that is connected to the respective port. This shared wrapper cell reuses an existing functional flip-flop. Again identification and modification of these flip-flops is typically done by a tool.

In theory there is no limitation as to where dedicated wrapper cells are inserted. In practice, however, it is advisable to place the dedicated wrapper cell as closely to the core's respective input or output port as possible. This minimizes the amount of logic sitting outside of the wrapped part of the core allowing the core-internal test to cover the core in its entirety.

Two major problems exist when using dedicated wrapper cells. The first problem applies to designs using multiple clock domains. If the port to be isolated is driven by flip-flops residing in

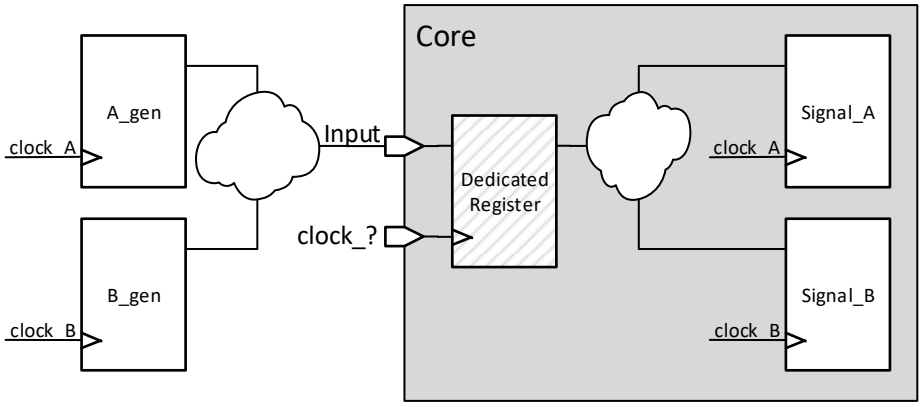


Figure 22: Dedicated wrapper cells vs. clock domains

different clock domains, there is no clear answer as to which clock signal should be driving the inserted wrapper cell. This is illustrated in Figure 22 where the input port "Input" that is to be isolated is affected by both clocks A and B. Typically in this scenario the clock driving the majority of the flip-flops in the vicinity of the port is chosen.

The second problem applies to at-speed testing. Consider again the input port “Input” in Figure 22 that is to be isolated by a dedicated wrapper cell. This port is part of a path going from flip-flops “A_gen” to “Signal_A”. In the functional mode of operation the inserted dedicated wrapper cell is transparent, i.e. the path is operating at speed. In test mode however, the path is interrupted by the flip-flop that is located within the wrapper cell (refer to Figure 16). Consequently, the path can only be tested in two segments. In the core’s EXTEST mode we cover the segment of the path going from “A_gen” to the wrapper cell. In the core’s INTEST mode we cover the segment of the path going from the wrapper cell to “Signal_A”. Thus due to the dedicated wrapper cell the full path can no longer be tested at-speed. The same problem applies to the path going from “B_gen” to “Signal_B”.

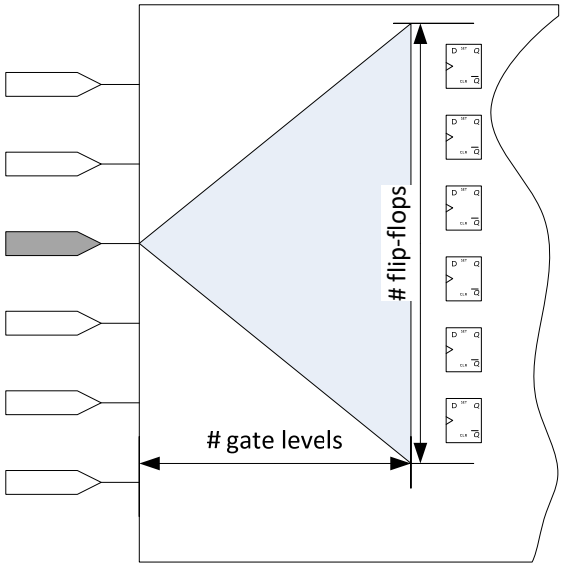


Figure 23: Criteria for inserting shared wrapper cells

Since shared wrapper cells reuse existing flip-flops these two problems do not apply. For shared cells, however, the crucial task is to first identify the best candidate flip-flops (see [20]). Typically, two criteria are used to guide the selection of functional flip-flops:

Depth The number of gate levels that have to be crossed when going from the port to be isolated to the first flip-flop, see Figure 23.

Width The number of flip-flops reached at that level.

In combination the parameters depth and width span the logic cone that may be reached from the port in question. This in turn gives an indication on how many flip-flops have to be converted into shared cells in order to isolate that logic and on how much logic would be placed outside of the isolation wrapper. As already mentioned it is desirable to keep the amount of non-isolated logic at a minimum. Yet, also the number of shared wrapper cells should be as low as possible to keep wrapper chains short (this translates into shorter test times).

Both shared and dedicated wrapper cells have their pros and cons, see Table 1. Therefore, there are three types of isolation strategies: i) use shared functional wrapper cells only, ii) use dedicated wrapper cells only, or iii) use “balanced mode” selecting the type of cell on a port-by-port basis. In any case a tradeoff decision between all three modes has to be made.

	Dedicated Cells	Shared Cells
Area and timing overhead	One scan flip-flop and at least one MUX gate are added for each port. At least one MUX is added in the functional path.	*Negligible area/timing overhead.*
Wrapper chain length	*Minimal*. Equal to the number of I/O ports included in the wrapper.	Chains length is *depending* on the fan-in/-out cones of ports and the amount of logic between the port and the wrapper cell.
INTEST (stuck-at coverage)	*All faults* can be tested in the core test (no logic exists between I/O port and wrapper cell).	*Interface* logic *cannot be tested*. This can be compensated in the EXTEST mode.
EXTEST (at-speed coverage)	Interface logic *cannot be tested* because the path will be split by the inserted wrapper cell.	*All faults* of the interface logic can be tested because functional paths will not be split.
EXTEST (size of the gray box)	*Smaller* gray box	*Bigger* gray box
Multi clock paths	*Impossible* to test.	Testability *not affected* by wrapper.

Table 1: Comparison between dedicated and shared wrapper cells

In summary the use of shared wrapper cells has many advantages and is the method of choice but might result in long wrapper chains. In case a port is not timing critical, it may be acceptable to insert a dedicated wrapper cell. This may result in a shorter wrapper chain.

3.5.3 Experimental Results

Core wrapper insertion using the optimized implementation of Section 3.4 was performed for two industrial soft cores using a commercial tool for DfT insertion. Industrial core A contains nearly 150,000 flip-flops; industrial core B contains nearly 70,000 flip-flops.

	Dedicated Cells Only	Shared Cells Only	Balanced Mode
Threshold (Width × Depth)	n/a	Max × Max	Med × Med
Total FF in wrapper chains	2,789	11,207	5,241
Number of dedicated wrapper cells	2,786	56	92
Length of input wrapper chain	1,231	4,395	645
Length of output wrapper chain	1,558	6,812	4,596

Table 2: Results of wrapper insertion for industrial core A

The experimental data for cores A and B are summarized in Table 2 and Table 3, respectively. Columns two to four show the results for the three isolation strategies as introduced in Section 3.5.2. When using the strategy "Dedicated Cells Only" no shared wrapper cells are allowed, i.e. the thresholds for width and depth are not used. For "Shared Cells Only" all flip-flops on the first register level should be included in the wrapper chain. Thus, the logic cone for each port (refer to Figure 23) should be as wide as necessary corresponding to both maximal width and depth. Finally, in "Balanced mode" we try to restrict the logic cone to a reasonable size, corresponding a medium setting for both width and depth.

	Dedicated Cells Only	Shared Cells Only	Balanced Mode
Threshold (Width × Depth)	n/a	Max × Max	Med × Med
Total FF in wrapper chains	2,183	17,380	3,633
Number of dedicated wrapper cells	1,971	167	797
Length of input wrapper chain	1,275	4,205	2,228
Length of output wrapper chain	908	13,175	1,405

Table 3: Results of wrapper insertion for industrial core B

The data in the tables shows that even for a wrapper configuration intended to use only shared wrapper cells, there are some dedicated cells inserted. This usually happens for core inputs in case the logic cone does not end only at flip-flops. Clock gating cells are a typical example. When the clock gate enable signal depends on a core input, a dedicated wrapper cell is required to ensure the controllability of the gated clock.

It can also be seen that the "Shared Cells Only" strategy can lead to a significant amount of design flip-flops to be part of the isolation wrapper (around 25% for the industrial core B, see Table 3). To reduce the wrapper chain lengths it can be beneficial to use at least some dedicated cells for the isolation as shown in the "Balanced Mode". Especially for industrial core A, see

Table 2, it can be seen that by adding only 36 dedicated wrapper cells the overall number of shared cells can be reduced by 5,966.

3.6 Section Summary

In this section the core wrapping requirements in terms of IEEE 1687 was discussed, as well as a match to IEEE 1500 and the requirements formed by the project BASTION. The need for the addition of so-called wrapper cells (placed around each core) was formulated, and their implementation was considered. Finally, a comparison between the implementation solutions was made on two industrial designs.

4 Solutions to test IJTAG networks

4.1 Introduction

When a device embeds an IJTAG network, the issue of how to test whether it is affected by any hardware defect arises. This issue is clearly of high practical importance, since a defect in the network may impair the possibility of correctly accessing the instruments it connects. A similar issue is commonly considered when performing a test on a generic Design for Testability structure before using it. For example, some works (e.g., [1] [23] [24]) faced the issue of testing the test circuitry mandated by the IEEE 1149.1 standard. The authors of [10] also propose a method to introduce fault tolerance in such a network.
Typical IEEE 1687 networks are chains of flip-flops interleaved with special modules (e.g., Segment Insertion Bits, or SIBs, and ScanMuxes), allowing to dynamically split the whole scan chain into segments that may be connected in series or in parallel, and to support a flexible access to the instruments. While the test of possible permanent faults affecting a standard scan chain can be easily done by resorting to well-known techniques (e.g., shifting into the chain a sequence of alternated 0s and 1s, and checking that the same sequence appears at the other extreme of the chain [1] [23] [24]), testing an IEEE 1687 network is more complex, as testing must also check whether the network can be properly configured and whether it works as expected after the configuration (i.e., whether the expected sub-network is made accessible), whichever legal configuration we enforce. Thus, each special module targeting the network configuration must be also tested.
Within the BASTION project some efforts have been done to devise an approach to test a sub-set of networks compliant with the IEEE 1687 standard with respect to permanent faults. This Section describes such an approach. For the most important components of an IEEE 1687 network (i.e., TDRs, SIBs, and ScanMuxes) we provide techniques for their test, and then we describe how to combine them into a single comprehensive test. This test is independent of the specific implementation of the network elements, and does not require any change in the hardware implementing the network. Once the sequence of operations required to test the network is known, the time required by the test can be computed using the methods described in [25]. Despite that the proposed strategy does not take into account all the possible constructs allowed by the standard, the main problem of testing such networks is tackled for the first time and can be easily extended to cover the missing cases.
Experimental results are reported on a set of representative benchmark cases, which practically demonstrate the correctness and feasibility of the above test approach, and provide an evaluation about the duration of the test.

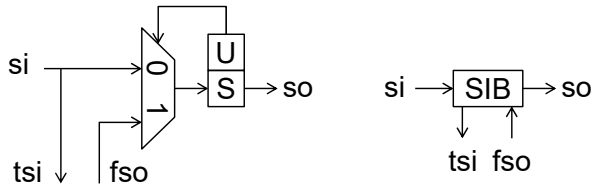
4.2 Related Work

Testing a regular (non-reconfigurable) scan chain for permanent faults can easily be performed by shifting a sequence of 0s and 1s through the scan chain. A reconfigurable scan chain, such as an IEEE 1687 network, is however far more complicated to test. When testing an IEEE 1687 network, one must not only test whether the flip-flops composing the TDRs can be correctly accessed, but also whether the modules introduced to support its reconfiguration (e.g., SIBs, multiplexers and flip-flops controlling them) work correctly.
In this section we will first briefly overview the key characteristics of an IEEE 1687 network, with special focus on the sub-set of structures considered in this work, and then explain why their test may turn into a complex task (especially if one wants to minimize the test time).

4.2.1 Overview of IEEE 1687 Networks

A key feature in IEEE 1687 networks is reconfigurability, i.e., the possibility to switch TDRs on and off the accessible scan path. Reconfiguration is done by incorporating programmable components into the network structure. One such programmable component is the SIB module which allows for bypassing a segment of a network. A segment can be simply one or several TDRs or a sub-network consisting of TDRs and other programmable components. Therefore, it is possible to create a hierarchical network with the use of SIBs.

Figure 24(a) shows a simplified schematic of a (possible implementation of a) SIB, which comprises a one-bit shift-update register and a two-input scan multiplexer (ScanMux). SIBs are programmed by shifting a bit into their S flip-flop and latching that bit into the parallel U latch. If the latched bit is 0, the SIB is de-asserted and the scan-path is from the si (ScanIn) terminal, to the so (ScanOut) terminal via the S flip-flop, bypassing the segment between the tsi (ToScanIn) and fso (FromScanOut) terminals. If, on the other hand, the latched bit is a 1, the SIB is asserted and the scan-path includes the segment connected between tsi and fso terminals of the SIB. In this section, the symbol shown in Figure 24(b) is used to represent a SIB.



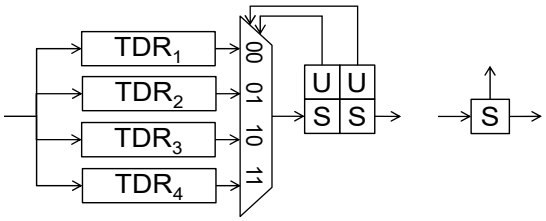
(a) Simplified schematic (b) Symbol

Figure 24: Segment Insertion Bit (SIB).

The reconfigurability in IEEE 1687 networks is not limited to the use of SIBs; others ad hoc reconfigurable networks can be constructed by the use of shift-update registers and ScanMuxes. As an example, consider the network shown in Figure 25(a) in which a two-bit shift-update register is used to select among four inputs of a 4-to-1 ScanMux. In a similar way as with SIBs, configuration of the ScanMux is performed by shifting the required values into the shift flip-flops of the control register (i.e., the S flip-flops) and latching the shifted bits into the parallel U latches. In the rest of this Section, the symbol shown in Figure 25(b) will be used to represent the shift-update register that controls a ScanMux.

To keep Figure 25 simple, the clock, reset, the control signals (namely, shift, update, and capture), and the select signal used to gate the control signals are not shown. To follow the examples in this work, it should suffice to assume that only the TDR connected to the selected port of a ScanMux receives (i.e., reacts to) the clock and control signals. It should be noted that the configuration of the network (i.e., the status of the latched bits) does not change when shifting a new vector through the shift cells, but only in the update phase where the shifted vector is latched into the U cells.

To operate an IEEE 1687 network from outside the chip, the TAP as defined by the IEEE 1149.1 (JTAG) standard can be used. The finite state machine (FSM) in the JTAG circuitry provides the control signals needed to configure IEEE 1687 networks and access the instruments through them.



(a) Using a ScanMux to select among four TDRs

(b) Symbol for control register

Figure 25: Construction of ad hoc reconfigurable networks supporting the mutually-exclusive parallel access to multiple instruments by the use of ScanMuxes and ScanMux control registers.

4.3 Motivations

When testing an IEEE 1687 network, one must not only test whether the flip-flops composing the TDRs can be correctly accessed, but also whether the configurable modules (e.g., SIBs, ScanMuxes and flip-flops controlling them) work correctly in whichever configuration they are forced. In order to better clarify the motivations for this work, let us consider a simple example, corresponding to a circuit which includes five instruments: the user can access them through the TAP port, reading or writing from/to the associated Test Data Registers (TDR1 to TDR5). In order to save time when accessing to the instruments, the designer may decide to adopt an IEEE 1687 network including three SIBs and one ScanMux, as shown in Figure 26; each of these four configuration modules can be configured to allow the access to a given subset of TDRs (and the associated instruments). Figure 27 reports the eight possible configurations supported by this network, which depend on how the SIBs and the ScanMux have been configured. In Figure 27, "A" means asserted, "D" means de-asserted, 0 and 1 correspond to the two possible positions of the ScanMux, and "-" appears when a module belongs to an inaccessible segment.

When facing the test of an IEEE 1687 network we should check whether any fault affects not only flip-flops of TDRs, but also SIBs and ScanMuxes. This means checking whether SIBs and ScanMuxes can be properly configured and work accordingly. Moreover, the adopted solution should guarantee that the required test time is minimized.

In order to achieve this goal, the BASTION partners developed an approach in which the test is organized in *sessions*: in each session we first configure the network (so that each SIB and each ScanMux is switched into a given position), and then check whether the expected path has been inserted between TDI and TDO, i.e., whether the right instruments can be accessed. Since the number of possible configurations of a network grows exponentially with the number of configurable modules, the problem of identifying a sequence of sessions which guarantees that 1) all the configurations modules and TDRs are fully tested, and 2) the total test duration is minimized, is not trivial. Coming back to the example of Figure 26, this means identifying the sequence of configurations (out of the 8 possible ones) that matches the two above goals.

This section first describes the constraints that must be fulfilled by the sequence of sessions to guarantee the full test of each TDR and configuration element, and then describes a heuristic algorithm for selecting a sequence of sessions producing a minimal duration test.

4.4 BASTION Contributions

In this sub-section we describe the approach developed by BASTION partners to test a sub-set of IEEE 1687 networks for permanent defects. More in details, in this sub-section we

describe the testing of TDRs, SIBs and parallel structures, as well as the test time calculation and the optimization of the test sequence.
Testing if such an IEEE 1687 network is affected by permanent defects requires testing two different sets of components:

- the flip-flops composing the TDRs the network makes accessible
- the modules allowing the network to dynamically reconfigure (e.g., the SIBs and the ScanMuxes), which are referred to as configurable modules.

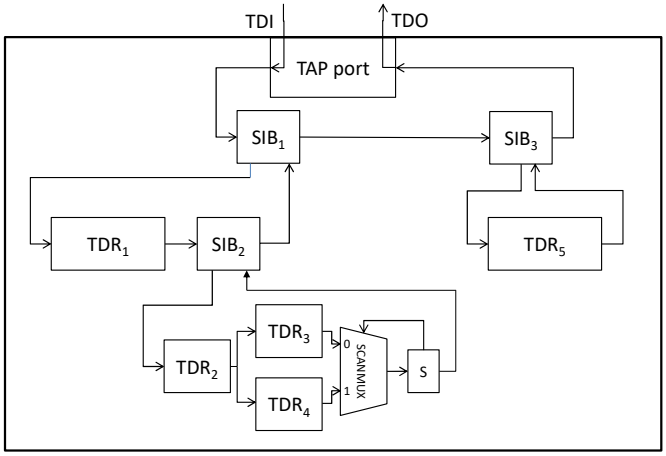


Figure 26: Example of 1687 network.

	SIB_1	SIB_2	SIB_3	ScanMux	Accessed TDRs
C0	D	-	D	-	-
C1	D	-	A	-	TDR_5
C2	A	D	D	-	TDR_1
C3	A	A	D	0	TDR_1, TDR_2, TDR_3
C4	A	A	D	1	TDR_1, TDR_2, TDR_4
C5	A	D	A	-	TDR_1, TDR_5
C6	A	A	A	0	TDR_1, TDR_2, TDR_3,TDR_5
C7	A	A	A	1	TDR_1, TDR_2, TDR_4,TDR_5

Figure 27: Set of possible configurations for the network in Figure 26.

Please note that we do not consider here the issue of testing the instruments connected to the TDRs, nor the connections between each instrument and the corresponding TDR. This task is typically performed resorting to ad hoc solutions which strongly depend on the kind of instrument and on possible solutions adopted at the system level (e.g., resorting to a loopback connection for test purposes). Therefore, we disregard whether the generic TDR is a Read-Only, Write-Only, or Read-Write TDR.
We also assume that the TAP controller works properly. An effective algorithm to detect possible permanent faults affecting the TAP controller is described in [27].

4.4.1 Test of the TDRs

The test of the flip-flops in a TDR can be performed by first picking a TDR, then configuring the network such that it is made accessible, and finally applying to the network input a sequence of bits able to test it.
More in details, once the target TDR has been identified, the following procedure performs the test:

- Configure the network so that the target TDR can be accessed
- Shift in a suitable sequence
- Check that the same sequence appears on the TDO signal.

The sequence to be used in step 2 depends on the kind of defects to be tested, following the techniques described in [1] [23] [24]: for those that can be modeled as stuck-at faults, a sequence composed of alternated 0s and 1s is fine.

4.4.2 Test of the SIBs

Our goal is to check whether a given SIB works correctly, or it is affected by any permanent fault (such as a stuck-at fault). The idea is to first configure the network in such a way that the SIB is asserted, checking then whether it works correctly; secondly, the network is configured in such a way that the SIB is de-asserted, checking whether it also works correctly in this new configuration. The check of the SIB behavior (to be performed after the first and the second configuration) can be performed in two steps. The first step forces the scan path from the configuration to a known status by shifting into the network a sequence composed of a number of 0s. Since a fault in a configurable module may change the selected path into another, the number of 0s shifted in the network in order to be sure that the path is forced to all 0s must be equal to the length of the longest path in the network. In the second step, a sequence composed of alternated 0s and 1s is shifted into the network. This same sequence should appear on the end of the path (i.e., on TDO) after a number of clock cycles equal to the path length. If this happens, it means that a path having the expected length exists between TDI and TDO; hence, the SIB is working correctly. The procedure is repeated but for the second path, where the SIB is de-asserted.
The test procedure is composed of the following steps:

1. Configure the network so that a first path is selected, where the SIB is asserted
2. Shift in a sequence of 0s into the network. The length of the sequence is equal to the length of the longest path
3. Shift in a sequence of alternated 0s and 1s, and check whether the correct sequence comes out of the network unchanged and starting at the due time; in this way we check that the SIB behavior is the expected one (i.e., the chain includes the corresponding segment)
4. Configure the network so that a second path is selected, where the SIB is de-asserted
5. Shift in a sequence of 0s into the network. The length of the sequence is equal to the length of the longest path
6. Shift in a suitable sequence of bits in the network, and check whether the correct sequence comes out of the network starting at the due time; in this way we check that the SIB behavior is the expected one (i.e., the segment corresponding to the SIB is bypassed).

In principle, the above procedure must be repeated once for every SIB. In practice, the test can be organized in a number of sessions, each corresponding to a configuration phase, in which a given scan path is connected between TDI and TDO, and a test phase, in which we

check whether all SIBs in the path work as expected. The set of sessions composing the test should be selected so that each SIB is at least once asserted and once de-asserted. In a network with SIBs, this constraint also guarantees that all TDRs are accessed, as required by the above procedure. Hence, by testing the SIBs we also test the TDRs.
As an example, let us consider the simple network shown in the left of Figure 28. Let us assume that TDR1 is composed of three bits, and TDR2 of four. A possible procedure for testing the network could be:

1. Configure the network so that SIB1 is asserted and SIB2 is de-asserted; hence, only TDR1 is accessed; the length of the path is five (three flip-flops for TDR1, one for each SIB)
2. Shift in a sequence of nine 0s (nine is the length of the longest path)
3. Shift in a sequence composed of alternated 1s and 0s from TDI; the same sequence should start appearing on TDO after five clock cycles
4. Configure the network so that SIB1 is de-asserted and SIB2 is asserted; hence, only TDR2 is accessed; the length of the path is six (four flip-flops for TDR2, one for each SIB)
5. Shift in a sequence of nine 0s
6. Shift in a sequence composed of alternated 1s and 0s from TDI; the same sequence should start appearing on TDO after six clock cycles.

Any fault affecting one of the two SIBs (e.g., forcing it to connect the wrong output to the input) can be detected through the above technique by looking at when the alternated sequence appears on TDO.
The reader should note that, due to the hierarchical structure of the generic IEEE 1687 network, selecting a set of configurations which allows each SIB to be asserted and de-asserted at least once requires that every subnetwork is accessed at least once. As an example, let us consider the network in the right of Figure 28. This network can be configured in six possible ways (shown in Figure 29), corresponding to the possible paths between TDI and TDO. “A” means asserted, “D” means de-asserted, and “-” appears when a SIB belongs to an inaccessible segment. For each configuration the set of accessed TDRs is also shown. A possible subset of configurations that fully tests the three SIBs is C1, C4, and C5. Hence, we can test all SIBs in the network with only three sessions, each corresponding to one of the three identified configurations.
The reader should also note that the time required by each session depends on the length of the TDRs lying along the path it activates; moreover, the time to configure the network so that a given path is activated depends on the previous configuration. Hence, the selection of the sequence of configurations allowing to test all TDRs and SIBs in a network with minimum test duration may turn into a rather complex task. For the purpose of this work we adopted a heuristic algorithm to solve this task, which will be described in the next sub-section.

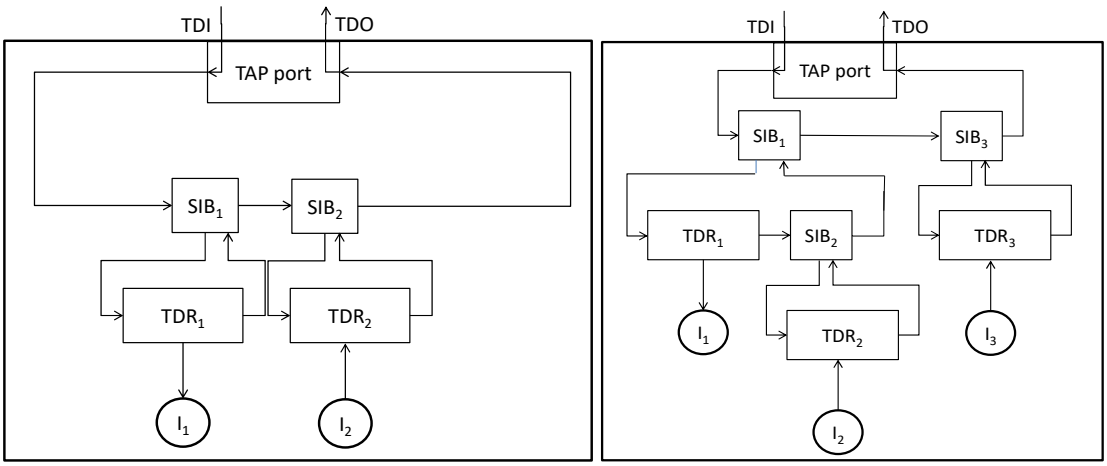


Figure 28: Example network #1 (left) and #2 (right).

	SIB_1	SIB_2	SIB_3	Accessed TDR_s
C0	D	-	A	TDR_3
C1	D	-	D	-
C2	A	A	D	TDR_1 TDR_2
C3	A	D	D	TDR_1
C4	A	A	A	TDR_1 TDR_2 TDR_3
C5	A	D	A	TDR_1 TDR_3

Figure 29: Set of possible configurations for network #2.

4.4.3 Testing the ScanMuxes

Testing the network also requires checking the correct behavior of structures intended to support the mutually-exclusive parallel access to different instruments, following the scheme of Figure 25. This means testing both the flip-flops storing the values of the ScanMux control bits and the associated multiplexer.

For this purpose we need first to test the flip-flops. Secondly, we need to test the ScanMux. To do so we can exploit the results of [28], where it was demonstrated that a given set of *2n* input vectors is able to fully test any *n-to-1* (i.e., *n* inputs, and *1* output) multiplexer (no matter its implementation) against any static fault.

The basic idea behind the test algorithm we propose is once again to first configure the network so that the ScanMux is switched to a given configuration, thus making a given path accessible. Secondly, a sequence of 0s is flushed into the network. Finally, a sequence of alternated 0s and 1s in shifted in the path, checking whether it emerges unchanged from TDO l clock cycles later, l being the length of the path. The procedure is repeated for every possible configuration of the ScanMux.

In the simple case of a module allowing to access two instruments TDR1 and TDR2 that are placed in parallel (as in Figure 30), the test procedure requires the following steps:

1. Configure the network so that the ScanMux makes TDR1 accessible; this means that a given path including TDR1 is introduced between TDI and TDO
2. Shift in the network a sequence composed of as many 0s, as the length of the longest path
3. Shift into the network a sequence of alternated 0s and 1s, checking that the ScanMux behavior is the expected one (i.e., the expected values emerge from TDO at the expected time)

4. Configure the network so that a new path is selected, in which the ScanMux makes TDR2 accessible
5. Shift in the network a sequence composed of as many 0s, as the length of the longest path
6. Shift in the network a sequence of alternated 0s and 1s, checking that the ScanMux behavior is the expected one (i.e., the expected values emerge from TDO at the expected time).

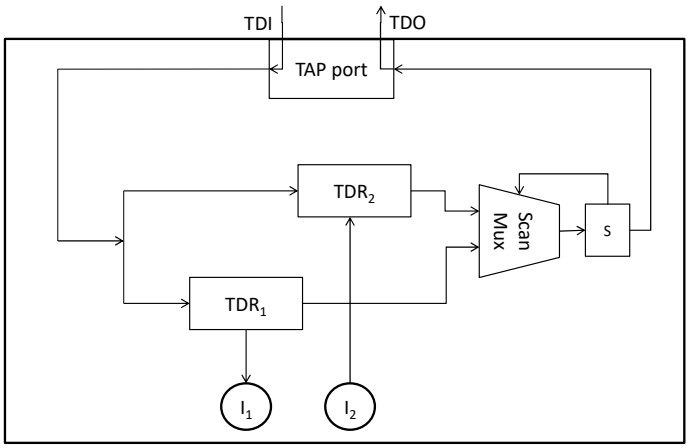


Figure 30: Example network #3.

In general, if the ScanMux connects m instruments, we will need to configure the network m times, each time switching the ScanMux to a different position, shifting in first the sequence of all 0s, and then the one of alternated 0s and 1s, and checking that the expected sequence emerges on TDO at the expected time. Although the detailed proof cannot be reported here for the lack of space, the above procedure guarantees that the input values required to fully test a multiplexer (as specified in [28]) are all applied, so that any possible static fault affecting the multiplexer is detected by the sequence.
The reader should note that if TDR1 and TDR2 have the same length, the above solution (nor anyone else) does not allow to test the multiplexer, because in this case there will be no way to check whether we are accessing the right TDR at any time.

4.4.4 Overall Test Strategy

Based on the above observations, the test of an IEEE 1687 network is composed of a number of sessions. Each session is composed of two phases:

- a configuration phase, in which control bits are shifted in to assert and de-assert each SIB and to load suitable values into the flip-flops controlling the ScanMuxes, thus selecting a path composed of a certain subset of TDRs
- a test phase, in which a suitable sequence composed of all 0s is first shifted in the selected path, followed by a sequence composed of alternated 1s and 0s; the first sequence is composed of as many bits, as the length L of the longest path; the second sequence is composed of l bits, l being the length of the selected path. While shifting in the second sequence, we should observe the values coming out of TDO and observe l bits equal to 0. By shifting the path for two more clock cycles we should observe a 1 and a 0. If this is the case we can assume that the right path has been selected and the TDRs composing it work correctly.

We denote by *tci* the duration (in clock cycles) of the configuration phase and by *tti* the time for the test phase for test session i.

The configuration time (*tci*) depends on the previous configuration. For example, let us assume that SIBi belongs to the segment controlled by the SIBj. If SIBi must be configured as asserted in the new configuration, and was not accessible in the previous one (since SIBj was de-asserted), we must first configure the network so that SIBi can be accessed (hence, SIBj must be asserted), and then access SIBi and configure it as required. Hence, at least two vectors must be shifted in for this purpose in the network. In the worst case, to configure a network for a given session we will require a number of vectors equal to the maximum depth of the network. Each vector requires the time for being shifted in (this time is denoted as SIB programming overhead in [25]), plus a few clock cycles (the exact number is implementation dependent) to capture it into the flip-flops of the corresponding path (JTAG protocol overhead in [25]).
On the other side, the duration of the test phase (*tti*) depends on the length l of the path selected by the configuration, and on the length L of the longest path in the network. The exact duration of the test phase is equal to $L+l+2$. In fact, for every configuration we need to

- fill the scan path with all 0s: since in the worst case a fault may turn the path into the longest one, this step requires L clock cycles
- fill the scan path with a sequence of alternated 0s and 1s; this step requires l clock cycles
- shift out the content of the flip-flops in the path, checking when the first two bits emerge from TDO: this step requires 2 clock cycles.

The total duration of the test of a network composed of N sessions is thus given by

$$T = T_c + T_t = \sum_{i=0}^{N-1} t_i^c + \sum_{i=0}^{N-1} t_i^t$$

where *Tc* is the sum of the configuration times and *Tt* is the sum of the test times. Each session activates a different path connecting TDI and TDO. The selection of the sequence of possible paths corresponding to the sessions should be performed in such a way that

- each TDR is accessed at least once;
- each SIB is at least once asserted and once de-asserted;
- each ScanMux assumes each possible configuration.

Different solutions can be followed for identifying the best sequence out of the total set of possible ones. Out of those matching the above constraints, the one requiring the minimal test time should be selected. The following sub-section describes a heuristic algorithm for selecting an optimized sequence with respect to test time.

4.4.5 Identification of an Optimized Sequence of Sessions

We now describe a heuristic method to select an optimized sequence of sessions to test an IEEE 1687 network, i.e., one corresponding to a minimized test time.
We first propose a representation of the network as a directed graph (denoted as Network Graph) whose vertices belong to three categories, corresponding to the elements of a generic network, and their interconnections:

- TDR: a vertex of this type has an incoming arc coming from the element feeding the TDR, and an outgoing arc going to the element it feeds
- SIB: a vertex of this type has two incoming arcs, one coming from the element feeding the SIB (*si*) and the other from the end of the segment it controls (*fso*), and two outgoing arcs, one (labeled as *A*) going to the first element of the segment

controlled by the SIB (*tsi*), and the other (labeled as *D*) going to the following element in the same segment of the SIB (*so*)

- ScanMux: a vertex of this type has an incoming arc coming from the element feeding the segments that can be accessed in parallel based on the position of the ScanMux, and as many outgoing arcs as the number of segments entering the ScanMux, each labeled with the corresponding value of the controlling signal of the ScanMux.

Additionally, a TDI and a TDO vertex exist, feeding the first element in the scan path and fed by the last one, respectively.

To provide the reader with an example, the Network Graph corresponding to the network in Figure 26 is reported in Figure 31.

The method we propose to identify the optimized sequence of sessions is based on performing a depth-first visit of the Network Graph starting from TDI. For each SIB vertex, first the child labeled with A is visited, and then the other. The order of visit for the children of the ScanMux vertices is irrelevant. Each path from TDI to TDO within the Network Graph corresponds to a possible configuration (i.e., scan path) within the network.

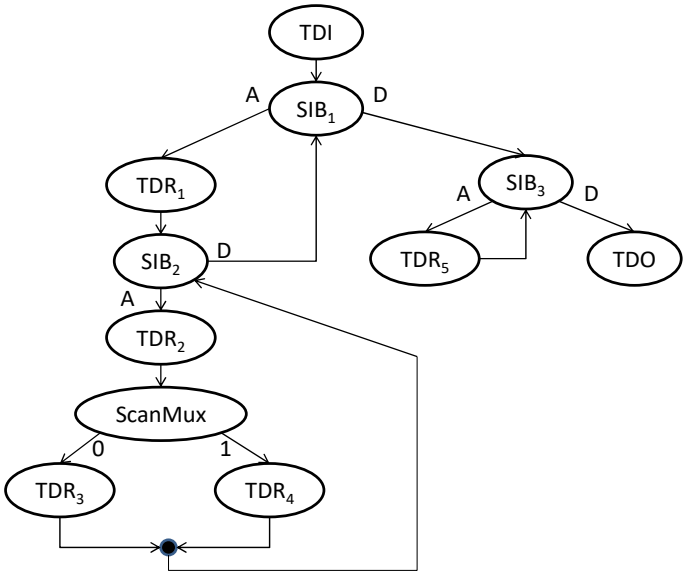


Figure 31: Network Graph for the network of Figure 26.

Each time the algorithm generates a new path (i.e., the TDO vertex is reached), the configuration assumed by each configurable module is recorded, and the corresponding session is added to the test. As soon as all the configurable modules have taken all the required configurations, the algorithm stops. The proposed algorithm guarantees that the identified sequence of sessions fully tests all TDRs, SIBs and ScanMuxes, according to the rules introduced in the previous sub-sections.

4.5 Experimental Results

To experimentally verify the correctness of the above algorithms and to evaluate the key parameters of the resulting test sessions, we wrote a prototypical tool implementing the proposed techniques. The tool is written in C# and amounts to about 700 lines of code.

We considered as a first set of benchmarks the same IEEE 1687 networks used in [29] and a few others generated with the same strategy, starting from the ITC02 benchmark SoCs [31]; these networks only contain SIBs. Their characteristics are summarized in Table 4.

By running the tool on the selected benchmark networks we got the results reported in Table 5. For our computations we assumed that the number of clock cycles required for vector application (which depends on the specific implementation) is equal to 5. For each network we identified the sequence of sessions required to test it according to the algorithm in the previous sub-section. In Table 5 we reported the sum of the configuration time (T_c), the test time (T_t) and the total time (T) for each network (in terms of number of clock cycles). For comparison, we also reported (Column *D-first*) the total time for a similar algorithm, in which for every SIB the child labeled with *D* is visited first (instead of the one labeled with *A*). The computation time has always been negligible (less than 1s) for all considered networks. As the reader can notice, the proposed algorithm is always able to produce a shorter test time than the D-first one; the difference is larger when the depth is higher and may achieve 15% for some of the considered networks.
As a second set of benchmarks, we randomly generated a number of networks including both SIBs and ScanMuxes.

Table 6 reports the characteristics of this new set. We then ran our tool on each network, and got the results of Table 7.
By looking at the results, the reader can note first that the number of sessions identified by the heuristic algorithm introduced in the previous sub-section is equal to *d+1* (*d* being the depth of the network) for the networks of the first set, while the number of sessions may be slightly higher when ScanMuxes exist in the network (as in the second set). Moreover, depending on the average length of the TDRs in each segment, the major component of the test time may correspond to the Configuration time or the Test time. Also for the second set of networks the proposed algorithm always outperforms the D-first one used as a reference, showing that a clever choice of the sequence of sessions may reduce (sometimes significantly) the configuration time, and thus the total test time.

Table 4: Benchmark networks – first set

	#SIBs	#TDRs	Depth
d695	147	147	1
p22810	30	28	2
p34392	22	19	2
p93791	49	32	3
a586710	6	5	2

Table 5: Test duration – first set of networks

	#Sessions	T_c [#cc]	T_t [#cc]	Total [#cc]	D-first [#cc]
d695	2	8,533	25,299	33,832	33,979
p22810	3	31,343	93,575	124,918	125,142
p34392	3	23,630	70,720	94,350	94,432
p93791	4	97,972	489,428	587,400	587,505
a586710	3	41,998	167,935	209,933	209,949

Table 6: Benchmark networks – second set

	#SIBs	#MUXs	#TDRs	Depth
N100D2	31	37	63	3
N132D4	39	40	92	5
N17D3	7	8	11	3
N49D0	16	18	31	1
N61D2	11	21	40	2
N88D8	32	32	56	4

Table 7: Test duration – second set of networks

	#Sessions	T_c [#cc]	T_t [#cc]	Total [#cc]	D-first [#cc]
N100D2	4	4,336	11,696	16,032	18,525
N132D4	6	7,781	23,153	30,934	36,150
N17D3	5	950	2,845	3,795	4,363
N49D0	2	1,417	3,329	4,746	5,204
N61D2	4	3,207	7,771	10,978	13,041
N88D8	5	4,188	12,408	16,596	19,137

4.6 Section Summary

The increasing adoption of the recent IEEE 1687 standard raises the issue of testing whether any permanent fault (e.g., stuck-at) affects the configurable scan chain mandated by the standard. Since in this scenario the scan path is configurable, testing an IEEE 1687 network requires testing also the configurable modules (e.g., SIBs and ScanMuxes) it includes. This goal can be achieved by organizing the test in a sequence of sessions, each configuring the network so that a specific path lies between TDI and TDO, and then checking whether the expected path can be accessed. This is accomplished by first shifting into the path a sequence of all 0s, followed by a sequence of alternated 0s and 1s, checking whether the same sequence appears on TDO at the end of the all 0s sequence after the expected number of clock cycles. In this Section we described a set of rules to be matched by the set of sessions in order to guarantee the test of all configuration modules, and then propose a method to identify a sequence of configurations, able to detect all possible static faults possibly affecting an IEEE 1687 network while minimizing the total test time. Some structures allowed by the standard are not covered in this work, and we are working to the generalization of the approach to the whole set. Extension to diagnosis is also being considered.

5 IJTAG Network Optimization and Adaptation, and Dynamic Pattern Retargeting

5.1 Introduction

In the following section, we will review the previous work in this regard, and will elaborate on how the BASTION project will enhance the state of the art.

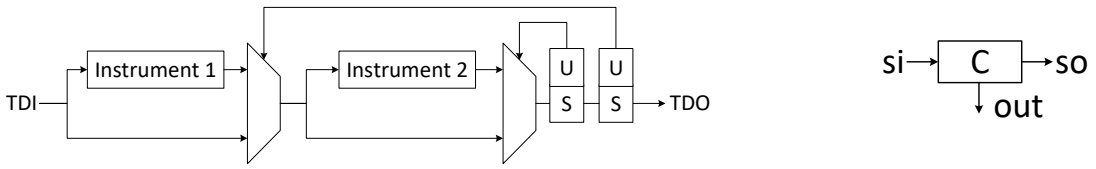
On-chip instruments are used in different stages of an integrated circuit's life cycle: from prototype debug and validation to in-field monitoring and test [31]. At any of these stages, the relevant instruments are accessed a number of times in arbitrary combinations with other instruments. In order to keep the access time low when there are many instruments, reconfigurable networks described by IEEE 1687 [1] can be used. Reconfigurable networks help to reduce access time by allowing to include only relevant instruments in the scan-path for each access. To minimize access time, especially when there are many accesses to perform, it is essential to reconfigure the network such that length of scan-path during each access is kept to a minimum. Therefore, many network reconfigurations might be performed. In IEEE 1687, finding the right reconfiguration is the task of retargeting algorithms. Briefly, retargeting is the translation of human-readable access procedures described at instrument terminals into scan vectors applicable at chip boundary. Retargeting process can be seen as a sequence of retargeting steps, where each step comprises generating scan vectors for reconfiguration as well as for performing the read and write operations on the instruments. The time it takes to generate scan vectors in a retargeting step can be used to gauge the efficiency of a retargeting algorithm. The effectiveness, on the other hand, can be evaluated by the application time (i.e., the volume) of the generated scan vectors. To increase efficiency and effectiveness, it is important to optimize the retargeting step. To perform retargeting, initial works have proposed the use of Boolean satisfiability problem (SAT) modeling for IEEE 1687 networks [32], as well as retargeting heuristics [33]. The work in [34] improved prior work w.r.t. effectiveness and efficiency via (1) using a more compact SAT modeling, (2) limiting the search space while maintaining optimality by using bounds on the number of capture-shift-update (CSU) operations, and (3) enabling minimized number of calls to the SAT solver. The bound applied in [34] was, however, applicable only to a subset of IEEE 1687 networks constructed from repetitions of certain structural patterns. In this contribution, we extend [34] by proposing an upper-bound computation method applicable to a wider range of IEEE 1687 networks. The method models the relevant properties, such as length of shift-registers, of an arbitrary IEEE 1687 network in the form of a Finite State Machine (FSM). From the FSM, the upper-bound is computed as the maximum number of CSU cycles needed to take the network from any initial configuration to any target configuration. In order to ensure the applicability of our approach to large networks, a set of reduction techniques are described and evaluated. We have implemented the method and have reported experimental results on a number of new and existing benchmarks. The results show that our method yields as good upper-bound as previous work [34], but can handle a wide variety of IEEE 1687 network designs while being still applicable to large designs.

5.2 Related Work

In this subsection, the relevant hardware features of IEEE 1687 are introduced in Subsection 5.2.1, and the retargeting concept is explained in Subsection 5.2.2.

5.2.1 Instrument Access Infrastructure (Network)

A strong feature in IEEE 1687 networks is the possibility of dynamic reconfiguration, which allows for reduction of the scan-path that are needed for current access. To enable dynamic reconfiguration in IEEE 1687 networks, ScanMux control bits are used, which are shift-update registers that can be placed anywhere on the scan-path to configure scan multiplexers (ScanMux components). Figure 32(a) shows two ScanMux control bits used to configure a network of two instruments. To program the control bits to any desired configuration, the right values should be placed in their shift cells (denoted by S) during the Shift phase, and copied to their parallel latch (denoted by U) during the Update phase. Clearly, for muxes with more than two inputs, multiple control bits are used. In the following, we will use the symbol in Figure 32(b) to represent a ScanMux controller, irrespective of how many bits it contains.
IEEE 1687 specifies the JTAG test access port (TAP) [35] as the primary interface between the chip boundary and the on-chip network of instruments. Interfacing is performed by connecting the IEEE 1687 network as a design-specific test data register (TDR) to the JTAG circuitry. Since the TAP FSM is primarily used to operate IEEE 1687 networks, performing each cycle of network configuration involves going through the capture, shift, and update states in the FSM, which is referred to as a CSU operation [1] (hereinafter CSU).



(a) A network of two instruments, configured via two ScanMux control bits (b) Symbol

Figure 32: ScanMux Control bit (SCB)

Figure 33 illustrates a small IEEE 1687 network consisting of three instruments (namely a DFT instrument, a sensor, and a debugging feature) and six ScanMux control bits. The instruments are interfaced to the scan-path through shift-registers with parallel I/O. To access the instruments, the control bits should be programmed to include the required shift-registers in the scanpath. For example, to access only the DFT feature, C_1 and C_2 should be set to logic value "1", and C_3 should be set to "0".
Reconfiguring the network to the desired configuration might need several CSUs. For example, assuming an initial configuration of $C_1 = \ldots = C_6 = 0$ in Figure 33, accessing the Debug instrument needs two CSUs. In the first CSU, only C_1, C_2, and C_3 are accessible, and by setting $C_2 = 0$ and $C_1 = C_3 = 1$, C_4, C_5, and C_6 become accessible. In the second CSU, C_4, C_5, and C_6 can be configured as $C_5 = 0$ and $C_4 = C_6 = 1$, so that the Debug instrument becomes accessible.
In Figure 33, the clock, control signals (namely, capture, shift, and update), and the select signals used to gate the control signals are not shown. In this work, it is assumed that only the components on the selected input of a mux get their select signal asserted. The select signal for C_1 is asserted when the TDR corresponding to this IEEE 1687 network is selected, i.e., C_1 is always accessible when working with this network.

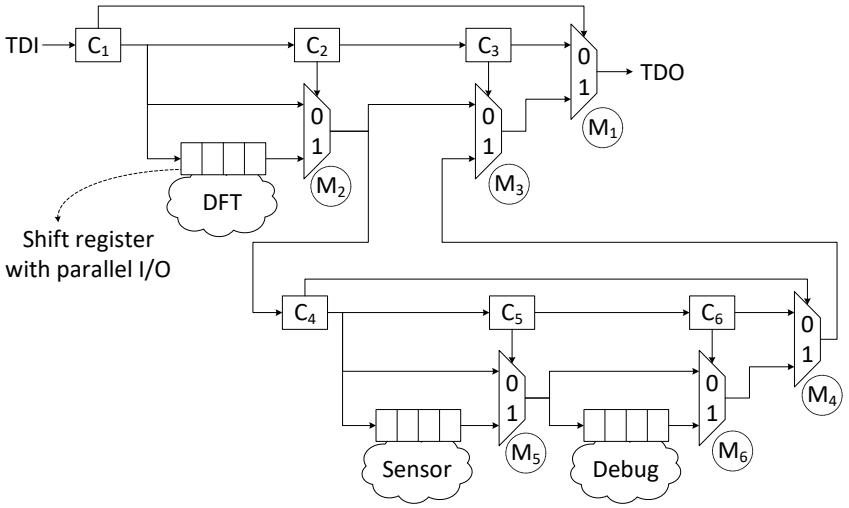


Figure 33: An IEEE 1687 network with three on-chip instruments

5.2.2 Description Languages and Retargeting

IEEE 1687 introduces two description languages: Instrument Connectivity Language (ICL) and Procedural Description Language (PDL). ICL is used to describe the network, i.e., how the instruments are connected to the TAP. PDL is used to describe the operation of instruments at their terminals. PDL commands allow to perform read/write operations on instrument shift registers and control bits, as well as to wait for an instrument (such as a BIST engine) to finish its operation. Using the ICL description of a network, a retargeting tool translates PDL scripts into scan vectors that configure the network and transport the required data between the TAP and the instruments. Retargeting tools relieve engineers from dealing with network configuration (i.e., directly writing PDL to configure ScanMux control bits). For example, assuming that the goal is to read the value from the sensor instrument in Figure 33, the PDL developer might simply use a write command to activate the sensor, a wait command to wait for the sensor to capture the value, and a read command to read the captured value out. It is then the task of the retargeting tool to generate (1) a scan vector to configure C_1, C_2, and C_3, (2) a vector to configure C_4, C_5, and C_6, (3) a vector to write to the enable bit in the sensor's shift-register, (4) a wait cycle of enough length, and finally (5) a vector to scan the captured value out.

In its basic form, a PDL script is a sequence of iApply groups. In each iApply group, there are a number of read and write operations to the registers in the network, which take effect upon encountering an iApply command. Translating an iApply group into scan vectors can be seen as a retargeting step. More specifically, a retargeting step is to generate a number of scan vectors to (1) change the configuration of the network from its current state to a target configuration in which the specified registers are accessible, and (2) to perform the read/write operation. Each vector is then applied to the network through a CSU operation. The complete retargeting flow for the PDL script comprises a number of such retargeting steps.

5.3 State-of-the-art in 1687 Retargeting

For complex IEEE 1687 networks and especially for long PDL scripts, it becomes desirable to both speedup the retargeting process and to generate scan vectors which are optimal with respect to the application time. To achieve these goals, a first measure would then be to optimize the retargeting step for both run-time efficiency and effectiveness of the generated vectors. There have been a number of works addressing retargeting for an IEEE 1687 network [32], [33], [3], [36], [37], [3], [38]. So far, only [32], [33], [34] have addressed efficiency or effectiveness in retargeting. What distinguishes [7] from the other works is addressing the efficiency of retargeting when applying interactive PDL (PDL Level-1, which supports programming language constructs such as conditions and loops) with the help of hardware acceleration. The only works that have so far addressed efficiency and effectiveness for a basic retargeting step are [32], [33], [34], which are discussed in this section.

Verification and pattern generation (retargeting) for reconfigurable scan networks were presented in [32]. The work in [32] models general reconfigurable scan networks using a structural SAT model which captures any arbitrary configuration of the network. In a typical retargeting step, several configuration cycles should be performed to take the network from an initial configuration to a target configuration (in which the shift registers of the required instruments become part of the active TDI to TDO scan-path). Therefore, to capture all the configuration cycles, the SAT model is unrolled over a number of time frames. Each of the time frames corresponds to an atomic CSU.
That is, each individual clock cycle spent on shifting input data (or performing capture and update operations) is not considered to be a separate configuration step, rather the whole cycle of capturing, shifting, and updating is seen as one step. The state of each bit (inside shift-registers and ScanMux controllers) in each time frame is then used to form a scan vector that should be shifted in and applied (by going through the update phase) for the transition from a frame to the next one. A sequence of such scan vectors is what a retargeting tool computes for taking the circuit from its current configuration to a target configuration. Using the above-mentioned scheme requires the algorithm to receive as input the number of times it should unroll the model (i.e., the number of allowed CSUs). The choice of the number of CSUs has a crucial impact on the resulting solution (i.e., the generated scan vectors). If the allowed number of CSUs is too small, the target configuration might be reachable from the current configuration (i.e., no feasible solution). Moreover, given that some solutions might be better than the others w.r.t. access time (in terms of test clock cycles), a too small value for the number of CSUs might exclude those better solutions from the search space. Therefore, finding the upper-bound on the number of CSUs is essential for effective retargeting (i.e., generating scan vectors which are optimal w.r.t. access time). On the other hand, if the number of allowed CSUs is too large, the generated model becomes unnecessarily large resulting in decreased runtime efficiency, yet with no guarantee on optimality. The work in [32] does not present an upper-bound derivation method for the number of required time frames and assumes that the user specifies a maximum allowable number of frames. Moreover, the generated scan vectors are not optimal regarding instrument access time. To address these issues, [33] presents an upper-bound for the number of time frames. The calculation of upper-bound on the number of frames, as presented in [33] can be explained as follows. The total access time is formulated as $\mathrm{t} = 2n \cdot \sum_{i=0}^{n} L_i$, where n is the number of frames, 2 represents the number of clocks spent on applying the stimuli and capturing the

responses for each frame, and L_i represents the length of the scan-path for frame i. The upper-bound for n, denoted by n_{bound}, is presented as

$$n_{bound} < [Cycles_n/2], \quad (1)$$

where $Cycles_n$ is the minimum access time achievable with n frames. According to the work in [33], finding the global minimum is an iterative process in which after finding an initial solution, the bound is calculated and iteratively lowered as we find solutions with smaller access times (i.e., smaller than $Cycles_n$ which was originally found).

Given that in real-life circuits, the access time might be in the order of thousands of clock cycles, the bound calculated using Equation (1) will not be helpful in practice. The reason is that, as discussed in [33], finding the optimal solution is NP-hard, hence requiring heavy computations to search the solution space, which is limited by the upper-bound on the number of frames. If this upper-bound is very high (that is, hundreds or even thousands of frames), the time that it takes to find the optimal solution will be very long. Therefore, the authors of [33] propose a heuristic for retargeting, which initially searches for the minimum number of CSUs required to get a solution, and from that point continues the search for a better solution by allowing a limited number of extra CSUs. There are two drawbacks with the heuristic proposed in [33], both negatively impacting the run-time efficiency. Firstly, searching for the minimum number of required frames involves multiple calls to the SAT solver, each with an incremented number of allowed CSUs. Secondly, allowing extra CSUs after an initial solution found (hoping to reach a local minimum) might be unnecessary if the solution already found is the globally minimum solution. The work in [34] demonstrated the possibility to calculate the upper-bound for a given network via structural analysis. The analysis in [34] is only applicable to a particular class of IEEE 1687 networks referred to as MUX-based in [32] [33]. The analysis performed in [34] leverages the repetition of a certain structural patterns in the same hierarchical level as well as across multiple hierarchical levels. This makes the application of such analysis very limited as in general a network might be any arbitrary connection of components.

5.4 BASTION Contributions

In this section, the contributions of the BASTION project regarding 1687 retargeting. In this work, we detail an upper-bound computation method, which is applicable to arbitrarily designed IEEE 1687 networks (as contrasted with [34]), and results in a bound low enough for real-life retargeting applications.

5.4.1 Motivational Example

In retargeting, a solution with minimum number of CSUs is not necessarily the optimal solution w.r.t. the number of clock cycles, as is shown in this section with the help of an example. It is also shown that the bound calculated by using Equation (1) can be large even for a very small example network. Moreover, it is discussed how length of instrument shift-registers affect the number of CSUs needed for obtaining the optimal solution (i.e., the upper-bound). It should be noted that the upper-bound analysis in [34] is not applicable to the example in this section, as it does not contain the structural patterns required by [34].
Figure 34 shows a network of five instruments. Lengths of instrument shift-registers in this network are as shown in

Table 8 for a number of instances we will consider. Assume that initially all control bits are set to zero, and that we aim to access instrument I_4. Accessing I_4 can be done by setting C_0 to "01". This, however, will not necessarily lead to minimum access time for I_4 since instruments I_2 and I_3 are then on the scan-path to I_4. Therefore, it might be better to first switch I_2 and I_3 off the scanpath before setting C_0 to "01". The reason for saying "might be" is that in this example, I_0 is always on the scan-path and for each access to the network, dummy bits should be shifted through it. If length of I_0 is comparable to the length of the shift-registers for I_2 and I_3, its contribution to overhead cancels out the benefit from switching I_2 and I_3 off the scan-path. To see how the length of shift-registers affect the search process, in the following, we will examine three instances (denoted by A, B, and C in Table 8) of this problem more closely, where each instance differs from the others only in length of shift-registers.

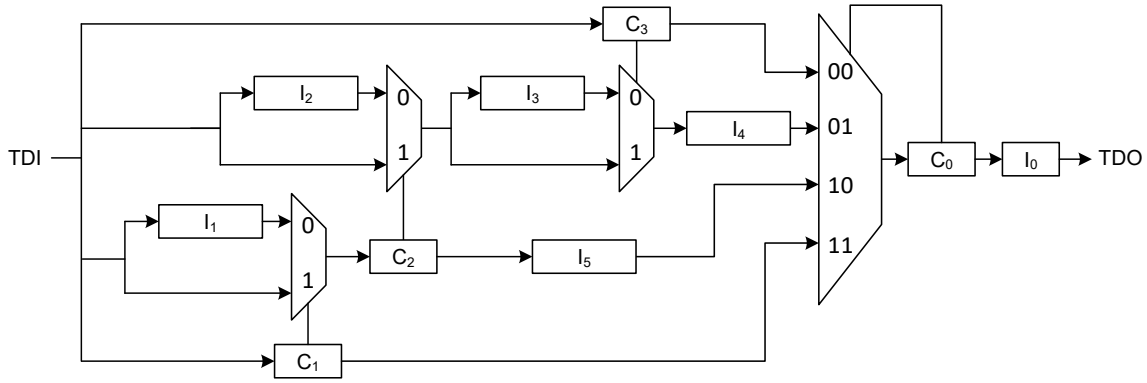


Figure 34: An example reconfigurable network used in the discussion in Section 5.4.1.

Table 8: Shift-registers' length for the instruments in Figure 34

	Length of instrument shift-registers					
	I_0	I_1	I_2	I_3	I_4	I_5
Instance A	20	50	100	20	20	5
Instance B	20	50	**70**	20	20	5
Instance C	**50**	50	100	20	20	5
Numbers in boldface denote where the instances differ.						

Instance A

The lengths of shift-registers for this instance are reported in the corresponding row in Table 8. Assuming that initially all control bits are set to zero and the goal is to perform a read/write operation on I_4, we calculate the access time for different configuration alternatives of the network. First, we consider the case where the only configuration performed is setting C_0 to "01". Here, two CSUs are needed and access time is calculated as the sum of number of clock cycles needed to (1) configure C_0 in the first CSU and (2) perform one read/write on I_4 in the second CSU. The number of clock cycles for the first CSU is 1 (for C_3) + 2 (for C_0 which is a two-bit register) + 20 (for I_0) + 2 (to perform the update and capture operations). The number of clock cycles for the second CSU is 160 (for instruments I_2, I_3, I_4, and I_0) + 2 (for C_0) + 2 (for the update and capture operations). In total, it takes 189 clock cycles to perform these two CSUs (marked on

the plot shown in Figure 35). Alternatively, since C_3 is initially on the scan-path, it can also be set to '1' in the first CSU. In this case, I_3 will not be on the scan-path in the second CSU and it thus takes 169 clock cycles in total to perform the two CSUs (also marked in Figure 35).

The two alternatives discussed above used two CSUs to access I_4. That is, if we limit the retargeting tool to unroll the model twice, the pseudo-Boolean optimization explores the above solutions and picks the one with the lowest access time, i.e., the one with 169 clock cycles. In the following, we explore alternative configurations with more than two CSUs.

If instead of switching C_0 to "01", we set it first to "10", we gain access to C_2 and can switch I_2 off the scan-path before performing the read/write operation on I_4. In this case, we use three CSUs and the access time is calculated as 149 clock cycles in total. If we allow the retargeting algorithm to use three CSUs, all the solutions marked with two and three CSUs on the plot are explored and the minimum which is 149 will be chosen.

If we switch I_1 off the scan-path before configuring C_2, access time might be further reduced. In this case, four CSUs are required in total and the access time is calculated as 124 clock cycles. The plot in Figure 35 shows access time for other solutions obtainable by using four CSUs, as well.

For this example, allowing further increase in CSUs will not yield lower access time, but will result in growingly complex models that lower the efficiency of the retargeting algorithm. In this regard, for this instance of the problem, the bound calculation in [33](see Equation (1)) calculates the bound on the number of CSUs as [169/2] = 85. Since there are five control bits, unrolling the model 85 times would result in a model with $2^{5\times85}$ decision variables, which should be compared to $2^{5\times4}$ variables when the model is unrolled only four times.

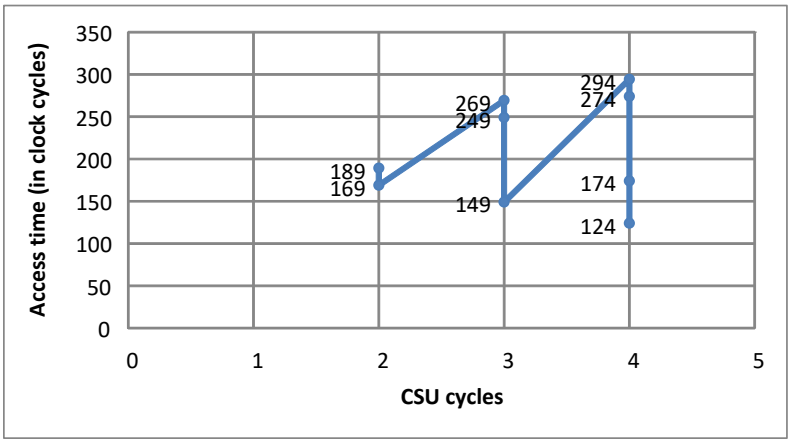


Figure 35. Access time vs. number of allowed CSUs for Instance A.

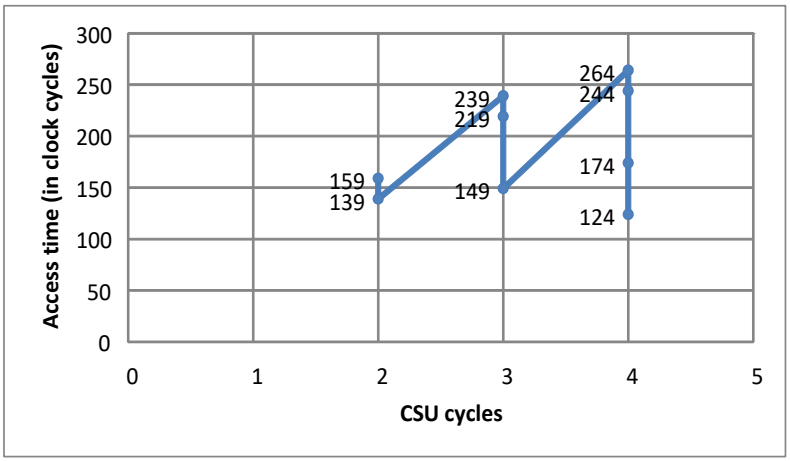


Figure 36: Access time vs. number of allowed CSUs for Instance B.

Instance B

Figure 36 shows how the solution space would look like if the length of shift-register for I_2 was 70 instead of 100. It is interesting to see that access time does not decrease when three CSUs are allowed but decreases when four CSUs are allowed. This entails that a heuristic searching the solution space by incrementing the bound on CSUs gets stuck at a local minimum. If, however, the search algorithm is aware of a bound on the number of CSUs, it can do enough unrollings of the model and let the pseudo-Boolean optimization find the minimal access time (as well as the right number of CSUs).

Instance C

Figure 39 shows how the solution space would look like if the length of shift-register for I_0 was 50 instead of 20. In this case, the overhead caused by shifting dummy bits through the shift-register for I_0, cancels out any potential benefit from using more CSUs used for removing I_2 and I_3 from the scan-path to I_4. It is important to note that in this example, if the aim was to access I_2 instead of I_4, the optimal solution would be obtained by using a different number of CSUs. The same can be said for other starting configurations (i.e., other than all control bits set to zero). In this work, however, our aim is to find an upper-bound on the number of CSUs which enables reaching the optimal solution for any retargeting step, regardless of the starting configuration and the set of instruments to be accessed. Therefore, in the following section, we propose a method which computes the upper-bound on the number of CSUs as the maximum number of CSUs needed to take the network from any initial configuration to any target configuration. Note that the retargeting algorithm should unroll the model one extra time to account for the actual read/write operation.

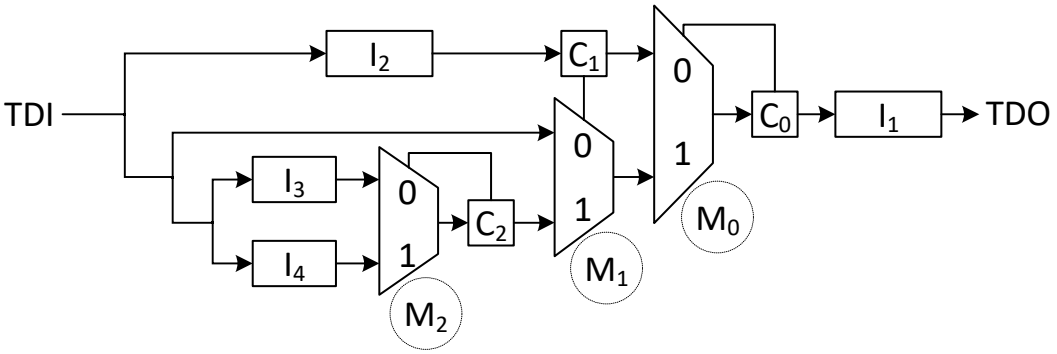


Figure 37: Example network used to describe UCC (5.4.2)

5.4.2 Upper-Bound Computation Core (UCC)

As was mentioned earlier, the aim of this work is to provide a method for computation of an upper-bound on the number of CSUs for a given network. The upper-bound helps in the retargeting process by shrinking the solution space without removing the optimal solution from it. In this section, we explain our generalized Upper-bound Computation Core (UCC) and discuss how its output can be used for optimal retargeting.

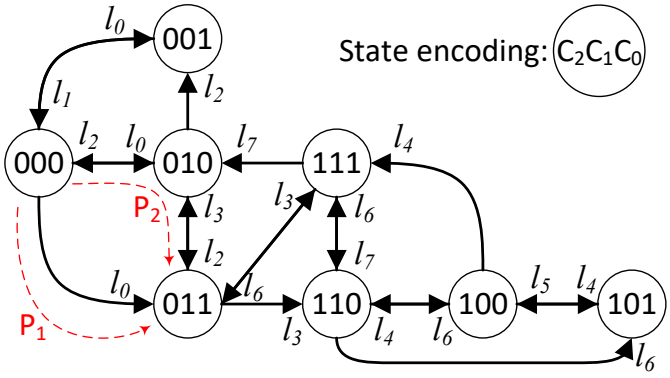


Figure 38: FSM showing the transitions for the network in Figure 37. Labels beside each arrowhead represent the number of clock cycles needed to perform each transition.

5.4.2.1 The Core: UCC

UCC consists of two steps: (1) modeling the network with an FSM, and (2) computation of the upper-bound. In the following subsections, each of these steps is detailed. We will use the example network in Figure 37 to describe UCC.

Table 9: pPaths corresponding to each state

Sate	Active components
000	I_2, C_1, C_0, I_1
001	C_0, I_1
010	I_2, C_1, C_0, I_1
011	I_3, C_2, C_0, I_1
100	I_2, C_1, C_0, I_1
101	C_0, I_1
110	I_2, C_1, C_0, I_1
111	I_4, C_2, C_0, I_1

1) *Modeling with an FSM:* The network in Figure 37 has three one-bit mux controllers C0–C2 and thus eight possible configurations. The FSM in Figure 38 models the network in Figure 37, where each state (encoded as the bit sequence C2C1C0) represents one of the eight configurations, and each edge models a transition between two states. Transitions which are from a state to itself are not considered in the model. The labels l_i beside transition arrowheads represent the number of clock cycles needed to perform the transition. The required number of clock cycles is calculated as the sum of length (in number of flip-flops) of components on the active scan-path (namely, shift-registers and control bits) plus the number of clock cycles needed to perform capture and update operations. Table 9 lists the components that are active in each of the states. For example, l_0 which corresponds to state 000 is calculated as sum of the length of components I_2, C_1, C_0, and I_1, plus two clock cycles for capture and update operations. It is worth noting that not all transitions are bidirectional, and that length of a transition is not necessarily equal to the length of the transition in the opposite direction.

2) *Computing the Upper-Bound:* The FSM in Figure 38 can be used to calculate the number of CSUs needed to transition from each of the states to any other state. The number of CSUs is equal to the number of transitions between two states. There might be multiple paths for transitioning between a pair of states. For example, both paths marked with P_1 and P_2 on the FSM in Figure 38 can be taken to change the state from 000 to 011, where

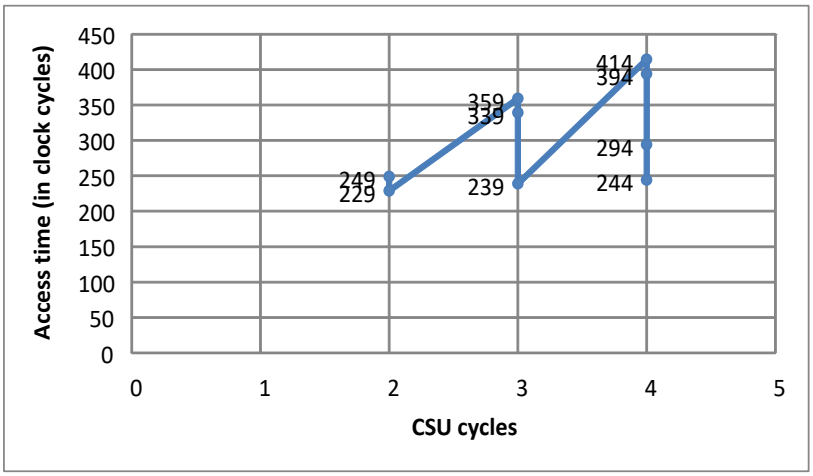


Figure 39: Access time vs. number of allowed CSUs for Instance C.

P_1 takes l_0 clock cycles and P_2 takes $l_0 + l_2$ clock cycles. We are, however, only interested in the number of transitions for the path that uses fewer clock cycles (which is not necessarily the path with the fewer number of transitions). Therefore, if we derive the shortest path between any two states s1 and s2, and compute the number of transitions (a.k.a. number of hops) needed to achieve that shortest path, we will know how many CSUs are needed for the transition from s1 to s2. The upper-bound, i.e., the number of CSUs which allows to take the network from any state to any state with the smallest number of clock cycles, can then be calculated as the maximum among the number of hops corresponding to each pairwise shortest path. Assuming a length of 20 flip-flops for instrument shift-registers I1–I4, Table 10 presents the number of transitions corresponding to the shortest path between each pair of states in the FSM (Figure 38), where the first column lists the source states and the first row lists the target states. Based on Table 10, the upper-bound on the number of CSUs is found to be four.

5.4.2.2 Optimal Retargeting for Small Networks

The pairwise shortest paths information obtained as described in Section 5.4.2.1 can be used to *directly* generate the optimal scan vectors (w.r.t. test clock cycles) needed for retargeting. That is, instead of using the upper-bound to unroll an ILP model and solving the resulting pseudo-Boolean optimization, one can use the shortest paths information to find what configuration steps should be taken for taking a network from its current configuration to a target configuration. Since in many target configurations a superset of the desired instruments might be accessible, an approach merely based on the shortest paths information should choose the smallest among the shortest paths from current configuration to all those target configurations. Moreover, the length of the scan-path for those configurations should also be taken into account. The reason, as was discussed in Section 5.4.1, is that the actual goal in retargeting is performing read/write operations on the instruments. Therefore, for optimal retargeting, not only the transition time between states should be taken into account, but the time it takes to perform (at least) one read/write should also be considered.

Table 10: Number of transitions (hops) corresponding to the pairwise shortest paths among the states in Figure 38

State	000	001	010	**011**	**100**	**101**	**110**	**111**
000	0	1	1	1	3	3	2	2
001	1	0	2	2	4	4	3	3
010	1	1	0	1	3	3	2	2
011	2	2	1	0	2	2	1	1
100	3	3	2	2	0	1	1	1
101	4	4	3	3	1	0	2	2
110	3	3	2	2	1	1	0	1
111	2	2	1	1	2	2	1	0

This method of retargeting is, however, only applicable to small networks for which the pairwise shortest paths can be computed efficiently. For large networks, the computation time and memory requirements make the use of this method inefficient.

5.4.2.3 Pessimism in the UCC Results

There are two types of transitions that might increase the upper-bound unnecessarily. The first types are transitions that do not change the set of active components, such as transition from state 001 to state 101. The second type are transitions that do change the set of active components, but

the new set is achievable via other transitions with smaller number of CSUs and less than or equal number of clock cycles. For example, states 000 and 100 activate the same set of components, but it takes fewer clock cycles to go from 001 to 000 than from 001 to 100. These two transition types make the computed upper-bound slightly pessimistic.

Removal of such pessimism from the UCC results should be done with care otherwise the optimal solution to retargeting might be removed from the search space. As the aim of the current work is to provide guarantees for optimal retargeting we leave the pessimism removal for future work.

5.4.3 Handling large networks

The method we described in Section 5.4.2 is not directly applicable to large networks as the number of states in the FSM model grows exponentially w.r.t. the number of control bits. In this section, we describe three techniques (referred to as *reduction* here) which can help in handling large networks. Due to the lack of space, we only detail the implementation of the decomposition technique. We conclude this section by explaining how these reduction techniques are used in a complete upper-bound computation flow.

5.4.3.1 Reduction Through Decomposition

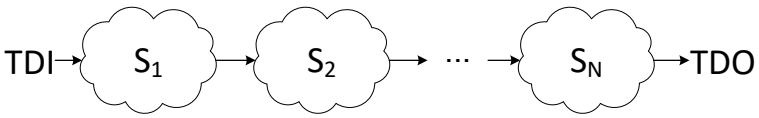


Figure 40: A network consisting of N isolated segments.

Figure 40 shows a network consisting of N segments S_1–S_N. Each of these segments is connected to the rest of the network exclusively via a scan-in/scan-out pair. In this work, any such segment is referred to as an *isolated* segment. In the network in Figure 40, a CSU applied to any of these N segments is also applied to the other N-1 segments at the same time. The reason is that the serial data goes through all the segments and the control signals are applied to (the currently active path) in each of them at the same time. Therefore, the segment requiring maximum number of CSUs determines the upper-bound. That is, the technique described in Section 5.4.2 can be applied to each segment S_i individually to compute the upper-bound for that segment (denoted as $n_{b,i}$), and the upper-bound for the whole network can be calculated as $max_{i=1}^{N} n_{b,i}$. Through decomposition, the worst-case complexity of upperbound computation for the original network is reduced to that of upper-bound computation for the segment containing the highest number of control bits.

5.4.3.1.1 Impact of Decomposition on Upper-Bound

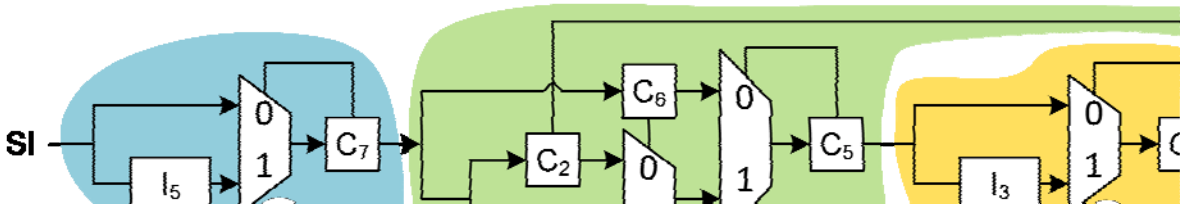
The upper-bound computed via decomposition might be slightly higher than what would be computed if UCC was directly applied to the original network (and therefore, higher than what is actually needed for optimal retargeting). The reason can be explained by referring to the motivational example network in Figure 34, which can be seen as combination of two isolated segments: s_1 containing instrument I_0, and s_2 containing the rest of components. We observed for Instance C of that example that an increase in the length of I_0 (from 20 to 50) caused a decrease in the number of CSUs needed for optimal access to I_4 (from 4 to 2). Seen the other way around, going from Instance C to Instance A, which decreases the length of I_0, causes an increase in the number of CSUs needed for optimal retargeting. The same effect is present in decomposition as it

removes other segments from each other's scan-path. This increased number of required CSUs calculated for each isolated segment, might make the upper-bound computed by the use of decomposition slightly pessimistic.

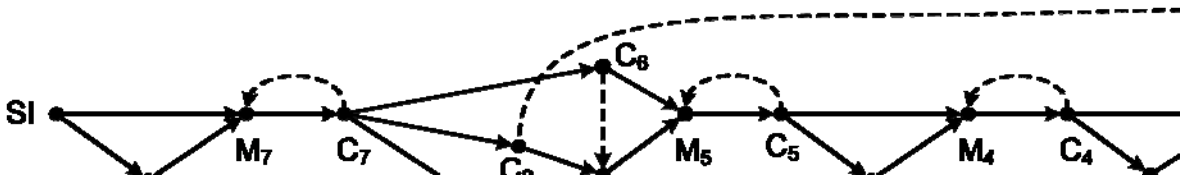
5.4.3.1.2 Performing Decomposition

We will now use the example network in Figure 41(a) to explain how to distinguish isolated segments. In this figure, the network components belonging to different isolated segments are marked with colored areas. For more clarity, each of the three isolated segments is also marked with a number. Compared to the conceptual illustration of isolated segments presented in Figure 40, in which it is clear where on the scan-path an isolated segment begins and ends, it is less straightforward to identify all isolated segments in the network in Figure 41(a). Given the exponential complexity of the presented UCC technique w.r.t. number of control bits, it is crucial to identify more (and consequently smaller) isolated segments in a given network.

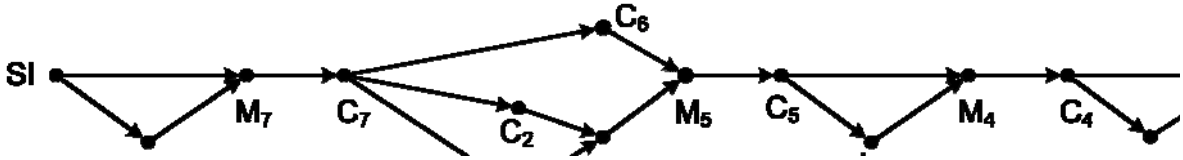
In the following, a two-step procedure for identification of isolated segments is presented. In the first step, we identify network segments connected to each other in series on the scanpath



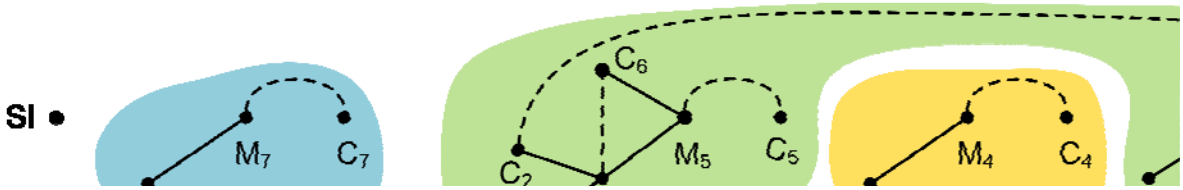
(a) Example network



(b) Corresponding directed graph with control signals denoted by dashed edges



(c) Corresponding directed graph without control signals (scan-path-only graph)



(d) Corresponding undirected graph after removing output edges of all vertices on the chain of idoms

Figure 41: Decomposition example

(hereinafter *candidate segments*). In the second step, based on the control dependencies between these candidate segments, we group them to form isolated segments.

a) *Step 1:* The graph in Figure 41(b) models the network in Figure 41(a), where the control signals are represented by dashed lines. In identification of candidate segments, we use the concept of *graph dominators*. In a directed graph, a vertex v_1 dominates vertex v_2 if all the paths going through v_2 pass first through v_1. For example, in Figure 41(b), vertex SI dominates all vertices in the network. However, SI is only *immediate dominator* (called *idom*) to I_5 and M_7. There are efficient algorithms to find idoms for vertices in a graph [39]. Dominators help to identify where on the scan-path a candidate segment starts and ends. For example, C_7 marks where isolated segment 1 finishes and isolated segment 2 begins on the scan-path. If, however, we apply the concept of dominators directly to the complete network graph in Figure 41(b), we fail to identify segment 3 as an isolated segment. Therefore, we instead apply the graph dominators algorithm to a scan-path only copy of the graph (in which control signals are removed) shown in Figure 41(c). Based on the results, we create a chain of idoms for the scan-path-only graph by going from the scan-out (SO) towards the scan-in (SI). The chain will be as SI → M_7 → C_7 → M_5 → C_5 → M_4 → C_4 → M_3 → M_1 → C_1 → SO, which reads as SO is immediately dominated by C_1, which is in turn immediately dominated by M_1, and so on. The vertices on this chain mark entry and exit points of candidate segments.

b) Step 2: The key to grouping candidate segments into isolated segments is detecting control dependencies between those candidate segments. That is, if there is a control signal connecting two candidate segments, those segments should be grouped and analyzed as one segment. To detect such dependencies, we use a copy of the network graph in which the output edges of all vertices on the chain of idoms are removed, as shown in Figure 41(d). Moreover, this graph is converted into an undirected graph, as the aim is to find connected network components irrespective of the order they appear on the scanpath. To identify which of the candidate segments should be grouped together, we use the concept of *connected components* in graph theory. A connected component in an undirected graph is a set of vertices in which any two vertices are connected. It should be noted that the a "graph component" is a set of vertices, and in our problem maps to an isolated segment, and not to a "network component". After applying the connected components algorithm, the isolated segments are identified as marked with the colored areas in Figure 41(d). The algorithm also identifies SI and SO as isolated segments, which we ignore.

In this example, there were no instruments in the chain of dominators, as there was no instrument directly on the scan-path between scan-in vertex SI and scan-out vertex SO. When there are instruments on the chain, they can be ignored, because if we form separate isolated segments for them, the upper-bound for that segment is zero (simply because there are no control bits in such an isolated segment).

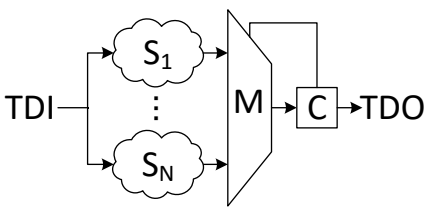


Figure 42: Example structures for the "lookup" technique Type I

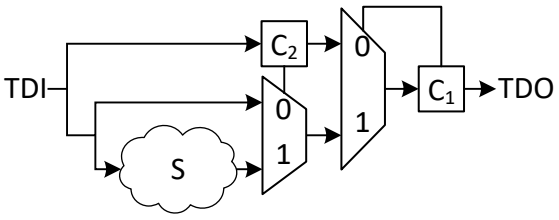


Figure 43: Example structures for the "lookup" technique Type II

5.4.3.2 *Reduction Through "Lookup"*

Another technique for handling upper-bound computation for large networks is to recognize structures for which we know how to calculate the upper-bound. As an example, consider the structure shown in Figure 42. Here, the assumption is that each of the segments S_1–S_N is isolated (in the sense defined in Section 5.4.3.1). The first point to note is that in any retargeting step, only one of the inputs to mux M can be active. That is, only one of the segments S_1–S_N should be configured, and therefore, it suffices to consider only the segment which requires the largest number of CSUs. For the structure shown in Figure 42, the upper-bound for the whole structure can be computed as $1 + max_{i=1}^{N} n_{b,1}$, where $n_{b,i}$ is the upper-bound computed for segment S_i, and 1 represents the CSU needed to configure mux M itself. Another example is the structure shown in Figure 43, for which the upper-bound for the whole structure is the upper-bound for segment S plus two [34].

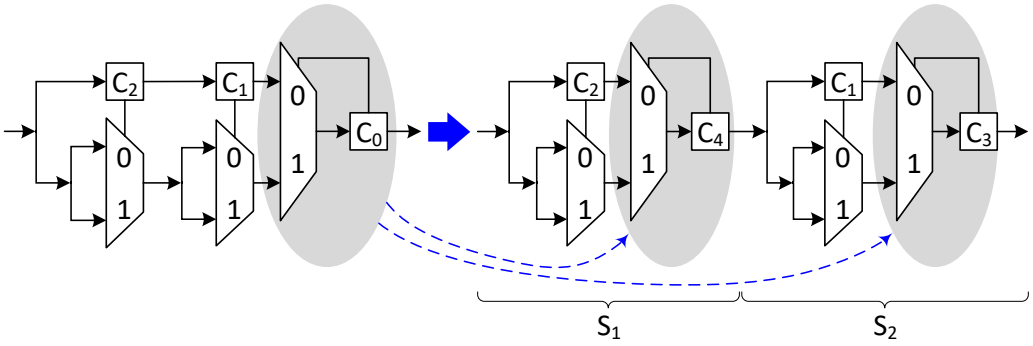
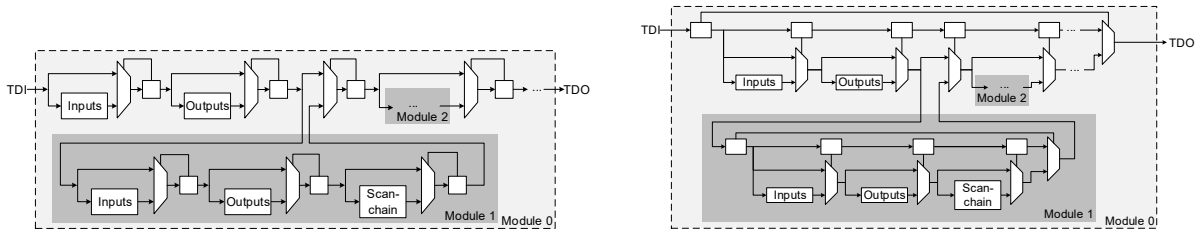


Figure 44: An example rewriting technique



(a) SIB-based (here, the internals of a SIB module are shown)

(b) MUX-based

Figure 45: The two variants of p34392 benchmark used in [32]

5.4.3.3 *Reduction Through Rewriting*

The idea in rewriting is to create a network which is equivalent to the original network w.r.t. the upper-bound on the number of CSUs, but can be handled by the above-mentioned decomposition techniques. An example of rewriting is presented in Figure 44 where the network to the left is rewritten by duplicating control bit C_0 along with its associated mux. The resulting network (to the right) can then be reduced by using the technique in Section 5.4.3.1, as each of the segments marked by S_1 and S_2 are isolated. Note that although the functionality of the rewritten network is different from the original network, the upper-bounds of both networks are equal.

5.4.3.4 *The Complete Upper-Bound Computation Flow*

In the following, we describe our complete upper-bound computation flow, which is based on the use of UCC (described in Section 5.4.2) and the reduction techniques described earlier in this section. Initially, the rewriting method is used to create a new network that has the same upper-bound as the original network. The computation of the upper-bound starts by applying decomposition, which distinguishes one or more isolated segments. The lookup technique is then applied to each of these segments. If the lookup does not recognize any known structures, UCC is performed on the segment. However, if the lookup recognizes a structure, it calls the decomposition technique on the isolated segments it has identified. In other words, the upper-bound computation consists of a number of mutual calls between the decomposition and lookup methods. When an isolated segment is not recognized by the lookup function, UCC is applied to it. The upper-bound computed for each segment is then used to compute the upper-bound for the whole network by using the formulas described for each of the reduction techniques.

5.4.4 Experiments

The described analysis and reduction techniques for computing the upper-bound for IEEE 1687 networks have been applied to a set of representative benchmarks which consists of three groups of networks. The first group contains networks which are constructed manually such that they are not decomposable by the methods described in Section 5.4.3 (see Figure 46). Here, the proposed upper-bound computation core (i.e., UCC) has to be applied on the complete network. The second and third groups are introduced in [32], and their networks are based on ITC'02 [30] benchmarks. In the second group, the networks are implemented by using Segment Insertion Bit (SIB) modules. As an example of such SIB-based implementation, Figure 45(a) illustrates the RTL network structure for the benchmark p34392. In the third group, the networks are implemented by using a daisy-chained architecture, referred to as MUX-based in [33] (see Figure 45(b)). Additionally, we report the results for the network shown in Figure 41(a), which was used to explain the decomposition technique. For all benchmarks, the length of instrument shift-registers is assumed to be 20 flip-flops.

The results obtained by evaluating the techniques proposed in this paper are summarized in Table 11. The first two columns of the table list the names and the total number of control bits for each benchmark network. The third column reports the maximum number of control bits required to model indecomposable sections within the network. This information is important since the number of control bits significantly impacts the run-time of UCC. It can be observed that for those benchmarks tailored to be indecomposable, namely, N_1–N_5, the proposed reduction techniques do not succeed to reduce the number of control bits. In contrast, the number of control bits is reduced for the network in Figure 41(a) from seven to five, which can be explained by the

decomposition of that network into three isolated segments the largest of which contains five control bits (Figure 41(a)). On the other hand, if the reductions are successfully applied, such as for the set of SIB-based and MUX-based benchmarks, the generation of an FSM and the application of UCC can be completely omitted. The reason is that for networks in the second and the third groups, the reduction techniques decompose the networks into a number of isolated segments each containing only one instrument shift register. As was mentioned in Section 5.4.3.1, the upper-bound for an isolated segment containing only instrument shift-registers is zero—hence no need for applying UCC.

The computed upper-bounds are listed in column four. The computed upper-bound denotes the maximum number of CSUs needed to reconfigure the network. In order to perform the actual

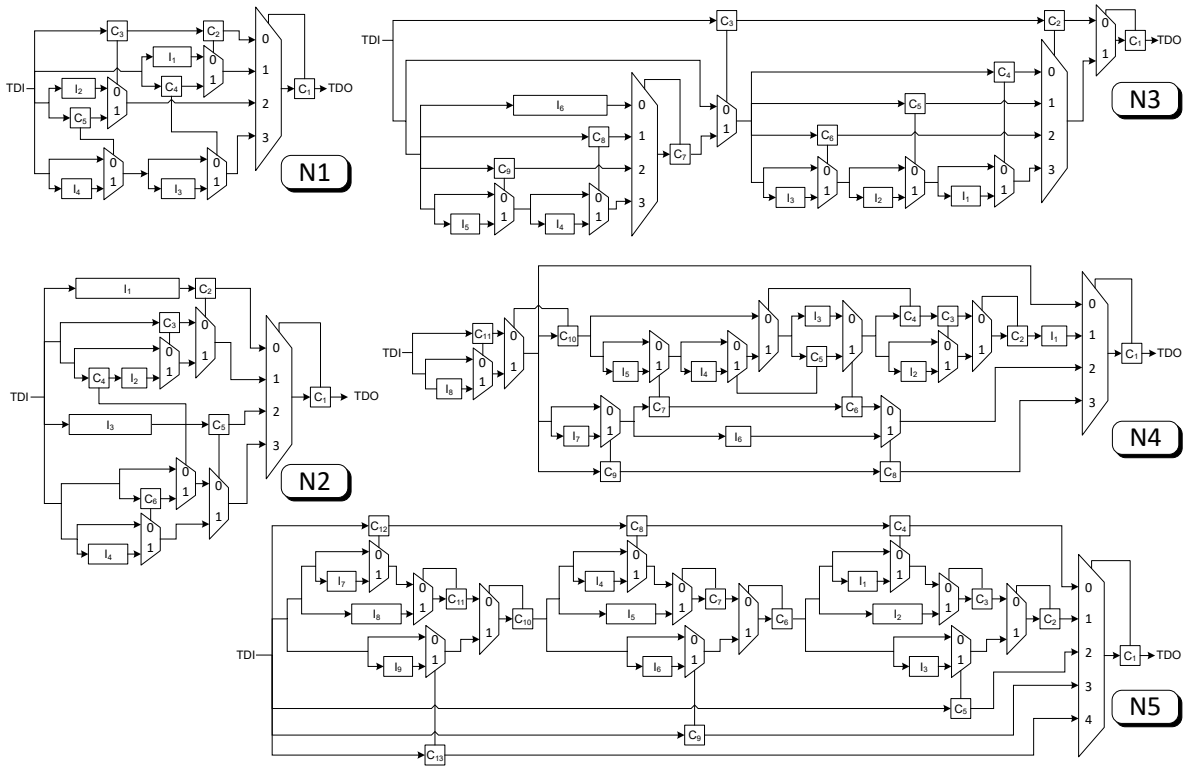


Figure 46: Five networks that cannot be decomposed by the methods mentioned in Section 5.4.3

read/write operation an additional CSU is required (see Section 5.4.1). In comparison to the upper-bound computation proposed in [34], which could only be applied to MUX-based benchmarks, the new method generates identical results while being applicable to a wide variety of IEEE 1687 network designs. The upper-bound values computed by the approach in [34] are reported in column five.

The described reduction techniques, such as rewriting, decomposition, and lookup, are evaluated in the columns six to eight. The reported run-times are the total sum over all the application cases of each of these techniques for each of the benchmarks. For the first set of benchmarks the run-time of the reduction techniques is negligible. Applying the reduction techniques to the largest among SIB-based and MUX-based benchmarks (i.e., p93791) requires up to more than a total of 10 minutes of run-time. As was mentioned earlier in this section, there is no UCC run-time required for the benchmarks in the second and third groups.

The run-time for generating the FSM after reduction is listed in column nine. As was explained in Section 5.4.2.1, to compute the upper-bound from the generated FSM, the shortest path between each pair of states should be computed. To do so, we evaluated two well-known shortest path computation algorithms, namely, Dijkstra and Floyd-Warshall. The Dijkstra algorithm finds the shortest path between a given source state and all target states, and is therefore run once for each state in the FSM. The run-time reported for Dijkstra algorithm in column 10, is the sum of the run-times for each source state. The Floyd-Warshall algorithm is, on the other hand, an all-pairs shortest paths algorithm and finds the shortest path between each pair of states in the FSM in one run. The run-time for the Floyd-Warshall algorithm is reported in column 11. The observation from our experiments is that the Dijkstra algorithm performed especially well on large FSMs (namely, for benchmarks N_3–N_5), whereas the alternative Floyd-Warshall algorithm required slightly less runtime on small FSMs. In general, the Floyd-Warshall algorithm has higher complexity (compared to running Dijkstra once for each source state) when the FSM is a sparse graph. Both algorithms delivered the same results.

Table 11: Experimental Results

Benchmark	Number of control bits	Largest number of bits seen by UCC	Upper-bound		Reduction run-time in milliseconds			UCC run-time in milliseconds		
			This work	[34]	Rewriting	Decomposition	Lookup	FSM generation	Computation (Dijkstra)	Computation (Floyd-Warshall)
The following networks are presented in the current work:										
Figure 41(a)	7	5	5	-	1.0	1.8	3.3	0.4	0.5	0.0
N1	6	6	5	-	0.7	0.7	1.1	1.0	1.7	0.2
N2	7	7	12	-	1.0	0.6	1.1	1.6	3.7	1.0
N3	11	11	8	-	0.8	0.7	2.5	44.2	643.2	2277.3
N4	12	12	7	-	1.2	0.7	0.0	122.3	4931.0	39548.5
N5	15	15	11	-	1.2	0.7	1.2	1799.1	529376.0	16730000.0
The following are SIB-based networks from [32]:										
a586710	39	0	3	-	3.1	56.7	185.5	0	0	0
d281	58	0	2	-	2.6	105.3	380.1	0	0	0
d695	167	0	2	-	8.0	998.1	3554.6	0	0	0
f2126	40	0	2	-	1.7	52.9	174.9	0	0	0
g1023	79	0	2	-	3.8	213.5	730.9	0	0	0
h953	54	0	2	-	3.9	104.3	353.7	0	0	0
p22810	282	0	3	-	20.3	3330.7	12535.4	0	0	0
p34392	122	0	3	-	5.3	478.8	1673.1	0	0	0
p93791	620	0	3	-	64.0	21743.3	89488.7	0	0	0
q12710	25	0	2	-	1.7	26.2	85.7	0	0	0
t512505	159	0	2	-	6.9	822.3	2873.8	0	0	0
u226	49	0	2	-	2.0	70.3	237.2	0	0	0
The following are MUX-based networks from [32]:										
a586710	47	0	6	6	7.5	95.1	402.3	0	0	0
d281	67	0	4	4	10.3	216.0	935.2	0	0	0
d695	178	0	4	4	39.0	2094.6	10068.8	0	0	0
f2126	45	0	4	4	4.8	96.3	407.5	0	0	0
g1023	94	0	4	4	15.8	401.9	1752.3	0	0	0
h953	63	0	4	4	8.1	187.3	818.6	0	0	0
p22810	311	0	6	6	84.8	7371.7	38070.4	0	0	0
p34392	142	0	6	6	42.9	991.0	4471.3	0	0	0
p93791	653	0	6	6	226.4	55051.0	312816.0	0	0	0
q12710	30	0	4	4	6.0	44.9	183.8	0	0	0
t512505	191	0	4	4	53.3	1802.3	8534.1	0	0	0
u226	59	0	4	4	8.2	138.6	613.7	0	0	0

5.4.5 Future Work

As was mentioned in Sections 5.4.2.3 and 5.4.3.1.1, there might be cases where the computed upper-bound is pessimistic. That is, the computed value is higher than what is actually needed for optimal retargeting. These cases should be investigated and addressed in our upper-bound computation method. Additionally, the proposed upper-bound computation method can be further developed to recognize more structures for lookup and rewriting. Finally, in computing the upper-bound in this work, we made no assumptions on the initial and the target configurations. The benefit of this relaxation is that the upper-bound computation needs to be done only once at the beginning of the retargeting process. The resulting upper-bound can then be used for all retargeting steps in that retargeting process. On the other hand, if the initial and target configurations are considered in the computation of the upper-bound, the computation should be performed once for each retargeting step. In this case, the result will be a tighter bound tailored to that step, which increases the retargeting efficiency. Therefore, the trade-off between (1) saving time by running the upper-bound computation once at the beginning of the retargeting process, and (2) saving time by faster retargeting steps should be investigated.

5.5 Section Summary

For the problem of optimal retargeting for IEEE 1687 networks, the shrinking of the solution space is highly important in order to ensure efficient generation of the shortest scan vectors. This can be done by providing bounds on how many capture-shift-update operations have to be considered in the retargeting process. To provide such bounds, we proposed in this work a method for the computation of upper-bound on the number of capture-shift-update operations. In comparison with prior work, the proposed method uses a number of techniques that make it applicable to a range of complex and large IEEE 1687 networks. By applying the approach to a set of benchmarks, it is shown that the method is able to efficiently provide tight bounds for complex and large benchmark networks.

6 Conclusions

This report concludes the BASTION activities regarding the task T3.2, which focuses on error detection and diagnosis in IJTAG networks. The presented contributions provide solutions to support several related issues in error detection and diagnosis. The contribution described in Section 3 proposes new methods to connect different blocks of complex designs in a hierarchical manner in order to enable effective and efficient testing while applying reconfigurable scan networks.
Additionally the contribution in Section 2 extends previously described contributions to efficiently propagate errors through IJTAG networks. The presented contribution provides details of the hardware implementation of the infrastructure in particular focusing on the implementation of the instrument manager that essentially performs online dynamic retargeting on regular hierarchical IJTAG networks. The remaining two contributions focus on generating and minimizing test patterns to test the IJTAG network as well as retargeting other test pattern in order to ensure efficiency and effectiveness of the resulting test pattern.

In the following, the discussed contributions are mapped onto the KPIs formulated in the DoW document.
KPI1: *Improvement of the efficiency of aging fault detection and relaxation by at least 30%.*
The efficiency in the context of aging fault *detection* and *relaxation* can be seen as a combination of the following factors:

a) ability to effectively *predict* accumulation of aging effects in a circuit (adequacy of checkers and sensors in terms of early detection of aging effects);
b) ability to effectively *prevent* accumulation of aging effects in a circuit (adequacy of proposed rejuvenation and mitigation techniques);
c) ability to quickly *detect* intermittent faults in aged modules of the circuit (speed of fault detection and localization);
d) ability to quickly *isolate* faults and *prevent* error propagation in the aged system (speed of recovery from a fault);
e) HW overhead introduced by the instrumentation and data collection infrastructure;
f) minimizing circuit performance impact caused by the instrumentation and data collection infrastructure.

This deliverable specifically addresses items c) through f), while items a) and b) are considered in D3.3. The contribution of this deliverable to item c) and d) in terms of fast alarming of the OS about fault detection is given in Section 2.5. Section 5 is further contributing here by providing optimization frameworks for IJTAG network design and retargeting. Fault isolation (item d)) is also addressed in Section 3. Item e) is covered by KPI6 (see below) and addressed in Sections 2 and 3. Item f) is achieved due to the fact that BASTION approach is relying on reuse of test infrastructure (IJTAG networks), which is not interfering with the system's normal functions.

KPI6: *Linear complexity with respect to the size of chip of the major monitoring and handling parameters.*
In BASTION, the Fault Management strategy relies on IEEE 1687 reconfigurable scan networks (RSN) and IJTAG-compliant embedded instrumentation (checkers, monitors, sensors, etc.) as described in Section 2. Arranging instruments in balanced tree-like RSNs allows keeping *HW overhead* linear with respect to the number of instruments, while *fault localization time* would follow a logarithmic (better than linear) trend. Assuming that in a multi-core system, the number of instruments in a core would be rather constant, the HW overhead caused by the instruments

and RSNs is also linear with respect to the size of the chip. The additional relative overhead of adding the Instrument Manager (FSM), one per chip is better than linear. The contributions in the area of test generation and retargeting support this KPI by efficiently using given test infrastructures and by providing test patterns which are minimized with respect to the test length.

7 Bibliography

[1] IEEE, "IEEE Standard for Access and Control of Instrumentation Embedded within a Semiconductor Device," p. 1–283, December 2014.

[2] N. Stollon, On-Chip Instrumentation: Design and Debug for Systems on Chip, Springer US, 2011.

[3] R. Baranowski, M. Kochte and H.-J. Wunderlich, "Modeling, Verification and Pattern Generation for Reconfigurable Scan Networks," in *International Test Conference*, Nov. 2012.

[4] J. Rearick and A. Volz, "A Case Study of Using IEEE P1687 (IJTAG) for High-Speed Serial I/O Characterization and Testing," in *Proc. IEEE Int'l Test Conf. (ITC)*, Oct. 2006.

[5] M. Keim, T. Waayers, R. Morren, F. Hapke and R. Krenz-Baath, "Industrial Application of IEEE P1687 for an Automotive Product," in *Euromicro Conference on Digital System Design (DSD)*, Sep. 2013.

[6] A. Jutman, S. Devadze and J. Aleksejev, "Invited paper: System-wide fault management based on IEEE P1687 IJTAG," in *Int. Workshop on Reconfigurable Communication-centric SoCs (ReCoSoC)*, 2011.

[7] E. Larsson and S. Kenneth, "Fault management in an IEEE P1687 (IJTAG) environment," in *Proc. IEEE Int'l Symp. Design and Diagnostics of Electronic Circuits Systems (DDECS)*, 2012.

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

[9] A. Jutman, S. Devadze and K. Shibin, "Effective Scalable IEEE 1687 Instrumentation Network for Fault Management," in *Design Test, IEEE*, 2013.

[10] K. Petersen, D. Nikolov, 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.

[11] K. Shibin, S. Devadze and A. Jutman, "On-line Fault Classification and Handling in IEEE1687 based Fault Management System for Complex SoCs," in *Proc. 17th IEEE Latin-American Test Symposium (LATS'2016)*, 2016.

[12] G. Ali, A. Badawy and H. Kerkhoff, "Online Management of Temperature Health Monitors using the IEEE 1687 Standard," in *Test Standards Application Workshop (TESTA)*, 2016.

[13] F. Zadegan, D. Nikolov and E. Larsson, "A Self-Reconfiguring IEEE 1687 Network for Fault Monitoring," in *Proc, European Test Symposium (ETS)*, 2016.

[14] A. Ibrahim and H. Kerkhoff, "Analysis and Design of an On-Chip Retargeting Engine for IEEE 1687 Networks," in *Proc, European Test Symp. (ETS)*, 2016.

[15] M. Portolan, "A Novel Test Generation and Application Flow for Functional Access to IEEE 1687 instruments," in *Proc, European Test Symp. (ETS)*, 2016.

[16] A. Jutman, K. Shibin and S. Devadze, "Reliable Health Monitoring and Fault Management Infrastructure based on Embedded Instrumentation and IEEE 1687," in *Proc. of AUTOTESTCON'2016*, 2016.

[17] M. Portolan, B. Van Treuren and S. Goyal, "Executing IJTAG: Are Vectors Enough?," in

IEEE Design & Test, 2013.

[18] E. J. Marinissen and Y. Zorian, "Challenges in Testing Core-Based System ICs," *IEEE Communications Magazine,* vol. 37, no. 6, 1999.

[19] Y. Zorian, E. J. Marinissen and S. Dey, "Testing Embedded-Core-Based System Chips," *IEEE Computer,* vol. 32, no. 6, pp. 52-60, 1999.

[20] IEEE Standard 1500-2005 "IEEE Standard Testability Method for Embedded Core-based Integrated Circuits", New York: IEEE, 2005.

[21] O. Sinanoglu and T. Petrov, "Isolation Techniques for Soft Cores," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems,* vol. 27, no. 8, August 2008.

[22] T. L. McLaurin, "The Challenge of Testing the ARM Cortex-A8 Microprocessor Core," in *IEEE International Test Conference (ITC'06)*, 2006.

[23] A. L. Crouch, "IJTAG: The Path to Organized Instrument Connectivity," in *IEEE International Test Conference*, 2007.

[24] S. Makar and E. McCluskey, "ATPG for Scan Chain Latches and Flip-Flops," in *IEEE VLSI Test Symposium*, 1997.

[25] F. Yang, S. Chakravarty, N. Devta-Prasanna, S. Reddy and I. Pomeranz, "On the Detectability of Scan Chain Internal Faults – An Industrial Case Study," in *IEEE VLSI Test Symposium*, 2008.

[26] F. Ghani Zadegan, U. Ingelsson, G. Asani, G. Carlsson and E. Larsson, "Test Scheduling in an IEEE P1687 Environment with Resource and Power Constraints," in *Asian Test Symposium (ATS)*, Delhi, Nov. 2011.

[27] A. Dahbura, Ü. Uyar and C. Yau, "An Optimal Test Sequence for the JTAG/IEEE P1149.1 Test Access Port Controller," in *IEEE International Test Conference*, 1989.

[28] S. Makar and E. McCluskey, "ATPG for Scan Chain Latches and Flip-Flops," in *IEEE VLSI Test Symposium*, 1997.

[29] S. Makar and E. McCluskey, "On the Testing of Multiplexers," in *IEEE International Test Conference*, 1988.

[30] F. Ghani Zadegan, U. Ingelsson, G. Carlsson and E. Larsson, "Access Time Analysis for IEEE P1687," *IEEE Transactions on Computers,* vol. 61, no. 10, pp. 1459-1472, 2012.

[31] E. J. Marinissen, V. Iyengar and K. Chakrabarty, "A set of benchmarks for modular testing of SOCs," *Proc. ITC,* p. 519–528, 2002.

[32] J. Rearick, B. Eklow, K. Posse, A. Crouch and B. Bennetts, "IJTAG(Internal JTAG): A Step Toward a DFT Standard," in *Proc. International Test Conference (ITC)*, 2005.

[33] R. Baranowski, M. A. Kochte and H.-J. Wunderlich, "Scan Pattern Retargeting and Merging with Reduced Access Time," in *IEEE European Test Symposium (ETS'13)*, 2013.

[34] R. Krenz-Baath, F. G. Zadegan and E. Larsson, "Access Time Minimization in IEEE 1687 Networks," in *Proc. International Test Conference (ITC)*, 2015.

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

[36] Y. Fkih, P. Vivet, B. Rouzeyre, M.-L. Flottes, G. D. Natale and J. Schloeffel, "2D to 3D Test Pattern Retargeting Using IEEE P1687 Based 3D DFT Architectures," *IEEE Computer Society Annual Symposium on VLSI (ISVLSI),* p. 386–391, July 2014.

[37] M. Portolan, B. Van Treuren and S. Goyal, "Executing IJTAG: Are Vectors Enough?," *IEEE,* vol. 30, no. 5, pp. 15-25, Oct 2013.

[38] F. G. Zadegan, U. Ingelsson, E. Larsson and G. Carlsson, "Reusing and Retargeting On-Chip

Instrument Access Procedures in IEEE P1687," *IEEE,* vol. 2, no. 29, p. 79 –88, April 2012.

[39] T. Lengauer and R. Tarjan, "A Fast Algorithm for Finding Dominators," *Transactions on Programming Languages and Systems,* vol. 1, no. 1, July 1979.

[40] F. Ghani Zadegan, U. Ingelsson, G. Carlsson and E. Larsson, "Design Automation for IEEE P1687," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, Mar. 2011.

[41] F. Ghani Zadegan, G. Carlsson and E. Larsson, "Robustness of TAP-Based Scan Networks," in *International Test Conference*, Seattle, Oct. 2014.

[42] K. Shibin, S. Devadze and A. Jutman, "Asynchronous Fault Detection in IEEE P1687 Instrument Network," in *NATW*, 2014.

[43] K. Yamasaki et al., "External Memory BIST for System-in-Package," in *International Test Conference*, 2005.

[44] A. Carbine and D. Feltham, "Pentium(R) Pro Processor Design for Test and Debug," in *International Test Conference*, 1997.

[45] Z. Conroy et al., "Board Assisted-BIST: Long and Short Term Solutions for Testpoint Erosion – Reaching into the DFx Toolbox," in *International Test Conference (ITC)*, 2012.

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

[47] S. Alstrup, D. Harel, J. Clausen and M. Thorup, "Dominators in linear time," *SIAM Journal on Computing,* vol. 28, no. 6, p. 2117–2132, 1999.

[48] A. Biere, A. Cimatti, E. M. Clarke and Y. Zhu, "Symbolic model checking without BDDs," *5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'99),* p. 193–207, March 1999.

[49] N. Eén and N. Sörensson, "An extensible sat-solver," *Theory and Applications of Satisfiability Testing, 6th International Conference,* no. Selected Revised Papers, p. 502–518, 5-8 May 2003.

[50] N. Eén and A. Biere, "Effective preprocessing in SAT through variable and clause elimination," *Theory and Applications of Satisfiability Testing,* pp. 61-75, 19-23 June 2005.

[51] N. Eén and N. Sörensson, "Translating pseudo-boolean constraints into SAT," *Boolean Modeling and Computation,* vol. 2, pp. 1-26, 2006.

[52] S. Eggersglüß, K. Schmitz, R. Krenz-Baath and R. Drechsler, "Optimization-based multiple target test generation for highly compacted test sets," *19th IEEE European Test Symposium, ETS 2014,* p. 1–6, 26-30 May 2014.

[53] M. Gebser, B. Kaufmann, A. Neumann and T. Schaub, "Conflict-driven answer set solving," *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence,* p. 386, 6-12 January 2007.

[54] R. Krenz-Baath, A. Glowatz and F. Hapke, "Fault collapsing of multi-conditional faults," *16th IEEE International Symposium on Design and Diagnostics of Electronic Circuits & Systems, DDECS 2013,* p. 42–47, 8-10 April 2013.

[55] C. Liu, A. Kuehlmann and M. W. Moskewicz, "CAMA: A multi-valued satisfiability solver," *2003 International Conference on Computer-Aided Design (ICCAD'03),* p. 326–333, 9-13 November 2003.

[56] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang and S. Malik, Chaff: Engineering an efficient SAT solver, Las Vegas, Nevada: Proceedings of the 38th ACM/IEEE Design Automation Conference, 2001, p. 530–535.

[57] J. P. Marques-Silva and K. A. Sakallah, "GRASP: a search algorithm for propositional satisfiability," *IEEE Transactions on Computers,* vol. 5, no. 48, p. 506–521, 1999.

[58] N. Eén and N. Sörensson, "Temporal induction by incremental SAT solving," vol. 4, no. 89, 2003.

[59] D. Tille, R. Krenz-Baath, J. Schloeffel and R. Drechsler, "Improved circuit-to-CNF transformation for SAT-based ATPG," *IEEE European Test Symposium,* 2008.

[60] K. Yang, K.-T. Cheng and L.-C. Wang, "Trangen: a SAT-based ATPG for path-oriented transition faults," *Proceedings of the 2004 Asia and South Pacific Design Automation Conference,* p. 92–97, 2004.

[61] J. Baumgartner, A. Kuehlmann and J. A. Abraham, "Property checking via structural analysis," *Proc. 14 Intl. Conference on Computer Aided Verification (CAV'02).,* p. 151–165, 2002.