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

Test Programs Development Techniques

(Deliverable D4.1, generated by Task 4.1 - Functional test programs exploiting the embedded instrumentation)

Due date of deliverable: June 30, 2015

Start date of project: January 1, 2014 Duration: Three years

Organisation name of lead contractor for this deliverable: Poltecnico di Torino

Revision 1.0

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 information, please contact Matteo SONZA REORDA, e-mail: matteo.sonzareorda@polito.it

This document is intended to fulfil the contractual obligations of the BASTION project concerning deliverable D4.1 described in contract 619871.

Table of Revisions

Version	Date	Description and reason	Author	Affected sections
1.0	June 5, 2015	Structure created	M. Sonza Reorda	Contents
1.1	June 15, 2015	Sections 2 and 5 introduced	C. Lotz, M. Sonza Reorda	Section 2 and 5
1.2	June 15, 2015	Subtitle added General structure modified by adding Section 3 First draft content of Section 4 introduced	S. Devadze, A. Jutman, C. Lotz, M. Sonza Reorda	Section 3 and 4
1.3	June 18, 2015	Content added in section 1 and 3	M. Sonza Reorda	Sections 1 and 3
1.4	June 19, 2015	Content added/updated in section 4 and 5	S. Devadze, A. Jutman, J. Raik	Sections 4 and 5
2.0	June 23, 2015	General review of the document. Updated version of Section 6. Content added in section 7.	C. Lotz, M. Sonza Reorda	All
2.1	June 26, 2015	Tuning of the format	M. Sonza Reorda	All

Author, Beneficiary

Matteo Sonza Reorda, Politecnico di Torino
Jaan Raik, Tallinn Technical University
Artur Jutman, Sergei Devadze, Testonica Labs
Christophe Lotz, Aster Technologies

Executive Summary

This document focuses on the techniques developed within the BASTION project to develop functional stimuli to be used for the test of devices and systems. Special emphasis is devoted to the use of embedded instruments to increase the achieved defect coverage and to take into account aging phenomena.

List of Abbreviations

ASIC	- Application-Specific Integrated Circuit
ATE	- Automated Test Equipment
ATPG	- Automatic Test Pattern Generation
BIST	- Built-In Self-Test
CAD	- Computer Aided Design
DFT	- Design For Testability
EDA	- Electronic Design Automation
FP7	- European Union's 7th Framework Programme
FPGA	- Field-Programmable Gate Array
FMECA	- Failure mode, effects and criticality analysis ()
IJTAG	- Internal JTAG, a short name for IEEE 1687 standard and infrastructure collectively
JTAG	- Joint Test Action Group; often used as a short name of the IEEE 1149.1 standard and respective infrastructure including test access port and respective header on the board; also Boundary Scan
IST	- Information Society Technologies
NBTI	- Negative Bias Temperature Instability
NFF	- No Failure Found (also No Fault Found)
PCB	- Printed Circuit Board
PCBA	- Printed Circuit Board Assembly
SBST	- Software-Based Self-Test
SoC	- System on Chip

Table of Contents

1 Introduction

This report describes some of the activities performed so far within the WP4 (*Instrument-Assisted testing for NFF*) of the BASTION project, with special emphasis on the results achieved within the Task 4.1 (*Functional test programs exploiting the embedded instrumentation*).

The main focus of the reported activities has been on the integration of functional test (which plays a major role in PCB test) with embedded instruments existing both at board and device level. More in details, these activities found motivation in the assumption that more effective techniques for developing and supporting the functional tests may allow the detection of a wider spectrum of defects, thus successfully attacking the NFF issue.

As a first step, the BASTION partners performed an overview of what has been proposed and done in the past in the area of functional test, both at the device (where it is often denoted as *Software-Based Self-Test*, or SBST) and board level. A summary of this overview is reported in Section 2. A key message stemming from the overview of this Section is that functional test may represent an important tool to detect a wider set of defects at the device and board level. However, more knowledge about the relevant defects is required, and more mature techniques for generating functional tests (and possibly automating the generation) are definitely necessary.

Secondly, the partners performed an experimental analysis of which benefits could come to functional test at board level from the adoption of enhanced observability techniques. This enhanced observability could significantly increase the defect coverage, and could come at a relatively low-cost if we use some features already provided by most processors. These features include the debug ones, which allow the on-the-fly monitoring of the processor behaviour, for example accessing to the values flowing on the bus, and the so-called performance counters, which exist in many processors, mainly to support design and silicon validation. An experimental analysis to assess the benefits (in terms of stuck-at fault coverage) stemming from the adoption of different observability solutions when a functional test is performed is reported in Section 3. Special emphasis is devoted to assess the impact coming from the adoption of SBST solutions when applied for in-field test.

Section 4 is devoted to the description of an environment which allows to support the implementation of instruments at the board level using existing FPGAs. This environment could provide the support required to implement some of the observability solutions described in the previous section, as well as others. The work reported in this Section clearly paves the way for further increasing the observability features, e.g., by adding ad hoc observation points and monitors that can be accessed at the board level.

Section 5 describes a method which allows to slow-down the aging effects in a device by carefully selecting the functional stimuli applied during the idle periods of an application. The method is based on an electrical-level analysis of the values to be applied to the transistors lying on the critical paths, and describes some preliminary experimental results showing that suitable stimuli may significantly slow-down the aging effects. Once again, the method lies on the assumption that we can generate suitable functional stimuli, demonstrating that this could not only allow the detection of a significant amount of defects, but also prevent their occurrence.

When defining the test for a PCB, a crucial point lies in the generation of the whole test program resulting from the combination of the tests related to different components. Moreover, it would be highly beneficial if we could re-use any

information about the fault coverage provided by a given test for computing the global fault coverage at the board level. Section 6 describes a method devised by ASTER in cooperation with Eriksson that faces this issue, based on a hierarchical approach which re-uses existing functional tests, as well as the related information.
Finally, Section 7 draws some conclusions.

2 Overview of functional test techniques

This Section of the report overviews the state of the art in terms of functional test, starting from its definition and ending with the hot topics in the area.

2.1 Definitions and motivation for functional test

Different definitions exist for the concept of "*Functional test*".
Definition D1: in some case, a functional test is meant as a test, which does not rely on any Design For Testability (DFT) structure: hence, this test only acts on the system functional inputs, and only observes the system functional outputs.
Definition D2: in other cases, a functional test is intended as a test, which has been generated by only exploiting functional information about the target system (i.e., without knowing its structure). As a consequence, this test does not rely on any structural fault model, leading to possible limitations in its defect coverage capabilities and in significant difficulties even in computing a defect coverage figure.
The two definitions can sometimes be adopted simultaneously: for example, a test can be generated starting only from functional information about the system, and the test is only applied resorting to functional inputs and outputs.
Functional test may be adopted in different test scenarios and may be motivated by different reasons. Examples of usage of functional test include the following cases

- During the manufacturing test of a System on Chip (SoC), functional test may complement structural test because it may cover some defects that are not detected by the latter, e.g., because the former typically works at the system operational speed (while some DFT techniques do not), or because the functional test exercises the system exactly in the same conditions of the operational phase. Since this test is typically performed by the manufacturer, who owns full information about the device structure, it mainly matches definition D1.
- Before mounting a device on a board, it may be required by regulations or economically convenient to perform a test to check whether the device is fault free (independently on the test performed by the device manufacturer). This test (sometimes called *Incoming Inspection*) is performed by the system company and it is often based on a functional approach, only (typically because possible Design for Testability features are not documented by the device provider). Hence, this test matches definition D2.
- When addressing end-of-manufacturing board-level test, functional test is typically considered as the final step, which is supposed to complement the previous ones with specific goals (e.g., testing the interfaces), allowing to achieve the target defect coverage. This test is often based on stimuli generated by application engineers and intended to stress the board as in the normal operation phase. Once again, this test mainly matches definition D2.
- During the in-field test of a board (or PCB), it may happen that the DFT features of the composing devices are not accessible any more (e.g., because they require an ATE), or are not documented by the device providers. Hence, the only feasible solution for the system company in charge of developing the test is often based on the functional approach. Sometimes, the device provider does deliver a proper test (developed according to definition D1), and guarantees that it achieves a given Fault Coverage. This solution matches well the situation, when standards and regulations for safety-critical domains specify the fault coverage it must achieve. Otherwise, the functional test has to be developed (according to definition D2)

starting from the functions performed by each device, only. In any case, the availability of effective solutions (e.g., based on a functional approach) for performing an in-field test where the system works in the same conditions as in the operational phase is a key step to effectively face the NFF issue.

2.2 Functional test principles

In most cases, a functional test for an electronic system requires a suitable test program TP to be executed by the processor(s) inside the system; this test program is expected to produce different results when the system is affected by a fault; results may be observed on a suitable output port or correspond to values left in a specified area of memory. When peripheral modules are targeted, suitable data stimuli TD may be required to be applied to specific inputs, or some output data have to be observed on specific output signals.

When functional test is the selected solution, two major issues have to be considered:

- How to apply the functional test, i.e., where to store the test program TP, how to trigger the processor to execute it, how to retrieve and check the produced results; a common solution lies in storing the TP in a memory accessible by the processor (or directly in the processor cache), triggering in some way its execution, and then checking the results left in the data memory (*Software-Based Self-Test*, or *SBST*) [2];
- How to generate the functional test (in particular, the test program TP). Solutions to the first issue are typically dependent on the targeted system and to the existing constraints. Moreover, when in-field test is addressed, most of these tasks are commonly orchestrated by the Operating System.

2.3 Functional test generation

Generating suitable test programs for functional test has been the subject of numerous research efforts, starting from [1], where the authors proposed a method to manually generate a test program for a simple processor, knowing its Instruction Set Architecture, only. Hence, the method in [1] fully matches the characteristics of definition D2. A major advantage of that method was that it was experimentally shown to be able to reach a good stuck-at fault coverage (around 90%).

In the last decades, the approach was extended to target processor cores of increasing complexity, as well as specific system components, such as memories, peripheral components and interconnection networks, as it will be better detailed in the next sub-sections.

2.3.1 Processors

A good overview of methods targeting processor cores is reported in [2]. More recently, researchers focused on specific modules within modern processor cores, such as Branch Prediction Units (BPUs), Memory Management Units (MMUs), Reorder Buffers (ROBs), and Cache controllers, showing that in most cases it is possible to develop test program that are guaranteed to reach a high fault coverage, without requiring the knowledge of the detailed implementation of such modules. Interestingly, some of the faults affecting these modules do not produce wrong results, but rather force the processor to behave in a temporarily different way, typically requiring a longer time to complete the test program execution (performance faults). The detection of these faults may be particularly challenging, since it requires some techniques to observe the time behavior of the processor in a precise manner [39].

Recent efforts also targeted the development of functional test programs for multi-core processors [4] and GPUs [5].
The above methods mainly correspond to algorithms, allowing a skilled engineer to manually write a test program targeting a given module or a whole processor. However, the effort and time for achieving this result may be significant, and represents a major drawback of the functional approach. Previous efforts to automate the process, for example based on extensive simulation and evolutionary techniques [6], had a limited success, mainly due to the huge computational effort they require. Recently, it was shown in [7] and [19] that formal techniques can be successfully exploited to automatically generate functional test programs for a pipelined processor.

2.3.2 Memories

Since memories correspond to an increasingly large fraction of a system, their test may represent an important target. Although the typical solution lies in the adoption of BIST, there are cases in which the functional approach is also of interest. In these cases the common solution lies in developing a test program, which performs on the target memory modules the same sequence of read and write operations mandated by a given March algorithm. In principle, this guarantees that the same defect coverage is achieved, although some defects may be missed due to the longer time between two consecutive accesses to memories [9]. An interesting extension of the same idea allows the test of cache memories resorting to suitably written test programs: [10] proposes a set of rules which allow to automatically transform any March algorithm into the corresponding test program. Reference [11] extends the same approach to L2 caches.

2.3.3 Peripherals and interconnections

When targeting communication peripheral components, the functional approach requires the combined action of the processor, programming the component and exercising/observing it on one side, and that of an external body (e.g., an ATE), exercising/observing the component on the other side [18]. For in-field solutions, where the ATE can hardly be exploited, a loop-back connection is often adopted. A similar approach can be adopted for system peripherals, such as Interrupt and DMA controllers [9].
Several methods have been proposed to develop a functional test able to effectively detect faults in the interconnection structures within a system. As an example, the work in [12] targets structural faults in a Network on Chip.

2.4 Current trends in functional test

In the last decade methods to generate functional test programs running under specific constraints (e.g., in terms of power [13]) or providing diagnostic information [14] were developed.
Both academia and industry are also exploring the cost and benefits stemming from the integration of the functional approach with a limited hardware supported, as proposed in [15] and [16].
Another important topic in this domain is the one of how to reduce the size and duration of functional test programs while preserving their fault coverage. Preliminary solutions and results in this direction are shown in [20].
Finally, it is worth mentioning that when more regular processor architectures are targeted, such as those of VLIW processors, it is possible to adopt a hierarchical

approach, in which the global test program can be built, once the test program for each composing unit is known [17]. In this way the major drawback of the functional approach, corresponding to the huge cost for manually generating the test (as a consequence of the lack of automated tools), can be successfully faced.
Following this approach, new efforts are expected to be taken in the close future, aimed at automating the generation of functional test programs (to be adopted in different scenarios) exploiting some limited but well-defined information coming from the core or device producer, as well as from the system designer.
When targeting board test and NFFs, combining functional test with the access to embedded instruments and on-board FPGAs is also crucial to increase the defect coverage with reasonable costs.

3 Functional test and embedded instruments

This section will explain how functional test may benefit from the access to embedded instruments and report some figures to support this claim. For the purpose of this section we do not enter into details about how the observability is obtained: different solutions may be followed (depending on the adopted scenario) including the usage of the new IEEE 1687 standard.

3.1 Introduction

As anticipated in the previous sections, in several domains (e.g., automotive, biomedical, space, aircrafts) electronic systems are now commonly used for mission- or safety-critical applications. In these domains, a misbehavior due to a defect affecting the hardware may cause catastrophic effects (e.g., hurting humans, or provoking huge economical losses). Hence, solutions must be devised and adopted to minimize the probability that a defect arises, and to suitably face it when it appears. Moreover, solutions are adopted, able to effectively detect possible faults, hopefully signaling their presence before they cause serious consequences. When considering the latter point, different solutions have been proposed. Some of them are based on changing the hardware (Design For Testability). Others only act on the software. Clearly, the best solution depends on the specific constraint of each scenario. Standards and regulations (e.g., IEC 61508 for generic safety-related systems, ISO 26262 for automotive applications, DO-254 for avionics) also play a key role in this scenario.

SBST is currently adopted in different test scenarios (both for end-of-manufacturing and in-field test) and may be motivated by different reasons. Sometimes it complements other solutions (e.g., those implementing some Design for Testability technique), in other cases it is an alternative to them.

In particular, when considering the in-field test of a system, it may happen that the Design For Testability structures of the composing devices are not accessible any more (e.g., because they have been destroyed or made inaccessible to better protect the system safety), or are not documented by the device providers. Hence, the only feasible solution for the system company in charge of developing the in-field test is to adopt the functional approach.

It must also be underlined that, since the functional approach by definition tests the system in the operational mode (without moving to a test mode and without reconfiguring in any manner the system), it is guaranteed not to produce any form of overtesting, and may provide advantages in terms of defect coverage with respect to some DFT solutions.

When comparing the SBST solutions adopted for end-of-manufacturing test with those for in-field test, a major difference lies in the fact that the former ones use a tester, and can thus benefit of full accessibility to the input and output signals of each device. This greatly increases the fault controllability and observability with respect to the latter case, where the test must be performed without resorting to any tester and must comply with the constraints given by the application. As an example, the memory area usable by the test will be limited to a specific size and location. On the other side, in-field constraints may be quite severe, and in some cases cause some faults to become functionally untestable [38].

Concerning observability, in in-field SBST the effects of any possible fault are typically observed by just looking at the values left in some specified memory locations at the end of the test program execution. This limited observability may

significantly reduce the achievable fault coverage. Moreover, some specific fault categories are known to be untestable by just looking at the final memory content. In particular, faults that only affect the time behavior of the processor (e.g., by delaying some operation) cannot be detected in this way. These faults are known as *performance faults* [36] and can be easily found in modules such as cache controllers [25] and Branch Prediction Units [24]. The test of these faults can be successfully faced by resorting to the so-called *performance counters* existing in most of the current microprocessors and microcontrollers [3]. Alternatively, one can resort to special hardware modules that can be easily and inexpensively added to a processor, able to monitor the bus during the execution of a program and computing a signature.
This section first of all describes the different solutions that can be adopted in practice to support the observation of fault effects when SBST is adopted for in-field test, discussing the advantages and limitations of each of them.
Secondly, we use a test case to quantitatively evaluate the benefits and cost of each solution, for a given functional test that targets the cache controller logic in a dual-core LEON3 system.
Thanks to the reported information, the reader can have a better understanding of the advantages and disadvantages provided by the different solutions.
This section is organized as follows: sub-section 2 describes the different observability solutions we considered here. Sub-section 3 describes the experiments we performed on a system based on two LEON3 processors to quantify the effects stemming from the adoption of the different observability solutions. Finally, sub-section 4 draws some conclusions.

3.2 Observability Solutions

In the following, the main solutions that can be adopted to guarantee observability during the in-field SBST testing of a processor-based system are described. We also list a few solutions assuming that we consider them just as references, although they usually cannot be adopted in in-field SBST. All the considered solutions are graphically shown in Fig. 1.

We assume that the targeted faults are those inside an internal module of the processor core. The design and interface of the target module usually cannot be modified by the test designer and are assumed to be given. For every solution, the adopted mechanism as well as the main advantages and disadvantages are detailed, and a preliminary analysis about the forecasted coverage is reported.

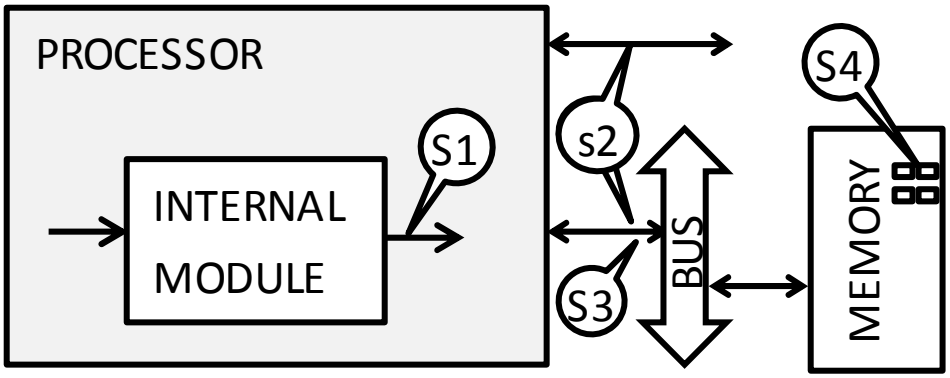

Figure 1*:* Generic system under test - the observation points adopted by the techniques described in the text are highlighted

3.2.1 S1: observability at the module level

This is the maximum level of observability that the testing process can achieve. At this level, it is assumed that all the output ports of the considered module are available for being observed. This observability approach is commonly used during simulation and fault simulation processes. However, this approach is not feasible during in-field testing because the module output ports usually do not coincide with the circuit pinout, and even in that case, it is not affordable to continuously observe the circuit behavior without resorting to additional hardware.

This observability solution is thus introduced here only as a reference, because it establishes an upper bound to the results obtainable through functional testing approaches.

3.2.2 S2: observability at the processor level

This level assumes that the observability is performed at the processor level, i.e., the processor core outputs are monitored.

Clearly, the observability we can get with S2 is lower than with S1 because the fault effects should be propagated from the internal module outputs to the processor outputs. In most of the cases, propagating the faulty behavior requires an additional effort in order to reach the processor outputs without masking the fault behavior. In the case of a functional testing approach based on test programs, this additional effort implies the addition of specific instructions able to propagate the fault effects to the processor outputs. As an example, faults within an arithmetic unit can be easily activated by executing suitable arithmetic instructions (thus propagating their effects on the module outputs), and can then be made observable on the processor outputs via any instruction writing to memory the result of the arithmetic operation.

Faults may also exist that, even with the addition of instructions, cannot be observed on the processor outputs. This situation may happen when the processor design includes some redundant circuitry, for example left from previous releases or included for future extensions of the design. Clearly, the related faults can be classified as untestable. However, their identification may often represent a relevant problem.

This observability solution is the one adopted during end-of-manufacturing test, when the processor is tested resorting to an ATE. It can hardly be adopted by in-field SBST and it is once again considered here only as a reference.

3.2.3 S3: observability at the system bus level

At this level we observe the system bus. When adopted in-field, this solution requires a dedicated hardware able to monitor the bus during the relevant time slots, possibly complemented with a compaction mechanism to be used during the same process. When considering SoC testing, this solution may require the system to include a hardware I-IP, such as the one described in [36].
Compared to the coverage results obtained in the previous levels, this observability solution may produce a new coverage reduction. This fault coverage loss is motivated by the reduction of the observed signals, as they are a subset of the signals observed with S2. Faults affecting the circuitry for supporting an external coprocessor may represent a good example of this situation. The effects of such faults can be observed at the processor level on the processor output ports to the coprocessor, but cannot be observed at the bus level (unless the test program propagates the faulty values to the memory with suitable instructions).
The expected fault coverage reduction will be more or less important, depending on how much related is the module under analysis with the bus activity.

3.2.4 S4: memory content observation

This observation mechanism assumes that the test program collects the testing related information and saves this information in the system memory. Hence, at the end of the test program run the processor itself or another module (e.g., another processor) may perform an access to the specific memory cells in order to check the memory content, verifying their values with respect to the expected ones.
The information collected by the test program may be based, for example, on a single signature, collected, compacted and then at the end of the process saved in the memory in few memory cells. On the other hand, the information saved by the program may be written on a collection of several memory cells where the test program saves the testing results, according to the targeted module characteristics [37].
This solution is the most commonly adopted for in-field SBST.
Since the testing results are saved only at the end of the test program execution, some performance faults may escape when using this observability mechanism with respect to solutions S1 to S3. For example, in the case of Branch Prediction Units, some performance faults may not modify the final test program results, but only delay the actual execution time [3].

3.2.5 S5: observability via Performance Counters

Performance counters (PCs) exist in many processors. These counters measure the number of occurrences of different internal events, making easier their observation from the outside, mainly for design validation, performance evaluation and silicon debug. Their values can normally be accessed via software. Hence, a test program may read the value of a given PC, execute a sequence of instructions exercising a given module, and the read again the value of the PC, comparing it with respect to the expected value. Possible differences may allow the detection of faults inside the module.
The most common types of PCs include those that count internal events related to:

- caches, counting the number of miss and hit events;
- branch prediction units, counting the number of correctly or incorrectly predicted branches;
- pipeline stages, counting the different types of stalls;
- memory management units, counting the number of hit/miss accesses to the TLB;
- exception units, counting the number of triggered exceptions, often divided by type.

These counters are quite common in general-purpose high performance processors, while their adoption is rarer (although growing) in microcontrollers for embedded applications.

The usage of these counters as part of the observability mechanism adopted by a testing procedure has been first proposed in [3]. They are crucial for the detection of some specific types of faults, such as performance faults. Moreover, they can make easier to test faults belonging to some modules, such as Branch Prediction Units, Cache Controllers, TLBs. They may also be used to support the test of specific modules within the pipeline, such as those controlling the activation of stalls. Regarding observability issues, the PCs may provide deeper details on internal module events that may not reach the output ports.

3.3 Experimental results

In order to experimentally validate the previous assumptions, we implemented a dual-core multiprocessor system based on the LEON3 processor core, and performed different fault simulation campaigns, targeting the stuck-at fault model. Every one of the previously introduced observability solutions was separately implemented and evaluated. The different fault coverage results were gathered in different observation conditions, exploiting the same excitation patterns generated by a test program running on the system processors. In the rest of the present section the experimental setup is better detailed and the fault coverage results are presented and analyzed.

3.3.1 System setup

The experimental platform is a dual-core system based on the open source LEON3 processor provided by Aeroflex Gaisler AB [38] and is shown in Fig. 2.

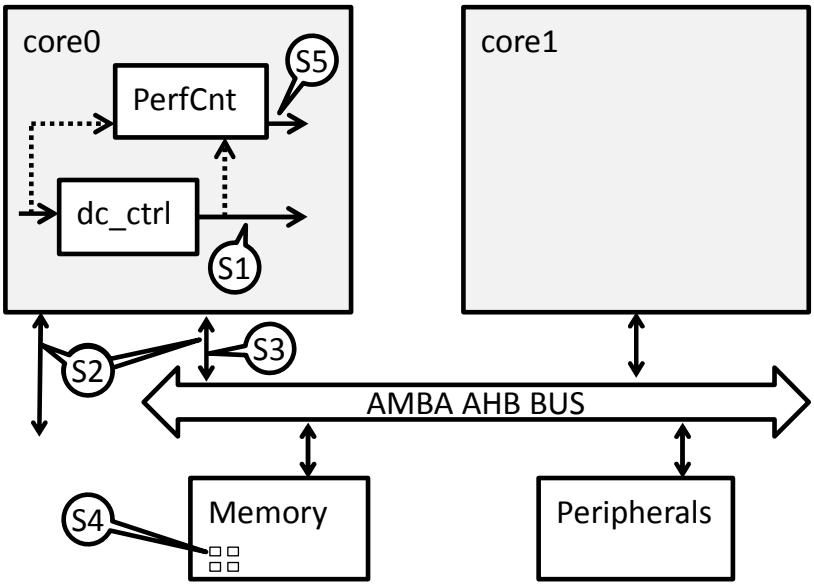

Figure 2*:* Dual core system under test

LEON3 is a highly configurable SPARC V8 7-stage pipelined processor with independent instruction and data caches. Both caches were configured as 1 way, 1Kbyte, 16 bytes/line. The data cache policy is always write-through, no allocate on miss. The snoop mechanism was enabled to assure cache coherency.

The internal module we selected for our experiments is the data cache controller (*dc_ctrl* in 0) of *core0*, one of the processors in the dual-core system, and the targeted faults are the stuck-at faults inside this module. The whole system was synthesized using the Synopsys SAED32 standard cells library [39]; the *dc_ctrl* module counts with 23,958 faults.

A test program was created based on the techniques described in [37] and [39]; in these papers, the derived test programs targeted specific functionalities inside the data cache controller. In addition, the test program used herein includes the programs provided by Aeroflex Gaisler AB to verify the functionalities of the cache controller.

More in details, the part of the test that targets the snooping logic, as described in [39] runs concurrently on both processors, while the rest of the program runs only on *core0*. The test program execution time is about 300k clock cycles.

In order to better compare the results, the test program was exactly the same in all the experiments, even when parts of it, like the signature evaluation, are only used in one of the observability solutions.

In order to better assess the observability solution based on performance counters, the cache controller in *core0* was enriched by adding some of the performance counters usually available on commercial processors. For this purpose, a new block, called *PerfCnt* in 0, fed by a subset of the data cache controller ports was designed. The set of performance counters can be classified in the following two groups:

- *AMBA bus events*: a group of 5 counters triggered on the occurrence of specific bus transfers (write, read, byte, half word, and word transfers). These counters take their inputs from the connections between the data cache controller and the AMBA AHB interface.
- *Cache operation events*: 3 counters for cache read hit, cache write, and cache line invalidation because of snoop events. The inputs of these counters come from signals connecting the data cache controller with the cache memory and the integer unit.

Using the system described above, a set of fault simulation experiments was performed, changing each time the observed signals to mimic the different observability solutions presented in the previous section. We used Synopsys TetraMAX ATPG tool (version J-2014.09-SP2) in our experiments.

The different observation solutions considered are listed below, indicating the set of observed signals for each case. The observation points are also shown in 0

- S1: all the output ports of the data cache controller module *dc_ctrl*.
- S2: all the output ports of *core0*.
- S3: the outputs of *core0* that are connected to the AMBA AHB bus.
- S4: it only observes the final signature computed by the test program out of the data written in memory.
- S5: only the output ports of the performance counters module described above.

As we mentioned in the previous sub-section, in an in-field SBST approach the S4 and S5 solutions are the only ones feasible, without requiring the use of any additional hardware devices.

By construction, it can be stated that the higher coverage is the one obtained resorting to the S1 solution. For all the other solutions, the observed signals are a subset or a transformation of the signals observed in S1. So, if the effect of a fault inside the *dc_ctrl* module is observable for example at the AMBA bus level, it can also be observed on the *dc_ctrl* ports. In a similar way, the following inclusion relationships can be assured between the sets of detected faults and between the sets of undetectable faults in the different solutions. The name in parentheses identify the solution and the two letter code denotes the fault class (*DT*: detected, *UD*: undetectable).

$$DT(S1) \supseteq DT(S2) \supseteq DT(S3) \supseteq DT(S4) \quad (1)$$

$$UD(S1) \subseteq UD(S2) \subseteq UD(S3) \subseteq UD(S4) \quad (2)$$

$$DT(S1) \supseteq DT(S5) \quad (3)$$

$$UD(S1) \subseteq UD(S5) \quad (4)$$

3.3.2 Results

The fault simulation results are summarized in Fig. 3 and TABLE I. The inclusion relationships introduced in the previous sub-section were verified by checking the detailed fault lists for each solution, and are reflected in the sizes of the fault sets shown in the figure.

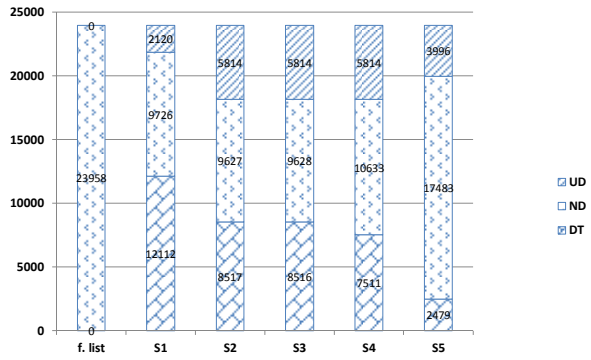

Figure 3*:* Fault simulation results

TABLE I. FAULT SIMULATION RESULTS

Class	f. list	S1	S2	S3	S4	S5
Detected (DT)	0	12,112	8,517	8,516	7,511	2,479
Not Detected (ND)	23,958	9,726	5,814	9,628	10,633	17,483
Undetectable (UD)	0	2,120	9,627	5,814	5,814	3,996

The gathered results show, as expected, a fault coverage reduction trend on the different experiments. The achieved fault coverage (even by S1) is mainly due to the high amount of redundancy (hence, untestability) existing in the module, which could not be removed by the synthesis tool. More in detail, the reader can notice that the fault coverage achieved by the S2 and S3 solutions is about 70% of the one of the S1 solution. Interestingly, the S2 solution covers only one fault more than the S3 solution. This was partly expected, because the data cache controller module we are considering is strongly memory related. A more appreciable coverage difference between the S2 and S3 solutions would be expected for a non-memory related module (e.g., an arithmetic unit, or the interrupt controller).

The provided results also show that S4 covers about 88% of the faults covered by S3, and S5 can cover only 33% of the faults covered by S4. However, it is important to mention that S4 and S5 are usually adopted together. Thus, it is pertinent to analyze the intersections and differences between both of the sets of detected faults.

0 schematically shows the effect of adding the results obtained by S4 and S5. Including the S5 observability adds about 3.5% to the fault coverage obtained by S4 alone, reaching 91.5% of the coverage obtained by S3. Remarkably, there are some faults (104 faults) covered by the S5 and S3 that escape to S4; these faults are associated to the group of performance counters related to AMBA bus events. This group of faults shows a contribution of S5 which partly compensates the loss in fault coverage moving from S2 to S3.

On the other side, there is another set of faults (181 faults) covered only by S5 that escape not only to S4 but also to S3. This means that during the execution of the test program the effect of these faults is unobservable at the bus level, but modifies the behavior of the second group of performance counters, those that count events related to the cache operation. The test program is successful in exciting these faults but is not able to produce observable modifications at the bus level. In this case the observation of the performance counters contributes to the overall coverage by making these faults observable.

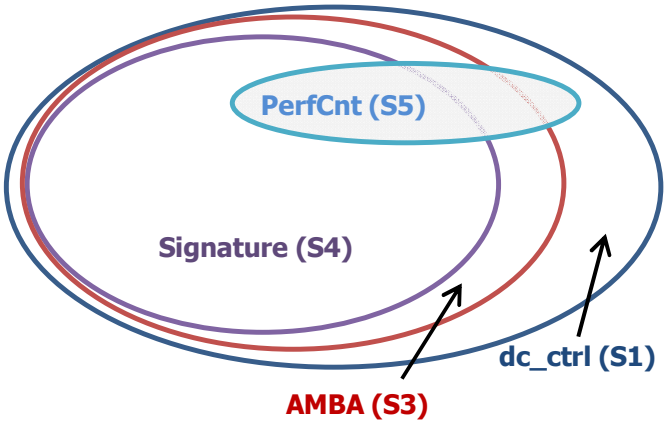

Figure 4*:* Coverage of combined Signature (S4) + Performance counters (S5) observation solutions

An additional experiment was performed trying to understand if a wider set of performance counters may increase the final fault coverage, and how much it can be enhanced. In this experiment, the data cache controller subset of signals used as inputs by the performance counters module were observed. Denoting by S6 this new observability solution, the following extensions to the above inclusion relationships can be stated:

$$DT(S1) \supseteq DT(S6) \supseteq DT(S5) \quad (5)$$

$$UD(S1) \subseteq UD(S6) \subseteq UD(5) \quad (6)$$

Consequently, the resulting coverage is an upper bound to the coverage that can be observed by processing the *PerfCnt* module inputs. The value of this bound was 511 faults, meaning that the 2.4% coverage enhancement provided by the performance counter block can potentially be extended to a maximum of 4.2% of the reference coverage.

Another interesting result from this new fault simulation is that, when observing the performance counters inputs, the set of undetectable faults is identical to the one obtained with S1. This means that the same faults detected by S1 could potentially be detected by adding new Performance Counters and devising a suitable input sequence.

3.4 Conclusions

When adopting SBST for in-field test of a processor-based system, as it is often required by standards and regulations for safety-critical applications, the achieved fault coverage is often affected in a strong manner by observability issues.

In this Section we analyzed the different solutions that can be adopted, both from a theoretical and from an experimental point of view. We adopted a dual-processor system based on the LEON3 processor to gather quantitative data about the faults that can be detected using the different solutions.

We showed first that moving from end-of-manufacturing to in-field SBST the fault coverage may significantly drop. Secondly, we experimentally demonstrated that a careful combination of observability mechanisms based on checking the memory content at the end of the test program execution with the information coming from the Performance Counters existing in many processors allows achieving a fault coverage

not far from the maximum one. More in general, a suitable set of Performance Counters may allow achieving nearly the same Fault Coverage achieved by end-of-manufacturing test.

4 In-field PCBA test framework

This section provides a description of BASTION contributions to the topic of in-field testing of Printed Circuit Board Assemblies (PCBAs) with the help of embedded instruments. We present an approach for using instrumental IPs that can be temporally embedded into on-board programmable devices (FPGAs) to facilitate in-field (online) test of PCBA. In particular, the developed framework can be used for testing of PCBAs at power-up. In the end of the section we will also describe the combined usage of functional test techniques and embedded instruments for increasing defect coverage of functional tests.

4.1 Introduction

Despite the fact that modern complex electronic devices are thoroughly tested after manufacturing at production facility, there is a notable lack of methods applicable for in-field testing of electronics. In particular, in order to ensure that produced Printed Circuit Board Assemblies (PCBAs) contain no defects they are tested at manufacturing line by various methods, such optical and X-Ray inspections, in-circuit test, functional test, Boundary Scan (JTAG). In particular, Boundary Scan (BS) test technique is widely used at production to verify that manufactured PCBAs have no defects (i.e., shorts and opens) on interconnection lines between digital components (integrated circuits) mounted on PCB [41].

However, testing at production will not help to identify defects that can emerge during in-field operation of an electronic device. The lack of in-field PCBA test methods is partially stipulated by high costs of overhead needed to provide this functionality, including extra hardware (on-board test controller), requirements for special design-for-test (DFT) structures on PCBA, additional engineering efforts for development of in-field test application scheme, etc. As a result, in-field PCBA test concept remains to be relatively unpopular among designers of electronic products.

In this section we present a technique for *Board Built-In Self-Test* (BBIST) which is capable to carry out in-field PCBA tests with the help of newly developed embedded instruments. The proposed technique relies on the usage of FPGA devices that are often used on modern complex PCBAs. The idea behind BBIST is to embed Boundary Scan test procedures into an FPGA, so that BS test can be repeated each time electronic device is powered-up or reset.

The proposed method does not require any extra FPGA device to be mounted on PCBA, as the existing FPGA can be re-used as a test controller. During normal (functional) mode of operation of an electronic device, the FPGA is loaded with functional design and is carrying out its normal operational tasks. However, during test stage (at power-up or after reset) the same FPGA is configured with a special BBIST instrumental IP which converts it into an embedded tester. Despite certain amount of DFT still needs to be introduced for BBIST implementation (e.g., extra I/O lines on the FPGA to control the tests application, extra space in storage device to accommodate the BBIST instrument IP bitstream), this overhead is very small in comparison with to the overall complexity of an average PCBA.

Another advantage of BBIST is that this solution is capable to re-use JTAG/Boundary Scan patterns that are typically generated for the purpose of manufacturing testing. This allows to automatically convert test procedures created for a test station at production site into patterns suitable for in-field BBIST. As a result, extra development efforts required to deploy the solution are minimized.

Moreover, it is possible to combine BBIST instruments with functional tests applicable to programmable ICs (such as CPUs and microcontrollers) in order to increase defect coverage of functional test programs and reduce the test time. In the latter case, the BBIST IP acts like an additional observation module which monitors functional test program running on a microprocessor using the debug and trace port of this CPU. By analyzing the trace information arriving via the debug channel, the BBIST instrument is capable to detect inconsistencies in functional test program execution flow that are caused by faults inside the microprocessor.

4.2 Board BIST instrument concept

The BBIST instrument IP is a special FPGA design intended to test on-board components and interconnections during the board boot-up process, i.e., to carry out *Power-On Self-Test* (POST) test procedures on PCBA. The developed test technique is intended to be applied at the PCBA power-up/reset and does not require any user interaction or usage of external test and measurement hardware. The overall concept of BBIST is given in Figure 5.

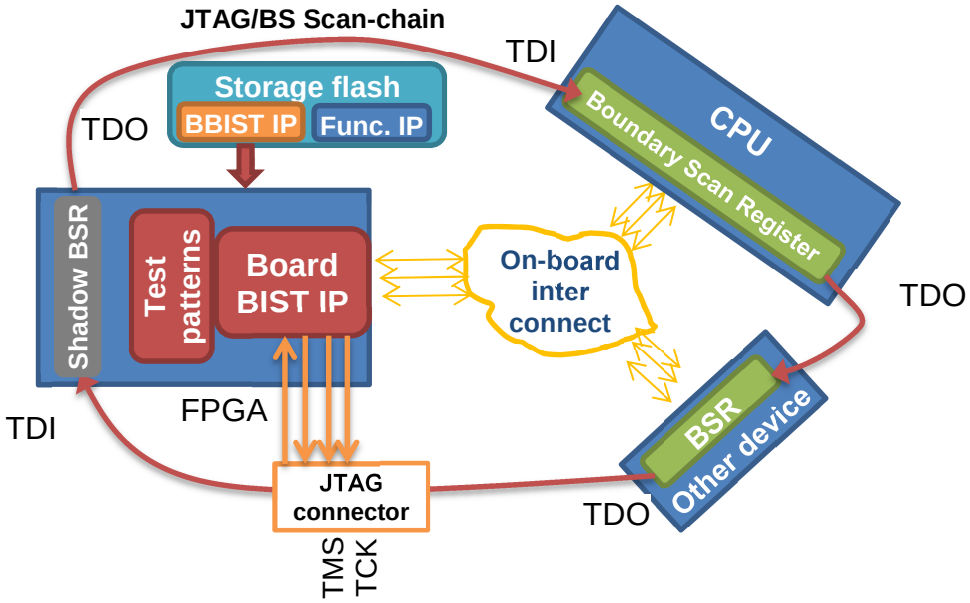

Figure 5. Overall Board BIST concept

Figure 5 presents a typical configuration of a PCBA. This example PCBA contains an FPGA, a CPU and some other complex digital devices. Typically, all such devices incorporate IEEE 1149.1 (Boundary Scan) facilities [42] that allow execution of BS interconnection tests on PCBA. These tests are applied by means of an external BS controller that is attached to an on-board JTAG connector. The test patterns are shifted into the JTAG scan-chain via the Test Data Input (TDI) port of the connector and are captured on the Test Data Output (TDO) port. Two more ports – Test Mode Select (TMS) and Test Clock (TCK) are used to control JTAG tests application flow.

Normally, BS test is carried out only at the production facility. After the board is produced, the JTAG infrastructure on the PCBA is rarely used and typically remains in idle state. However, it still exists and can be used also for in-field testing. For this purpose, a special DFT has to be implemented on PCBA (Figure 5) that would allow

the BBIST IP to gain control over the JTAG infrastructure. The overall behavior of the BBIST is described in the following.

When PCBA is powered up in the field, the on-board FPGA device becomes initially configured with a Board BIST instrument IP, which performs the required test procedures acting as an embedded Boundary Scan controller. After the test is finished successfully, the BBIST IP triggers the re-configuration of the FPGA with a functional IP and hand-offs control to it. Both functional and BBIST IPs can be stored as separate binary images in a single configuration storage device (e.g., Xilinx Configuration Flash, SPI flash or similar storage devices) at different addresses. Modern FPGAs support methods for triggering reconfiguration by means of currently operating design and loading new configuration image from the storage device. This scheme ensures small overhead, since no extra devices should be placed on board (however the storage device should be large enough to accommodate both types of firmware).

The goal of the BBIST instrument is first to check the integrity of the JTAG infrastructure and then test the connection of ICs that are included into the JTAG scan-path. In particular, standard BS tests are able to detect opens/shorts on pins of these devices as well as test interconnection lines between them. In certain cases, the BBIST IP can also detect faults on the interconnection between BS-enabled devices and non-BS devices (e.g., RAM, Flash, sensors). The Boundary Scan test patterns (from manufacturing tests) can be re-used by the BBIST instrument (however, some extra conversion may be required to adapt them).

Furthermore, the BBIST instrument can also be connected to the debug and trace port of the CPU device. In that case, the BBIST IP is able to monitor the trace information and detect faults occurring in the CPU during the execution of the functional test program.

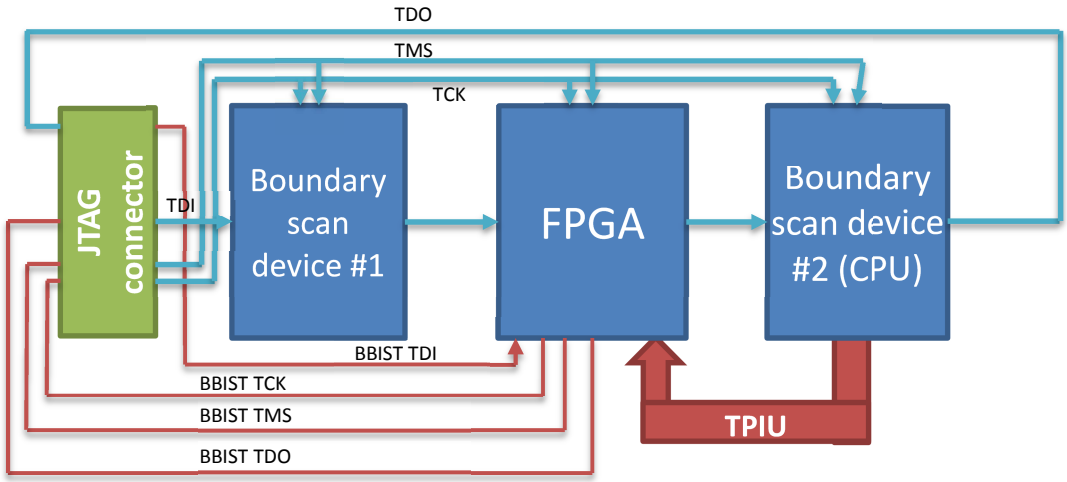

Figure 6. Extra on-board interconnect required for embedded BBIST

A small amount of on-board DFT has to be added to PCBA in order to support the usage of the BBIST instrument. These changes are listed below and are schematically presented in Figure 6 (extra DFT structures are shown by red lines):

1) Spare FPGA I/O pin (BBIST TMS) connected to the JTAG TMS line (the FPGA becomes the JTAG master)
2) Spare FPGA I/O pin (BBIST TCK) connected to the JTAG TCK line (the FPGA drives the test clock)
3) Spare FPGA I/O pin (BBIST TDO) connected to the JTAG TDI line (the FPGA can drive test data into the scan-chain)
4) Spare FPGA I/O pin (BBIST TDI) connected to the JTAG TDO line (the FPGA can capture test data coming out of the scan-chain)
5) (Optional) Connect TPIU (Trace Port Interface Unit[1]) of CPU to spare I/O pins of the FPGA. This is required only if the BBIST IP is used in combination with functional test programs executed on the CPU.

In total, 4 extra pins of the FPGA have to be reserved for BBIST (and 4 extra nets on PCB have to be routed). In addition, the debug and trace port (TPIU) can be connected to the FPGA I/O pins to allow BBIST instrument to monitor the execution of functional test programs on the CPU. The width of the TPIU depends on the implementation of debug facilities of a particular microprocessor.

Note that the above mentioned changes in PCBA design do not block the capability of carrying out normal BS/JTAG test afterwards. This is especially useful in case the BBIST IP has detected any failure on the PCBA and the failed product is being returned to the manufacturer for further inspection and repair.

4.3 Board BIST procedure

During the application of BBIST the following power-on self-test stages are performed after PCBA initialization:

1. The target FPGA (the one which will hold is BBIST IP) is started, while other devices on the board are kept in functional reset;
2. The FPGA configures itself with the BBIST instrument stored in the first part of the configuration Flash (which is the default source for FPGA configuration);
3. The BBIST IP performs infrastructure, interconnect and other self-tests on the PCBA;
4. If no failures are detected, the content of the FPGA is overwritten with the functional IP and the board start-up sequence proceeds in the normal way.

Test patterns for in-field self-tests are also loaded from the FPGA configuration Flash and stored in an internal memory of the FPGA (BRAM). On every iteration, the BBIST instrument IP takes the next test pattern and starts to shift this pattern into the JTAG scan-chain (using the Shift-DR state of the JTAG TAP). In this way, the Boundary Scan Registers (BSRs) of each of the BS-devices on-board are loaded with test stimuli. The only exception is the FPGA itself, which cannot use its BSR for test purposes since the usage of BSR will disconnect the internal BBIST IP logic from the FPGA I/O pins controlling the JTAG scan-chain. Instead of the BSR, the FPGA with BBIST IP contains the so called shadow BSR register (i.e., a shift register implemented in FPGA logic realizing the same functions as BSR, see Figure 6).

[1] For simplicity, we adopted here the vocabulary used by ARM cores.

Shadow BSR will drive the corresponding test-stimuli directly to the I/O pins of the FPGA involved into the test.

After the test pattern is applied (Update-DR state of JTAG TAP) and new values are captured (in Capture-DR state), the IP starts to shift-in the next pattern. At the same time, the results of the previous test are shifted-out and compared with the expected values (that are loaded in the BRAM along with the test patterns). In case of mismatch, the system is considered to be defective.

In the next sub-sections we will discuss the embedded BBIST stages in details.

4.3.1 Infrastructure self-test

In this stage, the BBIST IP verifies that the on-board JTAG infrastructure is operational and is capable to perform the test. This is a prerequisite for all other types of tests. In particular, the BBIST IP has to verify that it is capable to shift-in patterns into the scan-chain and receive the correct responses back (i.e., shift the scan-chain through).

At the same time, the BBIST IP identifies all the other devices in the scan-chain to ensure that its configuration matches the actual scan-chain on board. As it is specified by the IEEE1149.1 JTAG standard [42], all BS-capable devices on the PCBA should start-up in JTAG "IDCODE" state. Therefore, the BBIST IP is capable to scan the IDCODE values out of the scan-chain and compare them with the expected ones.

The next step is to verify that the BS devices can work in test mode. For this purpose, the BBIST IP forces all other BS devices to switch into "SAMPLE/PRELOAD" JTAG mode that is used to pre-load a test pattern into the scan-chain. Afterwards, the scan-chain is shifted through to determine its length. If the length of the scan-chain matches the expected one, then it means that all the devices (except the FPGA with the BBIST IP) were successfully switched into the "SAMPLE/PRELOAD" mode.

4.3.2 Interconnect self-test

In this test step the BBIST IP places all devices in the scan-chain (except the FPGA) into the "EXTEST" JTAG mode. The FPGA itself is forced into the "USER1" JTAG user mode in which a special register (shadow BSR, discussed in the following) implemented inside the BBIST IP is placed between the test data input and the test data output of the FPGA. This shadow BSR is used to drive the test stimuli to I/O pins of the FPGA and capture the responses back.

The BS test patterns are applied one-by-one and the test responses are fetched back and analyzed by the BBIST IP. In case of mismatch, the failure is detected on the interconnect. Note, that the BBIST IP is only capable to detect the presence of interconnect faults, but is unable to give any further diagnosis about the fault type or location.

A special attention should be paid to the pins of the FPGA that are controllable in traditional Boundary Scan test (i.e., have a BS cell behind) but are inaccessible from the FPGA logic (in fact, they cannot be controlled by the BBIST instrument). These are typically special-purpose configuration-related pins (such as reset). As the BBIST IP is unable to access them (i.e., unable to sense or drive the value on the

corresponding net), these have to be excluded from the interconnect test during the conversion Boundary Scan test patterns for embedded BBIST.

4.3.3 Other tests

In the same manner the BBIST IP can perform all other types of BS tests that are typically developed for testing a PCBA at the end of the production process. These include tests for interconnection with on-board memories (e.g., DDR RAM and SRAM), cluster tests (for on-board simple logic devices) as well as testing the presence and connections of peripherals: I2C devices (e.g., temperature sensors), flash devices, etc.

4.3.4 Extended functional test of processors

In this stage the BBIST IP clears the reset signal of the microprocessor and as a result the CPU starts to execute its boot program. This could be either operational code (e.g., bootloader) or some kind of specially prepared functional test program targeted to test the main components of the processor core. At the same time, the BBIST IP uses the debug and trace port of the CPU to fetch information about program execution provided by the debugging facilities inside the processor (i.e., program counter, information about memory accesses). In this way, the BBIST IP can monitor the program execution flow and detect inconsistences caused by possible faults inside the processor.

4.3.5 End of test stage

At this point, all test procedures are finished and board is being switched into functional mode. The FPGA clears the resets signals of all devices on the PCBA and forces the execution of the normal initializing sequence. At the same time, the board BBIST IP triggers the re-configuration of the FPGA with a functional IP and hand-offs control to it.

4.4 BBIST IP architecture

The general architecture of the Board BIST IP is shown in Fig. 7.

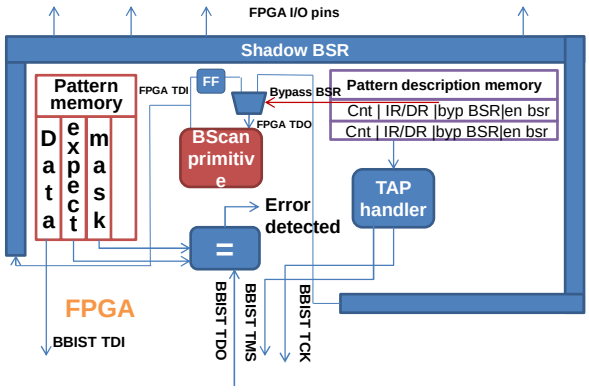

Figure 7. General architecture of embedded BBIST instrument

The BBIST IP architecture consists of the following main components (see Figure 7):

- *Shadow Boundary Scan Register* (Shadow BSR) which is used instead of the normal BSR to deliver test data to/from the FPGA I/O pins. The shadow BSR is a

shift-register implemented in FPGA logic. The shadow BSR is intentionally made equal in BS cells count and order to the normal BSR (to simplify test pattern conversion/portability).

- *Pattern memory*. This memory is used to store test data (patterns) that have to be driven to scan-chain. The memory is divided into three storage parts:
    - "data" test data for BBIST TDI (test data input) that should be shifted into scan-chain
    - "expected data" used for comparison with data coming back from scan-chain via BBIST TDI (test data output)
    - "mask" (that indicates if the comparison of a particular bit has to be performed or not).
- *Pattern description memory*. It contains the information on how each pattern should be handled. It consists of the following fields:
    - Count – number of bits in the pattern.
    - IR/DR – indicates whether IR-shift or DR-shift should be performed.
    - Bypass BSR – indicates whether the test data of this pattern should go also into the shadow BSR or bypass it. The latter corresponds to the situation when the FPGA is placed into the "BYPASS" state during the normal Boundary Scan test.
    - Enable BSR – indicates whether the Shadow BSR should be enabled (drive values to external I/O pins) or disabled during the application of this pattern. Bit one in "Enabled BSR" field corresponds to the situation when FPGA is in "EXTEST" [2] state during normal BS test procedure.
- *Comparator*. It compares the test responses with the expected data (taking into account the "Mask" bits) and reports error on mismatch. In case, if error is detected the BBIST IP interrupts further booting process of the system.
- *BScan primitive*. A native part of the FPGA device used to provide access to FPGA TDI and FPGA TDO pins from the user logic. The BBIST IP uses the BScan primitive to provide linked scan-chain, i.e., either to include the Shadow BSR into the scan-chain or to bypass the Shadow BSR and connect the FPGA TDO and TDI via 1-bit long bypass shift-register.
- *BBIST TMS* (Test Mode Select) and *BBIST TCK* (Test Clock). These signals are driven by the BBIST IP to emulate the standard IEEE1149 [1] TMS and TCK signals. The BBIST TMS and BBIST TCK lines are connected to the respective pins of the JTAG connector on the PCBA (Figure 6).
- *BBIST TDI* (Test Data Input) and *BBIST TDO* (Test Data Output). These signals are used by BBIST IP to send (TDI) and receive (TDO) test data into/from the scan-chain. BBIST TDI and BBIST TDO lines are connected to the respective TDI and TDO pins of JTAG connector on PCBA (Figure 7).
- *TAP handler*. It generates the correct sequence of BBIST TMS and BBIST TCK signals depending on the type of the current pattern (IR- or DR-shift) being applied.

The BBIST IP architecture is developed in a universal way: this means that the same IP is capable to carry out BS tests on different PCBAs without changes in the design. The behavior of the BBIST IP is fully determined by the test patters (i.e., by the content of the Pattern Memory and Pattern Description Memory). This content in its turn is derived from the normal Boundary Scan test generated by the BS development software.

5 Rejuvenation of nanoscale logic at NBTI-critical paths using evolutionary TPG

This Section describes a method developed by BASTION partners to attack the aging phenomenon. The idea is to identify the most critical devices in a circuit, and develop functional stimuli that, when applied to the device inputs, may allow these devices to rejuvenate, thus delaying the ware-out phase. Preliminary results are shown, supporting the effectiveness of the method.

5.1 Introduction

With nanoscale semiconductor manufacturing processes, lifetime reliability has become one of the key design challenges to guarantee CMOS integrated-circuit robustness. One of the most critical downsides of technology scaling beyond the 65nm node, in turn, is non-determinism of the devices' electrical parameters due to time-dependent deviations in the operating characteristics of the device. In [43], a brief overview of major sources of time-dependent variations is presented. One of them is Negative Bias Temperature Instability (NBTI) [44].

NBTI is defined as the effect that occurs when a pMOS transistor is negatively biased. The effect manifests itself as an increase of the threshold voltage V_{th} over time. This results, both, in drive current reduction and noise increase, which in turn causes an increase of the device delay. NBTI's effect on the long-term stability of functional logic expresses itself through the incapability of storing a correct value at memory elements such as flip-flops due to the de-synchronization between clock distribution and signal propagation through the logic paths of a circuit. Therefore, in several years of circuit operation time the NBTI induced aging may cause, first, transient faults and, ultimately, permanent circuit functional failure.

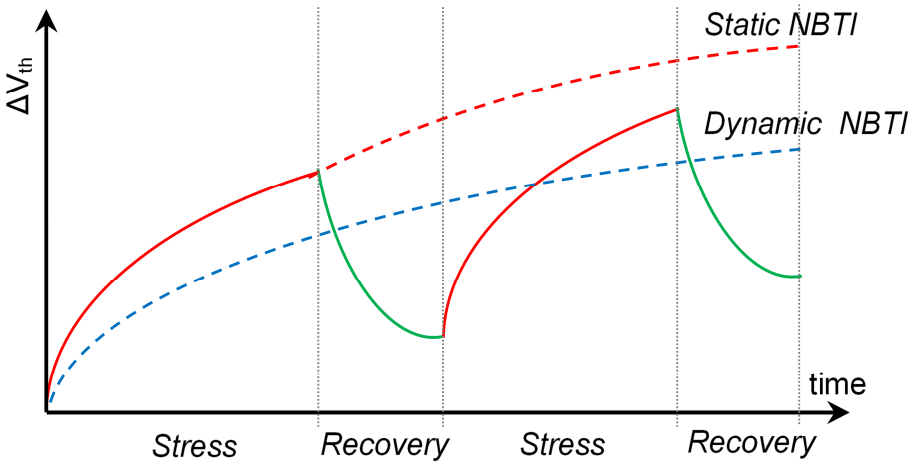

Fig. 8 Illustration of NBTI stress and recovery phases (based on the R-D model by Alam [63])

The variation of threshold voltage of pMOS transistors due to *dynamic NBTI* (i.e., when the stress and recovery phases are iterating) is estimated to be 5-15% per year [45][46], depending on the targeted technology and its environment. The threshold voltage shift for devices at *static NBTI* (i.e., when a transistor is under constant stress) can be significantly higher. The path delay degradation follows the same trend, though with a smaller magnitude. It has been shown that NBTI depends on many

factors [47] with strong correlation to the signal probability P_z (input duty cycle) and the output load capacity C_L [48]. The signal probability P_z for a logic gate's input is defined as the ratio of time where that input signal is set to logic 0. An estimation for 65nm pMOS V_{th} degradation for different input signal probabilities is shown in Fig. 9 [48].

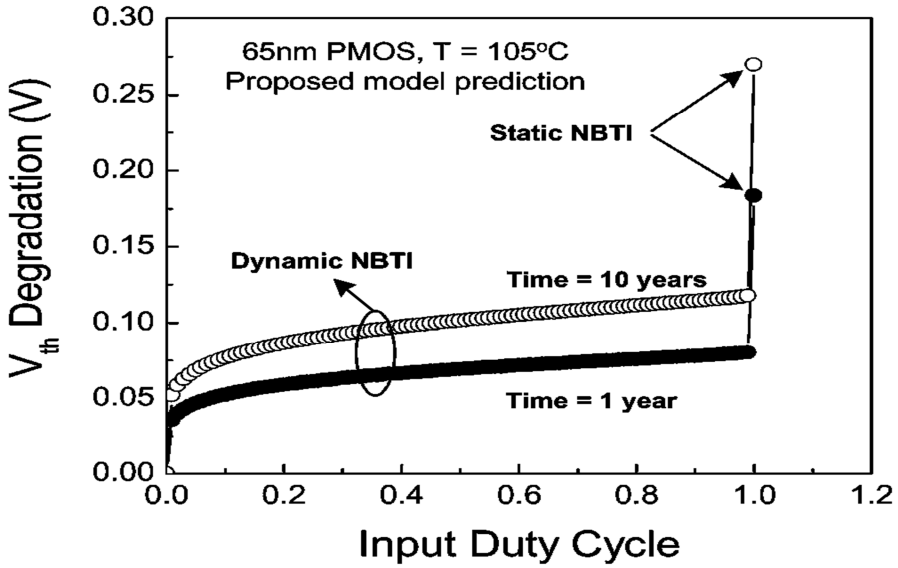

Fig. 9 Dynamic and static NBTI for different input signal probabilities [48]

In BASTION, we propose a novel *NBTI mitigation approach by rejuvenation of nanoscale logic at NBTI-critical paths* with dedicated stimuli sequences. The method is based on fast hierarchical NBTI-critical paths identification at gate level and rejuvenation stimuli generation using an Evolutionary Algorithm. The rejuvenation stimuli is targeted to be applied as execution overhead at precalculated periods of time to drive to the recovery phase the pMOS transistors that are the most significant for the NBTI-induced path delay. The proposed approach aims at extending the reliable lifetime of nanoelectronics.

Evolutionary Computation has been used by the Computer Aided Design community for years, also involving it for the tasks of automatic test-pattern generation. In BASTION, we exploit a general-purpose evolutionary toolkit called µGP [49][50] and find a suitable fitness function using an open source hardware analysis framework zamiaCAD [51]. The advantage of such flow is its ability and flexibility in solving the dependencies of impacts by individual gates to the most critical NBTI-induced path delay using the evolutionary optimization process.

5.2 Progress beyond the state-of-the-art

Previous works found in the literature address the NBTI problem both for memories [45][52] and functional logic. Targeting the second one, [53] proposes a *redesign approach based on transistor sizing technique* that not only mitigates NBTI induced delay of the gate under consideration, but also minimizes its impact on the adjacent gates. This technique seems to be very effective, but it is also mandatory to identify the critical gates and paths to which it should be applied. Otherwise, this technique will result in an unacceptable area overhead and eventually, excessive power

consumption. In [54] the authors present a method characterizing the delay of every gate in a standard cell library as a function of the signal probability (P_z) of each of its inputs and suggest an *NBTI-aware synthesis* accordingly. It demonstrates an average of 10% area recovery for 65 nm under pessimistic assumption that *all pMOS transistors in the design are under constant static NBTI stress*. [55] has proposed an approach for temporarily hiding NBTI-induced aging by applying *changes to voltage and frequency* of the circuit.

Approaches to analyze the efficiency of *controlling input signal probability* for mitigating NBTI at circuit level were proposed in [56][57]. Works in [58][59] propose to exploit idle time of processors and unused bits in source operands [60].

A very relevant approach for processor circuits rejuvenation is presented in [61] where authors propose to replace the default NOP with a special Maximum Aging Reduction NOP instruction that while having no effect on the program state minimizes the NBTI effect. The results show that this method can extend lifetime by 37% in average, with performance, power, and area overhead within 1%.

Different from the works existing in the state of the art, the approach proposed in BASTION:

- is based on accurate and fast hierarchical gate-level identification of NBTI-critical paths and particular gates where rejuvenation has to be applied;
- proposes efficient rejuvenation stimuli generation with evolutionary algorithm;
- does not require redesign and can be applied to the existing circuit, exploiting the existing design-for-testability (DFT) instruments (e.g., scan-chains).

5.3 Experimental results

The proposed approach has been demonstrated on an ALU (Arithmetic Logic Unit) core extracted from a MIPS processor design Plasma [62] described in RTL VHDL. The gate level was synthesized with the Synopsys Design Compiler. For the purpose of the rejuvenation experiments, a number of circuit Aging Profiling Sets (APSs) were generated:

- APS1-9: these are realistic exploitation scenarios of the ALU circuit, when only one, two or three functions out of the 16 implemented in the ALU logic were used.
- APS1 uses only function OR with all possible combinations for 4-bit operands. APS2 is for NOR, APS3 is for AND and APS4 is for ADD.
- APS5 allows activation of ADD and NOR functions only. ADD and OR in APS6 and OR and AND in APS7.
- APS8 activates 3 functions OR, NOR and ADD, APS9 allows OR, NOR and AND.
- APS10-12: these are stimuli sets with repetitions of random sequences of length 5, 10 and 150.

- APS13-15: these are stimuli sets providing close to the highest NBTI degradation to the logic, where sequences of 1, 5 and 10 vectors are repeated.

The ALU circuit was first simulated with the given aging profiles to obtain values for P_z at each node (gate input). Further, NBTI-induced path delays were estimated based on these P_z values and corresponding structural details such as gate output loads (dependent on number of fanout branches) and initial path delays calculated by static analysis. Rejuvenation stimuli sequences were generated for each APS individually and their contribution to reduction of path delays was calculated. Table II demonstrates the experimental results.

In case of different profiles, the path delay in 10 years was estimated to increase by 12% to 86%. The highest path delay increase due to NBTI (APS13) is less realistic (a single stimuli vector is kept all the time with many gates on the aging-critical paths set to static NBTI state) and is provided mainly for reference of a possible worst case scenario. However, the increase of paths delays with APS1-9 profiles can be considered realistic and still provide for very high delay increments Δt, in the range of 19%-40%. The aging profiles with random stimuli provide for very smooth distribution of P_z probabilities close to 0.5 in the whole design structure and therefore in case of longer random sequences can cause only small and well-distributed NBTI-induced path delays (around 12%). The last three rows in the table demonstrate the efficacy of the generated rejuvenation stimuli to mitigate the paths delays under given constraints for the allowed overhead. As it can be seen from the table, application of the rejuvenation stimuli with 1% execution overhead was capable to reduce path delay increments (i.e., Δt to Δt^R) in 1.5 to 5.6 times for aging profiles with static NBTI involved.

The time required for the proposed approach to generate the individual rejuvenation stimuli sequences for each of APS profiles was 20 minutes (3GHz iCore7 Windows 64 bit, 1 GB of memory used by JVM). It included iterative execution of the evolutionary algorithm with the circuit simulation by aging profiling stimuli and NBTI-critical path identification calls.

TABLE II. EXPERIMENTAL RESULTS

Aging profiling sets (APS)		*Executions of one to several ALU functions*									*Random*			*The highest NBTI degradation*		
		1 func	*1 func*	*1 func*	*1 func*	*2 func*	*2 func*	*2 func*	*3 func*	*3 func*						
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
# of unique vectors		256	256	256	256	256	256	256	256	256	5	10	150	1	5	10
nodes at static NBTI (%)		25	21.25	23.75	16.875	8.7	13	14.2	6.88	5.625	20	8	0	50	49	50
Δt by NBTI (%)		36.32	24.96	22.96	39.84	25.4	36.8	20.7	21.1	18.9	32.48	21.16	12.16	86.08	81.13	64.9
Δt^R after rejuvenation for the given overhead, (%)	***0.1%***	18.56	22.96	22.88	17.44	17.2	17.7	19	17.3	18.9	23.04	17.37	12.16	19.2	30.79	39.8
	1%	13.68	10.96	10.96	13.68	13.4	13.9	13.3	13.5	12.9	12.64	13.55	12.16	15.44	14.39	11.45
	10%	13.52	11.36	11.36	13.44	13.2	13.7	13.15	13.3	12.7	12.72	13.34	12.08	15.12	14.26	11.44

5.4 Conclusions

This section describes a method which is able to face the aging phenomenon in electronic circuits. The key idea is that this phenomenon can be delayed or even recovered (*rejuvenation*) if suitable values are applied to the most critical transistors (i.e., those on the critical paths). To achieve this goal we need first to identify these transistors and the required values, and then to devise suitable functional stimuli to be applied in the field to the inputs of the device in order to generate them.
The experiments reported in this Section show that this is feasible, and the method can achieve interesting results.

6 Automatic Test program Generation for PCBA test

This Section describes a new method which is currently being developed and evaluated by the BASTION partners to automate the generation of functional stimuli at the board level, starting from existing test stimuli and information for specific components.

6.1 Introduction

A functional test solution is usually developed within the hardware design environment, as a test vehicle for verifying that a PCBA meets its design criteria.
Once the design validation and prototype test phases are complete, the test vehicle is transferred to the Test Engineering department, to be also used as a functional test platform, to verify that manufactured products meet their performance specification and are 'fit for purpose', before being shipped to the customer.
The functional test stage is the final PCB quality gate.
However, functional test is a challenge, because:

- The test programs are developed manually with graphical guidance but with limited or no automation.
- It is extremely difficult to predict the test coverage provided by a test program, unless fault simulation has been undertaken.
- Measurement statements from the test reports are difficult to correlate against defects.
- Fault diagnosis is either extremely limited, or in some cases non-existent, with the majority of tests simply providing PASS/FAIL status notification.

The estimation of the defect coverage provided by functional test often requires far more elaborate calculations. If coverage information is available for each of the defect classes, then it makes sense to re-use this information in the functional test coverage calculations, across all designs.

6.2 ATPG for functional test

Within the BASTION WP1, a dedicated task (T1.4) is targeted to "Improvement of test coverage metrics". As test coverage is a direct measurement of the test quality, it is taken into account by the majority of the BASTION Work Packages.
In 2006, ASTER developed a test coverage method named "Coverage by Inheritance". Based on discussions and results performed with the BASTION project, it appears that it will be possible to use that work as a basis for the development of an ATPG for functional test at the PCB level.

ATPG (**A**utomatic **T**est **P**attern **G**enerator) is an electronic design automation method used to find test sequences that, when applied to a Printed Circuit Board Assembly, enable automatic test equipment to activate differences between the correct circuit behavior and the faulty circuit behavior caused by defects, and hence to detect them.

ATPG is popular for IC and PCBA test, though it is usually limited to ICT, FPT, or traditional Boundary-Scan test. The ASTER research is based on an innovative approach.

The ASTER method is designed to re-use test coverage predictions and test program data, associated with functional blocks (also named *modules*) within a design.
Each functional block is described in a sub-folder, with a set of files, as presented in Figure 10 below:

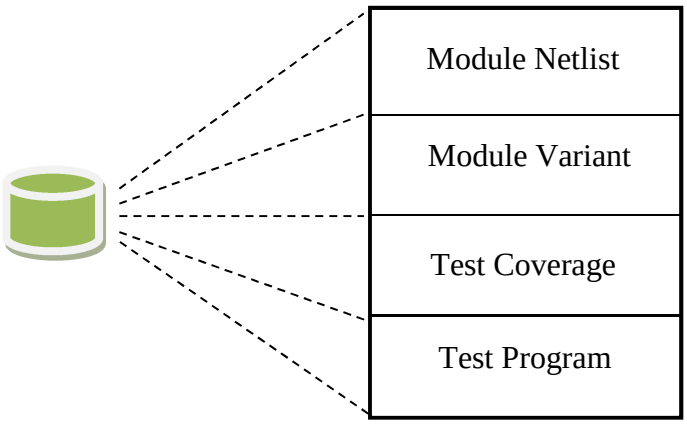

Figure 10: Synopsis of functional block / module

This is particularly useful within hierarchical design flows that reuse several instances of the same functional blocks/modules. Information can be transferred to multiple instances of the functional block, thus eliminating the need to reproduce the fault coverage predictions and test program development for every instance of the design module.

During the design analysis phase, the CAD design library will be interrogated for instances of known design modules that are re-used within the board design, so that the corresponding predicted fault coverage/test program can be transferred to the board design (Figure 11).

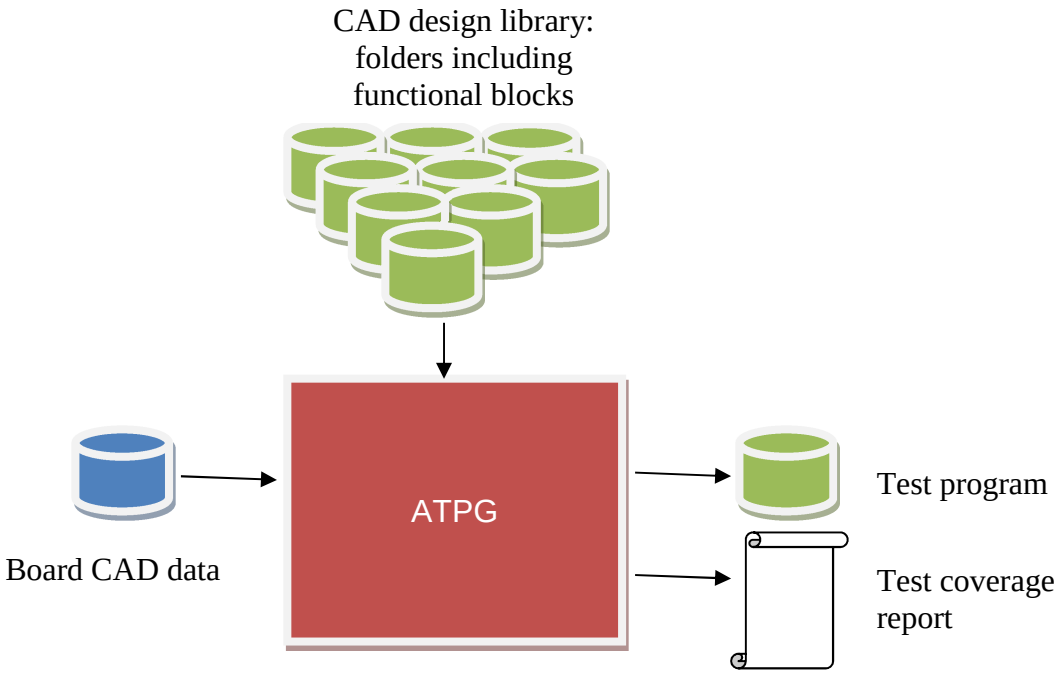

Figure 11: Synopsys of the Functional ATPG

6.3 Module matcher

The most critical part of the ATPG is the "module matcher". It checks if a known module corresponds to a subset of the board. The module connectivity is compared with the board connectivity and computes a matching percentage between the functional block and a subset of the board. When the matching percentage equals 100%, the module test coverage & test program is transferred to the board.

In order to recognize and match a module or function, the following module information is used:

1. Component reference designator
2. Component type or component function
3. Pin number
4. Net name.

Points 1 and 2 are linked with the component, while points 3 and 4 are linked with its connectivity.

When there is more than one instance of a module or function, the reference designator (point 1) and the net name (point 4) are updated automatically by the CAD design tool.
Under these circumstances, it is clear that the module matcher cannot use the reference designator (point 1) and net name (point 4) and should only use component type (point 2) and pin number (point 3).

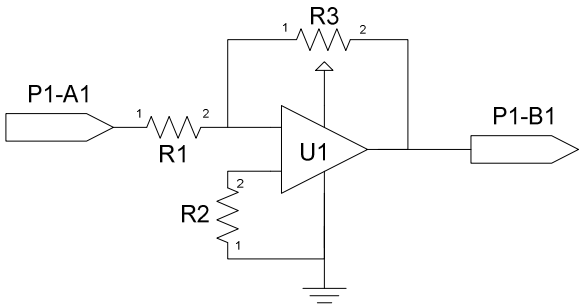

Figure 12: Example circuit

The current implementation of the "module matcher" makes it possible to properly manage variations in design instances that have identical functionality. Referring to the example circuit shown in Figure 12, possible variations are:

- Resistor R1 could be mounted in reverse (swapping between pin 1 and pin 2).
- The resistors (R1, R2 & R3) could be packed in a resistor array. In this case even component type (point 2) and pin number (point 3) become irrelevant.
- Connector pins (P1-A1 or P1-A2) could be handled differently in either of the following cases:
  a) When it is mandatory to use a certain pin, only (standard connector).
  b) The function is still the same even if a pin is connected to another connector point.

6.4 Conclusion

Experimentation on coverage analysis for BIST, in partnership with Ericsson Sweden, has been conducted. It demonstrates that when an ASIC includes a processor IP core, connected to a bank DDR3, the module matcher is able to identify that the functional block is testable with embedded instruments. The test coverage is automatically transferred to the IP Core and DDR3 memories, even in the case of multiple instances on the same board, or multiple boards. This first result can be used to generate the functional test program.

A second experiment has been conducted, with Continental Toulouse on Automotive PCBA. They use a wide variety of PCBA for power-train control, depending on the engine characteristics (i.e., on the car manufacturer). With a library of the functional blocks used in design flow, the ASTER ATPG successfully created a different variant of the test flow allowing for the various configurations.

For the next step, it is being discussed with National Instruments, one of the world-wide functional test leader, in order to automatically generate TestStand instructions from the ASTER ATPG. A new experiment is being initiated with a common customer (Häger Electro à Obernai).

7 Summary

This report summarizes some of the activities performed by the partners of the BASTION project involved in WP4.

A major goal of WP4 (*Instrument-Assisted testing for NFF)* is to investigate the opportunities offered by the availability of instruments at the IC and board level to successfully face the NFF issue. Given the key role that functional test plays at all levels in increasing the achievable defect coverage, the plan of activities of BASTION includes a preliminary phase, in which the partners are expected to explore new solutions for combining functional test with the use of embedded instruments (Task 4.1, *Functional test programs exploiting the embedded instrumentation*).

Based on this plan, several activities were launched, whose preliminary results are reported in this document.

First, an overview of functional test was presented in Section 2, followed by an experimental analysis of the effects on the fault coverage that can be obtained by adopting different observation mechanisms when the SBST approach is followed. Although the current results only refer to the stuck-at fault model, they show that the availability of different observation mechanisms can lead to quite different fault coverage results, especially when considering the in-field scenario. Moreover, a clever choice of the observation mechanism may allow to achieve a fault coverage which is comparable with the one that can be obtained for end-of-manufacturing test, when full controllability and observability of the device signals can be easily obtained.

Section 3 describes a set up that using any available FPGA can provide easy and inexpensive access to embedded instruments of different nature when performing in-field test at the board level.

Section 4 describes a method which resorts to functional test to perform rejuvenation: suitable functional stimuli are generated, able to delay or even reverse the aging process by acting on the most critical transistors in a device. Preliminary results show the effectiveness of the method.

Finally, Section 5 describes how it is possible to re-use existing functional tests and the related information (e.g., about their fault coverage) to automate the generation of a functional test at the board level. Special emphasis is devoted to some key implementation issues that have to be faced in order to automate the approach and integrate it into a real test flow.

The reported activities and preliminary results are the basis for further actions currently being performed by the BASTION partners.

8 References

[1] S. Thatte and J. Abraham, “Test Generation for Microprocessors”, IEEE Transactions on Computers, vol. 29, no. 6, pp. 429–441, June 1980.

[2] M. Psarakis et al., “Microprocessor Software-Based Self-Testing”, IEEE Design & Test of Computers, vol. 27, no. 3. May-June 2010, pp. 4-19.

[3] M. Hatzimihail et al., “A methodology for detecting performance faults in microprocessors via performance monitoring hardware”, IEEE International Test Conference, 2007.

[4] M. Kaliorakis et al., “Accelerated online error detection in many-core microprocessor architectures”, IEEE 32nd VLSI Test Symposium (VTS), 2014.

[5] S. Di Carlo et al., “A software-based self test of CUDA Fermi GPUs”, IEEE 18th European Test Symposium (ETS), 2013.

[6] F. Corno et al., “Automatic Test Program Generation: a Case Study”, IEEE Design & Test of Computers, vol. 21, pp. 102-109, 2004.

[7] A. Riefert et al., “An effective approach to automatic functional processor test generation for small-delay faults”, Conference on Design, Automation and Test in Europe (DATE), 2014.

[8] M. Grosso et al., “Software-Based Testing for System Peripherals”, Journal of Electronic Testing (JETTA), vol. 28 n. 2, pp. 189-200, 2012.

[9] A. J. van de Goor et al., “Memory testing with a RISC microcontroller”, Proc. of Design, Automation and Test in Europe (DATE), 2010.

[10] S. Di Carlo et al., "Software-Based Self-Test of Set Associative Cache Memories", IEEE Trans. on Computers, vol. 60, issue 7, pp. 1030-1044, July 2011.

[11] M. Riga et al., “On the functional test of L2 caches”, IEEE International On-Line Test Symposium (IOLTS), 2012.

[12] A. Dalirsani et al., “Structural Software-Based Self-Test of Network-on-Chip”, IEEE 32nd VLSI Test Symposium (VTS), 2014.

[13] J. Zhou and H.-J. Wunderlich, “Software-Based Self-Test of Processors under Power Constraints”, Conference on Design, Automation and Test in Europe (DATE), pp. 430-436, 2006.

[14] P. Bernardi et al., “An Effective technique for the Automatic Generation of Diagnosis-oriented Programs for Processor Cores”, IEEE Transactions on

Computer-Aided Design of Integrated Circuits and Systems, vol. 27, pp. 570-574, 2008.

[15] P. Bernardi et al., “Exploiting an infrastructure-intellectual property for systems-on-chip test, diagnosis and silicon debug”, IET Computers & Digital Techniques, vol. 4(2), pp. 104-113, 2010.

[16] F. Reimann et al., “Advanced Diagnosis: SBST and BIST Integration in Automotive E/E Architectures”, 51st ACM/IEEE Design Automation Conference (DAC), 2014.

[17] D. Sabena et al., “On the Automatic Generation of Optimized Software-Based Self-Test Programs for VLIW Processors”, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 22, n. 4, pp. 813-823, 2014.

[18] A. Apostolakis et al., “Test Program Generation for Communication Peripherals in Processor-Based Systems-on-Chip”, IEEE Design & Test of Computers, vol. 26 n. 2, pp. 52-63, 2009.

[19] A. Riefert, R. Cantoro, M. Sauer, M. Sonza Reorda, B. Becker, “On the automatic generation of SBST test programs for in-field test”, Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015, pp. 1186 – 1191.

[20] M. Gaudesi, M. Sonza Reorda, I. Pomeranz, “On Test Program Compaction”, IEEE European Test Symposium (ETS), 2015.

[21] M. Psarakis et al., "Microprocessor Software-Based Self-Testing", IEEE Design & Test of Computers, vol. 27, no. 3. May-June 2010, pp. 4-19.

[22] W. Lindsay, E. Sanchez, M. Sonza Reorda, G. Squillero, "Automatic Test Programs Generation Driven by Internal Performance Counters", IEEE International Workshop on Microprocessor Test and Verification, 2004, pp. 8-13.

[23] S. Almukhaizim, P. Petrov and A. Orailoglu, "Faults in processor control subsystems: Testing correctness and performance faults in the data prefetching unit", IEEE Asian Test Symposium, pp. 319 - 324, 2001.

[24] E. Sanchez, M. Sonza Reorda, "On the Functional Test of Branch Prediction Units", IEEE Transactions of VLSI Systems, 2015.

[25] W.J. Perez, D. Ravotto, E. Sanchez, M. Sonza Reorda, A. Tonda, "On the Generation of Functional Test Programs for the Cache Replacement Logic",

18th IEEE Asian Test Symposium 2009 (ATS09), Taichung, Taiwan, November 2009. pp. 418-423.

[26] P. Bernardi, M. Bonazza, E. Sanchez, M. Sonza Reorda, O. Ballan, "On-line functionally untestable fault identification in embedded processor cores", Design, Automation & Test in Europe Conference & Exhibition (DATE), 2013.

[27] S. Thatte and J. Abraham, "Test Generation for Microprocessors", IEEE Transactions on Computers, vol. 29, no. 6, pp. 429-441, June 1980.

[28] P. Parvathala, K. Maneparambil, W. Lindsay, "FRITS - a microprocessor functional BIST method", IEEE International Test Conference, 2002, pp. 590 – 598.

[29] S. Gurumurthy, M. Pratapgarhwala, C. Gilgan, J. Rearick, "Comparing the effectiveness of cache-resident tests against cycleaccurate deterministic functional patterns", IEEE International Test Conference, 2014, pp. 1-8.

[30] L. Chen and S. Dey., "Software-based self-testing methodology for processor cores," IEEE Trans. on Computer-Aided Design, vol. 20, no.3, March 2001, pp. 369-380.

[31] F. Corno et al., "Automatic Test Program Generation: a Case Study", IEEE Design & Test of Computers, vol. 21, pp. 102-109, 2004.

[32] A. Riefert, R. Cantoro, M. Sauer, M. Sonza Reorda, B. Becker, "On the Automatic Generation of SBST Test Programs for In-Field Test", Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015, pp. 1186 – 1191.

[33] G. Theodorou, S. Chatzopoulos, N. Kranitis, A. Paschalis, D. Gizopoulos, "A Software-Based Self-Test methodology for on-line testing of data TLBs", IEEE European Test Symposium (ETS), 2012.

[34] D. Appello, P. Bernardi, M. Grosso, E. Sanchez, M. Sonza Reorda, "Effective Diagnostic Pattern Generation Strategy forTransition-Delay Faults in Full-Scan SOCs", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 17 (11), pp. 1654-1659 .

[35] A. Paschalis, D. Gizopoulos, "Effective software-based self-test strategies for on-line periodic testing of embedded processors", Design, Automation and Test in Europe Conference and Exhibition, 2004, pp. 578 – 583.

[36] Bernardi P; Grosso M.; Rebaudengo M; Sonza Reorda M., “Exploiting an I-IP for both test and silicon debug of microprocessor cores”, Microprocessor Test

and Verification, 2005. MTV '05. Sixth International Workshop on IEEE International Workshop on Microprocessor Test and Verification MTV, Austin, TX, USA, 3-4 Novembre 2005, pagine 55-62.

[37] P. Bernardi, L. Ciganda, M. De Carvalho, M. Grosso, J. Lagos-Benites, E. Sanchez, M. Sonza Reorda, O. Ballan (2012), “On-Line Software-Based Self-Test of the Address Calculation Unit in RISC Processors”, 17th IEEE European Test Symposium (ETS) 2012, ISBN: 9781467306966.

[38] Gaisler, J., Catovic, E., Isomaki, M., Glembo, K., Habinc, S., "GRLIB IP core user's manual. Version 1.3.7 - B4144". Gaisler research, 2014.

[39] Goldman, R., Bartleson, K., Wood, T., Kranen, K., Cao, C.; Melikyan, V., Markosyan, G., "Synopsys' open educational design kit: Capabilities, deployment and future," Microelectronic Systems Education, 2009. MSE '09. IEEE International Conference on, July 2009.

[40] Perez Acle, J., Cantoro, R., Sanchez, E., Sonza Reorda, M., "On the Functional Test of the Cache Coherency Logic in Multi-core Systems.", 6th IEEE Latin American Symposium on Circuits and Systems, LASCAS, 2015.

[41] K.P. Parker. The Boundary-Scan Handbook. Boston, MA, USA, Kluwer Academic Publisher, 2003, p. 373.

[42] IEEE Standard test access port and boundary-scan architecture, IEEE Std. 1149.1-2001, 2001.

[43] S.Hamdioui, D.Gizopoulos, G.Guido, M.Nicolaidis, A.Grasset, P.Bonnot, “Reliability Challenges of Real-Time Systems in Forthcoming Technology Nodes”, ACM/IEEE DATE, March 2013, pp. 129-134.

[44] S. Mahapatra, D. Saha, D. Varghese, P. B. Kumar, “On the Generation and Recovery of Interface Traps in MOSFETs Subjected to NBTI, FN, and HCI Stress”, IEEE Trans. Electron. Dev. 53, 7, 1583-1592, 2006.

[45] C. Ferri, D. Papagiannopoulou, R. Iris Bahar, A. Calimera, “NBTI-Aware Data Allocation Strategies for Scratchpad Memory Based Embedded Systems”, IEEE 12th Latin American Test Workshop (LATW), March 27-30, 2011, Porto de Galinhas, Brazil.

[46] Ceratti, A.; Copetti, T.; Bolzani, L.; Vargas, F.; , "Investigating the use of an on-chip sensor to monitor NBTI effect in SRAM," 2012 13th Latin American Test Workshop (LATW), pp.1-6, 10-13 April 2012.

[47] M. Khan, P. Weckx, P. Raghavan, S. Hamdioui, et al., “Comparison of Reaction-Diffusion and Atomistic Trap-based BTI Models for Logic Gates”, IEEE Transactions on Device and Materials Reliability, June 2013.

[48] W. Wang, S. Yang, S. Bhardwaj, S. Vrudhula, F. Liu, Y. Cao, “The Impact of NBTI Effect on Combinational Circuit: Modeling, Simulation, and Analysis”, IEEE Trans. on VLSI, vol. 18, no. 2, 2010, pp. 173-183.

[49] G. Squillero, “MicroGP—An Evolutionary Assembly Program Generator”, Genetic Programming and Evolvable Machines, pp. 247-263, vol. 6, issue 3, Sep 2005

[50] E. Sanchez; M. Schillaci; G. Squillero, Evolutionary Optimization: the µGP toolkit, Springer, ISBN: 978-0-387-09426-7, 2011

[51] zamiaCAD framework web page, [http://zamiaCAD.sf.net]

[52] F. Ahmed, L. Milor, “Reliable Cache Design with On-Chip Monitoring of NBTI Degradation in SRAM Cells using BIST”, 2010 28th IEEE VLSI Test Symposium, pp. 63-68.

[53] S. Khan, S. Hamdioui, Modeling and Mitigating NBTI in Nanoscale Circuits, 2011 17th International On-Line Testing Symposium, pp. 1-6.

[54] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar, “NBTI-aware synthesis of digital circuits,” Proc. Design Autom. Conf., 2007, pp. 370–375.

[55] A. Tiwari and J. Torrellas, “Facelift: Hiding and slowing down aging in multicores”, International Symposium on Microarchitecture, pages 129–140, 2008.

[56] F. Firouzi, S. Kiamehr, and M.B. Tahoori, “A linear programming approach for minimum NBTI vector selection”, Great lakes symposium on VLSI, pages 253–258. 2011.

[57] Y. Wang, X. Chen, W. Wang, V. Balakrishnan, Y. Cao, Y. Xie, and H. Yang, “On the efficacy of input Vector Control to mitigate NBTI effects and leakage power”, Quality Electronic Design Int’l Symp., pages 19–26, 2009.

[58] J. Abella, X. Vera, et al., “Penelope: The NBTI-aware processor”, Micro, pages 85–96. IEEE Computer Society, 2007.

[59] L. Li, Y. Zhang, J. Yang, and J. Zhao, “Proactive NBTI mitigation for busy functional units in out-of-order microprocessors”, Design, Automation and Test in Europe, pp 411–416, 2010.

[60] X. Fu, T. Li, and J. Fortes, “NBTI tolerant microarchitecture design in the presence of process variation”, Int. Symposium on Microarchitecture, pp 399–410, 2008.

[61] F. Firouzi, S. Kiamehr, and M.B. Tahoori, “NBTI mitigation by optimized NOP assignment and insertion”, Design, Automation and Test in Europe Conference, pages 218–223, 2012.

[62] Open Cores Plasma CPU project, [http://opencores.org/project,plasma]

[63] M. A. Alam and S. Mahapatra, "A comprehensive model of PMOS NBTI degradation," Microelectronics Reliability, 45(1), pp. 71-81, Jan. 2005.