



*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

# Reuse adaptation, and integration of embedded instruments at the board level test environment (Deliverable D4.3)

Due date of deliverable: April 30, 2017
Ready for submission date: June 26, 2017

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

Organisation name of lead contractor for this deliverable: Testonica Lab

Revision 1.0

| Project co-funded by the European Commission within the Seventh Framework Programme (2014-2017) | | |
|---|---|---|
| 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 Artur Jutman, e-mail: artur@testonica.com

This document is intended to fulfil the contractual obligations of the BASTION project concerning deliverable D4.3 described in contract 619871.

© Copyright BASTION 2017. All rights reserved.

## Table of Revisions

| Version | Date | Description and reason | Author | Affected sections |
|---|---|---|---|---|
| 0.1 | March 24, 2017 | Structure created | A. Jutman | All |
| 0.2 | April 4, 2017 | Section 6 added | M. Sonza Reorda | Mainly Section 6, plus minor changes elsewhere |
| 0.3 | April 11, 2017 | Section 2 added | H. Ebrahmi, H.G. Kerkhoff | Section 2 & references |
| 0.4 | April 25, 2017 | Section 5 added | I. Aleksejev | Section 5 |
| 0.4a | April 26, 2017 | Section 7 added | I. Aleksejev | Section 7 |
| 0.5 | April 28, 2017 | Section 4 added | E. Larsson | Section 4 |
| 0.6 | May 24, 2017 | Section 3 added | J. Raik | Section 3 |
| 0.7 | May 31, 2017 | Integration of all contributions together | A. Jutman | All |
| 0.8 | June 8-9, 2017 | Correcting references and numbering | A. Jutman | All |
| 0.9 | June 22, 2017 | Writing Introduction and Conclusions | A. Jutman | Section 1 and Section 8 |
| 1.0 | June 26, 2017 | Final polishing and preparing for submission | A. Jutman | All |

## Author, Beneficiary

R. Cantoro, E. Sanchez, M. Sonza Reorda, Politecnico di Torino
I. Aleksejev, S. Odintsov, A. Jutman, S. Devadze, Testonica Lab
J. Raik, TU Tallinn
H. G. Kerkhoff, H. Ebrahimi, University of Twente
E. Larsson, University of Lund

## Executive Summary

This document reports on the activities performed within two tasks of the BASTION project: T4.3 and T4.4. Being the last deliverable in WP4, it concludes the work done in the frames of this work package, presenting solutions for reuse and adaptation of various embedded instruments both in the field and in a system/board test environment. These results are mainly based on combined achievements already reported in D4.1 and D4.2 but also rely on key research done in the rest of the project, especially in WP1 and WP2. The document first presents techniques for wear-out and aging fault detection including IRFs and Low-Latency SEUs. Secondly, we describe instruments and methods for testing marginal and delay faults at the board level. Finally, automation and integration into board test environment is described.

# List of Abbreviations

| | |
|---|---|
| ADC | Analog-to-Digital Converter |
| ATE | Automatic Test Equipment |
| BBIST | Board Built-In Self-Test |
| BIST | Built-In Self-Test |
| BSDL | Boundary-Scan Description Language |
| BS | Boundary-Scan |
| BST | Boundary-Scan Test |
| CAD | Computer Aided Design (also EDA) |
| CPU | Central Processing Unit, also Processor |
| DFT | Design For Testability |
| DNL | Differential Non-Linearity |
| DPM | Defects Per Million |
| DRAM | Dynamic RAM |
| DRC | Design Rule Check |
| EDA | Electronic Design Automation (also CAD) |
| EMS | Enhanced Manufacturing Services |
| FIFO | First In, First Out (data buffer) |
| FPGA | Field Programmable Gate Array |
| FP7 | European Union's 7th Framework Program |
| HW | Hardware |
| IC | Integrated Circuit |
| ICT | In-Circuit Test |
| IJTAG | Internal JTAG, a short name for IEEE 1687 standard and infrastructure collectively |
| INL | Integral Non-Linearity |
| IP | Intellectual Property (hardware module in FPGA or SoC) |
| IRF | Intermittent Resistive Fault |
| JTAG | Joint Test Action Group; also Boundary Scan; often used as a short name of the IEEE 1149.1 standard and respective infrastructure including test access port and header on the board; |
| LSSD | Level-Sensitive Scan Design |
| NBTI | Negative Bias Temperature Instability |
| NFF | No Fault Found or No Failure Found (also NTF) |
| NTF | No Trouble Found (also NFF) |
| PCB | Printed Circuit Board |
| PCBA | Printed Circuit Board Assembly |
| PDL | Procedural Description Language |

| | |
|---|---|
| POST | Power-On Self-Test |
| RAM | Random-Access Memory |
| RTD | Research and Technological Development |
| SAR-ADC | Successive-Approximation-Register ADC |
| SBST | Software-Based Self-Test |
| SEU | Single Event Upset |
| SIB | Segment Insertion Bit |
| SoC | System on Chip |
| SVF | Serial Vector Format |
| SW | Software |
| TDR | Test Data Register |
| UUT | Unit Under Test |
| VHDL | VHSIC Hardware Description Language |
| VHSIC | Very High Speed Integrated Circuit |

# Table of Contents

# 1 Introduction

This deliverable is the last one in BASTION project to report research and development activities. The results presented in this document are, hence, connected to all other work packages. Fault coverage metrics developed in WP1 help assessing the results reported in Sections 5 and 6. Results of WP2 contributed to Sections 2, 3, and 5. Section 4 is based on previous results reported in WP3. Prior activities of WP4 were the basis of activities described in Sections 3 and 6. In its turn, the work presented in this deliverable contributes to the major part of the demonstrator, described in D6.2

Board and system level test today cannot any longer rely fully on traditional dedicated test technologies like In-Circuit Test, Optical or X-Ray Inspection and even Boundary Scan. A contemporary high-performance system board is a complex 3D object that may contain dozens of hidden layers, stacked microvias, high density interconnect, with all above not contributing to ease of test and reliability. High-speed signals are normally fine-tuned or even calibrated to deliver pitch perfect timing even in case of now-ubiquitous DDR3 memories. Today, data transmission rates on the board may be reaching multi-gigabit ranges on a single channel. Such defects like dewetting, cold solder, head-in-pillow, voiding/crack in micro-via or excessive solder voids may result in system performance issues, increased error rates, intermittent faults and other sporadic stability issues observed in certain operation modes, at certain workloads or manifesting in a seemingly stochastic manner, which contributes to increasing test escape rates and No Failure Found.

Every mission-critical system today has to successfully pass a fit-for-function qualification test, which is based on a combination of functional tests and application-based tests. Stress testing or endurance testing is often used afterwards to eliminate marginal devices, such that fail due to parametric deviations, dynamic faults, intermittent or marginal defects introduced during final assembly of printed circuit boards. Functional test alone does not any longer guarantee fault-free operation of the target system in the field.

One of the key contributions of WP4 is the concept of combined usage of functional tests and IC-level instruments during the post-assembly system test (board level or product test). Although, D4.2 already addressed this topic, here we present follow-up activities in Section 3 and Section 6. While the former one addresses aging fault detection in the field, the latter one assesses the ability of traditional techniques to cover board-level delay faults and compares the typical test coverage with the BASTION approach. The chip-level instruments help increasing the observability and hence improve the coverage of functional test and the ability of in-situ fault detection both in-field and during the manufacturing test.

Reuse of IC-level instrumentation is also considered in Section 2, which elaborates on IRF monitoring in the field for early detection of mechanical wear out on the board as well as aging on the chip – well before the damage would accumulate to the critical level resulting in system malfunction.

Section 5 addresses the challenge of systematic discovery of marginal defects, timing related faults and stability issues in board assemblies as a part of end-of-line testing in volume production environment. We have develped a set of embedded instruments on FPGA, which reuse some of the typical functional blocks of the IC converting them into powerful embedded test instruments. The instruments have been evaluated on industry-grade high-end products demonstrating stable detection of injected marginal faults, even such that would otherwise escape the normal functional test.

Section 3 further improves the work done in WP3 by introducing a concept of self-reconfiguring instrumentation networks that would improve the ability of the system to quickly detect and localize faults in the field, including such scenarios as Health Monitoring, Software Based Self-Test, and Power-On Self-Test.

## 1.1 *Structure of the document*

This document concludes activities of WP4 reporting the results achieved in two tasks: T4.3 “Chip-Level instrumentation reuse at the board level” and T4.4 “Automation and integration in a board-level test environment”. These results are mainly based on combined prior achievements reported in D4.1 and D4.2 as well as in the rest of the project, especially in WP1 and WP2.

This report is structured as follows. First, Section 2 describes embedded instruments for IRF detection as a result of wear out on board and the IC. Section 3 continues the topic of aging aiming for NBTI effects and other issues. Section 4 elaborates on the concept of self-reconfiguring on-chip instrumentation networks for fast on-line fault detection and localization. Section 5 and 6 describe how IC-level instruments can improve delay fault and marginal defect detection on the board. Section 7 presents some details regarding instrument integration into a board-level test environment. Finally, Section 8 draws some conclusions and discusses the progress on KPI5.

# 2 Embedded instruments for IRF detection on ICs and boards

## 2.1 *Introduction*

Shrinking feature sizes in semiconductor manufacturing technologies lead to more reliability issues in interconnections. Smaller interconnect dimensions, shrinking transistor geometries, and reduced voltage and noise margins threat the dependability of electronic integrated systems. In electronic systems like system chips and printed-circuit boards, interconnection wiring is heavily dominating the infrastructure and therefore faults in these parts are extremely important.

One of the most challenging interconnect reliability issues that threat dependability of highly-dependable systems are intermittent resistive faults (IRF). IRFs are a specific category of No Fault Founds (NFFs). Like other types of NFFs, they result in many product returns in car and avionic industries; moreover, evoking and detection of these faults are highly time and cost consuming [Dav05].
Marginal or unstable interconnections are the most likely cause of IRFs. In advanced integrated circuits, as well as circuit boards, there are high number of interconnection such as tracks, vias, bumps and junctions. All of these interconnections are vulnerable to IRFs. Moreover, electro migration, corrosion, temperature and mechanical stress cause more increased instability. Cracks and voids in interconnections are susceptible to the above issues.

IRFs manifest themselves as a sequence (burst) of low-level resistance changes in an interconnection. An interconnection can be a chain of interconnect *segments*, including regular chip interconnect, TSVs, vias and micro-bumps in 2.5 and 3D silicon structures. It also extends to (complex) printed-circuit boards level, and can include e.g. cold-solder connections in that case. IRFs might occur randomly during system operational time in any interconnect. IRFs emerge repetitively in a location and they gradually become more severe during the lifetime of the system. Finally they can evolve in a permanent fault [Ria13]. Therefore, it is very important to detect and repair these faults before they become permanent and result in a system failure especially in safety-critical systems.
Intermittent fault (IF) detection is very challenging. IFs are not deterministic and may (probably) not appear during testing. Therefore, for IF detection, highly time and cost consuming techniques such as periodic testing [Kra06] or online monitoring are required. In the case of periodic testing, the probability of detecting IRFs increases whereas in online monitoring technique the behavior of circuits is monitored by different embedded monitors during operational time.

IRF online monitoring may be performed at board- or chip-level. One approach for IRF detection at board-level has been published by us in [Ker15a]. We suggested an enhanced version of IEEE standard 1149.4 to allow online monitoring of wiring tracks in boards. In [Ebr16b] and D2.1 we have proposed a digital monitor which is able to detect IRFs at chip-level. In this deliverable, the proposed IRF monitor has been reused to detect IRF at board level and it has been validated by a programmable hardware IRF generator.
The remainder of this research is organized as follows. First, a review of related works and the state-of-the-art is been introduced. Then, the proposed monitor is presented. In Section 2.3.2, first, a generic simulation model for IRFs is introduced, and then fault simulation results for IRFs as well as aging detection are presented. Last, a hardware implementation of the IRF monitor has been introduced and validated.

## 2.2 *State-of-the-art*

Several papers have studied the influence of intermittent faults on digital systems [Gra14], [Gra10]. In [Gra14] the authors studied the impact of intermittent faults on the behavior of a reduced instruction set computing

(RISC) microprocessor by using VHDL-based fault injection. The authors of [Pan12] proposed a metric intermittent vulnerability factor to characterize the vulnerability of microprocessor structures to intermittent faults. Intermittent fault models at logic and RTL abstraction levels have been generated in [Gil12] and [Gra10]. In addition, the behavior of a microcontroller has been studied in the presence of intermittent faults by simulation-based fault injection.

However, none of the previous work has considered the problem of IRFs at circuit level. We have proposed a model for IRFs and analyzed the influence of IRFs on analogue [Wan14] as well as digital circuits [Ker15b] at the transistor level. In [Ker15b], we presented an extension of the mixed-signal boundary-scan standard, IEEE 1149.4, to detect IRFs in boards. In this deliverable, we continue the investigation of IRF detection and introduce an IRF detection technique at chip-level.

IRFs occurrences in interconnections of a chip result in timing deviation in circuit's paths. There are several approaches toward measuring path delay by using dedicated measurement circuits such as tunable replica circuits (TRC), ring oscillator (RO) and time-to-digital converter (TDC). They will be briefly discussed subsequently.
A TRC is a method for detecting (timing) errors due to voltage and delay degradation. Process variations and aging affects replicas in a similar way in the critical path. However, the random component of process variations will result in differences between the TRC and the actual critical path [Bow09].
The ring-oscillator-based architecture can be used more efficiently for path delay measurement by making the path-under-test a part of the oscillator. The ring oscillator frequency can be monitored *continuously* and any degradation in frequency with time can be attributed to device aging affects [Kim15]. Ring-oscillator based test structures can simply be used to measure the aging degradation effects in digital circuits [Kim15], [Hsi15].
A TDC measures the time interval between two transitional signals. It is typically constructed from registers and delay elements and, like ROs, can be used to measure the effect of process, temperature and voltage variation and aging degradation. However, as the TDC is an on-chip circuit, it suffers from both process variation and aging effects. Hence, it needs self-calibration [Kat15].

However, *none* of the above-mentioned methods can be used for IRF detection. As IRFs *occur randomly in time*, an online monitor should always monitor path activities to detect IRF occurrences. TRC is not applicable for IRF detection as an IRF can occur in every interconnection wire independently. As a result, an IRF in a replica does not mean there is an IRF in the actual path. Similarly, a RO needs a dedicated path and therefore it is not suitable for IRF detection. The usage of TDC for IRF detection is not cost effective in terms of area and power consumption.

Yet another approach towards measuring the timing violations is to use slack monitor accompanied by a guard-band. Most of the aging faults change transistor parameters and increase interconnections' delay. In a synchronous system, a delay increment can result in timing failure. Timing failures occur if the delayed data does not meet a flip-flop's setup requirement and has a late transition near to the clock edge. Hence, online delay (or slack) monitoring in an integrated circuit is a suitable metric for measuring the aging of synchronous circuits [Bow09], [Rah14].
The timing slack or guard-band is defined as the delay, between the data arrival time and active edge of the clock, minus the flip-flop setup time. Generally, guard-band assignment is part of pre-silicon design phase. A guard-band for a monitor is determined based on the slack of the target path and the clock frequency of the system. This deliverable does not focuses on time-windows (guard-band) generation. A practical approach to design a time-windows generator has been proposed in [Reb11]. The same approach can be used for the IRF monitor as well.
Recent works [Lai14], [Sad15] show slack monitoring methods are effective in detecting timing errors such as aging and process and voltage variations. In [Rah14] a timing slack monitoring methodology of inserting

monitors at both path ending nets and path intermediate nets is presented. In [Sad15], the authors presented a digital slack monitor which is able to measure the slack of a selected path. However, their monitor is able to detect either rise or fall transitions and it needs re-initialization for each slack monitoring procedure.
The aforementioned methods cannot be used for IRF detection because they need a pre-initialization and calibration; moreover, they do not provide continuous online monitoring on a data path.

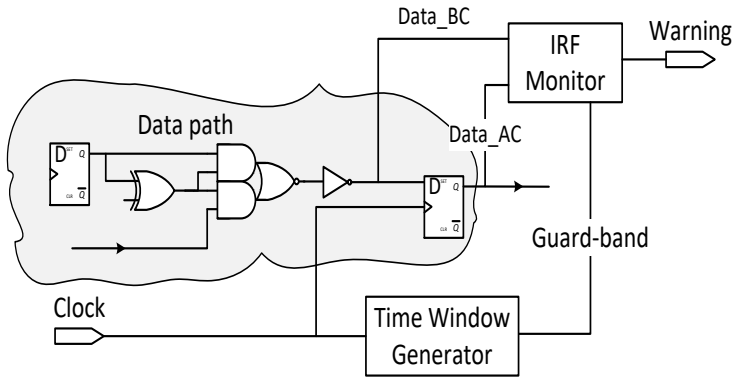



**Figure 2-1. An example of a data path with IRF monitor and time window generator**

In this deliverable, the usage of slack monitors for IRF detection has been investigated. We have proposed a digital IRF monitor, which is able to detect timing variation and distinguish if it is because of aging or IR faults.

## 2.3 *IRF Detection at Chip-level*

### *2.3.1 The proposed monitor*

Timing slack is an excellent measure for the health of a circuit. Measuring timing slack can allow an early warning of deterioration and trigging pre-emptive actions to avoid failure because of aging or IRFs. Basically, a positive timing slack indicates that a circuit is operating safely, with a margin by which the delay can increase before causing a failure. A small or negative quantity of slack is an indication that the circuit is close to, or beyond the point of failure.

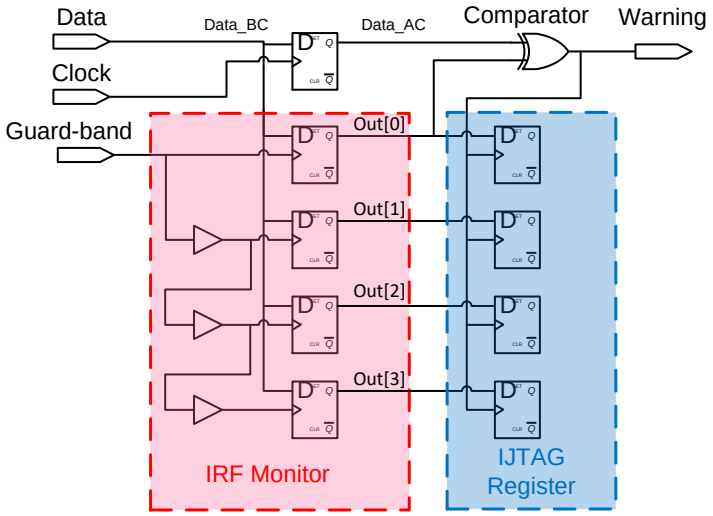



**Figure 2-2. The schematic of the proposed IRF monitor**

The proposed IRF monitor is based on detecting and measuring slack violation on a data path. It is composed of a slack monitor and a time window generator (Figure 2-1). The monitor is located at the end of a data path of the design. It continuously monitors signal at the end of the data path during a timing window provided by the time window generator. It makes sure there is always no slack violation during operation to avoid wrong data be captured.

The schematic of the proposed IRF monitor is presented in Figure 2-2. It raises a warning flag in case of a specified guard-band (safety margin) is violated, which is a sign of an impending timing failure. It consists of four D-flip-flops. Each flip-flop receives its clock from the previous output of a buffer, except the first flip-flop whose clock is connected to the *Guard-band* line. As a result, each D-flip-flop clock is delayed by the delay of one buffer. As all D-inputs are connected to the *Data* line, all D-flip-flops in the monitor can capture the signal on the Data line at different times. *Data_AC* and *Data_BC* are data at the end of the data path after and before capture, respectively.

The IRF monitor uses a very simple concept. If there is not any slack violation in the data line, then correct data is captured in the flip-flop in the path and all flip-flops in the monitor capture the same value. Therefore, the *warning* signal remains zero. In the case of slack violation, e.g. because of aging or IRFs, the *warning* signal becomes one and the degree of violation is captured in the monitor's flip-flops. This degree can be "1000", "1100", "1110" and "1111" in case of fault occurrence in the falling signal or "0111", "0011", "0001" and "0000" in case of fault occurrence in the rising signal. "1000" and "0111" indicate a small timing violation while "1111" and "0000" indicate large timing violation.

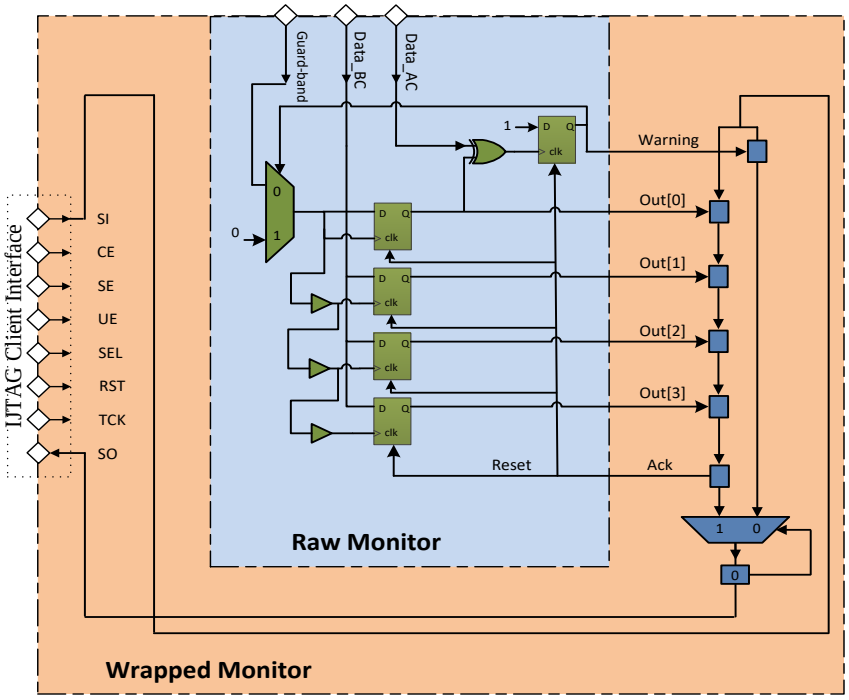



**Figure 2-3. An IRF monitor wrapped by using the IJTAG standard**

The stored data in the monitor's flip-flops will be read by fault-management software through an IJTAG [Shi14] network. Based on this data and the monitors' location, the fault management software can distinguish the type of the detected timing fault: if it is caused by a transient, aging or intermittent fault. If during a certain duration, several faults with different amounts of captured timing violation occurs on a path, it can be classified as an IRF. If only one timing violation happens, it can be classified as transient fault. Finally, it can be an aging fault; in this case the IRF monitor detects several timing violations with the same violation degree.

```
1   iPDLLevel 1
2   iProcsForModule IRF_Monitor
3   set Status 0
4   while { $Status == 0} {
5       iRead IRF_Monitor.Warning
6       iApply
7       set Status [iGetReadData IRF_Monitor.Warning]
8   }
9   iRead IRF_Monitor.Out
10  iWrite IRF_Monitor.Ack 0b1
11  iApply
12  set IRF_Out [iGetReadData IRF_Monitor.Out]
13  iWrite IRF_Monitor.Ack 0b0
14  iApply
```

**Figure 2-4. Simple example of a PDL file of the IRF monitor wrapped by IJTAG**

**Figure 2-3 shows the raw IRF monitor wrapped with IJTAG registers into an embedded instrument. It receives the signals *Data_BC, Data_AC* and *Guard_band.* A simple access procedure written in PDL for the IRF monitor is shown in** Figure 2-4. It consists of a simple logical synchronization mechanism between the IJTAG controller and the monitor. This controller *continuously polls* on the warning flag until a warning is raised (lines 4-8); then it reads the output data and concurrently writes a '1' in the acknowledge register in order to notify the instrument to reset its warning and data outputs (lines 9-11). Finally, the controller resets the acknowledge flag in the next scan cycle (lines 13-14).

### 2.3.2 Intermittent Resistive Faults Model

We have developed a software module to generate IRFs in a Cadence Virtuoso environment, based on some real IRF measurements in [Ker15b], [Acc08]. A basic scheme of the IRF injector has been introduced in D1.2, 0, and 0. The IRF injector generates a burst of IRF pulses based on six parameters. These parameters consists of the number of pulses in a burst (burst length), start and stop times of the burst, active and inactive times, and resistance value for each resistance pulse. After determining the range of these parameters by the user, the IRF injector randomly generates resistance pulses according to these parameters.

One example of the values and distributions applied for the simulations in this deliverable are listed in Table 2-1. After a random start-time such as 1ns or more passes in simulation time, the burst of resistance pulses starts. Each pulse of the burst has a random resistance value *R* witch is active during a random activation time (T-active). After each pulse, an inactivation time (T-inactive) is randomly generated in which a fault-free situation exists (R=100Ω). In the case of a burst (burst length > 1), there is a feedback loop and the same procedure will be followed again. The IRF generation procedure is completed by generating a fault-free situation at the end.

**Table 2-1: Range of used parameters in the IRF generator during fault simulation**

| Parameter | Minimum | Maximum | Distribution |
|---|---|---|---|
| Start time | 1ns | 10ns | Uniform |

| | | | |
|---|---|---|---|
| Resistance | 100 Ω | 100 kΩ | Uniform |
| T-active | 0.1 ns | 2 ns | Uniform |
| T-inactive | 0.1 ns | 2 ns | Uniform |
| Burst length | 1 | 20 | Uniform |
| Safe time | 1 ns | (years) | Uniform |

In this model, the concept of seeds in random variable generation is being used to allow an easy replication of the same intermittent fault for comparisons during simulations. The model has been implemented in Verilog-A and allows replacing a normal wire in the net list by one including an IRF.
By using this model, analogue mixed-signal [Wan14] as well as digital circuits [Ker15b] can be evaluated at the transistor level. The next sub-section will show some results of IRF simulation and the validation functionality of the proposed IRF detection monitor.

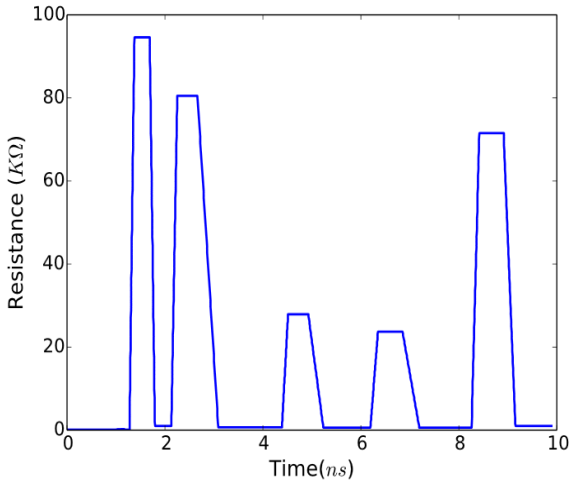



**Figure 2-5. Used IRF for the simulations**

### *2.3.3 Intermittent Resistive Fault Simulation*

In order to evaluate the ability of the proposed IRF monitor to detect IRFs, a well-known concept of fault injection, simulation-based fault injection [Gil12], is used. As an example, an AES-128 encoder circuit in the 45nm Nangate CMOS technology [Nan08] has been used. After synthesis by Synopsys Design Compiler, the circuit operates at a clock frequency of 1.6 GHz.
IRF simulation and the monitor validation has been performed at transistor-level using Virtuoso Cadence. Several critical path and near critical paths were selected based on area constraint and the design's timing information. The proposed monitor was inserted at the endpoint of the target paths.
One example of a simulation fault injection is shown in Figure 2-6 and Figure 2-7. For these fault simulations, the injected IRF in this simulation is as shown in Figure 2-5. It is a burst of five resistance-related pulses from 1kΩ to 100kΩ during 10ns.

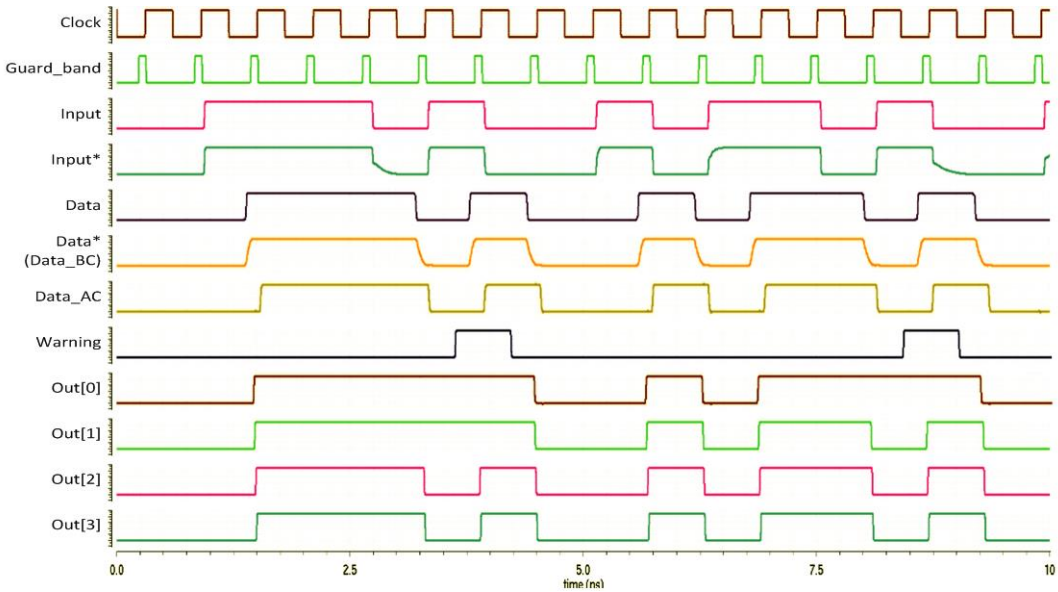



**Figure 2-6. Simulated waveform results of the proposed IRF monitor under IRF injection**

In Figure 2-6, the clock of the system is shown on the top, other signals from top to down are as follows. *Guard-band* indicates when the proposed monitor should be active. *Input* is a sample of data transitions in the beginning of the critical path. *Input** shows the signal Input after the injection of the IRF (Figure 2-5). Signals *Data* and *Data** are the outputs of the path in fault-free and faulty cases, respectively. *Data_AC* is the signal captured by the flip-flop at the *end* of the path in a case of fault, as it can be seen; there is not any functional error in the signal although two small delay degradations are detected by our monitor. Signal *Warning* is the output of the proposed monitor. It shows the monitor detected two late transitions on the data (signal *Data**). The last four signals (*Out[0]*, *Out[Dav05]*, *Out[Ria13]* and *Out[Kra06]*) are the outputs of the flip-flops of the monitor. The captured values by these flip-flops show the value of degradation and path-slack reduction.
At the time 3ns, the monitor captures a timing violation. The value of "1100" is stored by flip-flops of the monitor. Two flip-flops captured a wrong value. It means that the degree of the path's slack is violated as much as two buffer delays. Figure 2-5 shows that the IRF at this time induced by a very high resistance (80 kΩ). Similarly, at 8 ns, a slack violation is detected as much as 1 buffer delay because the value of '1000' is stored in the monitor's flip-flops. By referring to Figure 2-5, it can be seen, the value of the IRF has been near 70 kΩ at 8ns.

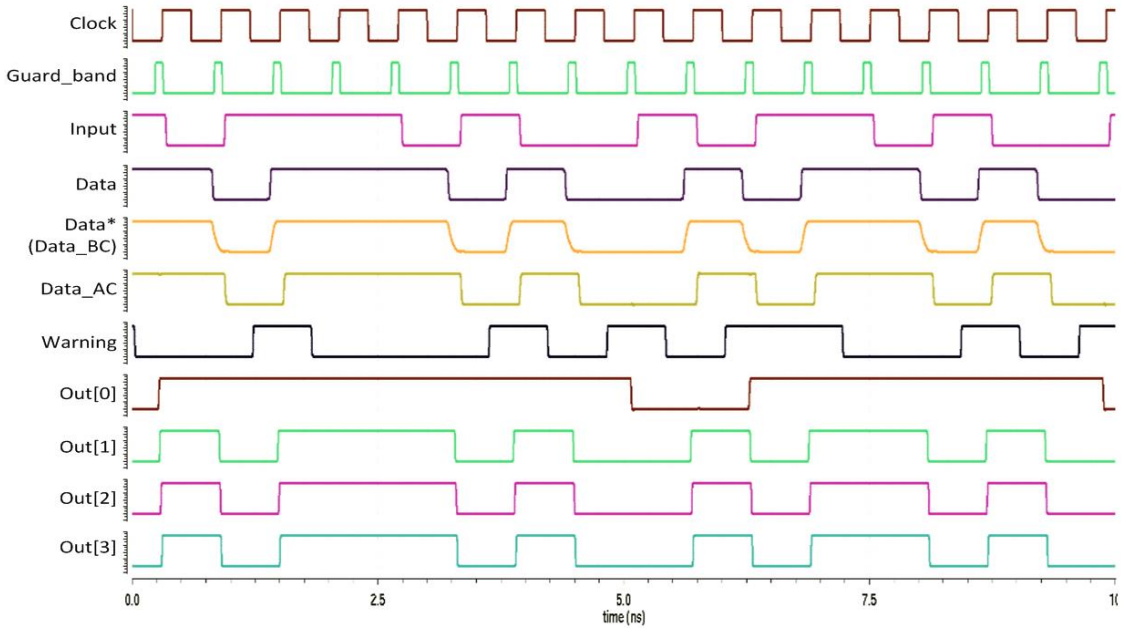



**Figure 2-7. Simulated waveform results of the proposed IRF monitor under NBTI aging conditions**

### *2.3.4 Aging Simulation*

The experimental results in [Sto10] show that NBTI is the primary factor leads to timing degradation in current technologies. NBTI has been shown to cause shifts in threshold voltages of up to 50mV over an operating lifespan of 10 years in 65nm technologies. This translates to more than 20% deterioration in circuit operating speed [Wan07]. To model the delay induced by aging faults (NBTI), the Vth of all PMOSs in the target circuit and the monitor have been increased by 50mV in our Cadence simulator.

Figure 2-7 shows the Cadence Virtuoso simulation results of a combinational circuit where its transistors are subjected to the aging faults. The output (logic) behavior of the circuit as well as the proposed IRF monitor has been evaluated. The signals in Figure 2-7 are the same as ones in Figure 2-6. Signal *Data* shows the simulation results of the target circuit before aging. In this case, there is not any timing (guard-band) violation detected by the monitor, as to be expected. Signal *Data** shows the simulation results of the target data path after aging. By looking at the *Warning* signal it can be seen that the monitor has detected some timing violations. The first flip-flop of the monitor has captured wrong values but other three flip-flops have not captured any violation.

From the above simulations, it can be concluded that for aging detection one flip-flop for each monitor is sufficient. Because, the degree of timing violations in aging faults are not random and in fact gradually increase during a system lifetime. Whereas in IRFs, the degree of timing violations change randomly, hence a number of flip-flops are required to capture violation.

In both aging and IRF detection, after a timing violation, a warning flag raises. It enables the degree of timing violation be sorted in the monitor's flip-flop be transferred to the IJTAG [Shi14] registers. Therefore, by extracting timing violation information and the fault localization using the IJTAG standard, a fault management software environment can distinguish the type of occurred faults whether it is an aging, intermittent or transient fault.

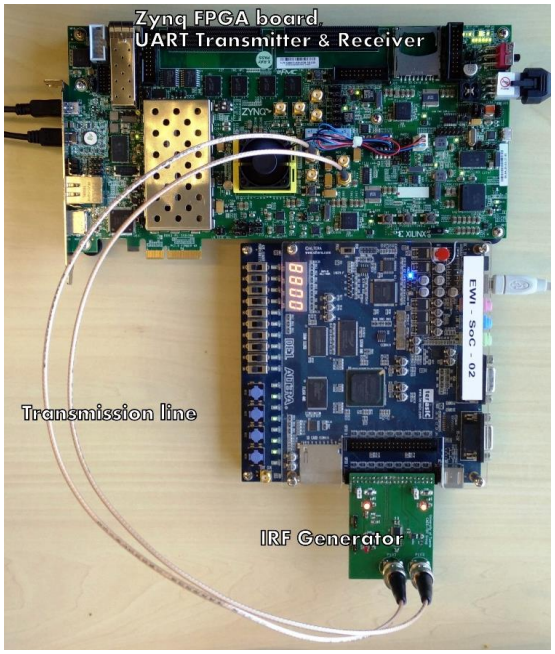



**Figure 2-8: Measurement set-up of the IRF generator and IRF monitor**

## 2.4 *IRF Detection at Board-level*

In this section, the ability of the on-chip IRF monitor in board-level IRF detection has been investigated. For this purpose, the IRF monitor has been implemented on a FPGA and has been evaluated by hardware-based fault injections.

In this deliverable a Universal Asynchronous Receiver Transmitter (UART) has been used as an example to investigate IRF detection at board level. A UART transmitter and a receiver was made at logic gate level and implemented on an FPGA (Zynq-7000 board). This is the standard FPGA board used within the BASTION consortium. IRFs have been injected in the transmission line between the transmitter and the receiver by our hardware IRF generator [Ebr16b] (see Figure 2-8).

A number of different IRF sequences were applied to the data input (RX) of the UART. A nominal power supply Vdd of 3.3V was used and the clock frequency was around 5.5MHz (baud rate ~921 kHz). The UART first receives a start bit and then the less significant bits of 8 bits of data and at the end a parity bit and two stop bits. The parameters of the injected IRF are as depicted in Table 2-2. The injected resistance pulses are randomly chosen from a range between 2.5 Ω and 5 kΩ, which is rather low and hence less likely to cause logic errors.

**Table 2-2: Range of used parameter in IRF generator during fault emulation**

| Parameter | Minimum | Maximum | Distribution |
|---|---|---|---|
| Start time | 0.1 $\mu$s | 1 $\mu$s | Uniform |
| Resistance | 2.5 Ω | 5 kΩ | Uniform |
| T-active | 0.6 $\mu$s | 3 $\mu$s | Uniform |
| T-inactive | 0.3 $\mu$s | 1 $\mu$s | Uniform |
| Burst length | 1 | 2 | Uniform |
| Safe time | 1 $\mu$s | (years) | Uniform |

In Figure 2-9, the result of hardware IRF injection is shown. The time scale is 2 $\mu$s and voltage scale 2 Volt. The *busy* signal shows that when the UART receiver receives data, *Data* and *Data** show the bit stream

received by the receiver before and after fault injection on the data line, respectively. *IRF_active* indicates at what time the IRF is injected. The *Guard_band* signal has been generated by an (on-chip) PLL on the FPGA with a frequency 10 times faster than the system clock frequency. As can be seen, the IRF monitor has detected the timing violation and has raised a *Warning* signal. Here, the detected IRF has been active for 1.8 $\mu$s with a resistance value of 3.2 k$\Omega$. The degree of violation has not been shown in this figure but it has been restored in the monitor's flip-flops with a "0111" value. It should be noted that the IRF has *not caused a logic error* and the sent bit stream "01010110" has been transferred intact without parity or frame error. However, the IRF monitor detected the timing violation caused by injected IRF. This monitor's ability helps a system to avoid going down from a failure.

The comparison of the simulation and hardware-based fault injection shows a significant speed enhancement in IR fault injection. In Section IV, the fault simulation was carried out using a 64-bit Linux machine with 8 GB RAM running at 3.4 GHz. The simulation time was about 300 seconds. Whereas, the hardware-based fault injection can be done in about 30 $\mu$s. This means that the hardware-based injection IRF is up to seven orders of magnitude faster than simulation-based IRF injection.

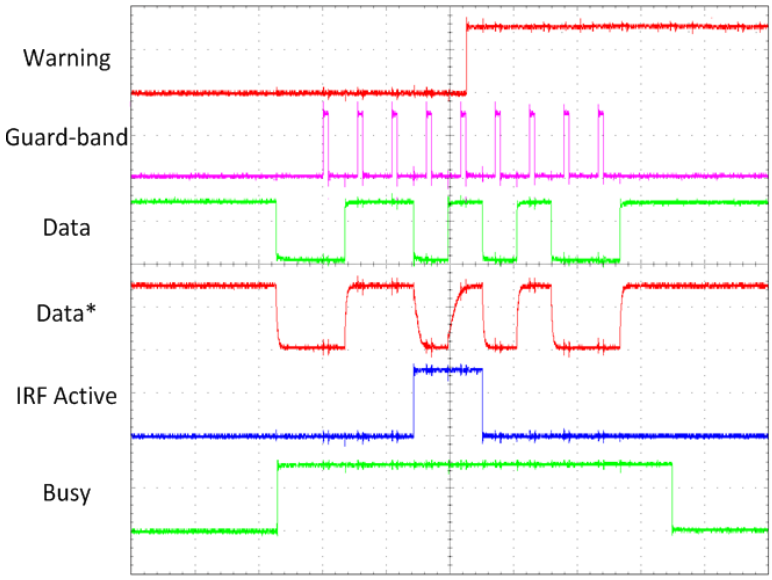



**Figure 2-9. Measured waveform results of the proposed IRF monitor under hardware IRF injection**

## 2.5 *Section Conclusions*

We have presented a digital online IRF monitor to detect one of the most challenging interconnection reliability issues i.e. intermittent resistive fault (IRF). Early-stage detection of this fault can prevent catastrophic failures in safety-critical systems. The experimental results show that the presented monitor can detect small timing deviations produced by IRFs in data paths. In addition, the proposed monitor in actual hardware by hardware-based fault injection has been evaluated. The experimental results show the ability of the proposed monitor to be reused for IRF detection at board level.

# 3 A Framework for Combining Concurrent Checking and Functional Test for Low-Latency SEU and Aging Fault Detection

The focus of this research carried out in BASTION is detection of faults in chip-level modules by combining concurrent checkers with on-board or on-chip functional test to enable cost-effective trade-offs between area-overhead and test coverage. First, we propose a framework of tools for formally evaluating the quality of the checkers and for optimizing the overhead area with given fault coverage constraints. The stress is in particular on the minimization of the error detection latency, which is a crucial aspect in order to eliminate (or limit) error propagation. Second, the concurrent checkers will be complemented by the functional test to be applied as a periodic routine during the idle periods in router operation. The framework together with the corresponding methodology has been successfully applied to a realistic case-study of a fault tolerant NoC router design. The case study shows that combining concurrent routers with functional test allows reducing the area overhead of the checkers from 31-35% down to 1.5-10% without sacrificing the fault coverage.

## 3.1 *Motivation*

One of the main challenges related to the design of modern integrated circuits is the extreme down-scaling of modern technologies that increases the probability of the components to wear-out as well as their vulnerability towards environmental effects. These are phenomena occurring during the life-time of the system and cannot be screened out by manufacturing testing. Thus, cost-efficient mechanisms for detecting faults during system's life-time are needed. These mechanisms should detect errors within routers and enable reconfiguration of the routing network in order to isolate the problem and provide graceful degradation for the system. In BASTION, we propose combining concurrent on-line checkers with functional test in order to achieve early and cost-effective detection of faults.

Regarding the development of on-line checkers, we introduce a new framework and a methodology with a stress on the level of automation, fault coverage, detection latency and area-efficiency. The methodology consists of four main steps. The first step of the methodology is formal checker qualification which includes identification of control-intensive parts of the architecture, converting them to pseudo-combinational counterparts, preparation of the checkers synthesized from verification assertions and specifying the environment in terms of valid input stimuli for the pseudo-combinational circuit. As a result, the faults detected by each individual checker will be calculated.

Second, the number of checkers within the set will be minimized by applying the checker optimization step. As a starting point is the fault detection characteristics for each individual checker as well as their weights in terms of silicon area. Further, a heuristic minimization method is applied resulting in a minimal selection of checkers to achieve a target fault coverage level. The minimization technique is based on a divide-and-conquer approach of partitioning the checkers' fault table into independent clusters. This approach is very effective as the checkers devised for different modules normally do not have overlapping fault sets.
Third, and optional, step of the methodology includes devising additional checkers from temporal assertions for modules that do not achieve 100% fault detection. For these checkers the formal qualification step described above is not possible and traditional fault injection experiments are carried out by a sequential fault simulation tool included to the framework. Finally, the checkers are to be complemented by the functional test.

## 3.2 *State-of-the-art*

Online detection of errors in logic is a thoroughly studied research area. Traditional Triple-Modular Redundancy (TMR) and duplication based approaches are too costly in terms of multiplying the area and correspondingly the power consumption. An alternative to minimize this overhead is the selective TMR that identifies Single Event Upset (SEU) sensitive sub-circuits that are to be protected [Tut1].

In addition, there exists a variety of solutions based on coding techniques such as Berger [Tut2] or Bose-Lin [Tut3] codes. In many works the coding techniques are combined with synthesis [Tut4, Tut5]. The approaches suffer from significant area overhead to the design to be checked.

Concurrent on-line built-in self-test techniques such as Built-In Concurrent Self-Test (BICST) [Tut6] and Reduced Observation Width Replication (ROWR) [Tut7] provide high fault coverage at low area overhead but only consider a limited subset of pre-computed test vectors. Hence these approaches are likely to miss faults occurring in a normal circuit operation.

Several alternatives based on checkers that do not require modification of the circuit under test have been developed. Creating checkers automatically based on logic implications derived from the circuit structure [Tut8] is feasible but suffers from low fault coverage and high area overhead, often exceeding the duplex solutions. On the other hand, deriving checkers from functional assertions, or reusing verification assertions, is similarly known to yield low coverage of structural faults as it is difficult to correlate functional coverage to structural one [Tut9].

Many previous works have focused on addressing faults in the control logic of NoC routers. In [Tut16], Yu et al. have addressed fault tolerance for NoC topologies and proposed an error control method for detecting transient errors in routing logic implemented using Logic-Based Distributed Routing (LBDR) mechanism and its extension for high-radix topologies, LBDRhr. The proposed error control method utilizes the inherent information redundancy (IIR) to reduce the error control overhead. However, the method does not guarantee full fault coverage.

Authors of [Tut17] have presented a method for online error detection and diagnosis of NoC switches. The proposed method deals with routing faults that cause packets to be forwarded to unintended output ports. Regarding modeling routing faults in switches, a high-level fault model has been introduced in this work. The fault coverage is measured only at the functional level and there is no estimates on correlation to gate-level fault coverage.

In order to deliver correctness guarantees for the complete network, Parikh et al. have proposed a network-level detection and recovery solution ForEVeR [Tut14] that monitors the traffic in the NoC and protects it against functional bugs that were not detected during design time. To this end, ForEVeR augments the baseline NoC with a lightweight checker network that alerts destination nodes of incoming packets ahead of time and is used for the recovery process. The approach suffers from extremely high latency. Only 30% of the faults will be detected during the first clock cycle by the approach.

The work in [Tut15] proposes checkers synthesized from a set of 32 verification assertions. The checkers detect most of the injected faults. The faults that are not covered correspond to non-catastrophic failures. The work proposed in [Tut15] is not automated and lacks the completeness and minimization aspects present in BASTION.

In [Tut18] a hybrid method is introduced for synthesis of fault-secure NoC switches utilizing error detecting codes for the data path (data flits) and a concurrent error detection structure for dealing with faults not covered by the flit encoding (using multiple parity trees). However, the work still results in more than 50% area overhead.

The use of embedded test configurations for testing the datapath of NoC routers has been proposed in [Tut19], with design-for-testability structures included in [Tut20] and built-in self-test application in [Tut21]. However,

all the mentioned approaches are targeting the global network and not a concrete router. Furthermore, only off-line test scenarios have been considered in [Tut19, Tut20, Tut21].

BASTION exceeds the existing state-of-the-art in fault tolerant router design by proposing:

- a framework for formal checker qualification. The underlying approach is complete, i.e. it allows proving the absence or presence of true misses by the checkers. In addition, it provides minimal fault detection latency due to the fact that the circuit is transformed into a pseudo-combinational one and therefore only checkers with a single clock cycle latency are considered.
- automated minimization of checkers. The formal qualification of the combinational checkers provides the fault detection capabilities for them. These, along with the checker area requirements are applied in an automated minimization process resulting in a minimal area overhead checker solution under certain fault coverage constraints.
- complementing the resulting checkers with temporal checkers and functional test. This enables combining best of both worlds. In the case of NoC control part, where embedded test packet based approaches have proven inefficient, low area concurrent checkers are applied. On the other hand, in the datapath, the functional test yields full fault coverage whereas error correcting codes would be expensive.

Experimental results on a realistic NoC router design demonstrate the efficiency of the proposed approach.

## 3.3 *The Concept of Concurrent Checkers*

Fig. 1 presents the role of concurrent on-line checkers in detecting faults within a circuit. In addition to the original circuit (functional logic), a set of checkers (checker logic) will be connected to functional inputs/outputs of the circuit. These checkers are derived based on functional assertions obtained from relationships between variables corresponding to inputs and outputs of the circuit. The checker logic targets the faults at lines at the inputs of each gate within the functional logic (marked by green circles). The lines at the functional outputs succeeding the checker inputs (marked by a red cross) cannot be detected by the checker. In addition, the checkers are not targeting the faults at functional inputs preceding checker inputs, since the checker may not detect that the input value has been altered by a fault (such functional input lines are also marked by a red cross in Fig. 3-1). In BASTION, the concurrent on-line checkers target Single Event Upsets (SEUs) and permanent stuck-at faults caused by aging (e.g. BTI).

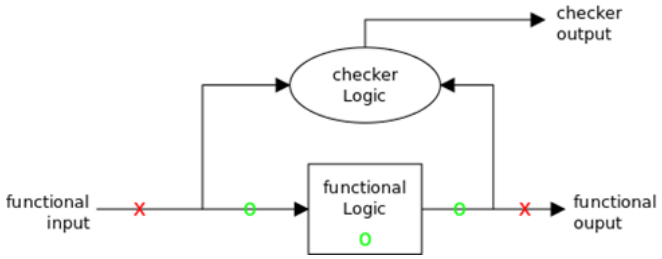



**Figure 3-1. The concept of concurrent checking**

Given a fault at a line within the functional logic and a set of input stimuli, four possible scenarios may occur:

***Case 1***: Fault occurs at an internal line and is visible at functional output(s) and checker logic flags a violation. The term *True Detection* is used to describe this situation, since a critical fault is effectively detected by the checker.

***Case 2***: Fault occurs at an internal line but is not visible at primary output(s). Checker catches the fault and flags a violation. The term *False Positive* is used to describe this situation. False positive is not harmful because

an error is flagged which did not have any effect. However, it has negative impact on design's performance because normally it causes re-execution of the task.

***Case 3***: Fault occurs at internal line but is not visible at primary output(s) and the checker logic does not detect the violation. The term *Benign Miss* is used to describe this situation. Benign miss shows correct operation by the checker.

***Case 4***: Fault occurs at internal node and is visible at primary output(s). Checker does not detect violation. The term *True Miss* is used to describe this situation, which is the worst possible case. True miss means that the fault propagates to the functional outputs and onwards to the system. However, the system has no information that a critical fault has occurred.

Traditionally, in order to evaluate the fault detection quality of the checkers, *fault injection* has been applied. Fault injection refers to injecting faults into a circuit at a certain time step and simulating it with the input stimuli to see whether any functional output of the circuit changes and whether any of the checker output fires. Due to the fact that it is generally impossible to inject and simulate all the faults at each circuit line at each time step, a statistically significant sample of random faults would normally be injected and simulated.

However, in BASTION a methodology is proposed which is based on automated extraction of a pseudo-combinational circuit out of the original functional logic by breaking the flipflops and converting them to pseudo primary inputs and pseudo primary outputs. Further, an exhaustive test for the extracted circuit is fed through a filtering tool in order to derive the complete valid set of input stimuli which will serve as the environment for checker evaluation. This means that in BASTION, full formal qualification of the combinational checkers with all possible stimuli and faults can be obtained.

Let $D$ be the number of true detections, $X$ be the number of benign misses, $F$ be the set of false positives and $W$ be the number of true misses over all the injection runs. In order to evaluate the fault detection capabilities of the checkers we define the metrics of *Fault Coverage* (*FC*), *Checkers' Efficiency Index* (*CEI*) and *False Positive Ratio* (*FPR*) as follows.

$$FC = \frac{D + X}{D + X + W} \qquad (1)$$

$$CEI = \frac{D}{D + W} \qquad (2)$$

$$FPR = \frac{F}{F + X} \qquad (3)$$

Here, FC shows the probability of the checkers behaving correctly over all possible fault cases, CEI shows the probability of checkers ability to detect critical faults whereas FPR reports the ratio of false positives over all the cases a fault did not propagate to circuit outputs. The mentioned three metrics are calculated for checkers by the automated checker qualification framework proposed in BASTION.

## 3.4 *Framework and Methodology*

This Section presents the framework for fault tolerant NoC router design that has been developed as an extension of the Turbo Tester test framework [Tut10]. The proposed methodology of combining concurrent checkers with functional test consists of three main steps:

- Checkers' qualification and minimization (combinational checkers);
- Checkers' evaluation by fault injection (temporal checkers);
- Fault simulation of the functional test.

In the following, these steps are explained in more detail.

### 3.4.1 Checker Qualification and Minimization

Fig. 3-2 presents the qualification and minimization flow for the checkers. The flow starts with synthesizing the checkers from a set of combinational assertions. Thereafter, a pseudo-combinational circuit will be extracted from the circuit of the design under checking. The pseudo-combinational circuit is derived out of the original circuit by breaking the flipflops and converting them to pseudo primary inputs and pseudo primary outputs. Note, that at this point additional checkers that also describe relations on the pseudo primary inputs/outputs may be added to the checker suite in order to increase the fault coverage.

Subsequently, the checkers' qualification environment is created by generating exhaustive test stimuli for the extracted pseudo-combinational circuit. This stimuli are fed through a filtering tool that selects only the stimuli that correspond to functionally valid inputs of the circuit. As a result, the complete valid set of input stimuli that will serve as the environment for checkers' qualification is obtained.

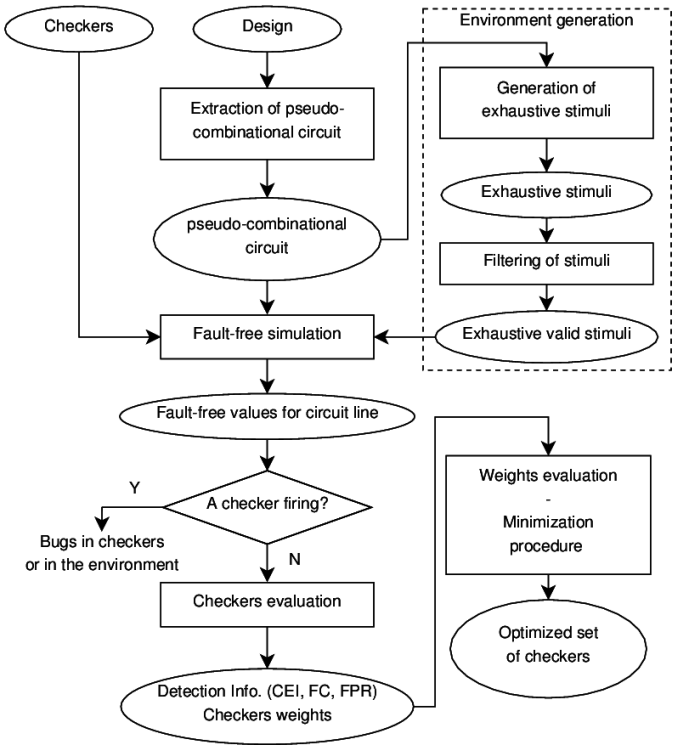



**Figure 3-2. Checkers' qualification and minimization flow**

The obtained environment, pseudo-combinational circuit and synthesized checkers are applied to fault free simulation. The simulation calculates fault free values for all the lines within the circuit. Additionally, if any of the checkers fires during fault-free simulation it refers either to a bug in the checker or an incorrect environment.

If none of the checkers is firing in the fault-free mode then checkers' qualification takes place. The tool injects faults to all the lines within the circuit one-by-one and this step is repeated for each input vector. As a result, the overall fault detection capabilities for the set of checkers, in terms of FC, CEI and FPR metrics will be calculated. In addition, each individual checker will be weighted by summing up the total number of true detections by the checker.

The weighting information will then be exploited in minimizing the number of checkers, eventually allowing to outline a trade-off between the fault coverage, and the area overhead due to the introduction of checker logic.

### 3.4.2 Checkers' Evaluation by Fault Injection

There are cases when a module under checking cannot be handled by the combinational checker qualification and minimization approach. For example the module may have a large number of inputs so that the set of generated valid input stimuli would be too large (e.g. datapath modules) and/or the fault coverage reached by the combinational checkers is too low.

In those cases, the checkers are to be evaluated by traditional fault injection. Here a test bench is created for the design and the circuit with the checkers is simulated by a sequential fault simulator with a sufficiently large random sample of faults injected into the circuit. In BASTION, all the datapath checkers and the FIFO checkers were evaluated using this approach.

### 3.4.3 Fault Simulation of the Functional Test

Finally, the stuck-at fault coverage of the functional test for the NoC router is measured by a fault simulator belonging to the framework. As experimental results show, full fault coverage for the datapath with the test application time of 196 clock cycles is achieved.

## 3.5 Experimental Results

Fig. 3-3 demonstrates the high-level overview of a 5-port 2D NoC router that we have chosen as a target architecture for applying the checkers. The router consists of a datapath and a control part. The datapath is composed of input buffers (implemented as FIFO), one for each input port, a crossbar switch and an output buffer for each output port. The control part contains routing units, arbiters and FIFO control. For the routing unit of our target architecture, we have opted for Logic-Based Distributed Routing (LBDR) [Tut13], which is considered as a scalable solution compared to routing tables. As an arbiter, round-robin arbitration was implemented.

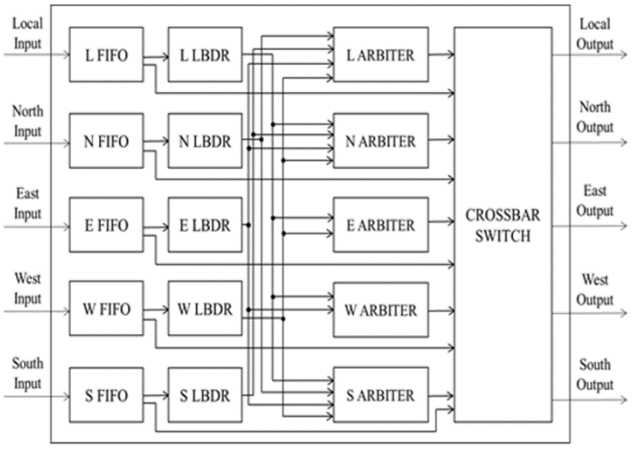



**Figure 3-3. High level architecture of the NoC router**

### 3.5.1 Checker Qualification/Minimization for LBDR/Arbiter

The pseudo-combinational circuit for ELBDR has 11 input bits, as mentioned in the previous section, thus the exhaustive set of stimuli presents $2^{11}$=2,048. A filtering scheme based on the following statements was devised:

- if input buffer's empty signal is high, any other input bit is meaningless, and therefore any value is allowed for it;
- if the incoming flit is a header, the destination address has to be valid according to the XY routing and turns restrictions;
- if the incoming flit is a body or tail flit, the previous output values must be valid, they must follow a one-hot fashion, according to XY routing.

This allowed to obtain a valid and complete set of stimuli consisting of 1536 vectors, which forms 75% of the exhaustive set. The run-time for generating the stimuli was 2 seconds. (All the experiments were carried out on an Asus ux32vd-r4002v computer with a 1.9 GHz Intel Core i7-3517U processor and 10 GB RAM.)
Table 3-1 lists the obtained minimized set of three checkers for the LBDR. Reducing the set of checkers to the three most significant ones allows to limit the area overhead to 78.57% over the ELBDR circuit, far lower than 185.71% imposed by the initial non-minimized set of checkers, while the CEI and FC remain at 100%.

**Table 3-1: A minimized list of checkers for the LBDR**

| Checkers for Routing Logic (LDBR) | | |
|---|---|---|
| 1 | Valid LBDR output | If there is a request to the routing logic (the corresponding input buffer is not empty), LBDR has to compute at least one valid output direction (according to XY routing). |
| 2 | No LBDR output | If no flit arrives (the corresponding input buffer is empty), all the output port signals of LBDR should remain zero. |
| 3 | Single LBDR output | If the corresponding input buffer is not empty (there is a request to LBDR), because of using XY routing, at most only one output port signal of the LBDR logic can become active. |

Similarly, Table 3-2 lists the minimized set of two checkers for the Arbiter that was obtained from an initial set of 28 verification checkers by applying the checker qualification and minimization framework.

**Table 3-2: A minimized list of checkers for the Arbiter**

| Checkers for Arbiter logic | | |
|---|---|---|
| 4 | Valid Grant output | If there is a request from LBDR, arbiter has to assert at least one of the grant |

| | | |
|---|---|---|
| | | signals for the corresponding output direction. |
| 5 | Invalid arbiter State | State variable of the arbiter FSM has to respect one-hot encoding. |

### *3.5.2 Fault Injection Experiments for the FIFO*

Table 3-3 lists the set of 8 checkers generated from the verification assertions for the FIFO control part. The checkers were evaluated by the fault injection tool of the framework. A set of input stimuli for the FIFO was devised, aiming to cover all the possible situations for the control logic. The following conditions were considered in the pattern generation procedure:

- reset condition;
- filling the FIFO, followed by reading up to empty condition;
- smooth traffic condition, i.e. concurrent writing and reading operations, avoiding the FIFO to get full;
- idle condition, i.e. write and read enable signals low, during reading and writing operations, in different conditions of fulfillment of the buffer.

100% CEI and FC were achieved on the control part of the FIFO, considering the patterns derived from the previously listed conditions, amounting to 134. Run time for the experiment was 0.06 s. No false positives were encountered in this experiment.

**Table 3-3: Checkers for the FIFO Control Part**

| Checkers for FIFO control part | | |
|---|---|---|
| 6 | Reset checker | Whenever reset goes high, at the next clock cycle empty flag should be high (reading and writing pointer are reset to the same value). |
| 7 | Flags checkers | Empty and full flags should never be high at the same time. Whenever the defining condition occurs, the corresponding flag should go high at the next clock cycle. |
| 8 | One-hot pointers checkers | Reading and writing pointers have to respect one-hot encoding. |
| 9 | Registers enable DMR checker | Duplication and comparison for the logic enabling the writing operation in data registers. |

| | | |
|---|---|---|
| 10 | Reading pointer update checker 1 | Whenever read enable is high and the FIFO is not empty, at the next clock cycle the reading pointer should be updated. |
| 11 | Reading pointer update checker 2 | If either read enable is low or the FIFOis empty, at the next clock cycle the reading pointer should preserve its value. |
| 12 | Writing pointer update checker 1 | Whenever write enable is high and the FIFO is not full, at the next clock cycle the writing pointer should be updated. |
| 13 | Writing pointer update checker 2 | If either write enable is low or the FIFO is full, at the next clock cycle the writing pointer should preserve it value. |

Table 3-4 lists the set of 3 additional checkers which were included in order to achieve the full fault coverage after fault injection experiments for the control part identified uncovered faults in the interconnections of control part modules.

**Table 3-4: Control Part Infrastructure Checkers**

| Control Part Infrastructure Checkers | | |
|---|---|---|
| 14 | FIFOs read enable DMR checker | Logic producing read enable signals for the FIFOs (5 OR gates) is duplicated, then real and duplicated outputs are compared. |
| 15 | Output registers enable DMR checker | Logic producing enable signals for the output registers (5 OR gates) is duplicated, then real and duplicated outputs are compared. |
| 16 | Flit type LBDR error | Flit type field of a flit has to respect one-hot encoding. |

### *3.5.3 Checkers for the Datapath*

In order to fully cover the faults in the NoC datapath two types of concurrent checkers were introduced (listed in Table 3-5). First, for each input port an even parity bit is included, whereas each output port has a checker

evaluating the even parity. Second, since fault injection experiments for the whole router identified undetected faults within the crossbar multiplexers, dedicated checkers for the crossbar were devised.

**Table 3-5: Checkers for the NoC Datapath**

| Datapath Checkers | | |
|---|---|---|
| 17 | Even parity checker | An even parity bit is computed and added to data entering each input port, which is later evaluated before data leaves the router through any of the output ports. |
| 18 | Crossbar checker | Crossbar MUXs are duplicated, then real and duplicated outputs are compared. |

### *3.5.4 Putting It All Together*

Fig. 3-4 reports the area overhead required by the checkers for routers of varying bitwidth (from 32 bits to 256 bits). It can be observed from the Figure that the required area for the control part checkers stays constant while the overhead area of datapath checkers (parity and crossbar) grow proportionally to the router size.

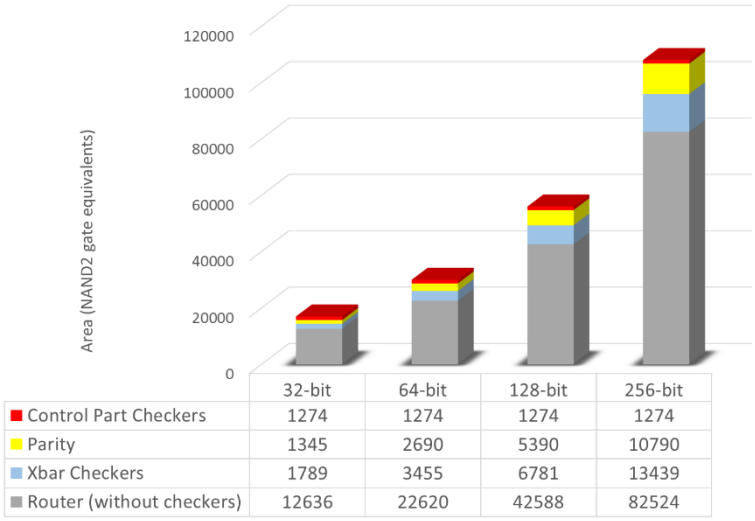



**Figure 3-4. Area consumption for different datawidhts**

The same trend is revealed in Table 3-6. It can be seen that if datapath checkers are included then the required area overhead would be in the range of 31-35%. Whereas, the control part checker circuitry demands significantly less area, especially for larger bitwidths.

**Table 3-6: Overhead Area for Different Datawidths**

| | 32-bit | 64-bit | 128-bit | 256-bit |
|---|---|---|---|---|
| Router (w/o checkers) | 12636 | 22620 | 42588 | 82524 |
| Control part checkers | 1274 | 1274 | 1274 | 1274 |
| Xbar Checkers | 1789 | 3455 | 6781 | 13439 |
| Parity | 1345 | 2690 | 5390 | 10790 |
| Area overhead (contr. p. checkers), % | 10.08 | 5.63 | 2.99 | 1.54 |
| Area overhead (all checkers), % | 34.88 | 32.80 | 31.57 | 30.90 |

However, when combining the control part checkers with the functional test presented in Section 4, full fault coverage for the NoC router can be achieved with a minor area overhead. As it has been shown by experiments in [Tut21] a functional test of length K=196 clock cycles will achieve FC=100% within the NoC router datapath. Thus, combining the concurrent checkers for the control with functional test results in a cost-effective solution.

## 3.6 *Section Conclusions*

We have developed a framework for formal qualification of checkers and for minimizing the overhead area with the given fault coverage constraints. The goal is to achieve low-latency, low area overhead checkers for network on chip routers. In addition, we propose complementing the concurrent checkers with functional test to be applied as a periodic routine during the idle periods in router operation.

The framework together with the corresponding methodology has been successfully applied to a realistic case-study of a fault tolerant NoC router design. The case study shows that combining concurrent routers with functional test allows reducing the area overhead of the checkers from 31-35% down to 1.5-10% without sacrificing the fault coverage.

# 4 Fault Extraction in Self-Reconfiguring IEEE 1687 Networks

## 4.1 *Introduction*

The advances in semiconductor technology have enabled production of integrated circuits (ICs) with very small transistors. As smaller transistors are more sensitive to environmental conditions such as radiation or voltage and temperature fluctuations, it has become crucial to address reliability by handling errors that occur when the IC is in operation [Bo05]. One way to address reliability issues is to embed instruments capable of detecting errors [Su07]. Such instruments can be connected via a fault monitoring network to a fault manager that makes decisions based on the collected error data. In this article, we consider two types of latency in the fault monitoring network:

- *fault detection time*: the time interval between detection of an error by a monitoring instrument and when the fault manager gets aware of presence of errors in the system, and
- *fault localization time*: the time it takes between detection of fault(s) by the fault manager and when the fault manager identifies the faulty resource(s) and extracts the error code(s) reported by the respective monitoring instrument(s).

It is important that the latency in a fault monitoring network is kept low, as the earlier the fault manager identifies the faulty resources, the faster it launches recovery actions. Moreover, the latency should be deterministic to let designers of a system better assess its reliability.

A fault monitoring network can be either stand-alone, or part of an existing functional infrastructure such as network-on-chip or system bus. The advantage of using the functional network for the transport of fault management data is that no additional hardware cost is incurred. One drawback is that adding traffic to transport fault monitoring data may adversely affect the performance of the system. It is difficult at design time to know the amount of data traffic that is to be generated from errors, as the traffic depends on when errors occur. Another drawback is that the predictability of the system is reduced, as the traffic on the functional network might also affect the latency of the fault monitoring data. To be on the safe side, the network might be over-designed to ensure that performance is kept high. This is however costly. The advantage with a stand-alone network is twofold: it does not impact the performance of the system, and simplifies achieving a deterministic fault detection and localization time. The downside of using a stand-alone network is adding extra hardware cost, if it is added only for the purpose of fault management. However, many ICs have stand-alone networks accessed via the IEEE 1149.1 test access port (TAP) [Ieee01] to enable test, diagnosis, configuration, etc., which makes it attractive to reuse such networks for fault monitoring and error handling during operation as well.

There have been a number of works on networks for transporting monitoring data (for transient faults, timing errors, power estimation, etc.) using a dedicated infrastructure [Bou12] [Mad09] [Pet14] [Jut13] [Shi14] [Jut16] [Ibr16] [Zad16]. The works in [Pet14] [Jut13][Shi14][Jut16][Ibr16] [Zad16] stand out as they rely on reusing an IEEE 1687 [Ieee14] network, accessed via the TAP, for monitoring purposes. The works in [Ibr16] [Zad16] propose the use of networks that automatically reconfigure themselves (i.e., perform self-reconfiguration) to include the instrument registers that contain error codes in the active scan path.

In this section, we detail a hardware module for the self-reconfiguring networks proposed in [Zad16] that discovers the current configuration of the network (after self-reconfiguration) and extracts error information in real time. The proposed hardware module can also be utilized by the approach proposed in [Ibr16] during the localization process. To validate our proposed approach, as well as to give an idea of the associated hardware overhead, we implemented multiple self-reconfiguring networks for different number of instruments, and performed post-layout simulations in 65nm technology.

## 4.2 *Background and Prior Work*

Hierarchical 1687 networks have been used in fault management schemes to connect instruments to a fault manager [Pet14] [Jut13][Shi14][Jut16][Ibr16] [Zad16]. In [Pet14], a simulation-based platform for

experimenting with fault injection and fault management is elaborated, but no time analysis or network optimization method is presented. In [Jut13], methods for optimized design and calculation of error localization time are presented for their proposed fault management scheme. The work in [Shi14] extends [Jut13] by elaborating on how the fault manager can react faster to new faults while the instrument access network is in use for other purposes and how multiple faults can be addressed, but presents no time analysis method or experimental results for such cases. The work in [Jut16], extends [Jut13], [Shi14] by addressing the reliability issues in the fault monitoring network itself.

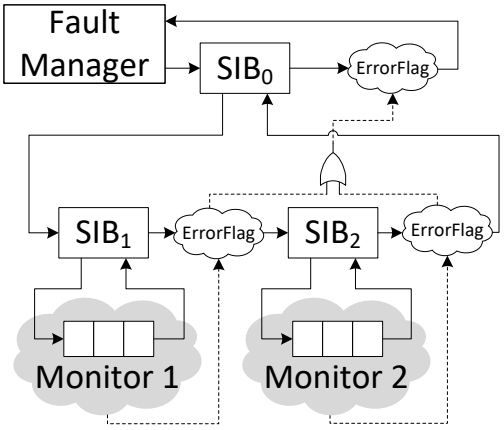



**Figure 4-1. A simplified representation of the basic idea in [Jut13]**

Along with the 1687 network, [Pet14] [Jut13] [Shi14] [Jut16] [Ibr16] [Zad16] use a fully combinational error flag propagation network that propagates error flags to the highest hierarchical level of the 1687 network. A simplified representation of the hierarchical networks used in [Jut13] is shown in Figure 4-1, where the dashed lines mark the error flag propagation network. The advantage with using such a propagation network is that by reading the ErrorFlag register in the highest level the fault manager gets informed of any error in the system without checking each and every instrument. To guide fault localization, [Pet14] [Jut13] [Shi14] added ErrorFlags at every level, resulting in dramatic increase in fault localization time. Also, fault localization in [Pet14] [Jut13] [Shi14] involves a number of CSUs to open hierarchical levels, each CSU performed over a scan path longer than the scan path used in the previous CSU, increasing the fault localization time.

In our previous work, [Zad16], to reduce the fault localization time, we considered a fault management scheme similar to those in [Pet14] [Jut13] [Shi14] and proposed self-reconfiguration. In the proposed scheme in [Zad16], the 1687 network is self-reconfigured (while maintaining compliance with IEEE 1687) to automatically include the instrument registers containing error codes in the active scan path. The proposed scheme achieves significantly faster fault localization compared to [Pet14] [Jut13] [Shi14].

The work in [Ibr16], which was published shortly after [Zad16], proposed an architecture with three modes of operation:

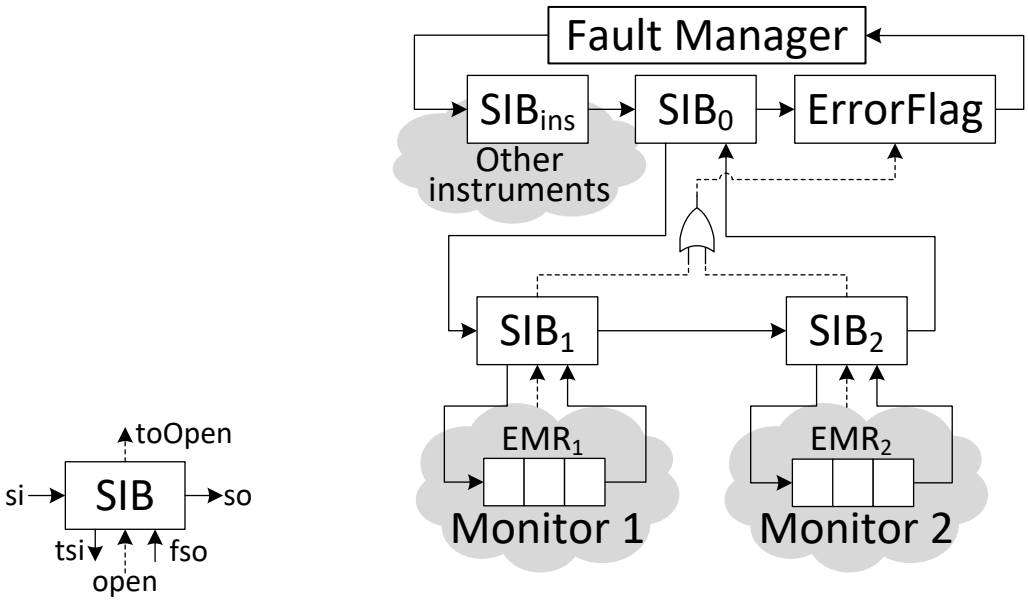
- In the first mode the architecture resembles a standard hierarchical SIB-based network.
- In the second mode—similar to the approaches in [Pet14] [Jut13] [Shi14]—error ***flag(s)*** corresponding to each of the SIBs are also included in the scan path (mainly for performing monitoring by an off-chip controller).
- In the third mode, the SIB's shift (S) flip-flop is also removed from the network and only the fault flag registers remain on the scan path. It is in this third mode in which—similar to our proposed self-reconfiguration scheme ***[Ibr16]***—the network reconfigures itself depending on the current state of errors in the system.

Neither [Ibr16] nor [Zad16] detail or discuss the hardware or software needed for the localization (i.e., analysis of the bit vector shifted out after self-reconfiguration in order to localize faults and extract error information). In this paper, we detail a hardware module that performs the localization and extraction of the error information in real time.

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

## 4.3 *Self-Reconfiguring Network*

In this section, we describe the hardware structure of the self-reconfiguring networks [Zad16]. For the hardware platform, we assume an IC equipped with embedded monitoring instruments (referred to as fault monitors) that can detect errors and produce error codes accordingly. Moreover, we assume there exists a hierarchical 1687 network interfacing all embedded instruments (test, debug, fault monitors, etc.) in the IC to the TAP circuitry. For the purpose of fault management, we consider that an on-chip entity, referred to as Fault Manager, can gain access to the 1687 network via the TAP. The Fault Manager detects and localizes errors that may occur in different components of the system over time, and initiates necessary fault handling actions. The basic idea in self-reconfiguration is that when a fault is detected by a fault monitor, the corresponding error code register is automatically included in the active scan path so that its contents can be readily shifted out and analyzed. Such scheme, improves the speed of fault localization via (1) avoiding to open layers of hierarchy one layer at a time, and (2) using only one single-bit ErrorFlag register instead of placing multiple such registers at each hierarchical level.




**(a)** **(b)**

**Figure 4-2. (a) Symbol for the *modified* SIB, and (b) an example self-reconfiguring network (the dashed line represents the error flag propagation network)**

Figure 4-2(b) shows an example of self-reconfiguring networks [Zad16]. The assumption is that among all instruments, there is a set of fault monitoring instruments. In the top level of the hierarchical network, the fault monitoring instruments are connected through a dedicated SIB, denoted by $SIB_0$, while all the other instruments (test, debug, etc.) are connected through another SIB, denoted by $SIB_{ins}$. The top level also includes a one bit shift-register (ErrorFlag) to indicate if any errors are detected by any of the fault monitoring instruments.

The work in [ Zad16] assumes the following for the fault monitors. A fault monitoring instrument has a *fault flag* output terminal that is set to logic '1' in case a fault is detected. The *fault flag* stays active until it is acknowledged via a *clear flag* input terminal. The fault flag signal will be used as an input to reconfigure the network, such that an access to the fault monitoring instrument is enabled. Furthermore, the fault flag signal is propagated across the hierarchical levels and is finally captured by the ErrorFlag register in the top level of the hierarchical network. Additionally, a fault monitoring instrument produces an error-code, which is parallel-loaded during the capture state into an error-code/mask shift-register (EMR) that interfaces the instrument to the 1687 network. An EMR is assumed to have capture and update features (similar to standard 1149.1 TDRs) and it contains an error-code field (written by the fault monitor) and a mask field (written by Fault Manager). Error masking is used to stop a permanent fault from constantly raising the fault flag. To be compliant with the IEEE 1687 standard, error masking should be enabled by default at reset to disable self-reconfiguration of the network. When the EMR of a fault monitoring instrument is selected and data is shifted through it, the *clear flag* is asserted to indicate that the fault from the fault monitor has been acknowledged. In Figure 4-2(b), the 3-bit registers, namely $EMR_1$ and $EMR_2$, are the EMRs associated to Monitor 1 and Monitor 2, respectively.

To enable self-reconfiguration, [ Zad16] proposes a *modified* SIB, which is the core component in a self-reconfiguring network. A *modified* SIB, while being IEEE 1687 compliant, can additionally be opened asynchronously via a dedicated terminal. The symbol shown in Figure 4-2(a) will be used in the rest of this section to represent a *modified* SIB.

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

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

## 4.4 *Fault Manager*

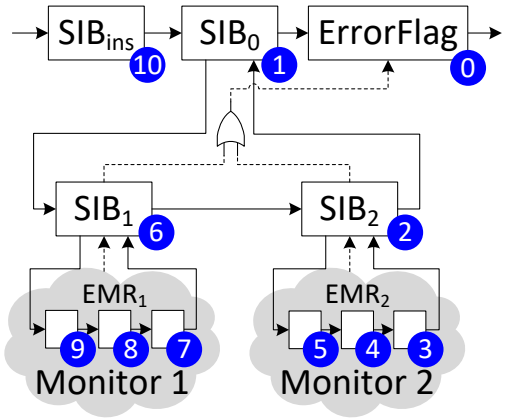
In this section, we elaborate on the tasks of Fault Manager, and propose one possible implementation. To operate the on-chip monitoring instruments and take necessary actions upon detection of an error, Fault Manager should perform the following tasks:

1. activation of the monitoring instruments (after reset) by clearing their mask bits,
2. polling ErrorFlag and launching the localization process in case faults are reported in the monitored system,
3. analysis of the bit sequence shifted out from TDO during localization, to determine which monitoring

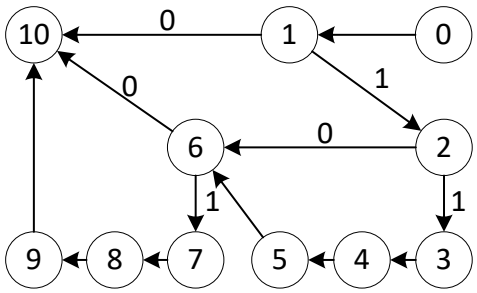
instrument has raised the error flag and to store the reported error code,
4. taking necessary actions based on the error code reported by the monitoring instrument that has raised the flag, and
5. disabling a fault monitor that keeps raising the fault flag (either due to a permanent fault or due to that the monitor itself is defective) by setting its mask bit.

Except for the first item in the above list, the way each of these tasks is carried out affects the fault detection and localization time. This effect is particularly more dramatic for the third item above as that task might involve processing long sequences of hundreds of bits. If the analysis of the shifted out bit sequence (during localization) is done after the shifting is complete, the analysis time is added to the fault localization time, which can increase the localization time significantly. Moreover, for such post processing, the bit sequence should be stored first, which requires allocation of buffers of adequate length. If, on the other hand, the processing is done at the same time as the bit sequence is shifted out, the need for the buffer is obviated and the analysis can overlap in time with the shift-out. This, however, enforces a constraint on the maximum amount of time that processing of each bit can take, otherwise shifted-out data might be lost. Let us take a closer look at how this analysis can be done by taking the example network in Figure 4-2(b) presented again in Figure 4-3(a).

Assume that a fault is detected by a monitoring instrument, the network has performed self-reconfiguration accordingly, and Fault Manager has detected the fault by reading the ErrorFlag and has subsequently opened SIB0 to start the localization process. The numbered circles next to the components in Figure 4-3(a) denote the order that those bits appear at TDO during the shift-out (under the assumption that all components are part of the active scan path). The finite state machine (FSM) in Figure 4-3(b) shows how by looking at the values shifted out, Fault Manager can discover the current configuration of the network, identify the faulty resources, and collect the error codes. Values read at TDO, when corresponding to SIBs, are used to determine to which component the next bit in the sequence belongs. These values are therefore used to label the transitions in the FSM for the SIBs, that is, where more than one output transition from a given state exists. By using such an FSM, Fault Manager can be guided during the localization process to detect the current configuration of the network and to collect the bits corresponding to each error code. For example, assuming that Monitor 2 has detected a fault, SIB2 is automatically opened and the bits corresponding to EMR2 are included in the scan path. In this case, upon reading a value of '1' for bit number 2 at TDO, Fault Manager enters state 3 in the FSM. From this point, Fault Manager should start collecting bits 3–5 as the error code and store the collected code upon leaving state 5.




(a) A self-reconfiguring network




(b) An FSM that identifies which bit is being shifted out, for the network in Fig. 3(a)

**Figure 4-3. Detecting the current network configuration based on the values being shifted out can be done by an FSM**

The FSM shown in Figure 4-3(b) is not complete as it does not capture the actions that should be taken for each fault that is localized. We will shortly elaborate on this issue. For the moment, we should note that a standard way of implementing an FSM in software is by using a state transition table. For the simple FSM in Figure 4-3(b) one such table looks like what is shown in Table 4-1. For such a table, the current state is just a pointer to a row (i.e., it is not stored in the table), and each row contains pointers to the next state based on the value observed at TDO. This way, the next state can be computed instantly for each observed bit at TDO.

**Table 4-1: Storing the FSM in Figure 4-3(b) in memory as a state transition table**

| Current state | Next state | |
|---|---|---|
| | TDO=0 | TDO=1 |
| 0 → | 1 | 1 |
| 1 → | 10 | 2 |
| 2 → | 6 | 3 |
| 3 → | 4 | 4 |
| ⋮ | | |

In order for Fault Manager to process the localization bit sequence, it should either be running on a faster clock (compared with TCK) so that it does not fall behind in case it needs to perform other tasks while performing the localization, or allocate buffers for temporary storage of the shifted-out bit sequence. To avoid these two limitations, namely, the faster clock and buffer allocation, as well as to avoid storing a table such as Table 4-1, we propose and detail a hardware module that runs on same clock as the network (i.e., TCK) and performs all tasks related to the monitoring instruments. More specifically, the proposed module performs tasks 1, 2, 3, and 5 mentioned in the beginning of this section, and therefore, relieves Fault Manager from having anything to do with the monitoring network, which is the self-reconfigurable network connected to SIB0. We will refer to this hardware module as *Monitors Manager* and will show that its hardware area can be lower than the area of memory that its software-based counterpart would require.

### 4.4.1 *Monitors Manager's Interface*

**Error! Reference source not found.** shows how the proposed Monitors Manager is interfaced to the 1687 Network and Fault Manager. The assumption is that Fault Manager is implemented as software running on an on-chip microprocessor.

When the **mode** signal is set to 0, Monitors Manager is connected to the TAP controller in the network. After the reset, the Monitors Manager module waits (while keeping the network's TAP controller state machine in the Test-Logic-Reset state) until it receives the **unmask** signal from Fault Manager. It then starts opening the SIBs in the network level by level until the EMRs are part of the scan path. It will then clear all mask bits while closing all the SIBs. This is rather straightforward as the proposed network construction method in [Zad16] places all the EMRs at the same level, making it relatively easy to embed this unmasking feature into the Monitors Manager module. Monitors Manager signals the completion of the unmasking through asserting the **unmasked** signal.

After the initial unmasking, if the **goto-rti** signal is active, Monitors Manager keeps the network's TAP controller state machine in the Run-test/Idle state, otherwise it starts the fault detection and localization process. The purpose of **goto-rti** is to signal Monitors Manager to stop the fault detection and localization process and take the network's TAP controller state machine back to the Run-test/Idle state. This way, Fault

Manager can take over (by switching the **mode** to 1) and access the other instruments in the network (i.e., those connected to SIBins, in order to take actions based on the detected errors).

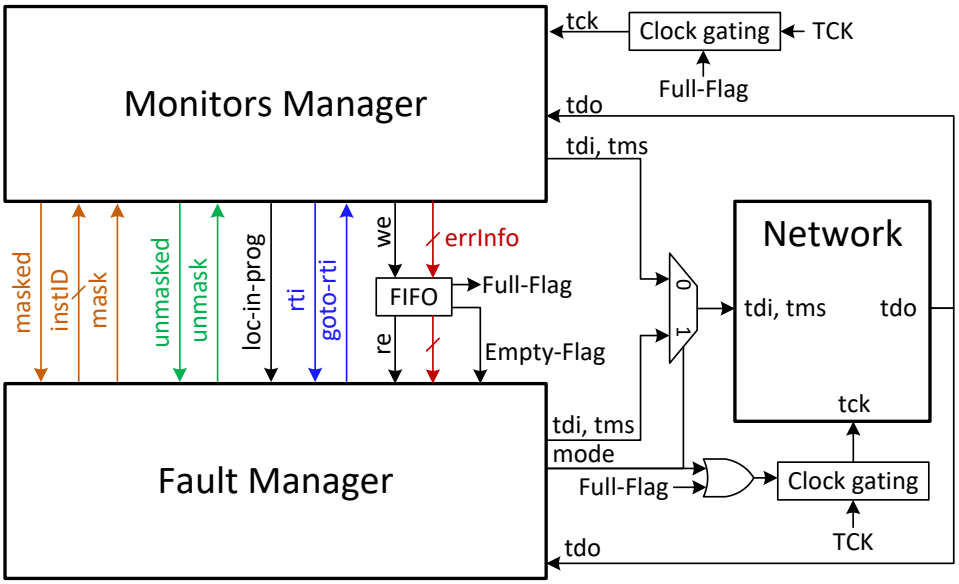



**Figure 4-4. A system view of the proposed fault management scheme illustrating the interfaces between Fault Manager, Monitors Manager, and the self-reconfiguring network**

When a fault is detected and localized by Monitors Manager, the instrument ID and the error code (i.e., contents of the corresponding EMR except for the mask bit) is pushed into the FIFO. Fault Manager is notified of existence of errors in the system by polling the **Empty-flag** of the FIFO. Alternatively, the **Empty-flag** can be interfaced as an interrupt signal. The **Full-flag** of the FIFO is used to *freeze* the operation of Monitors Manager in case Fault Manager has fallen behind in reading the error information from the FIFO. The freezing halts the clock to both Monitors Manager and the network, preventing the loss of the error information that is to be reported by Monitors Manager. This way, the errors that are detected and whose code is being shifted out stay in the scan path waiting to be shifted out, and new error flags will propagate and be detected in the next round of localization.

When a monitoring instrument keeps raising the error flag—either due to a permanent fault or due to that the monitoring circuitry itself is defective—Fault Manager can mask that instrument by placing its ID number on **instID** and asserting the **mask** signal. As will be detailed shortly, this masking is only possible when the requested instrument keeps raising the error flag and is thus part of the active scan path after self-reconfiguration. After receiving the masking command, Monitors Manager sets the mask bit in the EMR corresponding to the specified instrument, and asserts the **masked** signal.

Finally, the Monitors Manager module asserts the **loc-in-prog** signal whenever it detects a fault and starts the localization process. This signal can help Fault Manager in certain cases. For example, if a masking request is not acknowledged and localization is not in progress either, it means that the instrument that Fault Manager is trying to mask has cleared its error flag before being masked (i.e., the detected error has not been permanent).

### 4.4.2 *Internal Operation of Monitors Manager*

Internally, the Monitors Manager module is a state machine. In our experiments, we constructed this module automatically based on the description of the self-reconfiguring part of the network. In this section, we explain how such a state machine performs the fault localization as well as the masking/unmasking tasks.

**Table 4-2: The localization state transition table for the network shown in Figure 4-3(a)**

| Current state | Next state | | Output signal assignments | | | Other actions |
|---|---|---|---|---|---|---|
| | tdo==0 | tdo==1 | mask==0 | mask==1 && instID==1 | mask==1 && instID==2 | |
| 0 → | 1 | 1 | tdi=0; tms=0; masked=0; | | | |
| 1 → | 10 | 2 | | | | |
| 2 → | 6 | 3 | | | | |
| 3 → | 4 | 4 | | | | EMR=tdo; |
| 4 → | 5 | 5 | | | | EMR=(EMR<<1)\|tdo; |
| 5 → | 6 | 6 | errInfo=(2<<2)\|EMR; we=1; | errInfo=(2<<2)\|EMR; we=1; | tdi=1; masked=1; | |
| 6 → | 10 | 7 | tdi=0; we=0; | | | |
| 7 → | 8 | 8 | | | | EMR=tdo; |
| 8 → | 9 | 9 | | | | EMR=(EMR<<1)\|tdo; |
| 9 → | 10 | 10 | errInfo=(1<<2)\|EMR; we=1; | tdi=1; masked=1; | errInfo=(1<<2)\|EMR; we=1; | |

### 4.4.2.1 Performing the Initial Unmasking:

The network design method proposed in [Zad16] constructs the network such that all instruments are placed at the same depth (hierarchical level) in the tree. Based on this, the following is done to perform the unmasking:

- one CSU is applied to open each hierarchical level. For each CSU, a counter is loaded with the number of SIBs currently on the scan path (i.e., total number of SIBs on all currently opened hierarchical levels) plus one for the ErrorFlag. This counter is decremented with every clock cycle while a '1' is shifted out from Monitors Manager's tdi, until the counter reaches zero. Then a '0' is shifted out for SIBins to keep it closed, followed by the update operation.
- after all hierarchical levels are opened in the previous step, one final CSU is needed for clearing the mask bits and closing the SIBs. This time, the counter is loaded with the total length of the scan path and is decremented with every clock cycle while a '0' is shifted out from Monitors Manager's tdi, until the counter reaches zero. After this, an update is performed and the network's TAP controller state machine is taken back to the Run-Test/Idle state.

Implementing the above steps as a state machine is pretty straightforward and we skip detailing it further.

### 4.4.2.2 Performing the Localization and Fault Masking:

Before delving into the details in this section, we should mention that for the sake of simplicity in presentation, we disregard the mandatory tri-state delay element that is present at TDO [Ieee01]. In our implementation, we have taken the resulting half-cycle delay into account.

The Monitors Manager starts the localization process after detecting that ErrorFlag is set to one. If it is detected that the bit corresponding to ErrorFlag is '1', in the same CSU, a '1' is shifted in to open SIB0. As it is rather straightforward, we skip detailing the detection part of the FSM in Monitors Manager. We, however, detail how our proposed hardware implementation performs localization, while carrying out the instrument masking requests.

One important assumption behind our implementation is that when Fault Manager requests a monitoring instrument to be masked, the EMR for that instrument should be already part of the self-reconfigured active scan path. That is, that monitoring instrument should have raised the error flag. This assumption is justified by noting that if an instrument that reported an error earlier is not on the active scan path in the current localization round, the corresponding fault had not been a permanent one, hence no need for masking.

We use the state transition table presented as Table 4-2 to explain how the FSM in Monitors Manager performs the localization and masking for the example network in Figure 4-3(a). In practice, we have not used such a table in our implementation and have directly implemented the FSM in a hardware description language. The table, however, makes it easier to explain the localization process, and helps in getting a more realistic view of the memory usage for a software implementation of the Monitors Manager module. In the table, we have used the C language expressions to explain the low-level hardware operations.

In Table 4-2, the current state of the FSM is a pointer to a row in the state transition table. The next state is determined solely by the value observed at the **tdo** terminal. For example, in state 1, if the observed value is zero, the next state is 10 otherwise 2. The output signals, on the other hand, are determined by the values present at the **mask** and **instID** input terminals. Once an output signal is assigned it retains its value until next time it is assigned a new value. That is, empty cells in are in fact repetitions of the closest non-empty cell above them, and are left empty to reduce clutter. At the beginning of the localization process (namely, state 0) the **tdi**, **tms**, and **masked** outputs are all set to zero.

As an example, assume that both instruments in the network in Figure 4-3(a) have raised the error flag, but Fault Manager has requested Monitor 2 to be fault masked. That is, **mask**==1 and **instID**==2. In this case, once the FSM is in state 3, it starts buffering the bits corresponding to the EMR for Monitor 2—regardless of the **mask** and **instID** inputs—and it is only in state 5 when masking conditions are tested. If the monitoring instrument is not to be masked, its ID and error code are concatenated to form the error information and are pushed into the FIFO via the FIFO input terminal **errInfo** and by asserting the write-enable (**we**) signal. In the concatenation expression errInfo=(2<<2)|EMR;, the first 2 is the instrument ID and the second 2 is a two-bit left-shift. The reason for the shift is that in our example, the EMR has three bits, one of which is the mask bit, which is not needed to be included in the error information. If, as is the case for Monitor 2, the instrument is to be masked, a '1' is placed on the Monitors Manager's **tdi** output, to be shifted in for the mask bit, and the **masked** output is set to '1'. In state 6, the **tdi** and **we** outputs are set back to '0'. As Monitor 1 has also raised the error flag, the **tdo** input will have the value of '1' and the next state is set to 7. In state 7, similar to state 3 discussed above, the Monitors Manager module starts to buffer the error code and finally in state 9, the collected error code is pushed into the FIFO.

## 4.5 *Validation and Hardware Overhead*

To validate our proposed Monitors Manager module and give an idea of the hardware overhead associated with it, we implemented self-reconfiguring networks and their associated Monitors Manager for 27, 81, 243, 729, and 2187 instruments. We performed synthesis and place & route (optimized for a 100MHz TCK) using a 65nm technology. The target cell density was chosen as 70 percent, which was achieved for the target 100MHz clock frequency. Through post-layout simulations, we established the practicality of the proposed self-reconfigurable networks and the Monitors Manager module. In this section, we report on the operation of and the hardware overhead associated with the Monitors Manager module.

For the validation, during the post-layout simulations, we performed the following:

- Instructed the Monitors Manager module to perform the initial unmasking,
- inserted faults into the system by raising fault flags associated with some of the monitors, and observed that the Monitors Manager module correctly identifies the IDs of the associated instruments and inserts the right ID and error code into the FIFO,
- inserted a permanent fault into the system by constantly raising a fault flag, and instructed the Monitors Manager module to mask the corresponding monitor.

To justify the hardware overhead associated with the Monitors Manager module and its associated circuitry, such as FIFO and clock gating logic, we report the standard cell area occupied by these modules, and make a rough comparison with the SRAM area required for a "partial" software implementation of the Monitors Manager module. For the "partial" software implementation, we only consider the memory required to store the state transition table as presented in Table I (which does not include the actions detailed in Table II). The aim of the comparison is to show that the area of the SRAM bit cells required for a software implementation will be more than the area taken by the hardware implementation of the Monitors Manager module.

Table 4-3 presents the hardware area, as well as SRAM area estimation for a software implementation of Monitors Manager. In Table 4-3, the second and the third columns present the standard cell area taken by the 1687 network and the circuitry associated with the Monitors Manager module in square micrometers, respectively. Columns four to eight present how we have estimated the equivalent SRAM area required for the "partial" implementation of the software counterpart for the Monitors Manager module. Column four shows the number of states in the localization state diagram. Column five shows the number of memory locations required for the storage of the state diagram (two locations per state). Column six presents the number of bits required per memory location. When the number of memory locations is less than 256, we considered that eight-bit memory cells can be used, otherwise 16-bit cells. Column seven shows the total number of bits, and column eight presents the total area, assuming 0.499 $\mu$m$^2$ per cell [Tsmc16]. In this computation of the total area, we have only considered the area taken by the 6T SRAM bit-cells required for the localization process, and have disregarded the impact of these cells on the area taken by the decoding circuitry, sense amplifiers, etc. Comparison of the area reported in the third and the eighth columns shows that the area taken by a hardware implementation of the Monitors Manager module is less than the partial software implementation for 243 and higher number of instruments, and very close to the partial implementation for 81 instruments. It is only for the case of 27 monitoring instruments that a hardware implementation shows higher overhead compared to its software counterpart. Even in this case, a full software implementation of all tasks performed by Monitors Manager might result in more SRAM area compared with the hardware implementation of the Monitors Manager module.

**Table 4-3 Hardware area and estimated for SRAM area that would be used by the software-based approach**

| Number of instruments | Hardware area in $\mu m^2$ | | SRAM memory required for software-based localization | | | | |
|---|---|---|---|---|---|---|---|
| | 1687 network | Monitors Manager | Number of states | Memory locations | Bits per location | Total bits | Total area |
| 27 | 8779 | 4511 | 123 | 246 | 8 | 1968 | 982 |
| 81 | 26639 | 6691 | 366 | 732 | 16 | 11712 | 5844 |
| 243 | 77390 | 12646 | 1095 | 2190 | 16 | 35040 | 17485 |
| 729 | 231541 | 29793 | 3282 | 6564 | 16 | 105024 | 52407 |
| 2187 | 703424 | 81175 | 9843 | 19686 | 16 | 314976 | 157173 |

## 4.6 *Section Conclusions*

In this section, we detailed a hardware module to perform the localization that detects the configuration of the network after self-reconfiguration and extracts the error information in real time. We showed that for large number of instruments, this hardware-based localization requires less area compared with a software-based localization.

# 5 Employing Bit Error Rate Measurements for Board-Level Marginal Fault Test

The fault detection embedded instruments developed in D2.2 can be reused and adapted in a board environment. Within this task, we have implemented two instruments that reuse chip-level instrumentation and target timing-related faults studied in D1.3 for both serial and parallel data buses. The first one, Bit Error Rate Tester (BERT) instrument, can be applied to test high-speed serial links (HSSL), which are widely used in modern electronics for system-to-system communication. Second, DDR memory tester, is aiming at systematic discovery of marginal defects and stability issues in DDR3/4 parallel data bus.

## 5.1 *BERT instrument*

Bit Error Rate (BER) is often used as a quality metric for high-speed serial communication channels. It is measured by sending test patterns through the link and then comparing them to the patterns received on the other side. Traditional BER Testers require special Design for Test (DFT) test points to HSSL taken out to PCB surface for connection of BERT probes, except systems where link can be accessed through edge connectors or cables. In addition, probes connected to the test points leave out of testing some parts of HSSL that are behind the test points such as built into chip equalizers or BGA solder joints. To overcome the described issues our solution use chip built-in transceivers instead of external probes. High-speed transceivers, which are used for data transfer through these channels, have a fixed pre-defined location inside a FPGA. In our embedded instrument, we place a BER tester logic behind each transceiver to support all possible test cases. This makes our instrument be run-time reconfigurable, and does not require FPGA design synthesis and compilation after a product or specification change.

The BERT logic consists of equivalent Test Data Generators (TDGs) implemented behind transmitter (*Tx*) and receiver (*Rx*) parts of each transceiver. These run-time reconfigurable TDGs are able to produce test data patterns, which are either simple switching between zeroes and ones (clock patterns) or more sophisticated pseudo-random patterns. The receiver being complemented with pattern comparator waits for the first special test data pattern. When the pattern comes, the receiver starts to generate the same test data patterns sequence as *Tx* TDG and compare each received pattern bit with expected bit generated by *Rx* TDG. The BERT instrument can be used with the following connection schemes:

- Internal loopback (*Tx* is connected to *Rx* inside FPGA transceiver)
- External loopback (PCBA connector's *Tx* is connected to connector's *Rx* with a cable)
- Communication between two FPGAs with proposed instruments inside (In 1st FPGA only *Tx* active, in 2nd FPGA only *Rx* part active)
- Communication between FPGA with proposed instrument and external receiver (only *Tx* or *Rx* active depending on a setup)

The BERT instrument also provides an access to the configuration registers of each transceiver. By adjusting FPGA transceiver settings (emphasis, signal strength, link speed, etc) during run-time one is able to assess the quality of a gigabit link and find the optimal parameters for a particular test setup. This option is especially useful for debugging the prototypes when the best settings are unknown. Using an external tester one is also able to apply an automatic scanning of proper settings by dynamically changing transceiver settings and analyzing the achieved bit error rate. Such a procedure can considerably save the time needed to find optimal settings.

Besides the bit error rate signal quality characteristic, our solution can also measure BER diagrams. Depending on the capabilities of certain FPGAs, it can draw a bathtub diagram, signal waveform eye diagram or statistical BER eye diagram. These diagrams provide information about link quality at the margin worst case sampling points, along with information about how much noise the link can sustain until it fails. Diagram can be obtained relatively quickly and can be used by engineers for extrapolation to BER, otherwise achievable only by long

running measurement. Moreover, the need for long lasting BER measurements vanishes if assume that manufacturing defects increase BER dramatically, thus can be detect during quick BER tests.
Another benefit of proposed BERT instrument is that an FPGA-based embedded instrument can replace a costly external tester and yet deliver a better (more realistic) results as the instrument samples the data right inside the system eliminating an additional source of signal distortion.
To summarize, proposed BERT solution can help to find manufacturing defects in system high-speed interconnects early and with the resolution of a problem-causing link in the system.

## 5.2 *DDR memory tester*

Today, data transmission rates on the board may be reaching multi-gigabit ranges on a single channel. High-speed signals are normally fine-tuned or even calibrated to deliver pitch perfect timing even in case of now-ubiquitous DDR3 memories. The frequency of signals on DDR3 data bus goes up to remarkable 2.4 GHz and JEDEC further rises this value to 3.2 GHz in DDR4 standard. One should also remember that DDR is a parallel bus interface, meaning that several lines are synchronized and sampled simultaneously. Just recently, such speeds were achievable for serial links only.
At the same time, such defects like dewetting, cold solder, head-in-pillow, voiding/crack in micro-via or excessive solder voids may result in system performance issues, increased error rates, intermittent faults and other sporadic stability issues observed in certain operation modes, at certain workloads or manifesting in a seemingly stochastic manner. In this section, we present an instrument that is based on extended DDR memory controller in which we have converted the bus calibration mechanism into a fine-grain diagnostic tool. This tool instead of masking marginal discrepancies (the typical case) would unveil and report them.

### *5.2.1 State-of-the-art*

Usually memory interconnect test is either done by Boundary Scan (BS) or reusing in-system memory controller to get access to the PCB lines. The latter is possible either in semi-functional mode, when memory controller is accessed in debug mode, or as part of the at-speed functional test. Number of works examine different approaches to achieve desired coverage. In [Chen02] authors examine testing for gross crosstalk delays via common read and writes. In [Ta15] authors summarize the test patterns used to detect common static and dynamic faults, their application and diagnostic procedures. In [Kim04] authors propose to reuse memory controller to access memory bus.
These approaches came to replace BS-based techniques unable to deliver test patterns at-speed and reported to create additional difficulties related to memory interface test [Ge09]. System-level at-speed testing based on functional routines or applications has become extremely important due to its ability to catch dynamic faults, like transition delays and crosstalk, which is also valid for interconnect buses with their rapidly growing speeds.
The situation with memory interface (DDR3/4) today is even worse making common functional at-speed test unable to cover marginal defects that cause system field failures and may result in No Fault Found (NFF) scenario [Ju16].
At such a speed, process-voltage-temperature (PVT) variations and small delay defects have a big impact on data sampling correctness. In [Bla11] authors show how operating margin can vary from one produced device to another. Although, the authors come to conclusion that the operating margin variance is not an issue in practice (in commercial sense), the paper gives an idea of how the margin can vary.
Today, while even the at-speed test only covers gross delay faults, the gap in dynamic fault coverage becomes more and more obvious. No-Fault-Found phenomena, when the board passes all the tests at the production and still fails sometimes in the field, has become serious concern [Ju16]. Supporting this point in [Lai14], its authors describe the complex process of debugging and testing via separations on DDR bus. The defects lead to intermittent failures in the field, but stayed undetected by common test routines.

One of the factors contributing to NFF is a low operating margin, possibly caused by small delay faults or marginal PVT variations [Pa09][Ju16]. In [Bi09], authors show how cracks in vias are progressing through different stages, including the stage when the crack has not yet been developed into a complete open, but behaves like intermittent high resistance.
Of course, high-magnification X-ray inspection can help to detect such problems, but it is not always practical to use it. Moreover, X-ray inspection does not check for operating margin, that eventually defines the system's performance in the field.
The problem is aggravated by DDR3/4 memory controller's calibration logic. It was built to help systems to cope with undesired, but unavoidable PVT variations. As a downside, while calibrating, the memory controller would adjust the sampling point in a way that compensates and therefore masks the PVT caused delays. In the same way, calibration process can turn otherwise gross-delay faults into marginal delay faults, and make functional test pass, which is undesirable.
All the described issues support the idea, that new approaches for testing high-speed interface like DDR3 and DDR4 for marginal faults are needed.

### *5.2.2 Method description*

The method is based on the idea that small signal propagation delays and signal transition "edge" distortions cause shift of the signal transitions position in the unit interval (UI). First, we explain how to find out signal transition by composing Bit Error Rate (BER) diagrams for DDR3/4 data lines. Then we describe how BER diagrams can be used for analyzing signal transition delays. And finally, we will show that this information can be used to identify problems with DDR lines on Printed-Circuit Board Assemblies (PCBAs).

#### 5.2.2.1 BER diagrams

BER diagrams are often used to evaluate communication links quality. They show how error rate depends on the position of the sampling point (SP) in the UI. The error free interval represents line noise tolerance. Wider it is, more signal distortion can be tolerated. BER diagram composition is done by measuring Bit Error Rate (BER) in all possible sampling points.
We can also say that BER diagram shows signal transitions moments relative to the UI. Looking at BER diagram, one can find the interval in the UI, when signal always have a correct value. We will call this interval Operating Window (OW) and its size Operating Margin (OM).
If receiver samples data within Operating Window, it will always get the correct data. If receiver tries to sample data beyond the OW, it may happen that wrong data is sampled. In general, the best sampling position would be in the middle of OW, because then signal can sustain equal amount of noise in both directions.
We can divide BER diagrams into three types: a diagram with only rising BER edge, with only falling BER edge and classical BER "bathtub" diagram with both edges shown on Figure 5.1.

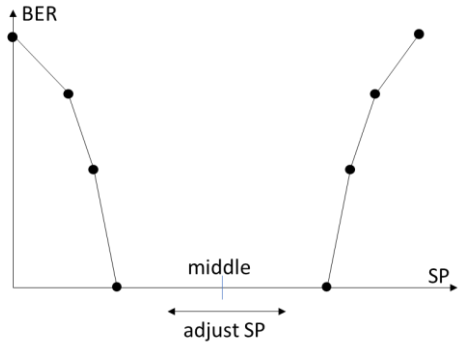



**Figure 5-1. Typical BER "bathtub" diagram**

Operating window borders are defined by worst case transitions: the fastest and the most delayed signal transitions. Ideally, signals would transition on the border of the UI, but in real environment, under different circumstances, transition can occur earlier or later in the UI. The early transitions are usually seen in the right part of the UI, where ideal signal should stay in its current value. The late transitions are seen in the left part of the UI, because they actually occur in the beginning of the next UI.
Normally, BER diagrams are measured while transmission of pseudo-random test data is running. However, by using a certain test pattern we can compose BER diagram that will represent behavior of a specific type of transition. For example, repeatedly transmitting a test pattern that contains transition of interest makes it possible to analyze this transition using BER diagram.
BER diagrams are usually composed iterating through all possible positions of a sampling point sequentially and measuring BER in each of them. Measuring BER is carried out by continuous acquisition of signal in a certain sampling point and comparing the received data to the expected.

#### 5.2.2.2 Adjusting sampling point with Memory Controller

DDR3/4 memory controllers are equipped with built-in calibration facilities. The calibration mechanism relies on the ability of the controller to adjust phase of the data strobe (DQS) signals for read and write operations on memory bus. Strobe signals are used as a clock for data sampling. Adjusting DQS phase has the same effect as adjusting sampling point used by DDR chip or memory controller itself. Further in the description, we will use term “adjust sampling point” instead of “adjust phase of the data strobe signal”.
Here, we will not discuss technical aspects and means related to the adjustment of sampling point. Instead, we assume that memory controller on the System under Test provides this capability. However, we have proven the described method using Xilinx MIG 7 series memory controller (soft-core) and Zynq-7000 SoC memory controller.
We should also mention that the presented method is aimed to test operating margin of data bus only (i.e. DQ lines DDR bus). This limitation comes from the nature of DDR memory calibration mechanisms. A typical memory controller can adjust phases of data strobes only. This is ensued by the frequency of switching activity on the lines: data bus switches at full speed, while address/command bus switches less frequently. Thus, there is no need to fine-tune address/command bus timings.

#### 5.2.2.3 Composing BER diagrams for DDR interface

In general, BER diagrams can be composed for different paths: when data is flowing from the memory controller to DDR chip and vice-versa.
To compose BER diagram for write path we need to continuously write test vector to memory and adjust write sampling point (which is used by DDR chip to sample data bus). However, in order to see the data that DDR chip had actually sampled with the altered position of sampling point - we will need to read this data from the memory.
We can also compose BER diagram for the read path. For this, we first write test vector to the memory (using non-altered sampling point) and then continuously read it back. Each time we read the vector from the memory we can also change the sampling point and calculate BER value.
By testing both directions, we cover both driver-receiver pairs variations. If the goal of the test is to only detect faults on PCBA, then testing in one direction is enough. But if test is also targeted to evaluate the operating margin of the bus, then BER diagrams for both directions are necessary.
The amount of reads and writes performed while calculating the BER equals to the number of times we send the signal through the bus. Every time the signal will make a transition, but the transition moment may fluctuate due to the PVT (process, voltage, temperature) variations. The more read or write cycles we will make during the test, the more PVT variations we will be able to catch. However, this would also increase the overall test duration.

Long-lasting BER measurements are common practice in test world, because they provide reliable test results and are the only mean to certify link to confirm a certain BER level. But for our method, long measurements are actually not needed. In our approach, we are only looking for signal transition moment. Hence, several thousands of read/write operations will be already enough to find signal transition moment and also gather the statistics for possible PVT variations.
Moreover, it is possible to reduce test time by performing the large number of iterations only when sampling point is on the "edge" of BER diagram (where number of errors starts to grow high from zero). For other positions, the number of iterations can be reduced.
As defined by the DDR3/4 standard, all the reads and writes from/to the DDR3/4 memory are done in bursts. The bursts can be interrupted by pauses or can consecutively follow after each other. To remove the unwanted transitions from the BER diagram we need to mask bits (in burst) that may produce them. This is explained in the following example.
Let us consider a situation, when we want to compose BER diagram using 8-bit long burst 00001111 on the line DQ0. The only transition that we want to be reflected in BER diagram is the transition from 1 to 0 in the middle of the burst. For that, we need to calculate BER values for the fourth and fifths bits in the burst (when this transition occurs). But, if we also include the first and the last bits into the BER calculation, we will eventually include transitions between two consecutive bursts (from Z or 0 to 1) into the BER diagram. To avoid this, the last and the first bits in the burst should be masked out during data comparison.
Below (Fig. 5-2) we bring an example pseudo-code for building BER diagram for read path and transition 1 to 0 in the middle of the burst.

```
write test vector 00001111 to the memory
for left and right direction:
        put read sampling point to the middle of the UI
perform a read 10 times, calculate BER
while BER != 1 increase(right)/decrease(left) SP
perform a read 10 times, calculate BER
perform a read 2048 times, calculate BER
while BER != 1 increase(right)/decrease(left)  SP
perform a read 2048 times, calculate BER
while BER != 0 decrease(right)/increase(left) SP
perform a read 2048 times, calculate BER
store BER and sampling point values

calculate BER:
                For each read iteration compare
                        Read burst bit 3 with 1 (00001111)
                        Read burst bit 4 with 0 (00001111)
```

**Figure 5-2. An example pseudo-code for building a BER diagram**

Building BER diagram for write path is a similar process, except we have to adjust write sampling point (while read sampling point is somewhere in the operating window) and perform both write and read operations to calculate BER in each sampling point.

### 5.2.2.4 Delay detection with BER diagrams

We have described how calibration mechanism of memory controller can be reused to compose BER diagrams for certain transitions in test vectors. Each composed BER diagram will represent signal transition moment for certain test pattern.

Let's assume we test two lines with slight difference in their signal propagation delays. We compose BER diagram using the same test pattern for both lines. The BER diagram for slower line will be shifted to the right comparing to the BER diagram built for the faster line.

The delay detection method we present is based on the process of building BER diagrams for all test patterns and then comparing the edge positions on the diagrams to the expected. Any constant or data-dependent signal delay or speedup will be seen on the BER diagram as a shift of the edge position.

We propose to calculate two parameters for each BER diagram:

- mean position of transition
- transitions range

Mean position of transition would be the main criteria for delay detection. It is calculated by finding the middle point of BER diagram bathtub edge.

Each sampling point has an error count measured for it. Each accounted error means that signal have not yet made transition to the expected state. By subtracting error counts from two neighbor sampling point, we find out how many times signal has made transition to the expected state between these two sampling points.

For example, we have measured BER for a line and got results (error count for each of positions of sampling point) shown in the second row of Table 5-4. We have made 2048 iterations in each sampling point, so the error count can be 2048 maximum. All the measurements made for sampling points before the $9^{th}$ position have 2048 errors and measurements on all sampling points after the $16^{th}$ position produce 0 errors.

**Table 5-4 Example BER diagram values**

| SP | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|
| Error count | 2048 | 2048 | 2047 | 2017 | 1884 | 61 | 0 | 0 |
| Number of transitions | 0 | 0 | 1 | 30 | 133 | 1823 | 61 | 0 |

To calculate how many transitions occurred between two sampling points we have to subtract the respective error counts from each other. For example, only one signal transition (i.e. 2048–2047) occurred between the positions 10 and 11. Following the same logic we can calculated how many transitions occurred between all the sampling points (shown in the third row of Table 2.4.1). In this example, we associate number of occurred transitions with higher position of sampling points, e.g. we associate one transition with position 11 and 30 transitions we position 12, etc. (Table 2.4.1, third row).

Now we can easily calculate the mean position of the transitions:

(1*11 + 30*12 + 133*13 + 1823*14 + 61*15)/2048 ~ 13.93

It should be noted, that we could associate number of occurred transitions with lower positions, e.g. one transition at position 10 and 30 transitions at the position 11. In this case, the mean position would be one unit less (~12.93). However, it is not that important which way we choose as long as we stay consistent for all tests.

Another important characteristic of the measurement is range of transition positions. To calculate this range, we just count the number of positions of sampling point where signal made transitions. In the example above, the range is five, since the first transition was registered in sampling point 11 and the last one in sampling point 15.

Basically, the range shows how the sampled signal jittered against sampling clock (UI) during measurement. Jitter is essential in such systems, but too large signal jitter against the clock can indicate serious problems in the system.

If detected that system under test shows large jitter on all data lines, then probably there is either a global problem with power delivery on PCBA or sampling clock jitters more than expected. On the other hand, detecting large jitter on one/few lines will indicate problems with return current or local power delivery. Together these two characteristics (mean position and range of positions) can be used to perform analysis on BER diagrams and for delay faults detection.

### 5.2.2.5 Testing PCBAs

The mean position of transition that we have calculated out of BER diagram represents an average signal transition moment in the UI.
If we calculate it for the same lines with the same test patterns, but on different devices under test, we expect the mean position to be approximately the same (altered only by PVT variations). If some line deviates too much from the typical behavior, this most probably indicates a problem in the system under test.
Our test approach consists of several steps:

- Defining test patterns
- Defining acceptable limits of deviations
- Performing measurements

Test patterns are defined by test engineer in accordance to the fault models. In this work, we will use common delay fault models listed below:

- Transition delay (T)
- Inter-Symbol Interference (ISI)
- Simultaneous Switching Noise (SSN)
- Crosstalk (CT)

By reusing this delay fault models, we would be able to detect marginal constant delays, as well as detect susceptibility to ISI, SSN or CT. The acceptable deviation of mean position of signal transition is defined by the test engineer.
One way to define acceptable deviation is to compute it analytically out of DDR chip specification and board layout. Another way is to perform analog simulations for this purpose. However, the values obtained by using these methods will rarely match to the real signal transitions on PCBA.
To overcome this problem the acceptable deviation can be defined either statistically or empirically. Statistical approach is suitable when vast majority of devices produce similar test results, while defective devices are rare and produce statistically distinguishable test results.
For example, Standard Deviation (SD) or Median Absolute Deviation (MAD) can be used to find out outliers and screen out suspicious boards. In this case, factory will have to build, store and analyze a database of signal transition statistics.
This method will probably work fine if “good” signal transitions will follow uniform distribution. But in case of normal distribution, statistical approach will start screening out good but rare boards (those that exceed defined amount of SD or MAD).
Another approach is to calculate acceptable deviation empirically by performing measurements for the set of known good systems. Later, limits of deviation can be redefined by learning on false negative results.
When limits for acceptable deviations are defined for all the test patterns, the process of testing becomes straightforward. We perform measurement and check whether transition mean position lies in the defined interval.
Failed systems can be also additionally tested with more accurate (and more expensive) test methods, e.g. with high-magnification X-ray inspection. In this way, the described method can be used to reduce overall test time by performing expensive tests only for a small amount of suspicious PCBAs.

### 5.2.2.6 Measuring Operating Margin

Operating margin (OM) represents the amount of noise that system can tolerate before wrong data will be sampled.

Operating margin is the size of the operating window, which is defined by the worst-case, the earliest and the latest, signal transitions.

Usually, the fastest and the slowest transitions are found on different BER diagrams built with different test patterns. In this case, to evaluate OM we have to extract the position of the earliest and the latest transitions from different BER diagrams.

To be more precise, the worst-case transition is represented by the edge ("rising" or "falling") on the BER diagram. The position of the worst-case transition in this case is either the first error-free position of sampling point after "falling" edge or the last error-free position of sampling point before the "rising" edge on the BER diagram.

The earliest signal transitions are represented by BER diagram with the leftmost "rising" BER diagram edge. In case there are no BER diagrams with "rising" edges, the BER diagram with the leftmost "falling" edge is used.

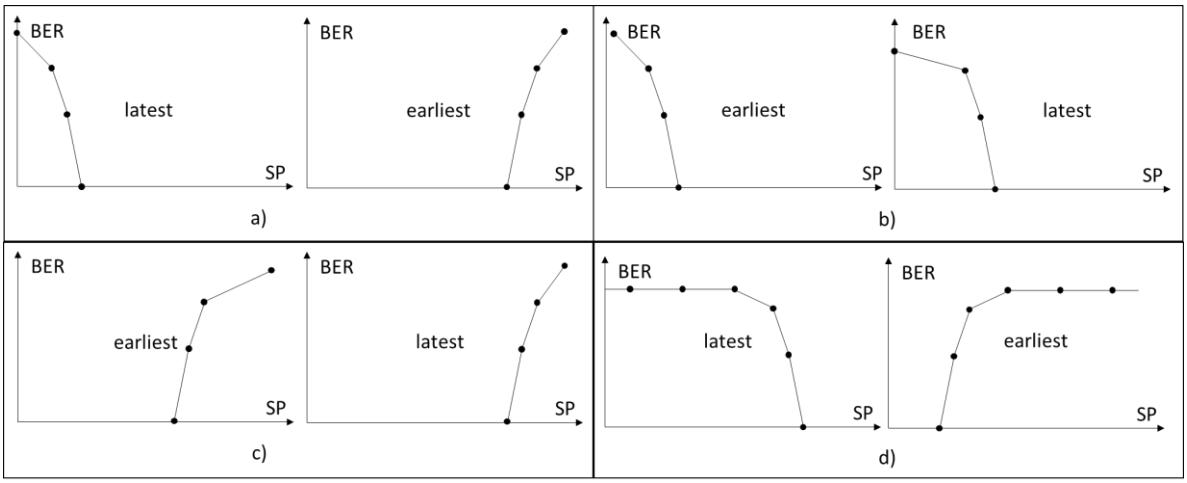



**Figure 5-3. Possible combination of BER diagrams representing earliest and latest signal transitions**

The latest signal transitions are represented by BER diagram with the rightmost "falling" BER diagram edge (or BER diagram with the rightmost "rising" edge if there are no BER diagrams with "falling" edges).

Figure 5-3 shows possible combinations of the BER diagram edges representing the earliest and the latest transitions: a) and d) show the case when transitions are represented by the edges of different type (e.g. "falling" and "rising), b) and c) show the cases when transitions are represented by the edges of the same type (e.g. "falling" or "rising").

In cases presented in Figure 5-3 a) and d), OM is computed by subtracting the position of the latest transition (LTP) from position of the earliest (ETP).

$$OM = ETP - LTP + 1$$

In case shown in Figure 5-3 d) the OM will be negative, indicating serious problem in the system under test. The computed value for the OM is expressed in sampling points. To properly evaluate OM, we first need to convert it into time units. For that, we need to know UI duration (UID) and number of sampling points in UI (N). The time distance (TD) between two sampling points can be calculated by dividing UID by the number of sampling points N.

TD = UID / N

To convert OM in sampling points into time units (OM_TU) we need to multiply OM expressed in sampling points by the time difference between sampling points.

OM_TU = OM * TD

For the cases presented in Figure 5-3 b) and c), OM is computed by subtracting difference between the earliest and the latest transitions from the UI duration.

OM = N – (LTP – ETP)

### 5.2.2.7 Byte-group Operating Margin

In the previous section, we presented a method to measure line operating margin out of data line with the help of BER diagrams. However, all the data lines that belong to a single byte-group of DDR bus are sampled simultaneously. Hence, the measured OM will not show real operating margin of bus.
The real operating margin is defined by OM with respect to byte-group operating margin. Computing byte-group operating margin is very similar to computing OM for a single line. The only difference is that we should choose the earliest and the latest transitions among BER diagrams composed for all data lines of single byte-group.

### 5.2.2.8 Test patterns

In this section, we examine the application of different test patterns when building BER diagrams. We also will discuss a typical BER diagram response on each type of patterns.
As we have mentioned in previous sections, we will examine four different fault models: transition delay, Simultaneous Switching Noise (SSN), Inter-Symbol Interference (ISI) and crosstalk (CT).
Transition delay test pattern is used to find out initial line delays or constant delay faults. The test pattern produces signal transition on the line under test while other lines are held in stable state. In this way, we are able to catch signal transition moment on target line and exclude possible influence of other lines.
Any kind of delay fault will add a constant delay to the signal transition timing, thus it will shift BER diagram to the right relatively to the diagram composed for a line without such fault. To test for all possible locations of delay faults we should repeat this test for every data line.
SSN test pattern is used to find how signal delay changes under “power stress” and even mode crosstalk. In literature, the effect of this test pattern is often called “Ground Bounce” or “VCC sag”. The test pattern assumes all the lines are switching in the same direction and thus put maximum stress on the power delivery network. Such test pattern will also cause even mode crosstalk.
We expect SSN test pattern to delay signals on all the lines even on defect-free system. Some lines may be delayed more than expected due to discrepancies in the power-delivery-network or if these lines are more susceptible to even mode crosstalk. Such delays will shift edges on BER diagrams to the right in comparison with the diagrams created for transition test pattern.
In case of SSN test pattern BER diagrams can be composed for all the lines simultaneously.
Inter-Symbol Interference (ISI) in an effect that occurs when previous signal value affects current signal transition. ISI test pattern assumes that line stays in one state for long time and then transitions to opposite state. ISI can cause this transition to be delayed.
Making a burst with a single transition that occurs at the last bit of the bursts should achieve the desired ISI effect. In our method, we will use ISI test pattern with long series of logical zeroes (or ones) followed by

transition to the opposite logical state. To avoid compromising the test results by crosstalk and/or SSN, it is safer to perform ISI test for each data line separately.
We expect that application of ISI test pattern will cause signals to transition slightly later than when transition test pattern is applied.
Crosstalk occurs when several lines switch their state simultaneously. If all signals are transitioning in the same direction we will see so called *even mode crosstalk*. If all signals switch in the direction opposite to the switching of one signal - *odd mode crosstalk* [Hall00] will be observed.
The effect of the even mode crosstalk is covered by SSN test patterns. Odd mode crosstalk assumes that we switch victim line in one direction, while aggressor lines into opposite direction. Like in the cases of transition test and ISI patterns, we have to make odd mode crosstalk test separately for each line.
The effect of the crosstalk could be signal delay or speedup depending on whether inductive or capacitive coupling dominates. Usually the signals are sped up due to odd mode crosstalk on the PCBs. We expect that victim line (under test) will slightly speed up, while aggressor lines will be delayed.

### 5.2.3 Experimental setup

To prove that the presented method can actually detect marginal defects, we have assembled an experimental setup and carried out a series of experiments on it. The setup is based on the Xilinx VC707 evaluation kit with Virtex-7 FPGA and SODIMM slot. For the FPGA device, we have implemented a design consisting of Microblaze core and Xilinx MIG memory controller.
As for experiments, we have selected five SODIMM DDR3 memories of the same part-name (KVR16S11S6/2). These memories were tested with four different types of test patterns:

- Transition test pattern
- SSN test pattern
- ISI test pattern
- Crosstalk test pattern

Moreover, we have also injected a defect into one of the SODIMMs (after it has been tested as defect-free) and repeated our test procedures with "defected" memory module. The defect was injected by isolating (taping) two ground pins (pin 3 and 9) on SODIMM module.
We will assume that initially all the SODIMMs are defect free, and all the variances in signals transition moments are caused by PVT variations.
First, we test five defect-free SODIMMs. The goal of this experiment is to analyze how test patterns affect signal transition delays. Next, we inject the defect and try to detect it. In order to detect the defect, we need to define limits for signal transition positions in the UI. We define the limits on the basis of the values that were measured for defect-free SODIMMs.
The defect is expected to generally delay lines DQ0 and DQ1, but mostly for SSN test pattern.

### 5.2.4 Experimental results

For the experiments, all the test patterns except ISI were made so that the transition occurs between 3$^{rd}$ and 4$^{th}$ bit of 8-bit burst. BER diagrams are constructed with 2048 iterations. Tests were done for read direction only. The memory controller used in te experiments is capable to adjust sampling point beyond the UI. SODIMMs under test are labeled with IDs from S1 to S5, the defective SODIMM is labeled as S1*.
First row in the Table 5-5 shows the mean transition position (in sampling points) for the all SODIMMs, except the defective one, per certain test pattern. Next five rows of the table show the deviation of the transition position relative to the calculated average value. The last row shows the deviation of the defective SODIMM. We see, that ISI test pattern slightly delays transition, since all the transitions made with ISI test pattern occurred later than for transition test pattern (for the same direction). The amount of delay varies for different SODIMMs and lines.

SSN test pattern also delays all the transitions, even more than ISI. Both presented lines were delayed by SSN transition and approximately the same for all SODIMMs. Line DQ0 was delayed by approximately four sampling points for all SODIMMs, while line DQ1 was delayed by approximately seven sampling points.
CT test pattern caused signals to speed-up. Line DQ0 was sped-up by approximately three sampling points while line DQ1 by approximately four sampling points.
Conducted experiments confirm that data dependent delays can be caught by the described method. As we can see, the results measured on defective SODIMM deviated from the average for all the test patterns, except CT pattern. Especially high deviation is seen for SSN test patterns.

**Table 5-5 Transition deviation per SODIMM per test pattern**

| | T01 D0 | T01 D1 | T10 D0 | T10 D1 | IS01 D0 | IS01 D1 | IS10 D0 | IS10 D1 | SS01 D0 | SS01 D1 | SS10 D0 | SS10 D1 | CT01 D0 | CT01 D1 | CT10 D0 | CT10 D1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Mean | 6.4 | 9.3 | 3.9 | 8.7 | 6.8 | 10.7 | 4.7 | 9.7 | 10.3 | 16.5 | 7.4 | 16.1 | 3.3 | 5.5 | 1.2 | 4.3 |
| S1 | 0.6 | 0.3 | 0.8 | 0.3 | 1.1 | 0.6 | 1.0 | 0.4 | 0.6 | 0.3 | 0.6 | 0.1 | 0.4 | 0.4 | 0.8 | 0.5 |
| S2 | 1.5 | 0.6 | 1.1 | 0.1 | 1.2 | 0.7 | 1.2 | 0.3 | 1.9 | 0.5 | 0.6 | 0.1 | 0.7 | 0.5 | 0.8 | 0.1 |
| S3 | 1.5 | 0.9 | 0.9 | 0.7 | 1.4 | 0.8 | 1.6 | 1.2 | 2.1 | 0.3 | 1.4 | 0.5 | 0.4 | 0.4 | 1.2 | 0.8 |
| S4 | 1.6 | 0.2 | 0.1 | 0.5 | 1.2 | 0.1 | 0.3 | 0.4 | 1.8 | 0.4 | 0.6 | 0.1 | 0.7 | 0.3 | 0.6 | 0.3 |
| S5 | 2.2 | 0.6 | 0.9 | 0.6 | 1.9 | 0.5 | 1.0 | 0.8 | 2.3 | 0.3 | 0.5 | 1.0 | 1.4 | 0.5 | 0.9 | 0.6 |
| S1* | 1.6 | **1.6** | **1.9** | **2.3** | 1.2 | **1.4** | **2.3** | **2.6** | **5.6** | **1.6** | **4.6** | **2.3** | 0.2 | 0.5 | 0.7 | 0.2 |

The first row in Table 5-6 shows the mean line operating margin for all the SODIMMs, except the defective one. The next five rows show the deviation of the OM value from the average for five good SODIMMs. The last row shows the deviation of the defective SODIMM.

**Table 5-6 Operating margin deviation per SODIMM per line**

| | Line DQ0 | Line DQ1 | Line DQ2 | Line DQ3 | Line DQ4 | Line DQ5 | Line DQ6 | Line DQ7 |
|---|---|---|---|---|---|---|---|---|
| Mean OM | 517.1 | 478.4 | 457.9 | 478.4 | 535.3 | 520.0 | 526.4 | 474.8 |
| S1 | 11.8<br>2.3% | 12.0<br>2.5% | 10.8<br>2.4% | 7.2<br>1.5% | -4.1<br>-0.8% | 38.9<br>7.5% | 15.6<br>3.0% | 7.2<br>1.5% |
| S2 | -12.3<br>-2.4% | -6.0<br>-1.3% | -4.8<br>-1.0% | -7.2<br>-1.5% | -2.9<br>-0.5% | -21.2<br>-4.1% | -12.0<br>-2.3% | -3.6<br>-0.8% |
| S3 | 8.2<br>1.6% | -7.2<br>-1.5% | -4.8<br>-1.0% | -3.6<br>-0.8% | 5.5<br>1.0% | 5.3<br>1.0% | -7.2<br>-1.4% | 2.4<br>0.5% |
| S4 | -19.5<br>-3.8% | -2.4<br>-0.5% | -10.8<br>-2.4% | -2.4<br>-0.5% | -6.5<br>-1.2% | -23.6<br>-4.5% | -10.8<br>-2.1% | -1.2<br>-0.3% |
| S5 | 11.8<br>2.3% | 3.6<br>0.8% | 9.6<br>2.1% | 6.0<br>1.3% | 7.9<br>1.5% | 0.5<br>0.1% | 14.4<br>2.7% | -4.8<br>-1.0% |
| S1* | **-60.3**<br>**-11.7%** | **-25.2**<br>**-5.3%** | -3.6<br>-0.8% | 2.4<br>0.5% | -10.1<br>-1.9% | -5.5<br>-1.1% | -6.0<br>-1.1% | 7.2<br>1.5% |

The injected defect had produced a considerable impact on the SSN test pattern response for line DQ0. This, in its turn, caused the line's operating margin to decrease. As a result, the operating margin of DQ0 line has

been decreased by more than 10%. In overall, the experimental results confirm that the injected defect (missing ground) can be detected by using the proposed test method.

### 5.2.5 Summary

- We can clearly see that the presented instrument can be used for detection of marginal defects on DDR3/4 memory bus.
- The same lines can have different signal propagation delays caused either by presence of a defect or by PVT variation. It is up to manufacturer to consider boards with larger delays as faulty or just bin them with a lower performance grade.
- All the boards that deviate from the expected behavior can be additionally tested by other more expensive and time-consuming methods (for example, X-ray inspection). In this way, the presented technique will significantly reduce overall test time and costs since the expensive testing should only be carried out for a small set of suspicious boards.
- Combination of defect and PVT variation can lead to marginal behavior that will escape functional test, but still may manifest itself in the field.

## 5.3 Section Conclusions

In this Section, we presented two instruments for marginal fault detection: 1) Bit-Error Rate Test (BERT) instrument incl. statistical eye composer and 2) defect detector on DDR3/4 memory bus. The memory bus test instrument is based on extended DDR memory controller in which we have converted the bus calibration mechanism into a fine-grain diagnostic instrument, which instead of masking marginal discrepancies (the typical case) would unveil and report them. This way of employing the DDR memory controller has never been reported in the literature. The second instrument (BERT + eye measurement) is based on the ability of gigabit transceivers in modern FPGAs to shift the signal phase and threshold voltage, hence enabling measurements in worsened conditions, demonstrating in this way the channel tolerance limits. Although, the FPGA's standard instrument hardware is reused, we demonstrate for the first time that this sort of measurement can be shifted from the laboratory to the production ground. We describe the way the instrument can be controlled and reconfigured automatically by the production test system. In our experiments the BERT procedure takes a few seconds instead of lengthy laboratory BER measurement typically taking at least several hours.

The novel instrumentation toolbox described in this Section helps to quickly identify outliers enabling reduction of test escape rates and improvement of production quality. The instruments populate the existing on-board FPGA device converting it temporarily into a fully-automated on-board embedded tester requiring no extra FPGA/DFT. The same principles can be further reused beyond FPGAs and find application in ASICs.

# 6 On the detection of board delay faults through the execution of functional programs: analysis and methods for improvements

## 6.1 *Introduction*

The test of a Printed Circuit Board Assembly (PCBA) is composed of several steps, including Optical and X-Ray Automatic Inspection, In-Circuit Test, Boundary Scan, Functional test. The different steps are combined in such a way to maximize their cumulative ability to detect possible defects, while minimizing their cost. Unfortunately, in the past decades the number of PCBAs passing all the tests at the end of manufacturing and then failing in the field (No Failure Found, or NFF) grew, mainly due to their increased complexity and to the appearance of new defect types. The economic impact of the NFF phenomenon is huge.

Hence, companies and researchers launched a number of efforts to identify its major causes and to devise effective countermeasures. Concerning the first point, there is a consensus on the fact that delay faults are major contributors to NFF, mainly because they can hardly be detected by most of the test approaches traditionally used in PCBA testing. In fact, most of these approaches are static, i.e., do not perform any at-speed test, and thus fail in detecting defects that only manifest themselves when the PCBA is used at the operational working frequency. A solution addressing this issue by adding suitable circuitry on-board the interconnected devices has been proposed in [Yi06], but its practical adoption has been limited so far.

In [Ju16], the author proposed some new fault models to effectively deal with delay defects affecting interconnections among devices, trying to extend to PCBA testing what has successfully being done in the recent past in the area of IC testing.

Functional test is commonly adopted as a countermeasure, since it is supposed to be able to detect delay defects [Ju14]. A common (and cheap) approach to functional test lies in adopting some existing application codes and testbenches, closely corresponding to the final application the PCBA has been developed for, possibly looking for the best trade-off between test duration and diagnostic capabilities [Ji16].

This work first aims at assessing on a simple but representative test case the effectiveness of such an approach. This goal is achieved by computing the fault coverage with respect to delay faults of several test programs which could be adopted for functional test (application programs, test programs addressing the processor logic, test program for memories), focusing on the delay faults possibly affecting the interconnections between the CPU and the memory devices on a PCB.

Reported results show that in most cases the achieved fault coverage is far from being complete, thus possibly identifying delay faults at the board level as one of the causes of NFF. As a second contribution, this work describes how to write a test program able to significantly increase the delay fault coverage, achieving 100% on the data bus lines.

The organization of this section is the following. Sub-Section 2 provides some background on the role of functional test in board test and some metrics used to quantify its quality, as well as on delay fault models. Sub-Section 3 describes the experimental environment we devised for our experiments. Sub-Section 4 reports the results we gathered and Sub-Section 5 draws some conclusions.

## 6.2 *Delay Fault Models and Observation Mechanisms*

In IC testing, delay defects are well-known and solutions for their detection have been investigated and successfully adopted. Commercial EDA tools allow to support their test, automating both the phase of inserting scan chains, and the one of generating test vectors addressing transition delay faults. The Launch on Capture (LOC) and Launch On Shift (LOS) techniques are widely adopted by companies. Although the achieved fault coverage is not always very high, and the risk of overtesting may reduce the effectiveness of this solutions, test

engineers can estimate the coverage of this kind of defects, and possibly act to tune the test effort in order to achieve the target Defect Level.

In the area of board testing, the scenario is more complex. First of all, there is much less consensus on the fault models to be adopted and on the metrics to be used for assessing the quality of a test [Hi02]. An effort in this direction is represented by the PCOLA/SOQ/FAM standard proposed by iNEMI [Ta09]. However, the relationship between the figures provided by the PCOLA/SOQ/FAM metrics and the achieved defect coverage is still weak, and may significantly vary depending on how the metrics are computed in practice. The author of [Ju16] listed several reasons why new and more accurate fault models are required to catch Time Related Faults affecting a PCBA, and proposed 3 fault models to better detail the "A" category in the PCOLA/SOQ/FAM standard. The simplest (denoted as pin-level fault) corresponds to a delay in the slow-to-rise or slow-to-fall transitions on each pin of a device and directly matches the Transition Fault Model on an internal signal which is widely adopted in the IC area.

When addressing this new fault model through a functional test, one should first activate the target transition on a given pin, and then propagate the effects of the fault (if any) up to an observable location. Commonly, the functional test corresponds to execute a given test program on the CPU existing on the board, and to check results produced by the program, often corresponding to the final results left in a given memory area at the end of its execution or to the sequence of values sent to a given output port.

In this work we investigate the fault coverage that can be achieved with respect to this fault model by several categories of test programs, focusing on the delay faults possibly affecting the interconnection signals between the pins of the memory devices and those of the CPU in a sample processor-based board. Each fault is supposed to be detected if it produces a difference in the final content of the memory area where the test program is supposed to produce its results at the end of its execution.

## 6.3 *Experimental Setup*

In this section we describe the sample processor-based system we considered for our experiments, as well as the technique we used to inject delay faults and to gather results about their detection. The system is based on the OR1200 soft-core processor [OR] and some memory modules, interconnected through the wishbone bus [WB].

### *6.3.1 Overview of OR1200 and Wishbone Bus*

At present the OR1200 is the only major RTL implementation of the OR1K architecture spec. The OR1200 is a 32-bit scalar RISC with Harvard micro-architecture, 5 stage integer pipeline, virtual memory support (MMU) and basic DSP capabilities. Default caches correspond to a 1-way direct-mapped 8KB data cache and a 1-way direct-mapped 8KB instruction cache, each with 16-byte line size. Caches were disabled for the purpose of our experiments. The OR1200 processor in its default configuration corresponds to about 1M transistors. The OR1200 core is mainly intended for embedded, portable and networking applications. It can successfully compete with the latest scalar 32-bit RISC processors in its class and can efficiently run any modern operating system. Competitors include ARM10, ARC and Tensilica RISC processors.

The wishbone system-on-chip interconnection architecture for portable IP cores [WB] is a flexible design methodology for use with several processor, memory and peripheral IP cores. Its purpose is to foster design reuse by alleviating system-on-chip integration problems. This is accomplished by creating a common interface between IP cores. This improves the portability and reliability of the system, and results in faster time-to-market for the end user. This specification can be used for soft core, firm core or hard core IPs. Since firm and hard cores are generally conceived as soft cores, the specification is written from that standpoint. This specification does not require the use of specific development tools or target hardware. Furthermore, it is fully compliant with virtually all logic synthesis tools.

The wishbone interconnect is intended as a general purpose interface. As such, it defines the standard data exchange between IP core modules. It does not attempt to regulate the application-specification functions of the IP core.

The OR1200 processor has been inserted into a system composed of a 2MB RAM module and a 2MB Flash memory. The processor has been synthesized with Synopsys Design Compiler targeting the NanGate 45nm Open Cell Library. The synthesized processor accounts for about 40k equivalent gates.

The programs we used for our experiments have been compiled with the or32-uclinux toolchain and loaded in the Flash memory during simulation. Logic simulation experiments have been performed using Mentor Modelsim.

### 6.3.2 Fault Simulation

Although several commercial EDA tools exist for performing fault simulation of delay faults, these tools are mainly intended to be used in combination with scan, and can hardly be used for evaluating the fault coverage achieved by functional stimuli, or even more by a processor executing a test program. Moreover, for our experiments we wanted to have full control on where and when to inject faults. Hence, we resorted to a different approach, based on modifying the description of the system at RTL, introducing some *saboteurs*, i.e., 1-bit fault injection modules [Gr01], [Gi99], [Gr11] on the 32 data bus lines connecting the RAM and Flash memory modules to the processor bus, respectively. Each fault injection module can inject a fault on the controlled signal when a read or write operation is executed by the processor on the related memory. In details, a saboteur controls the transition of a given line and is able to introduce a delay according to a time parameter. In case of a permanent fault, the saboteur applies the delay to all transitions on the line (e.g., all rising edges are delayed by 5 ns). Figure 6-1 shows the locations where the injection modules are placed.

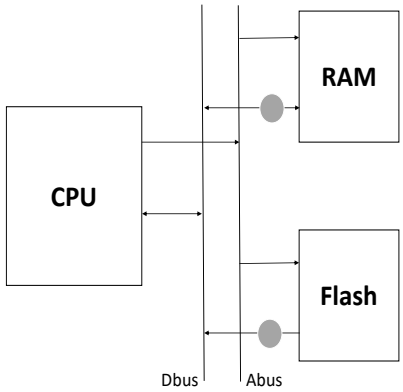



**Figure 6-1. Fault locations for the fault simulation campaign**

Based on this approach, some enable signals have been added in the top level processor description. For our experiments we considered the slow-to-rise (STR) and slow-to-fall (STF) faults. Hence, 4 enable signals for each fault injection module exist, activating the injection of a STR/STF fault when a Read/Write operation is executed. As a whole, 128 possible faults exist on the 32 data signals of the RAM memory, and 64 for the 32 data signals of the Flash memory, where the code is stored and which can only support read operations. All these faults can be activated at the due time during the simulation phase resorting to a *tcl* file, leading to the injection of a given fault on a given data line. For our experiments, we activated a single fault at a time, which is active for the entire program execution (i.e., single permanent fault). The framework is also able to inject multiple faults and transient faults. Each time the corresponding rising/falling transition happens on the target line during a Read or Write operation (respectively), the transition is delayed by 1 bus clock cycle[1] by the

[1] In the configuration used for our experiments, the CPU uses a faster clock. In this setup, the Bus Interface Unit samples the bus every 2 internal clock cycles. Thus, the STR / STF fault on the BUS is modeled to be at least 2 clock cycles long, otherwise it would be considered as a small delay.

injection modules. Finally, at the end of the simulation the tcl file is in charge of forcing the simulator to dump to a file the whole content of the memory and to compare its content with the expected one. Figure 6-2 shows the pseudo-code for the whole fault simulation campaign.

```
Perform fault free simulation;
Dump the final memory content;
For every fault
{  Activate the corresponding enable signal;
   Perform fault simulation;
   Dump the final memory content;
}
Compare the dump files and gather statistics;
```

**Figure 6-2. Pseudo-code of the fault simulation campaign**

## 6.4 *Experimental Results*

Using the fault simulation environment described in the previous Sub-Section we performed some experiments aimed at computing the fault coverage that can be achieved with four different types of test programs:

- A set of application programs
- Test programs developed to test the processor core with respect to stuck-at faults
- A program targeting the test of the memory
- An ad hoc test program.

### *6.4.1 Application Functional Programs*

In practice, functional test is often performed resorting to application programs mimicking in some way the code which is run on the system during the operational phase, and checking whether it produces the expected results.

In order to analyze this scenario, we considered 10 application programs, whose characteristics are reported in Table I. Three of them (bubble_sort, dijkstra and matrixMultiply) are simple applications implementing vector sorting, minimum path identification in a graph and integer matrix multiplication, respectively. The other programs are derived from the first 3 by introducing some of the hardening techniques described in [Ch15], aimed at allowing the detection of transient faults by introducing duplication and comparison instruction in the code.

Table II reports the results in terms of achieved delay fault coverage (*Delay FC(%)*). The table is divided in two parts: in the columns labeled as "Memory" we reported the number of delay faults that can be marked as detected when observing the final content of the memory at the end of the program execution. As a reference, we also reported in the table (columns labeled as "Bus") the corresponding figures, related to the case in which the bus is continuously observed. In this scenario, a fault is marked as detected if it produced at least one difference on the bus. This latter scenario is only provided as a reference, since the functional approach typically uses memory as the only observable point. We decided to deliver this set of figures because they provide a useful reference: faults that cannot be detected observing the bus can never be detected by observing the memory, only.

The main observation stemming from Table II lies in the fact that the achieved delay fault coverage is quite low, and is limited to about 70%, apart for bubblesort, which achieves 91%. As expected, in all cases the fault

coverage achievable by observing the bus is at least equal to the one achievable by observing the final content of the memory. In some cases (e.g., Dijkstra), the difference is higher than 17%.
These results empirically demonstrate that a generic application program may achieve a rather low delay fault coverage when considering the bus lines.

TABLE I. CHARACTERISTICS OF THE APPLICATION PROGRAMS

| | ***Duration (Clock Cycles)*** | ***Program size (KB)*** |
|---|---|---|
| bubble_sort | 2,026 | 1.7 |
| dijkstra | 7,921 | 6.4 |
| dijkVAR3 | 13,195 | 12.1 |
| matrixMultiply | 6,933 | 6.1 |
| var3_10 | 134,418 | 8.0 |
| bs_VAR3 | 3,502 | 3.6 |
| var4p | 6,681 | 6.3 |
| var4p_10 | 126,018 | 9.3 |
| var4pp | 6,473 | 6.6 |
| var4pp_10 | 121,134 | 9.6 |

TABLE II. RESULTS OF THE APPLICATION PROGRAMS

| | ***Memory*** | | ***Bus*** | |
|---|---|---|---|---|
| | ***#Detected faults*** | ***Delay FC (%)*** | ***#Detected faults*** | ***Delay FC (%)*** |
| bubble_sort | 175 | 91.15% | 182 | 94.79% |
| Dijkstra | 118 | 61.46% | 151 | 78.65% |
| dijkVAR3 | 135 | 70.31% | 167 | 86.98% |
| matrixMultiply | 120 | 62.50% | 122 | 63.50% |
| var3_10 | 114 | 59.38% | 114 | 59.38% |
| bs_VAR3 | 134 | 69.79% | 162 | 84.38% |
| var4p | 121 | 63.02% | 123 | 64.06% |
| var4p_10 | 115 | 59.90% | 115 | 59.90% |
| var4pp | 120 | 62.50% | 122 | 63.50% |
| var4pp_10 | 113 | 58.85% | 113 | 58.85% |

### *6.4.2 Stuck-at oriented Functional Test Programs*

These test programs (numbered from TP01 to TP15) were developed addressing the stuck-at fault coverage of the processor core. Their stuck-at fault coverage, duration and size are reported in Table III. These test programs were developed resorting to different approaches (random, deterministic, empirical), following the methods described in [Be16], and own quite different characteristics in terms of duration, size and achieved Fault Coverage with respect to the stuck-at faults. As the reader can notice from the table, the percent stuck-at fault coverage achieved by these test programs lies between 79% and 86%.
Table IV reports the results in terms of delay fault coverage achieved by the same test programs. Two key observations can be drawn from the above results:

1. There is no direct relationship between the stuck-at fault coverage achieved by a test program and the delay fault coverage on the data lines
2. The achieved delay fault coverage figures are anyway higher than for the previous application programs: in one case (TP12) it was even possible to reach 100% when observing the memory. Clearly, the limited fault coverage is due to observability problems, since full fault coverage is often achieved when directly observing the bus.

### 6.4.3 Evaluation of Delay Fault Coverage for Test Program Based on March-C Algorithm

A common test practice in board testing is to include in the test plan a test program addressing the memory system. This test program typically implements a March algorithm, often resorting to a software approach [vd10]. To mimic this scenario we developed a program implementing the March-C algorithm [vd91]. The characteristics of this test program are shown in Table V.

The results of the delay fault simulation experiments are reported in Table VI. It must be underlined that the implemented March-C algorithm code is able to detect all delay faults on the data lines connecting the RAM memory to the bus, while it only detects a few of those located on the Flash lines.

Hence, even a good test program for the memory is not able to provide a complete fault coverage when considering delay faults.

### 6.4.4 Ad hoc Test Program

In order to increase the efficiency of the test program and decrease the test time, an ad hoc test program is developed. This test program is mainly based on the same solution described in [Ta15] for testing Path Delay Faults in TSVs connecting dies in a 3D device. In practice, the test of the delay faults on the data bus lines of the RAM is performed by executing a proper test program implementing the following sequence of operations:

1. Write 00000000h to memory location X1
2. Write 11111111h to memory location X2
3. Write 00000000h to memory location X1
4. Read to a register R1 the content of memory location X1 (it should be 00000000h)
5. Read to a register R2 the content of memory location X2 (it should be 11111111h)
6. Read to a register R1 the content of memory location X1 (it should be 00000000h)
7. Write register R1 to memory location Z1
8. Write register R2 to memory location Z2.

When addressing the delay faults on the data bus lines of the Flash one can execute a proper test program implementing the above strategy. However, since the Flash is supposed to store the code, we cannot perform write operations to it. Moreover, read operations correspond to fetching instruction codes. Hence, the test must resort to a suitable sequence of instructions.

Let first focus on the test of the delay faults possibly affecting the least significant 16 bits of the Flash data lines. These bits in the machine code of the `l.movhi` instruction (writing an immediate into a register) do hold the value of the operand. Hence, the following instructions allow to test all the targeted faults, writing to a RAM memory location a value which is different in case a STR or STF delay fault affects one of the 16 target data lines:

```
1. l.movhi   r1, 0x0000 /* write to r1 */
2. l.movhi   r1, 0xffff /* write to r1 */
3. l.sw      0(r2), r1  /* write r1 to mem */
4. l.movhi   r1, 0xffff /* write to r1 */
5. l.movhi   r1, 0x0000 /* write to r1 */
6. l.sw      4(r2), r1 /* write r1 to mem */
```

Concerning the delay faults possibly affecting the most significant 16 bits of the Flash data lines, one should remember that the corresponding bits in the machine code of an OR1200 instruction do correspond to the operating code. Hence we need to

1. execute an instruction I1 with a given opcode O

2. execute an instruction I2 with an opcode O' whose bits are inverted wrt O
3. write to memory the results produced by I2

The above three operations must be repeated until all possible transitions are activated on the target bits.
Clearly, the purpose of the instruction I1 is only to prepare a transition, which is actually executed by fetching I2. In case of fault, the code for I2 becomes corrupted, thus its result is different than expected. For example, if I2 is an *add* instruction, the fault modifies either the operands or the instruction type. In the first case, the result of the operation is corrupted, while in the second case, a different instruction is executed, also in this case corrupting the expected result.
The properties of the ad hoc test program implementing the above strategies are shown in Table VII. With this ad hoc test program, the achieved delay fault coverage reaches 100%, while the size and duration are minimized.

TABLE III. STUCK-AT ORIENTED FUNCTIONAL TEST PROGRAM CHARACTERISTICS

| | ***Stuck-at Fault Coverage (%)*** | **Duration (Clock Cycles)** | ***Program Size (KB)*** |
|---|---|---|---|
| TP01 | 82.64 | 15,092 | 24.8 |
| TP02 | 80.28 | 85,292 | 25.2 |
| TP03 | 84.90 | 38,524 | 19.7 |
| TP04 | 81.33 | 31,830 | 59.0 |
| TP05 | 79.96 | 54,873 | 82.4 |
| TP06 | 85.46 | 171,435 | 103.4 |
| TP07 | 80.75 | 18,194 | 29.0 |
| TP08 | 82.56 | 24,806 | 39.4 |
| TP09 | 85.60 | 32,516 | 53.5 |
| TP10 | 79.19 | 27,709 | 33.5 |
| TP11 | 82.48 | 18,075 | 19.9 |
| TP12 | 83.93 | 66,247 | 59.6 |
| TP13 | 80.95 | 72,552 | 49.6 |
| TP14 | 85.51 | 47,788 | 51.9 |
| TP15 | 79.50 | 36,794 | 62.0 |

TABLE IV. STUCK-AT ORIENTED FUNCTIONAL TEST PROGRAM DELAY FAULT COVERAGE

| | ***Memory*** | | ***Bus*** | |
|---|---|---|---|---|
| | ***#Detected faults*** | ***DelayFC (%)*** | ***#Detected faults*** | ***Delay FC (%)*** |
| TP01 | 143 | 74.48% | 192 | 100.00% |
| TP02 | 129 | 67.19% | 167 | 86.98% |
| TP03 | 150 | 78.13% | 192 | 100.00% |
| TP04 | 123 | 64.06% | 192 | 100.00% |
| TP05 | 105 | 54.69% | 168 | 87.50% |
| TP06 | 96 | 50.00% | 96 | 50.00% |
| TP07 | 140 | 72.92% | 173 | 90.10% |
| TP08 | 113 | 58.85% | 192 | 100.00% |
| TP09 | 104 | 54.17% | 192 | 100.00% |
| TP10 | 109 | 56.77% | 192 | 100.00% |
| TP11 | 128 | 66.67% | 192 | 100.00% |
| TP12 | 192 | 100.00% | 192 | 100.00% |
| TP13 | 125 | 65.10% | 151 | 78.65% |
| TP14 | 127 | 66.15% | 152 | 79.17% |
| TP15 | 80 | 41.67% | 192 | 100.00% |

TABLE V. PROPERTIES OF A TEST PROGRAM IMPLEMENTING MARCH-C

| | *Duration (Clock Cycles)* | *Program size (KB)* |
|---|---|---|
| March-C | 25,542 | 2.2 |

TABLE VI. RESULTS OF A TEST PROGRAM IMPLEMENTING MARCH-C

| | *Memory* | | *Bus* | |
|---|---|---|---|---|
| | *#Detected faults* | *Delay FC (%)* | *#Detected faults* | *Delay FC (%)* |
| March-C | 132 | 68.75% | 168 | 85.42% |

TABLE VII. CHARACTERISTICS OF THE AD HOC TEST PROGRAM

| | *Duration (Clock Cycles)* | *Program size (KB)* |
|---|---|---|
| Ad hoc Test Program | 1,120 | 4.1 |

TABLE VIII. STUCK-AT ORIENTED FUNCTIONAL TEST PROGRAM DELAY FAULT COVERAGE

| | *Memory* | | *Bus* | |
|---|---|---|---|---|
| | *#Detected faults* | *Delay FC (%)* | *#Detected faults* | *Delay FC (%)* |
| Ad hoc Test Program | 192 | 100.00% | 192 | 100.00% |

## 6.5 *Section Conclusions*

NFF is a major economic and technical problem, mainly based on defects escaping the adopted test solutions. Delay faults are likely to be among the most common causes of NFF, and functional test is often selected as a possible solution.

The goal of this work is first to explore the ability of a generic program in detecting delay faults in the typical functional test scenario which is part of board testing.

By considering different application and test programs, we empirically demonstrated that the delay fault coverage is often much lower than 100%, despite the duration of a test program and its ability to test the processor stuck-at faults.

We also considered a test program targeting the memory and resorting to a March algorithm, showing that even in this case the achieved fault coverage is far from 100%.

Finally, we described an ad hoc test program which explicitly targets delay faults on the data lines connecting the memory to the processor bus, and showed that it can achieve full delay fault coverage with limited size and duration. The usage of suitable sensors and other techniques to increase observability may also increase the achieved fault coverage.

As a conclusion, functional test may represent an effective solution for NFF, provided that suitable test programs and observability solutions are used.

# 7 Automation and integration in a board-level test environment

In this section, we present the work carried out for the purpose of integration of embedded instruments into a board-level automated test environment. The overall architecture of the integrated embedded instrumentation and functional test technology is presented in Figure 7-1. The proposed architecture consists of three main independent elements. The software part (the leftmost in Figure 7-1) is used to control test execution flow and analyze the test results. Usually this is a functional tester with running operating system and test environment. The data exchange with the board under test is carried out using general-purpose digital I/O card capable to perform high-speed data transfers. The third component in the architecture is the board under test (BUT) itself. Board contains an FPGA with an embedded instrument IP inside which is intended to perform various kind of tests or measurements. This means that the combination of tester and JTAG controller can be used for any board under test. As a test automation software the list of possible solutions is:

- Test program written in C#
- Test program written in Python
- Test program written as a Windows batch script
- Test program automated with National Instruments' (NI) TestStand program
- Test program developed with NI LabView program

The last option is a de-facto standard in test and measurement domain. The well-known NI LabVIEW platform is heavily used by industry for building integrated control systems, functional testers and other kind of test equipment. The integration of the novel functional test technology into LabVIEW environment facilitates the usage of the developed methods in a standardized way and allows the integration with the wide range of the existing test and measurement hardware of NI and other third-party vendors.

It is worth to point out that the JTAG controller is not coupled with a software part (only a driver is required to run a test program). Below is a list of compatible hardware that can be integrated into the general platform:

- "C232HM-DDHSL-0" USB Hi-Speed - MPSSE Cable from FTDI [Ft17]
- "USB-DIO-16H" Pattern Generator Module from ACCES [Ac17]
- "USB-DIO32HS" High Speed Digital I/O USB Device from MCC [Mc17]
- RIO-series hardware from National Instruments (NI) [Ni17]

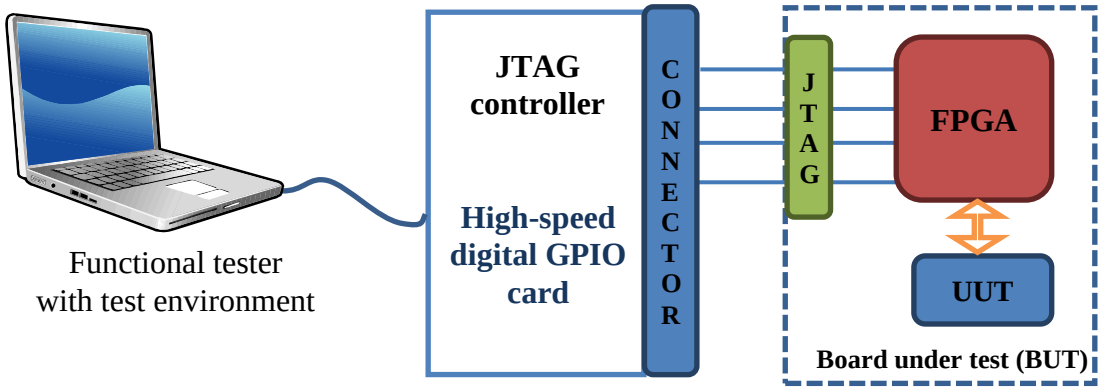



**Figure 7-1. Overall view on integrated functional test platform**

In this task, we have selected National Instruments LabVIEW (LV) environment and FTDI's "C232HM-DDHSL-0" cable as a main platform's components for the integration. This approach will allow users to easily augment existing NI-based test applications with the novel test methods.

For the software support we consider the development of special LabVIEW palette (toolbox) consisting of dedicated components (LabView Virtual Instruments (VIs)) for controlling embedded test instruments inside FPGAs. By placing the provided components (blocks) into LV program user will be capable to define the

access to the particular BUT and embedded instrument and then specify the type of test or measurement to perform. Similar components are used to configure different parameters of the test. As the embedded instrument control is done via standard LV VIs it could be easily added to the existing functional test programs developed on LV platform.

The JTAG controller is implemented on the FTDI's cable because of its fit-for-function characteristic. It is easy to install, it has a small size and low price (€25 as of 2017). This cable provides enough performance to communicate with board under test and transfer data back and forth on a relatively high speed. The usage of the NI-compatible hardware gives a possibility to use standard libraries and drivers (provided by LabView environment) to establish the low-level communication with the hardware. This approach provides a support of a large variety of different I/O cards without placing efforts into implementation of low-level driver functions. However, a JTAG-based communication protocol with the embedded instruments has still to be implemented. Besides that, the high price and complexity of installation of NI's hardware makes it profitable only if this hardware is already available or intended to be re-used later on.

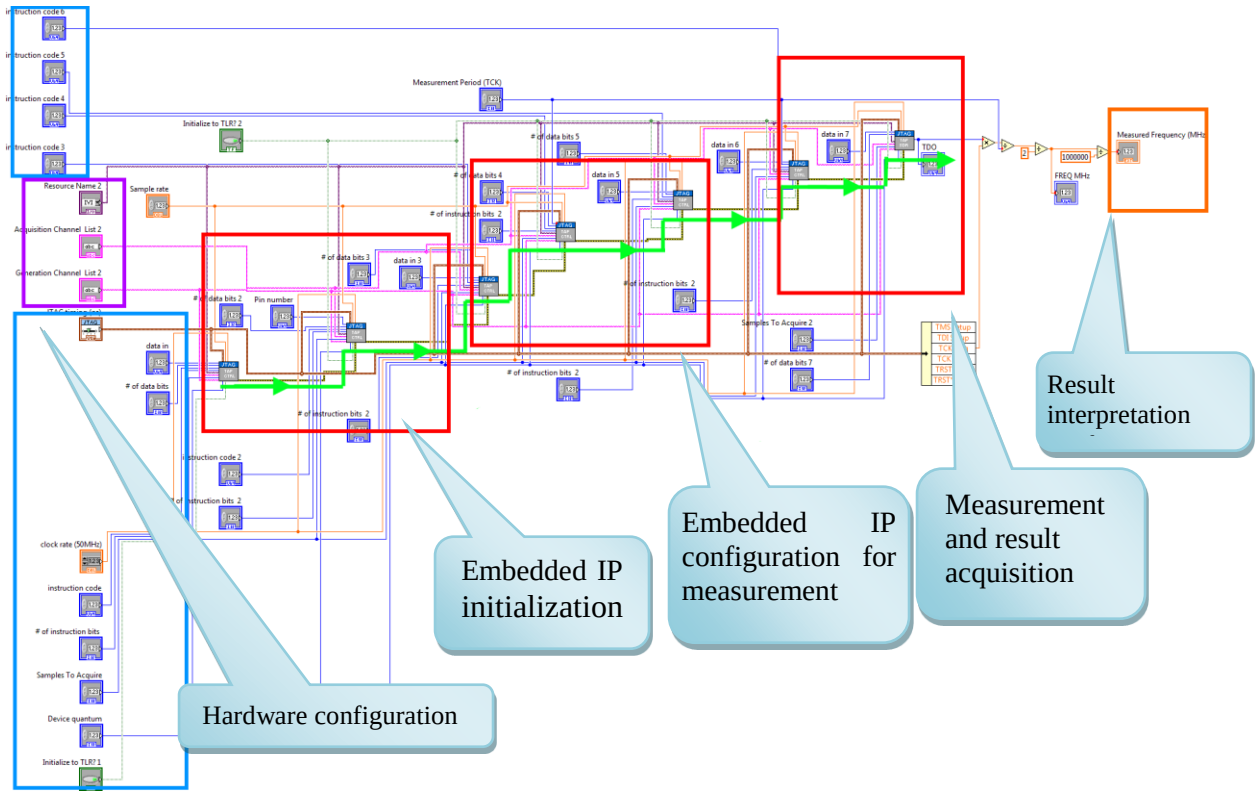



**Figure 7-2. Block diagram of the integrated embedded instrument**

As a prototype of integration of BERT embedded instrument into LabView test environment, we have made LabView Virtual instrument with a block diagram shown in Figure 7-2. The prototype consists of multiple parts:

- The controls highlighted by blue color are used for the initialization of communication protocol with the embedded instrument. It contains all the needed commands (JTAG operations) that put the embedded instrument into measurement mode and fetch the measurement result.
- The controls highlighted by purple color are responsible for the configuration of FTDI hardware: TCK speed and device ID;

- The output indicator and results interpretation is marked by orange color;
- The green arrow shows the workflow of the instrument;
- Three main phases of the VI workflow are high lightened by red color:
    - The first phase (leftmost) is responsible for the general initialization of communication between VI and Board under Test;
    - The second phase is responsible for the configuration of embedded instrument IP before a test;
    - The third phase is responsible for measurement process and the acquisition of the results.

The detailed description of this and other instruments is described in Deliverable D6.2.

# 8 Conclusions

The content of this deliverable summarizes the main achievements of the BASTION project partners involved in tasks T4.3 and T4.4. These activities conclude WP4 and contribute considerably to the integrated BASTION demonstrator (see D6.2 for details).

Board and system level test today cannot any longer rely fully on traditional dedicated test technologies like In-Circuit Test, Optical or X-Ray Inspection and even Boundary Scan. Every mission-critical system today has to successfully pass a fit-for-function qualification test, which is based on a combination of functional tests and application-based tests. Stress testing or endurance testing is often used afterwards to eliminate marginal devices, such that fail due to parametric deviations, dynamic faults, intermittent or marginal defects introduced during final assembly of printed circuit boards. Functional test alone does not any longer guarantee fault-free operation of the target system in the field.

In Section 6, we have explored the ability of a generic program to detect delay faults in a typical functional board test scenario and empirically demonstrated that the board-level delay fault coverage (as defined in D1.3) is often much lower than expected independent of the test duration and its ability to test the processor stuck-at faults. Even a typical memory test program (employing a March algorithm) has delivered the board level delay fault coverage far from 100%. Finally, we proposed an ad hoc test program of a limited size and duration that explicitly targets delay faults on the memory bus, and showed that it can achieve full delay fault coverage at least on data lines.

One of the key contributions of WP4 is the concept of combined usage of functional tests and IC-level instruments during the post-assembly system test (board level or product test) in order to achieve a better fault coverage or faster fault detection and localization. In Section 3, we have described a method of complementing the concurrent checkers with functional test to be applied as a periodic routine during the idle periods. In addition, we have described a framework for formal qualification of checkers and for minimizing the overhead area with the given fault coverage constraints. The goal of that study was to achieve low-latency, low area overhead checkers for network on chip routers. The framework together with the corresponding methodology has been successfully applied to a realistic case-study of a fault tolerant NoC router design. The case study shows that combining concurrent routers with functional test allows reducing the area overhead of the checkers from 31-35% down to 1.5-10% without sacrificing the fault coverage.

As a result, a functional test may represent an effective solution for NFF, provided that suitable test programs and observability solutions are used.

Along with the latter method suitable for detecting NBTI-type aging, we have also developed an on-chip monitoring framework for early detection of interconnect wear-out both on-chip and off-chip (board-level). Section 2 presents a digital online IRF monitor to detect one of the most challenging interconnection reliability issues i.e. intermittent resistive fault (IRF). Early-stage detection of this fault can prevent catastrophic failures in safety-critical systems. The experimental results show that the presented monitor can detect small timing deviations produced by IRFs in data paths. In addition, hardware-based IRF injection technique and demonstrator (see D6.2) has been developed and evaluated.

In addition to IRFs (Section 2), TRFs (Section 6) and other wear-out effects, marginal defects represent a real test challenge, remain easy to escape and hence contribute to the NFF problem. Section 5 presents a set of developed embedded instruments on FPGA that specifically target marginal defects on board assemblies. We demonstrate how to convert some of the typical functional blocks of the IC into powerful embedded test

instruments. The instruments have been evaluated on industry-grade high-end products demonstrating stable detection of injected marginal faults, even such that would otherwise escape the normal functional test. The instruments populate the existing on-board FPGA device converting it temporarily into a fully-automated on-board embedded tester requiring no extra FPGA/DFT. The same principles can be further reused beyond FPGAs and find application in ASICs.

In-field (especially on-line real time) fault detection requires fast alarm delivery from instruments to the management software. In Section 4, we detailed a hardware module to perform the localization that detects the configuration of the network after self-reconfiguration and extracts the error information in real time. We showed that for large number of instruments, this hardware-based localization unexpectedly requires less area compared with a software-based localization.

The work described in this deliverable contributes towards the following KPI.

**KPI5: Reducing test escapes and impact of NFF by, at least a factor of 2.**

Reuse of chip-level embedded instrumentation and integration with functional test routines to improve diagnostic resolution and performance testing capabilities, both leading to one of the main objectives of BASTION: relieving the NFF problem. Synergy between embedded instruments and functional test routines is expected to reduce the test escapes and impact of NFF at least by the factor of 2.

WP4 was expected to contribute by the following activities:

- Improvements in the techniques for detecting hardware defects, thus reducing the chance for defects to escape the test.
- Measurement of the performance (test coverage improvement) of the new instruments and test routines (also during combined usage of functional test and instruments).

The work performed actually contributed to KPI5 in the following respects:

- With our developed IRF detection circuit, the test escape can be reduced by a certain amount depending on the number of detectors employed, and the parameters of the IRF to be detected (minimum resistance value as well as minimum occurrence time).
- Our evoking developments have shown to be enhancing the detection of all kind of faults in a chip, among them the IRFs. The improvements are somewhat comparable with conventional burn-in tests.
- Instruments and methods for marginal defects and TRF detection presented in this deliverable further contribute to reduction of test escapes. The actual reduction ratio can be estimated based on the new class of TRFs introduced in BASTION deliverable D1.3 (and which is shown by the BASTION demonstration set up – D6.2). Getting real numbers requires however long learning loop that involves data from actual production where the new technologies would have been implemented.

# 9 References


[Dav05] S. Davidson, “Towards and understanding of no trouble found devices,” Proc. VLSI Test Symp. (VTS), Palm Springs, California, USA, 2005, pp. 147-152.

[Acc08] Accenture Report, “Big trouble with ‘no trouble found’ returns” (2008), http://www.accenture.com/SiteCollectionDocuments/PDF/Accenture Returns Repairs.pdf.

[Bow09] K.A. Bowman et al., “Energy-efficient and metastability-immune resilient circuits for dynamic variation tolerance,” in IEEE Journal of Solid-State Circuits, vol. 44, 2009, pp. 49-63.

[Ebr16a] H. Ebrahimi and H.G. Kerkhoff, “Detecting intermittent resistive faults in digital CMOS circuits,” in IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2016, pp. 87-90.

[Ebr16b] H. Ebrahimi and H.G. Kerkhoff, “Testing for intermittent resistive faults in CMOS integrated systems,” in IEEE Euromicro Conference on Digital System Design (DSD) , 2016, pp. 703-707.

[Gil12] D. Gil-Tomás, J. Gracia-Morán, J. Baraza-Calvo, L.-J. Saiz-Adalid and P.-J. Gil-Vicente, “Studying the effects of intermittent faults on a microcontroller,” Microelectron. Reliab. 52, 2012, pp. 2837–2846.

[Gra10] J. Gracia-Morán, D. Gil-Tomás, L. J. Saiz-Adalid, J. C. Baraza, and P. J. Gil-Vicente, “Experimental validation of a fault tolerant microcomputer system against intermittent faults,” IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN), 2010, pp. 413-418.

[Gra14] J. Gracia-Moran, J. C. Baraza-Calvo, D. Gil-Tomas, L. J. Saiz-Adalid and P. J. Gil-Vicente, “Effects of intermittent faults on the reliability of a reduced instruction set computing (RISC) microprocessor,” IEEE Trans. on Reliability 63, 2014, pp. 144-153.

[Hsi15] M. H. Hsieh, Y. C. Huang, T. Y. Yew, W. Wang, and Y. H. Lee, “The impact and implication of BTI/HCI decoupling on ring oscillator,” in IEEE International Reliability Physics Symposium (IRPS), 2015, pp. 6A.4.1-6A.4.5.

[Kat15] K. Katoh and K. Namba, “A low area calibration technique of TDC using variable clock generator for accurate on-line delay measurement,” in International Symposium on Quality Electronic Design (ISQED), 2015, pp. 430-434.

[Ker15a] H.G. Kerkhoff and H. Ebrahimi, “Detection of Intermittent Faults in Electronic Systems based on the Mixed-Signal Boundary-Scan Standard”, in Proc. of the Asian Quality Electronic Design Conference (ASQED), 2015, pp. 77-82.

[Ker15b] H.G. Kerkhoff and H. Ebrahimi, “Investigation of Intermittent Resistive Faults in Digital CMOS Circuits”, Journal of Circuits, Systems and Computers (JCSC), World Scientific Publishing, vol. 25, no. 3 , 2015, pp. 1640023.

[Kim15] T. T. H. Kim, L. Pong-Fei, K. A. Jenkins, and C. H. Kim, “A Ring-Oscillator-Based Reliability Monitor for Isolated Measurement of NBTI and PBTI in High-k/Metal Gate Technology,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 23, 2015, pp. 1360-1364.

[Kra06] N. Kranitis, A. Merentitis, N. Laoutaris, et al., “Optimal periodic testing of intermittent faults in embedded pipelined processor applications,” presented at the Proceedings of IEEE Conference on Design, Automation and Test in Europe (DATE), 2006, pp. 65-70.

[Lai14] L. Lai, V. Chandra, R.C. Aitken and P. Gupta, “SlackProbe: a flexible and efficient in-situ timing slack monitoring methodology,” in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 33, 2014, pp. 1168-1179.

[Nan08] Nangate Inc., Sunnyvale, CA, “Nangate Open Cell Library,” 2008. [Online]. Available: http://www.nangate. com/

[Pan12] S. Pan, Y. Hu and X. Li, “IVF: Characterizing the vulnerability of microprocessor structures to intermittent faults,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 20, 2012, pp. 777-790.

[Rah14] A. Rahimi, L. Benini and R. Gupta, “Application-adaptive guard-banding to mitigate static and dynamic variability,” in IEEE Transactions on Computers, vol. 63, 2014, pp. 2160-2173.
[Reb11] B. Rebaud, M. Belleville, E. Beigné, C. Bernard, M. Robert, P. Maurine, et al., "Timing slack monitoring under process and environmental variations: Application to a DSP performance optimization," Microelectronics Journal, vol. 42, 2011, pp. 718-732.
[Rid13] Ridgetop Group Inc., SJ BIST, White Paper Presentation, 2013.
[Sad15] M. Sadi, L. Winemberg, and M. Tehranipoor, “A robust digital sensor IP and sensor insertion flow for in-situ path timing slack monitoring in SoCs,” in VLSI Test Symposium (VTS), 2015, pp. 1-6.
[Shi14] K. Shibin, S. Devadze, and A. Jutman, “Asynchronous Fault Detection in IEEE P1687 Instrument Network,” in North Atlantic Test Workshop, 2014, pp. 73-78.
[Sto10] E. A. Stott, J. S. Wong, P. Sedcole, and P. Y. Cheung. “Degradation in FPGAs: measurement and modelling,” in ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA), 2010, pp. 229-238.
[Wan07] W. Wang, S. Yang, S. Bhardwaj, R. Vattikonda, S. Vrudhula, F. Liu, and Y. Cao. “The impact of NBTI on the performance of combinational and sequential circuits,” in Conference on Design automation (DAC), 2007, pp. 364-396.
[Wan14] J. Wan and H. G. Kerkhoff, “The influence of no fault found in analogue CMOS circuits,” Proc. IEEE Int. Mixed-Signal Test Workshop (IMSTW), 2014, pp. 1-6.
[Tut1] R. Sedmak and H. Liebergot. Selective triple modular redundancy (STMR) based single-event upset (SEU) tolerant synthesis for FPGAs. *IEEE Transactions on Nuclear Science*, 51:2957-2969, 2005.
[Tut2] J. M. Berger. A note on an error detection code for asymmetric channels. *Information and Control*, 4:68-73, 1961.
[Tut3] D. Das and N. A. Touba. Synthesis of circuits withlow-cost concurrent error detection based on Bose-Lin codes. *In VLSI Test Symposium*, pages 309-315, 1998.
[Tut4] K. Mohanram, E. Sogomonyan, M. Gossel, and N. Touba. *Synthesis of low-cost parity-based partially self-checking circuits*, 2003.
[Tut5] S. Ghosh, N.A. Touba, and S. Basu. Synthesis of low power ced circuits based on parity codes. *In VLSI Test Symposium*, pages 315-320, 1-5 May 2005.
[Tut6] R. Sharma and K.K. Saluja. An implementation and analysis of a concurrent built-in self-test technique. *In Digest of Papers Eighteenth International Symposium on Fault-Tolerant Computing FTCS-18*, pages 164-169, June 1988.
[Tut7] P. Drineas and Y. Makris. Concurrent fault detection in random combinational logic. *In Proc. Fourth International Symposium on Quality Electronic Design ISQED*, pages 425-430, March 2003.
[Tut8] Alves, N.; Shi, Y.; Dworak, J.; Bahar, R.I.; Nepal, K. "Enhancing online error detection through area-efficient multi-site implications", *IEEE 29th VLSI Test Symposium (VTS)*, 2011.
[Tut9] Marc Boule, Jean-Samuel Chenard, and Zeljko Zilic. Assertion checkers in verification, silicon debug and infield diagnosis. *In Proceedings of the ISQED '07*.
[Tut10] M Aarna, E Ivask, A Jutman, E Orasson, J Raik, R Ubar, V Vislogubov, HD Wuttke. Turbo Tester-Diagnostic Package for Research and Training. *The 1st East-West Design and Test Conference*, Alushta, 2003.
[Tut11] Art. Jutman, A Peder, J Raik, M Tombak, R Ubar. Structurally synthesized binary decision diagrams. *6th International Workshop on Boolean Problems.* pp. 271-278, 2004.
[Tut12] Raimund Ubar, Sergei Devadze, Jaan Raik, Artur Jutman. Parallel X-fault simulation with critical path tracing technique. *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pp. 879-884, 2010.
[Tut13] J. Flich, J. Duato, Logic-Based Distributed Routing for NoCs, *IEEE Computer Architecture Letters*, Vol. 7, No. 1, January-June 2008.
[Tut14] R. Parikh and V. Bertacco. Formally enhanced runtime verification to ensure NoC functional correctness. *In Proc. of the International Symposium on Microarchitecture (MICRO)*, 2011.

[Tut15] Prodromou, A.; Panteli, A.; Nicopoulos, C.; Sazeides, Y., "NoCAlert: An On-Line and Real-Time Fault Detection Mechanism for Network-on-Chip Architectures," *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 60-71, 2012.
[Tut16] Yu, Qiaoyan; Cano, J.; Flich, J.; Ampadu, P., "Transient and Permanent Error Control for High-End Multiprocessor Systems-on-Chip," *2012 Sixth IEEE/ACM International Symposium on Networks on Chip (NoCS)*, vol., no., pp.169,176, 9-11 May 2012.
[Tut17] Alaghi, A.; Karimi, N.; Sedghi, M.; Navabi, Z., "Online NoC Switch Fault Detection and Diagnosis Using a High Level Fault Model," *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems*, vol., no., pp.21,29, 26-28 Sept. 2007.
[Tut18] Dalirsani, A.; Kochte, M.A.; Wunderlich, H.-J., "Area-efficient synthesis of fault-secure NoC switches," *IEEE 20th International On-Line Testing Symposium* (*IOLTS*), pp.13,18, 7-9 July 2014.
[Tut19] J. Raik, V. Govind, R. Ubar. An External Test Approach for Network-on-a-Chip Switches. Proc. of the IEEE Asian Test Symposium, pp. 437-442, Nov. 2006
[Tut20] J. Raik, V. Govind, R. Ubar. Design-for-Testability- Based External Test and Diagnosis of Mesh-like NoCs. *IET Computers and Digital Techniques*, Vol. 3, Issue 5, pp. 476-486, September 2009.
[Tut21] Raik, J.; Govind, V. Low-area boundary BIST architecture for mesh-like network-on-chip, *IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, pp. 95-100, 2012.
[Bo05] S. Borkar, "Designing Reliable Systems from Unreliable Components: the Challenges of Transistor Variability and Degradation," IEEE Micro, vol. 25, no. 6, pp. 10–16, 2005.
[Su07] Sun Microsystems, Inc. (2016, Mar.), "UltraSPARC T2TM Supplement to the UltraSPARC Architecture 2007." [Online]. Available: http://www.oracle.com/technetwork/systems/opensparc/t2-14-ust2-uasuppl-draft-hp-ext-1537761.html
[Ieee01] IEEE Association, "IEEE Std 1149.1-2001, IEEE Standard Test Access Port and Boundary-Scan Architecture," 2001.
[Bou12] A. Bouajila, A. Lakhtel, J. Zeppenfeld et al., "A Low-Overhead Monitoring Ring Interconnect for MPSoC Parameter Optimization," in Proc. IEEE International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS), April 2012.
[Mad09] S. Madduri, R. Vadlamani, W. Burleson et al., "A Monitor Interconnect and Support Subsystem for Multicore Processors," in Proc. Design, Automation & Test in Europe Conference, April 2009.
[Pet14] K. Petersen, D. Nikolov, U. Ingelsson et al., "Fault Injection and Fault Handling: An MPSoC Demonstrator using IEEE P1687," in Proc. IEEE International On-Line Testing Symposium (IOLTS), 2014, July 2014, pp. 170–175.
[Jut13] A. Jutman, S. Devadze, and K. Shibin, "Effective Scalable IEEE 1687 Instrumentation Network for Fault Management," IEEE Design & Test, vol. 30, no. 5, pp. 26–35, Oct 2013.
[Shi14] K. Shibin, S. Devadze, and A. Jutman, "Asynchronous Fault Detection in IEEE P1687 Instrument Network," in Proc. IEEE North Atlantic Test Workshop (NATW), May 2014, pp. 73–78.
[Jut16] A. Jutman, K. Shibin, and S. Devadze, "Reliable health monitoring and fault management infrastructure based on embedded instrumentation and IEEE 1687," in AUTOTESTCON, Sept 2016.
[Ibr16] A. Ibrahim and H. G. Kerkhoff, "Efficient Utilization of Hierarchical iJTAG Networks for Interrupts Management," in Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), Sept 2016.
[Zad16] F. Ghani Zadegan, D. Nikolov, and E. Larsson, "A Self-Reconfiguring IEEE 1687 Network for Fault Monitoring," in Proceedings of the European Test Symposium (ETS), May 2016.
[Ieee14] "IEEE Standard for Access and Control of Instrumentation Embedded within a Semiconductor Device," IEEE Std 1687-2014, Dec 2014.
[Tsmc16] TSMC (2016, Dec.), "65nm Technology." [Online]. Available: http://www.tsmc.com/english/dedicatedFoundry/technology/65nm.htm

[Chen02] Chen, L., Bai, X., & Dey, S. (2002). Testing for interconnect crosstalk defects using on-chip embedded processor cores. Journal of Electronic Testing, 18(4/5), 529–538.
[Kim04] Kim, H. C., Jun, H.-S., & Chung, S. S. (2004). At-speed interconnect test and diagnosis of external memories on a system. 2004 International Conference on Test, 156–162
[Ge09] Geiger, P. B., & Butkovich, S. (2009). Boundary-scan adoption - an industry snapshot with emphasis on the semiconductor industry. In 2009 IEEE International Test Conference (pp. 1–10).
[Lai14] Lai, D., Chen, A., & Li, Y. L. (2014). MRC training vs. Board defect, 81–83.
[Bla11] Blankenbeckler, D., Norman, A., & Shepherd, M. (2011). Comparison of off-chip interconnect validation to field failures, (c), 121–125.
[Bi09] Birch, B. (2009). Reliability testing for microvias in printed wire boards. Circuit World, 35(4), 3–17.
[Pa09] Parker, K. P. (2009). The effects of defects on high-speed boards. In IEEE International Conference on Test, 2005. (pp. 180–187).
[Ha00] S. Hall, G. Hall, and J. McCall, High Speed Digital System Design, John Wiley & Sons, Inc. (Wiley Interscience), 2000, 1st edition
[Ch15] E. Chielle, G. S. Rodrigues, F. L. Kastensmidt, S. Cuenca-Asensi, L. A. Tambara, P. Rech, H. Quinn. "S-SETA: Selective Software-only Error-detection Technique using Assertions," IEEE Transactions on Nuclear Science, vol. 62, no. 6, 2015.
[Be16] P. Bernardi, R. Cantoro, S. De Luca, E. Sánchez, A. Sansonetti, “Development Flow for On-Line Core Self-Test of Automotive Microcontrollers”, IEEE Transactions on Computers, 2016, Volume: 65, Issue: 3, Pages: 744 - 754
[Gr01] J. Gracia, J. Baraza, D. Gil, P. Gil, “Comparison and application of different VHDL-based fault injection techniques”, Defect and Fault Tolerance in VLSI Systems 2001. Proceedings. 2001 IEEE International Symposium on, pp. 233-241, 2001
[Gi99] P. Girard, C. Landrault, V. Moréda, S. Pravossoudovitch, A. Virazel , “A New Scan-BIST Structure to Test Delay Faults in Sequential Circuits”, Journal of Electronic Testing, February 1999, Volume 14, Issue 1, pp 95–102, Springer
[Gr11] J. Grinschgl, A. Krieg, C. Steger, R. Weiss, H. Bock, J. Haid, "Modular fault injector for multiple fault dependability and security evaluations", DSD, 2011.
[Ju16] Artur Jutman; Igor Aleksejev; Sergei Devadze, “On coverage of timing related faults at board level”, 2016 21th IEEE European Test Symposium (ETS)
[Ta09] T. Taylor, “Functional Test Coverage Assessment Project”, IEEE 8th International Board Test Workshop (BTW’2009), Fort Collins, USA, Sept 15-17, 2009.
[vd10] A. J. van de Goor, G. Gaydadjiev, S. Hamdioui, “Memory testing with a RISC microcontroller”, Design, Automation and Test in Europe, Dresden, 2010.
[vd91] A. J. van de Goor, Testing Semiconductor Memories, Theory and Practice, New York: John Wiley & Sons, 1991.
[Ji16] Shi Jin; Fangming Ye; Zhaobo Zhang; Krishnendu Chakrabarty; Xinli Gu, “Efficient Board-Level Functional Fault Diagnosis With Missing Syndromes”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2016, Volume: 35, Issue: 6, pp. 985 – 998
[Ta15] Mottaqiallah Taouil; Mahmoud Masadeh; Said Hamdioui; Erik Jan Marinissen, “Post-Bond Interconnect Test and Diagnosis for 3-D Memory Stacked on Logic”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2015, Volume: 34, Issue: 11, pp. 1860 – 1872
[Yi06] Hyunbean Yi; Jaehoon Song; Sungju Park, “Interconnect Delay Fault Test on Boards and SoCs with Multiple Clock Domains”, 2006 IEEE International Test Conference
[Ju14] Artur Jutman; Matteo Sonza Reorda; Hans-Joachim Wunderlich, “High Quality System Level Test and Diagnosis”, 2014 IEEE 23rd Asian Test Symposium
[Hi02] K. Hird; K. P. Parker; B. Follis, “Test coverage: what does it mean when a board test passes?”, IEEE International Test Conference, 2002, pp. 1066 – 107

[OR] OpenRISC 1200 IP Core Specification, http://opencores.org/websvn,filedetails?repname=openrisc&path=%2Fopenrisc%2Ftrunk%2For1200%2Fdoc%2Fopenrisc1200_spec_0.7_jp.pdf.
[WB] Introduction to the WISHBONE Bus Interface, http://indico.ictp.it/event/a11204/session/35/contribution/22/material/0/0.pdf.
[Ft17] http://www.ftdichip.com/Products/Cables/USBMPSSE.htm
[Ac17] http://accesio.com/?p=../usb/usb-dio-16h.html
[Mc17] http://www.mccdaq.com/usb-data-acquisition/USB-DIO32HS-Series
[Ni17] http://www.ni.com/singleboard/