FP7-ICT-2013-11-619871

BASTION

Board and SoC Test Instrumentation for Ageing and No Failure Found

Instrument: Collaborative Project
Thematic Priority: Information and Communication Technologies

Report on Error Classification and Corresponding Error Handling (Deliverable D3.1)

Due date of deliverable: December 31, 2015
Ready for submission date: February 4, 2016

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

Organisation name of lead contractor for this deliverable: Lund University

Revision 2.0

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

Notices

For information, please contact Erik Larsson, e-mail: erik.larsson@eit.lth.se

This document is intended to fulfil the contractual obligations of the BASTION project concerning deliverable D3.1 described in contract 619871.

Table of Revisions

Version	Date	Description and reason	Author	Affected sections
1.0	October 19, 2015	Structure created	E. Larsson	Contents
1.1	November 5, 2015	Contribution by Testonica	A. Jutman	Parts concerned TL
1.2	November 20, 2015	Contributions by ULUND	E. Larsson	Parts concerned ULUND
1.3	December, 2nd, 2015	Contribution of Uni of Twente	H.G. Kerkhoff	Parts concerned UT
1.4	December 3, 2015	Contribution by IFAG	M. Beck	Parts concerned IFAG
1.5	December 7, 2015	Formatting	E. Larsson	All sections
1.6	December 11	Complementing contributions from UT	H.G. Kerkhoff	Parts concerned UT
1.7	December 16	Formatting	E. Larsson	All sections
2.0	December 21	Polishing	E. Larsson	All sections

Author, Beneficiary

Artur Jutman, Testonica Lab
Piet Engelke, Infineon
Matthias Beck, Infineon
Hans G. Kerkhoff, University of Twente
Alireza Rohani, University of Twente
Hassan Ebrahimi, University of Twente
Erik Larsson, University of Lund

Executive Summary

The document reports on the error classification and the corresponding error handling activities performed in the FP7-project BASTION. The document reports on the activities in T3.1, which aim at finding methods to classify the defects and for each class define appropriate error handling schemes. The overall bottleneck is low fault detection and localization latency in existing fault management systems. This has been addressed by the development of low-latency error detection monitors and optimized IEEE P1687 networks managed hierarchically. In respect to error classification, we contribute with aging fault classification, error classification from the perspective of the fault manager, design partitioning for better error localization and debugging in built-in self-test modeling. In respect to error handling, we contribute with health map implementation and error handling by the instrument manager and the fault manager as well as with quality metric for error handling.

List of Abbreviations

AET	- Average Execution Time
ATPG	- Automatic Test Pattern Generation
BTI	- Bias Temperature Instability
BIST	- Built-In Self-Test
CMOS	- Complementary Metal-Oxide Semiconductor
CPU	- Central Processing Unit, also Processor
CUT	- Circuit Under Test
DfT	- Design for Test
EDT	- Embedded Deterministic Test
EM	- Electro-migration
FM	- Fault Manager
FPU	- Floating point unit
FP7	- European Union's 7th Framework Program
GCDD	- Greatest Common Divisor of Deadlines
HCI	- Hot Carrier Injection
ICL	- Instrument Connectivity Language
IJTAG	- Internal JTAG, a short name for IEEE 1687 standard and infrastructure collectively
IM	- Instrument Manager
IRF	- Hardware
ICL	- Instrument Connectivity Language
IP	- Intellectual Property (hardware module in FPGA or SoC)
IRF	- Intermittent Resistive Faults
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;
LBIST	- Logic BIST
LOC	- Level of Confidence
MISR	- Multiple-Input Signature Register
NBTI	- Negative Bias Temperature Instability
NFF	- No Fault Found or No Failure Found (also NTF)
OS	- Operating System
PDL	- Procedural Description Language
PMOS	- p-type Metal Oxide Semiconductor
PRPG	- Pseudo-Random Pattern Generator
RRC	- Rollback Recovery with Checkpointing
RM	- Rate-Monotonic

RTS	- Real-Time System
STUMPS	- Self-Test Using Multiple-input signature register and Parallel Shift-register sequence generator
TAP	- Test Access Port
TDDB	- Time Dependent Dielectric Breakdown
WCET	- Worst-Case Execution Time
TDDB	- Time Dependent Dielectric Breakdown

Table of Contents

1 Introduction

This deliverable reports result performed by BASTION partners on error classification and corresponding error handling. The document reports on the activities in T3.1, which aim at finding methods to classify the defects and for each class define appropriate error handling schemes. The overall bottleneck that is addressed is the low fault detection and localization latency in existing fault management systems. This has by the BASTION project been addressed by the development of low-latency error detection monitors and optimized IEEE P1687 networks managed hierarchically.

In respect to error classification, we contribute with aging fault classification, error classification from the perspective of the fault manager, design partitioning for better error localization and debugging in built-in self-test modeling. In respect to error handling, we contribute with health map implementation and error handling by the instrument manager and the fault manager as well as with quality metric for error handling.

1.1 Structure of the Document

This report is structured as follows. We first give a background on error classification and then on error handling (Section 2). Second, we detail state of the art on error classification and then on error handling (Section 3). In Section 4 we describe our contributions on error classification and in Section 5 we detail our contributions in respect to error handling. Section 6 summarizes the deliverable.

2 Background

In this section we give background information on error classification (Section 2.1) and error handling (Section 2.2).

2.1 Error Classification

In this section, we detail the background on error classification, which includes modeling of negative bias temperature instability (NBTI), the usage of logic built-in self-test (LBIST) for error classification, and how error classification can be performed in-field.

When it comes to modeling of NBTI, the fact is that bias temperature instability (BTI) is a degradation mechanism that occurs in MOS devices as a result of interface traps between the gate oxide and silicon substrate at elevated temperatures, and it therefore degrade the dependability of associated electronic devices. This degradation mechanism results in device threshold voltage (V_{th}) shift and loss of drive current. The BTI in PMOSFETs is referred to as NBTI due to the negative gate-to-source voltage. NBTI has been shown to be the dominant effect in current process technologies. A model of NBTI fault has been presented in D1.1.

Logic built-in self-test (LBIST) solutions can be implemented in integrated circuits for detecting aging defects in the chip-internal digital logic. Reusing the LBIST as test instrument on the board-level allows testing the chip-internal logic in the system application. By default the LBIST provides only a simple pass/fail decision. An error localization or error classification is not easily possible. To enhance the LBIST

resolution for an improved error handling, design partitioning strategies allowing better error localization are analyzed taking into account design implementation aspects. For a more detailed error classification LBIST debugging approaches are evaluated considering the trade-off between accuracy and effort required for design and test execution.

In general, error classification for in-field on-line test has not been considered so far. In this report, we propose a general fault classification methodology for electronic systems, which is well-suited for on-line fault management tasks.

2.2 Error Handling

In this section we detail background in respect to error handling. We discuss error handling for aging and NFFs, on-line fault management and the quality of fault management.

For the error handling for ageing and NFFs, we have previously focused on two categories of faults that, under certain conditions, can result in errors. We distinguish between faults and errors as in our terminology it is important to point out that many faults do not result in errors. Especially in the digital case, many faults are masked, e.g. as result of redundancy. In respect to fault, in particular we have investigated aging faults, resulting from NBTI (and HCI), see deliverable D1.1 for details. Models have been developed, and we are capable of simulating aging behavior of digital CMOS transistor designs in commercial simulators. Several health monitors (error-detection circuits, embedded instruments) have been investigated, and some have been designed. A delay monitor has been introduced in deliverable D2.1. A delay monitor is motivated by the fact that delay is known to degrade as function of time (aging). The second category of faults is, a family within NFFs, called intermittent resistive faults (IRF). IRFs have been discussed in deliverable D1.1. Several designs of health monitors (error-detection circuits, embedded instruments) have been investigated and published [1].

On-line fault management is a methodology to handle errors and faults helping an electronic system to acquire, timely update, maintain and use the information about the faults that can occur in various system modules. In other words, the Fault Management process helps the system to be aware of the faults and thus to continue functioning, albeit with some of the modules being non-operational, instead of going fully out-of-order. When a fault happens in system's resource, the system becomes aware of the fault and can take actions to avoid using the faulty resource. Therefore it is important to classify faults into various categories. Depending on situation (and the characteristics of the detected fault), the system will reschedule the tasks to other resources or start using spare one if it was provisioned for. It will also initiate additional diagnostic procedures if necessary. On-line fault management is intended to be used in systems with CPU and OS where the scheduler is modified to take advantage of the fault management system.

Quality of error handling is important as many computer systems must provide correct response within given time constraints. These computer systems are known as real-time systems (RTS). RTS are often classified into soft RTS and hard RTS. For soft RTS, failing to meet the time constraints usually results in performance degradation,

while violating the time constraints can be catastrophic for hard RTS [2]. Therefore, for soft RTS, it is important to minimize the average execution time (AET), while for hard RTS it is more important to analyze the worst-case execution time (WCET). Obtaining the minimal AET has some important advantages. For example, reducing the AET can lead to a lower power consumption and thus save energy, which is very important for embedded systems. For soft real-time control systems, where tasks are executed periodically, reducing the AET is important as it can affect the control quality, i.e. tasks periods can be adjusted according to AET estimates while optimizing a control performance criterion [3]. While the minimal AET provides these advantages, having the minimal AET does not guarantee that time constraints (deadlines) are always met. WCET, on the other hand, along with schedulability analysis, can guarantee that deadlines are always met and thus catastrophic consequences can be avoided. However, the WCET may be very pessimistic and can result in having over-designed systems. As progressing towards sub-micron technologies, increasing soft error rates are observed [4] [5] [6] [7] [8]. These soft errors impact both soft and hard RTS. To cope with the presence of soft errors, RTS must incorporate some fault- tolerant techniques to ensure correct operation [9]. However, usage of a fault-tolerant technique usually results in a time penalty, and thus might result in a deadline violation in RTS. Due to that errors can occur unexpectedly at any moment in time and that the employed fault-tolerant technique may require a non-constant amount of time to handle the errors, non-determinism is introduced and thus makes it very difficult to perform accurate analysis on the WCET. To perform analysis on hard RTS most related works assume a fault model where the number of faults is bounded to a fixed number [10] [11] [12]. In this regard, using WCET analysis along with schedulability analysis provides guarantees that the deadlines will be met as long as the assumed fault model matches reality. Unfortunately this is often not the case, since errors can occur unexpectedly at any moment in time and therefore it is difficult to predict the number of errors that can occur within an interval of time.
Analyses on AET, on the other hand, are possible with a less stringent fault model. However, this is mainly applicable for soft RTS. Most of the work focused on soft RTS provides some optimization methods on the fault-tolerant techniques with the goal to minimize the AET [13] [14] [15] [16] [17] [18]. The drawback with the minimal AET is that we lack the knowledge on the distribution of the execution times, i.e. sometimes a job can complete much earlier in time, while sometimes it may take much longer time. From the discussion above one notes that the optimization objectives differ between soft and hard RTS, i.e. AET is used for soft RTS and WCET is used for hard RTS. Hence, the optimization objectives are clearly defined and separated for soft and hard RTS. This leads to a difficult problem for system designers. Prior to designing a system, designers must decide if the system that is to be designed is a soft or a hard RTS. In such regard, using analyses based solely on AET and WCET becomes insufficient. Therefore there is a need to have a unifying metric equally applicable for both soft and hard RTS. In our previous work we used the concept of Level of Confidence (LoC), to evaluate the probability that deadlines are met [19]. The LoC can be seen as a metric to measure to what extent a deadline is met. The advantage with this metric is that it is equally good to evaluate soft and hard RTS, which previously has not been considered, i.e. AET has been used for evaluation of soft RTS and WCET has been used for evaluation of hard RTS. By introducing this metric, a designer of an RTS can specify to what extent a deadline must be met. This metric can be used as an optimization goal when optimizing the fault- tolerant technique that provides the needed level of reliability.

3 State of the Art

In this section we describe state of the art in respect to error classification (Section 3.1) and error handling (Section 3.2).

3.1 Error Classification

For previous work on error classification, we detail in first modeling of NBTI and IRF and then we discuss error classification performed by the fault management, and finally work on error localization using logic built-in self-test (LBIST) is described.

In [20], the authors have studied the NBTI relaxation. They used a new ultra-fast (UF) technique to accurately characterize the recovery of the NBTI degradation from very short to medium relaxation times. Moreover, they have presented a framework of the NBTI physics modeling, and a procedure to transfer the NBTI degradation effects from device level to circuit level. As there was hardly any background on IRFs (D1.1), experimental measurements were carried out (cold soldering), and a Verilog-A model was developed.

In general, error classification for in-field on-line test has not been considered so far. There are specific ad-hoc approached developed for a particular type of devices, e.g. the well-known S.M.A.R.T technology that stands for self-monitoring, analysis and reporting technology [21], which measures hard drive characteristics such as temperature, spin-up time and data error rates. For hard drives there are further techniques for fault analysis and classification [22]. In this report, we propose a more general fault classification methodology for electronic systems, which is well-suited for on-line fault management tasks.

LBIST solutions for the test of chip-internal digital logic are often based on the STUMPS approach [23]. A pseudo-random pattern generator (PRPG) is used to provide the test stimuli. The test data is distributed within the circuit under test (CUT) by reusing the existing scan test architecture of the chip-level production test. A multiple-input signature register (MISR) is used to compress the test responses coming from the scan chains. The overall test result is determined by checking the actual MISR signature versus the “golden” reference value. Due to the test data compression in the MISR, the LBIST is usually a pure go/no-go test providing only very little information about the root cause of the failure. For many IC designs the scan test architecture is still implemented flat on the top-level. This means the test patterns are generated in a single automatic test pattern generation (ATPG) run for the overall design. This minimizes not only the setup effort for the pattern generation, but also has some benefits for the ATPG process. As the flat top-level scan approach usually enables very balanced scan chain lengths, there is no test time penalty due to inefficient scan chain loading. When using on-chip test data compression techniques as Embedded Deterministic Test (EDT) [24], there is another advantage of the flat ATPG approach. The test data provided by the tester through the EDT input channels can be distributed to all internal scan chains allowing an optimal usage of the test interface bandwidth. As the LBIST is usually reusing the scan test architecture of the production test the flat top-level scan architecture restricts its granularity. This leads to global self-test solutions for the complete design with only an overall pass/fail decision. For a better error localization and error handling it is desirable to replace the overall top-level LBIST by a hierarchical self-test allowing the identification which

component of the design has shown the failure. Some IC designs start implementing hierarchical test solutions mainly driven by design complexity and hierarchical design implementation flows. Even if this would enhance the LBIST resolution, the benefits for the error classification and error handling might be limited, when the circuit partitioning is only based on design aspects and not focusing also on enhancing the test resolution.

3.2 Error Handling

In this section, we detail previous work on error handling. We first describe previous works on handling data generated by embedded monitors. Secondly, we discuss error handling performed by the fault management. Finally, we discuss quality of error handling.

Previous work has addressed health monitors for NBTI-based faults, as well as IRF faults (D1.1, D2.1). In the case of NBTI, a delay monitor was linked to the IEEE 1687 environment, incorporating several muxes and registers. Also the organization of data-handling (rather than error-handling, see previous remarks) from different monitors was developed to the top-level, enabling support for final manufacturing test, as well as life-time (on-chip) test. The organization is IP-based.

Typical solution to handle errors today is tolerating them by redundancy or hardening. Aging faults are mitigated by removing stress or reducing the operational frequency. In BASTION we are developing a more general methodology to handle and manage faults based on a health map. Fault management is distributed across system levels from HW to operating system, which operates being aware of the current status of system integrity (health).

The quality of error handling is discussed in this section. We mainly discuss related work that addresses RRC used in RTS with the aim to maximize the probability to meet deadlines. Kim et al. provide probabilistic schedulability analysis for multiple real-time tasks, which employ check-pointing [25]. Using Markov model, the authors derive an expression to calculate the probability that all tasks will be able to meet their deadlines over a number of hyper-periods (shortest repetitive sequence of the schedule) for a given checkpoint interval (time between two consecutive checkpoints). The authors assume that tasks are scheduled by the rate-monotonic (RM) scheduling algorithm where higher priority tasks can preempt lower priority tasks. In this work, the authors have shown that there exists an optimal checkpoint interval that maximizes the probabilistic schedulability (probability that all tasks meet their deadlines). The weakness of this work is that it is only limited to real-time tasks with harmonic periods (the periods of all tasks is a multiple of the smallest period among the tasks), and the fact that the same checkpoint interval is used for all tasks.
Kwak et al. derive an expression to calculate the probability that a set of real-time tasks which employ checkpointing meet their deadlines during a hyper-period [26]. The tasks are scheduled according to RM. In contrast to Kim et al. [25], in this work, Kwak et al. relax the assumption of harmonic periods, and their approach is applicable for real-time tasks with arbitrary periods. Further, Kwak et al. assume different checkpoint intervals for different tasks. In [26], the authors show that there exists an optimal assignment of checkpoint intervals. However, obtaining the optimal checkpoint intervals requires exploration of many different assignments, and thus it may be very time consuming. In another paper, Kwak et al. provide analysis on the

reliability of a checkpointed real-time control systems [27]. In this work, the authors consider a control system that consists of a single control task for which the WCET, the period and the deadline are given. For the fault model, they consider that transient faults occur according to a Poisson process with a fault arrival rate λ and recovery rate μ. By utilizing Markov models they derive the reliability equation over a mission time (number of consecutive sampling periods) of the control system. Kwak et al. model the control system by a 3-state Markov chain, where within a sampling period the control system can be in one of the following states: (1) the control task has been correctly executed and the transient faults are in a fault-free state, (2) the control task has been correctly executed and the transient faults are in a fault-active state and (3) the control task has either been executed incorrectly or has not finished within the sampling period. The authors have shown the impact of the number of checkpoints on the system reliability and therefore they have proposed an algorithm to find the optimal number of checkpoints with the goal to maximize system reliability. In their work the authors used the assumption that the checkpoints are uniformly placed, i.e. an equidistant checkpointing scheme is used. Further, they extend the model to address system's reliability for a control system which consists of multiple control tasks, where for each control task the WCET and the period (which is equal to the deadline, Di) are given. For the case of multiple control tasks, they propose a task allocation algorithm. The algorithm computes the greatest common divisor of deadlines (GCDD) and divides each control task into Di/GCDD equal length smaller chunks (subtasks). Each subtask is then sequentially assigned on each GCDD interval. After running the task allocation algorithm, they apply the system reliability model for a single control task with a deadline equal to GCDD, assuming that all the subtasks assigned within the GCDD interval are equivalent to one control task. Out of this they can obtain the optimal number of checkpoints to be used for each task such that the system reliability is maximized. Related to this, we show in this paper that such modeling as described in [27] does not provide the optimal solution. First, computing the GCDD in [27] is very similar to what we call the Local Optimization approach (described in Section V), and second, the assumption, which is used in [27], i.e. assuming that all the subtasks assigned within the GCDD interval are equivalent to one control tasks, is very similar to what we call the Single Large Job approach. We show that both of these approaches, i.e. the Local Optimization and Single Large Job approach, do not provide the optimal solution.

4 Error Classification

In this chapter, we will detail our contribution on aging fault classification (Section 4.1), error classification from the perspective of fault management (Section 4.2), design partitioning for better LBIST error localization (Section 4.3), and debugging approaches in an LBIST environment (Section 4.4).

4.1 Aging Fault Classification

Aging faults are usually classified as recoverable and irrecoverable faults. NBTI is one of the most prominent recoverable aging effects. Electro-migration (EM) is another aging phenomenon that happens in metal wires, and it classified as recoverable fault. Other aging faults like Hot Carrier Injection (HCI) and Time Dependent Dielectric Breakdown (TDDB) can be classified into irrecoverable fault because their damage is usually permanent and cannot be recovered when stress conditions are removed.

4.2 Fault Management and Health Map Implementation

To manage the faults in the system, we classify them by several properties (persistence, severity, criticality, diagnostic granularity and location). The classification has a strong relation to fault management processes and the architecture of the Health Map (error handling in general). Taking into account the information from the health map, the Fault Manager would modify the current resource map used by the OS to identify currently available system resources. Hence, the OS scheduler becomes able to use this information in order to execute tasks only on healthy resources.

Classification of faults to be used in FM procedures consists of the following categories:

Persistence: Transient, intermittent, permanent – from instrument or from the fault statistics. This parameter shows what the nature of the fault occurrence is, and combined with the data from the fault statistics memory and the criticality of the tasks, it can be used to determine if the OS scheduler can assign this resource to tasks.

Severity: Faults can be different in their influence on the resource. While one fault can be benign (e.g. one of several similar execution units in a superscalar CPU fails), another can make the resource useless (e.g. program counter in a CPU core). Depending on how critical the fault is for the resource it was detected in, the scheduler may be allowed or not to assign the respective resource to tasks. This parameter will be determined by the OS during the diagnostic procedure.

Criticality: Depending on the resource where the fault has occurred, its consequences for operability and stability of the system as a whole can span from none to total system failure. Depending on the importance of the resource and hence the criticality of the fault, the fault management system can prioritize the need to launch a diagnostic procedure for the faulty resource to find out what is the persistence and severity of the fault.

Diagnostic Granularity: found by an instrument or deducted by diagnostic procedure. A fault in the data structure of fault management system should be assigned with the information about how it was found – either by hardware (F flag in

the IJTAG network **Invalid source specified.**), by RM (during the diagnostic procedure) or by the OS using high-level fault detection methods (e.g. temporal or modular redundancy).

Fault location: localization of the fault occurrence as a result of fault detection or fault diagnosis procedure. Depending on the information source, it can be of various types:

- location with respect to the instrumentation network hierarchy if detected by Instrument Manager (IM) hardware;
- fine-grain location in a sub-module, particular registers or logic gate if found by the diagnosis procedure;
- coarse grain location in system module if detected by OS

Fault locations from the perspective of IM and the OS may not be equivalent and a certain "translation" or mapping of fault locations between different agents is needed. System Health Map provides the basis for such a mapping mechanism.

4.3 Design Partitioning for Better LBIST Error Localization

Dividing the overall IC design into smaller sub-modules and testing these partitions with independent LBIST implementations can increase the diagnosis resolution of the overall chip-level self-test. Based on which LBIST controller has identified the failure a direct link to the faulty sub-module is given. Partitioning of the design does not only improve the rough error localization, but can also help for the classification of errors. If the design is partitioned based on the module functionality, this can help to judge the severity of a failure and allow an improved error handling. Based on the failing component it might be acceptable to switch off the defective part accepting some performance loss or limited functionality. In some cases it might be also possible to reschedule some tasks from the defective part to other functional modules with comparable capabilities. In case of available redundancy it might be even possible to completely replace the faulty component by an identical spare part.
For ICs in safety relevant applications the design partitioning could distinguish between design components that are essential to ensure the functional safety and other parts of the design that are less critical. Again the independent LBIST execution for sub-modules with different criticality can be helpful for the error classification regarding the severity of a failure. In addition this design partitioning based on safety aspects can consider the more strict self-test requirements for the safety critical components as requested by the new ISO26262 safety standard for automotive applications.

The resolution of the overall chip-level self-test increases with the number of design partitions independently tested. For the error localization a very large number of design partitions would be beneficial. This however would cause several drawbacks for the design implementation and handling effort. To allow an independent test of each partition some module isolation is required ensuring that the test execution is not disturbed by the surrounding logic. In a typical implementation style such an isolation wrapper could contain dedicated isolation cells or try to reuse existing registers. Even if the impact on area and timing varies, both isolation methodologies require some additional effort for the implementation. As each partition requires its own separate LBIST controller, the area overhead increases with the number of self-testable modules. Also the effort for the LBIST and ATPG tool setup increases, as it has to be

spent for each module. In addition to the LBIST controller each partition also requires a local clock control module providing the required clocking schemes for the LBIST capture phases. Especially for the delay test targeting defects causing performance degradation e.g. due to aging, this clock controller also needs to perform clock switching from a rather relaxed scan shift frequency which is limited by the power consumption to an at-speed clock source used for the launch and capture pulses. This local clock control module again contributes to the area overhead of each partition, but more importantly it also complicates the overall chip-level clock tree architecture.
As the insertion of test isolation wrappers is tightly linked to the design synthesis process, the number of partitions for a hierarchical test approach is usually directly coupled to the number of partitions used for the physical implementation. From the implementation point of view the design partitioning might be desired to deal with design complexity and tool runtimes, but on the other hand the required constraining of the module interfaces causes some additional effort and there might be an impact on the implementation results as the optimization capabilities over the module boundaries are reduced.

As a consequence a reasonable number of design and test partitions have to be chosen as a compromise considering error localization capabilities, design constraints, setup effort and handling aspects. Ideally the partitions should be of comparable sizes. If the number of test partitions is not sufficient for the error localization or an appropriate error handling, the resolution can be further improved by additional design measures within the partitions as discussed in the next section.

4.3.1 Design Measures to Increase LBIST Resolution

For a given hierarchical LBIST implementation the diagnosis resolution can be enhanced by selectively testing only certain parts of the test partition. By placing AND-gates at the end of the scan chains as shown in Figure 4.1, certain groups of chains can be masked before the data is propagated through the XOR-compactor to the MISR.

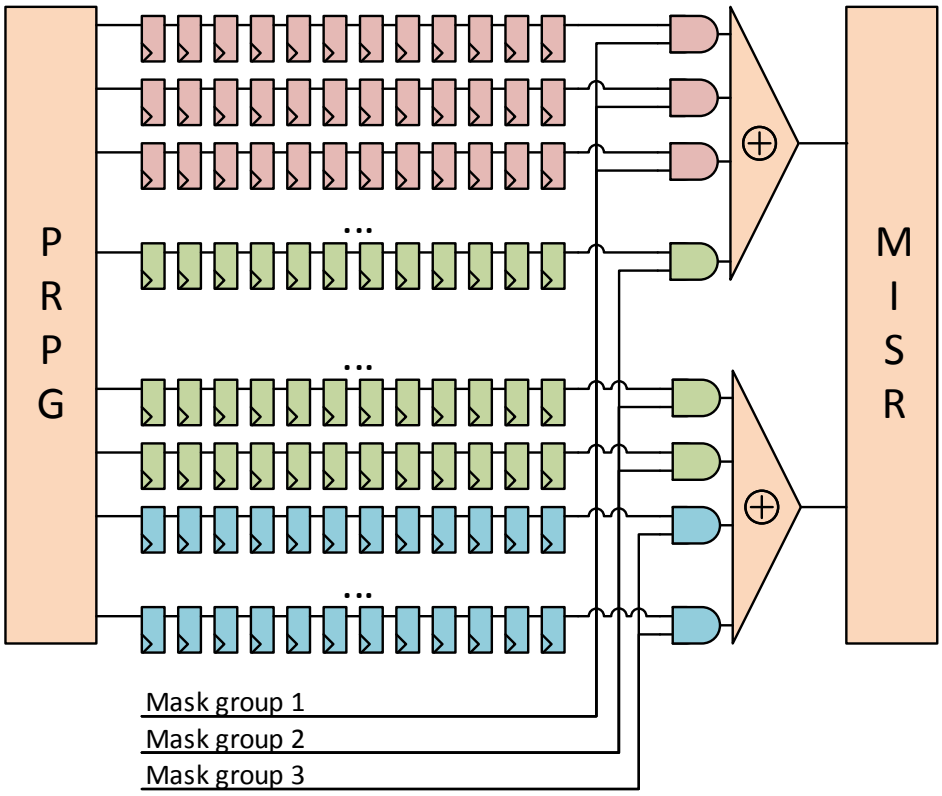

Figure 4.1: LBIST with chain group masking

The scan chain grouping of Figure 4.1 is very flexibly allowing each scan chain to be assigned to a certain mask group. Also the number mask groups to be implemented can be individually decided for each test partition without any limits. In an extreme case each scan chain could be masked individually. While this mask grouping can be implemented allowing a very flexible selective test approach, it should be taken into account that for each mask configuration the "golden" MISR signature needs to be calculated upfront to allow the pass/fail decision. This means the more test configurations are used the higher is the effort to provide the reference signatures and to handle them for the test evaluation.

The masking approach shown in Figure 4.1 is not only helpful to increase the diagnosis resolution, but could also be beneficial to deal with unforeseen design issues that would prevent the LBIST execution. In case one scan flip-flop would capture unreliable data during the test execution, the complete LBIST would be useless as the MISR signature would not be predictable. If such a situation would have been missed in the design verification, the affected scan chain or scan group could be masked and the LBIST could be performed at least for the remaining scan groups with reduced test coverage.

In Figure 4.1 one AND-gate is used for each internal scan chain. To reduce the LBIST test execution times it is usually desired to reduce the scan chain lengths by increasing the number of scan chains. Therefore, the proposed masking approach might require a significant number of additional AND-gates. To reduce the extra area caused by these gates an alternative approach as shown in Figure 4.2 can be applied.

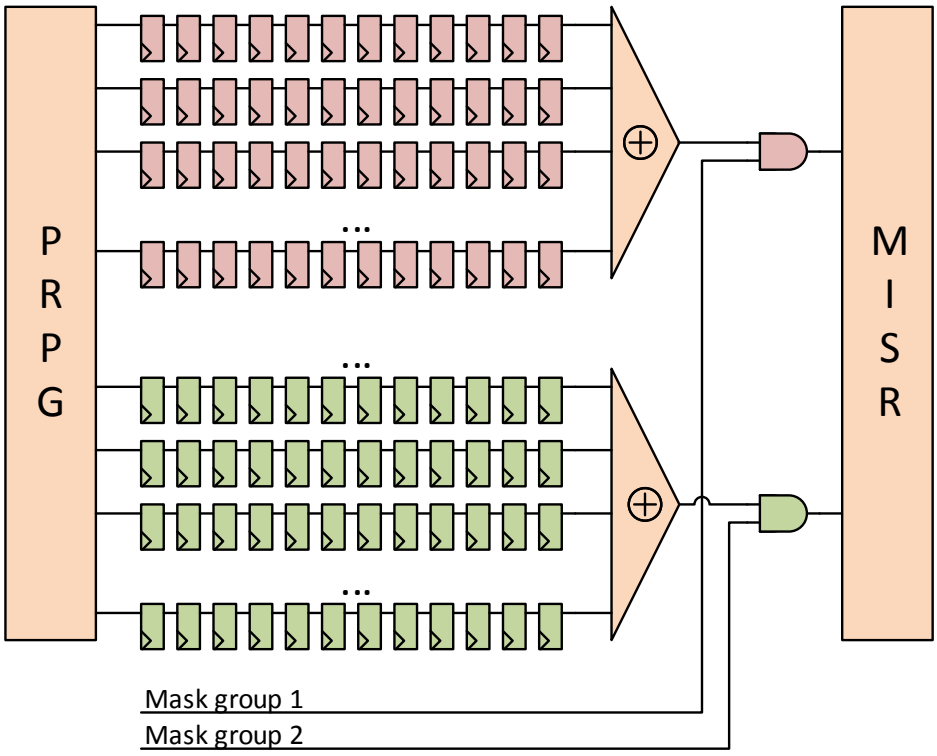

Figure 4.2: LBIST with channel group masking

Instead of masking the outputs of the scan chains the compactor outputs are masked before the data is propagated into the MISR. Even if this approach is less flexible and requires an appropriate grouping of the scan chains, it could reduce the AND-gate overhead. Of course the methods Figure 4.1 and Figure 4.2 can also be combined.

4.3.2 Evaluation of the Proposed Method

The hierarchical LBIST approach with the proposed masking schemes for the selective test execution has been successfully implemented and tried for a test case design. The circuit used for the evaluation is a small processor design with four CPU cores. At first a single CPU core has been implemented and equipped with the required design for test (DFT) features. The used LBIST controller has been generated together with EDT test data compression logic by using commercial "Tessent" DFT tools from Mentor Graphics [28]. The required clock control module has been developed in-house. It enables a flexible clocking scheme and takes care about the required switching between the slow scan shift clock and the at-speed test clock pulses. The insertion of the scan chains and the generation of the module isolation wrapper have been done using the commercial Synopsys synthesis environment [29]. After generating and verifying the ATPG patterns for the core and also performing the core-level LBIST fault simulations for the stuck-at and transition fault model, the CPU core has been instantiated four times in the top-level netlist. After synthesizing the top-level design an IJTAG network and TAP controller have been inserted in the gate-level netlist with the "Tessent IJTAG" tool [30].

The top-level test sequences for configuring and executing the four core-level LBIST controllers have been generated with the "Tessent IJTAG" tool based on the ICL and PDL description. At first the standard LBIST test patterns have been generated without masking any scan chains. Afterwards the test sequences with the selective

scan chain masking have been created. For the chain masking the available features of Mentor LBIST controller have been used, but also a manually designed custom implementation as shown in Figure 4.1. The control of the masking is performed by configuration registers as part of the IJTAG network. The proposed procedure of first running a normal LBIST followed by a selective test of some parts of the design has then been successfully verified in gate-level simulations.

4.4 LBIST Debugging Approaches

The LBIST execution is usually a pure go/no-go test with only very little diagnosis resolution and debugging capabilities. Due to the MISR used for the on-chip test data compression it is not easily possible to conclude what is the root cause of an observed fail. If the MISR value is directly compared on-chip versus the "golden" reference value and if only a pass/fail decision is provided as result of the test, further error localization is impossible. Even the final MISR signature can be streamed out of the device, the situation is not much better.

The design partitioning and selective scan masking described in the previous chapters allows at least a rough error localization. The design partitioning comes with the price of additional hardware for the module isolation and local LBIST and clock controllers, but it has no impact on test time when executing the different LBISTs in parallel. This approach is able to identify the defective test partition. The usability of this additional information, especially for the error handling, is highly dependent on the used partitioning criteria.

The proposed selective scan masking approach can be applied after the defective partition has been identified. This test would usually not be part of the standard test program or in-system test application. Instead it would be only executed, if additional debug information is required in case of a failure. In most cases this approach can be only used when operating the LBIST controlled by external test equipment. Otherwise it would be required to store the various MISR signatures on-chip and to control the selective test sequences by a chip internal algorithm reacting on the actual LBIST results. When controlled from external test equipment the memory requirements and the additional test time for the selective debug test are relaxed. When a test partition can be divided into n masking groups, in worst case the LBIST would need to be repeated n times and n MISR values would need to be compared to identify the failing scan group or scan chain.

For a detailed failure diagnosis the identification of the failing scan group would not be sufficient. In most cases it would be required to identify the failing scan registers. Such a detailed debugging would be usually performed in a two-step approach. At first the failing LBIST patterns need to be identified. This can be done by a kind of binary search approach. Instead of running the complete LBIST the test is only executed for half of the patterns. Depending on the test result it can be concluded if the first fail happens in the first or second half of the original LBIST sequence. Afterwards the identified faulty LBIST section is again checked in the middle and so forth. Such a debugging requires a LBIST controller being capable to execute a partial LBIST sequence from a given start pattern to a given end. In a second step after having identified which patterns of the LBIST are failing, the uncompressed content of the scan chains might be serially read out e.g. through the JTAG interface. For this purpose the internal scan chains could be concatenated forming a long single scan

chain. The first part of this debug approach is not critical for the tester requirements or test time. Here the main problem is that a real interactive debugging with a dynamic test program is required. The second part of the approach is more demanding for the test equipment as the amount of uncompressed scan data could be huge and the required tester memory for storing the expect data is limited. Again a dynamic test program would be required as it is not known upfront for which scan pattern the uncompressed data needs to be streamed out. To reduce the tester memory requirements this approach might be combined with the selective scan masking approach discussed before. After having identified the failing patterns the scan masking feature could be used to identify the failing scan group. In this case only the compressed scan data of the failing scan group would need to be shifted out.

Another debug approach is to run the LBIST for each scan pattern stand-alone and to always check the MISR signature. When having the actual MISR values for each LBIST pattern, some diagnosis would be possible with an appropriate diagnosis algorithm [31]. The main advantage is that such a debug test could be executed with a static test program independent of the failure. Also the tester memory requirements are acceptable as only the very compact MISR signatures need to be stored as expect values. The main drawback is that the diagnosis resolution might be limited and that the required tooling with the diagnosis algorithms needs to be available.

Not only the error localization is difficult when using a LBIST, but also the error classification capabilities are very limited. To decide if an observed failure is caused by a permanent fault the LBIST can be rerun and checked if the fail still happens. Distinguishing between static and dynamic faults is more complex. When using an at-speed LBIST for targeting delay defects, the same test is also used for the detection of static defects. For test time reasons there is usually no additional “slow” speed test. As the LBIST at-speed clock pulses are usually provided by an on-chip PLL, the test frequency cannot be changed easily. Modifying the frequency would usually mean to reconfigure the PLL settings. This is in most cases not possible in the system application, but could be allowed in a dedicated debug mode e.g. through an IJTAG network. As the functional PLLs are used for the overall IC design, it is not possible to adjust the test frequency for one LBIST partition without changing the clock for the remaining parts of the design. Instead of varying the test frequency it is often easier to change the clocking scheme used for the test. By adding an idle cycle between the launch and capture pulse and rerunning the test it can be checked if the LBIST still fails with reduced frequency.

5 Error Handling

In this section, we first detail the implementation of a health map and the error handling conducted by the instrument manager and the fault manager (Section 5.1), and secondly we detail the work on finding a metric to evaluate the quality of error handling (Section 5.2).

5.1 Fault Management and Health Map Implementation

The health map is a data structure in the system memory that holds the detailed information about the faults and fault statistics. The health map may be more detailed than in the resource map and may have more levels of hierarchy, to store the detailed diagnostic information. The structure of the health map is organized as a tree where

the children nodes are the sub-modules of modules represented by their parent nodes. This allows for coarse- and fine-grained health mapping.

The Health Map maintains the statistics of fault occurrence in a resource for a better reliability prediction capability. This information helps the operating system (OS) scheduler decide which resource is at the current moment more reliable and where the most critical task should be run. If an error threshold is set, then after a certain amount of faults within a certain timeframe the resource is marked as faulty or unreliable. In order to flexibly analyze the reliability of a resource across a wide range of environmental conditions, the health map may maintain separate records of the resource health status for each range of each environmental factor. To limit the size of the table, there should be a limited amount of ranges.

The Health Map is intended to contain information about the status of all system's resources and since faults are occurring in hardware and do not disappear after system restart, the health map also should not be lost. To retain the information about the known faults in the system when the system is powered off, the health map should be stored in a reliable non-volatile memory to maintain the prediction capability across the power cycles. However, the access to the memory is performed by the Fault Manager (FM) which is executed as process by the OS. To help protect the contents of this memory from error prone behavior of the OS, the error memory itself should not be directly accessible, but rather a special error-tolerant controller should be employed to control the access to the memory.

The resource map is also a data structure in the system memory, but it holds the information about the currently available (healthy) resources of the system. It should be modified on the fly during system's normal operation, should a fault be detected by an instrument or a diagnostic routine. The resource map operates with abstract modules which represent the actual resources in the system. These abstract modules do not contain specific information about the resource, besides its type (e.g. CPU core, FPU unit). An abstract module may have other sub-modules organized in a hierarchical manner. The type of module used by the scheduler to select the resources required by the task to be executed. Besides the health status information, the resource map is a good place to store the working characteristics of the resources, such as the power consumption or maximum clock frequency. The OS scheduler can then take into account these characteristics in order to perform e.g. power-aware scheduling. The initial state of the resource map (from factory, with all resources operational) can be stored in ROM and serve as a starting point for initialization during system start-up. During the initialization, information from the health map about the known resource faults is transferred to the resource map. The resource map is used by the OS scheduler to check which resources are available.

Role of Fault Manager (FM). FM is a part of OS (kernel) which is responsible for updating both health and resource maps. If a fault is detected in the system, FM must start a diagnostic procedure to find out the location of the fault as precisely as possible. This location information must be reflected in the health map by setting the fault flag for the appropriate resource and updating the fault statistics. In case if the resource usability is affected, FM must also update the resource map.

Role of Instrument Manager (IM). IM is a hardware module which is responsible for the communication with the instruments through IJTAG network. Whenever FM needs to access the instruments to get the diagnostic information, it gives a read/write command to IM which in turn opens the path to the instrument through the hierarchical IJTAG network and performs the requested operation. Besides instrument access, IM is responsible for reacting to the fault flags set by the instruments and propagated as an asynchronous interrupt signal. IM automatically opens the path to the instrument which raised the fault flag and provides the information about its location to FM or directly to the health map. However, IM can only provide coarse fault location information in this manner and FM should start the diagnostic procedure to find out the fine-grained fault location information and update the health and resource maps. Depending on the result of the diagnostic procedure, the error flag can be moved to the deeper level of hierarchy. Optionally, if the fault is critical and the resource cannot be used, the error flag is also retained at the top-level.

Task scheduling is performed by a modified OS scheduler which takes into account the information from the resource map. The OS scheduler is responsible for assigning the tasks to the appropriate resources. To do this, the scheduler must analyze the resource map and select the resources to be used for task execution based on 1) sub-resource availability (e.g. FPU inside a CPU), 2) information in the task descriptor, 3) reliability of the resource based on the fault statistics, 4) mission-criticality of the task. The task descriptor contains the information about resource requirements for the task execution. This is represented by a set of types of resources, optionally with some reliability or environmental requirements. Scheduler uses this information during task execution to select the available resources from the resource map.

5.2 Quality of Error Handling

The section details work on finding a metric to evaluate the quality of error handling. The section is organized as follows. First, the problem is stated. Then, preliminaries are given, followed by a problem formulation and a motivation. Finally, we briefly describe approaches to maximize LOC and experiments. The section is concluded with conclusions.

5.2.1 Problem

We address the following problem: given is a set of jobs where the objective is to find the optimal assignment of checkpoints for each job such that the LoC with respect to a global deadline is maximized. In particular, the contributions are as follows:

- We show that performing a local optimization for each job and combining these local optima together does not result in the maximal LoC with respect to the global deadline.
- We show that handling the set of jobs as a one single large job and obtaining the optimal number of checkpoints for the single large job does not result in the maximal LoC with respect to the global deadline.
- We provide, for a set of jobs, an expression to evaluate the LoC with respect to the global deadline.
- We show that a holistic solution (exhaustive search over all possible checkpoint assignments) is required to obtain the optimal checkpoint assignment that results in the maximal LoC.

- We propose a method (heuristic) to maximize the LoC and obtain the results in significantly shorter time com- pared to the exhaustive search method.
- We present experimental results to demonstrate that our method is capable of finding the optimal LoC while observing tremendous reduction in computation time compared to the exhaustive search method.

5.2.2 Preliminaries

In RRC, a job is duplicated and simultaneously executed on two processors. During the execution a number of checkpoints are taken. At each checkpoint the states of both processors are compared against each other. If the states match, the states are saved as a safe point from which a job can be restarted. If the states do not match, this indicates that an error has occurred in at least one of the processors and therefore, to handle the error, the job has to be restarted from the latest saved safe state. The main drawback of RRC is that it introduces time overhead due to checkpointing operations (comparing the states from both processors and storing\loading a safe state). We refer to this overhead as checkpointing overhead τ.

We define an execution segment to be the portion of a job's execution between two successive checkpoints. We assume that checkpoints are equally distributed, i.e. we assume an equidistant checkpointing scheme. In such scheme all execution segments are of same length T, where T denotes the nc processing (response) time of the job, i.e. the time required for a job to complete when no errors occur and RRC is not used, and nc denotes the number of checkpoints. Illustration of RRC, along with execution segments and checkpointing overhead, is presented in Figure 1.

First we present the assumed fault model and then we present the mathematical framework to evaluate, for a single job, the LoC with respect to a given deadline. For the fault model, we assume that occurrence of an error is an independent event. We assume that given is the probability that no errors occur in a processor within an interval equal to the processing time of the job T. Due to the fact that the occurrence of soft errors is an independent event, we calculate the probability that no errors occur within an execution segment in both processors.

Although errors can occur at any point in time, when RRC is used, errors are detected only at discrete moments in time, i.e. at each checkpoint. Detection of an error causes re-execution of the last executed execution segment. As all execution segments take the same amount of time when RRC is used, a job is expected to complete only at discrete equidistant moments in time.

Given the fault model, we derived a probability distribution function that evaluates the probability that a job that uses RRC completes at a certain time [19]. To evaluate the LoC with respect to a given deadline D, we calculate the probability that the job completes before the deadline. Therefore, the LoC is computed as a sum of intermediate terms from the probability distribution function.

Important to note is that the LoC depends on the number of checkpoints. A low number of checkpoints results in a poor LoC. Increasing the number of checkpoints, increases the LoC. However, at a certain number of check- points, an optimum is reached and further increase in the number of checkpoints will lead to decreased LoC, and may even result in a zero LoC. The reason is as follows. When the number of

checkpoints is low, the execution segments are longer. This limits the number of re-executions that can take place before the deadline. Hence, only few terms from the probability distribution function will be used to calculate the LoC, which results in a poor LoC. When the number of checkpoints is high, the execution segments become shorter, which in general allows more re-executions to be fitted before the deadline. However, the high number of checkpoints also increases the time overhead due to the check-pointing overhead, which may limit the number of re-executions that can be fitted before the deadline.

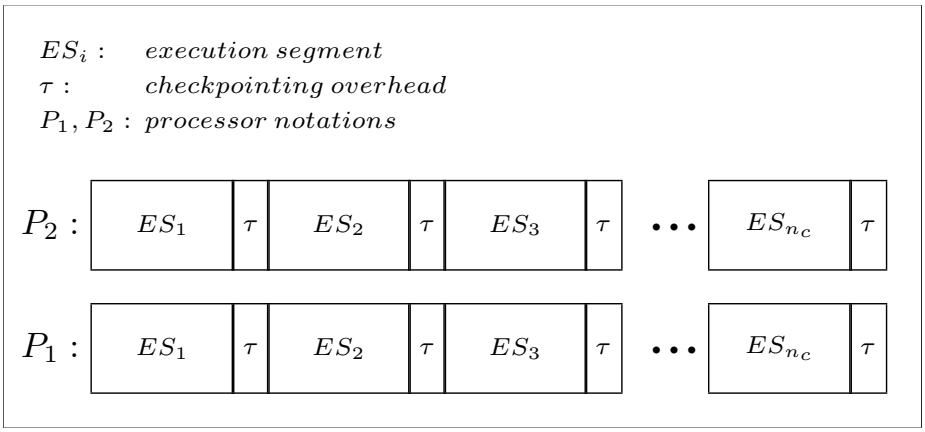

Figure 3. Graphical presentation of RRC scheme.

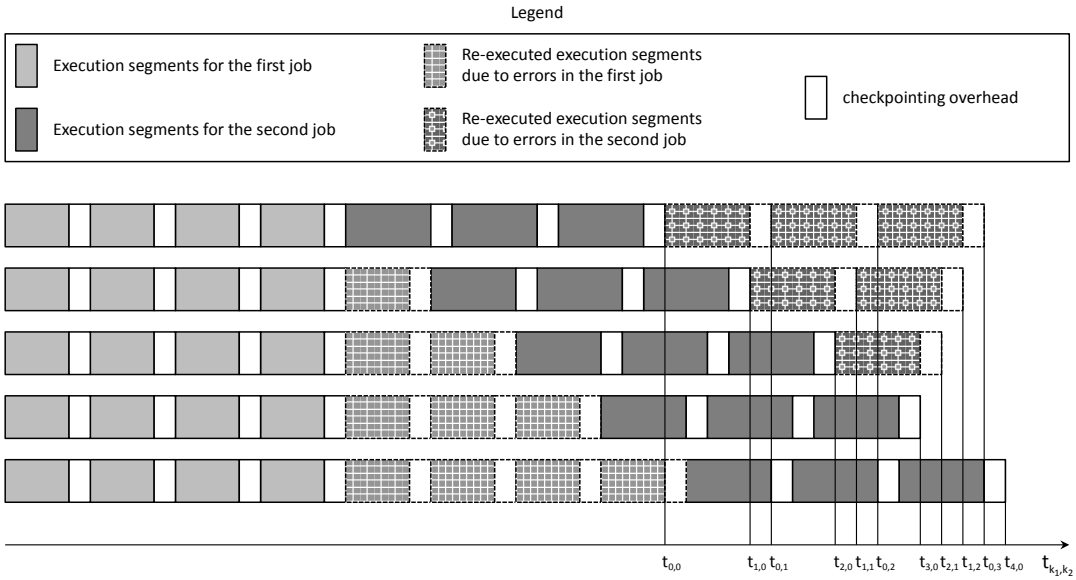

Figure 4. Execution of two jobs where termination time differs depending on the number of errors

5.2.3 Problem Formulation

We analyze the LoC with respect to a given global deadline for a set of jobs. We assume that the following inputs are given:

- a set of m jobs, where each job has a processing time T,
- a checkpointing overhead, τ,
- a global deadline, D, and
- a probability, PT , that no errors occur in a processor for an interval of time T.

The goal is to obtain the optimal checkpoint assignment that results in the maximal LoC with respect to the global deadline D.

5.2.4 Motivation

We investigate two approaches that directly apply the solution for a single job to solve the problem for a set of jobs. The approaches are:

- Local Optimization: perform local optimization for a single job and apply the results to all jobs, i.e. find the optimal number of checkpoints for one job, such that the LoC with respect to a local deadline is maximal, and apply this optimal number of checkpoints to all jobs.
- Single Large Job: assume that the set of jobs is equal to one single large job and find the optimal number of checkpoints for this job, such that the LoC with respect to the global deadline is maximal.

The advantage of the Single Large Job approach in comparison to the Local Optimization approach is that we are able to identify to optimal number of checkpoints without introducing local deadlines, which are not part of the original problem formulation. However, there are two main disadvantages. First, the optimal number of checkpoints for the single large job, does not necessarily need to be a multiple of the number of jobs, and thus we cannot guarantee that by adopting a checkpoint assignment such that each job provide the maximal LOC which we aim to find.

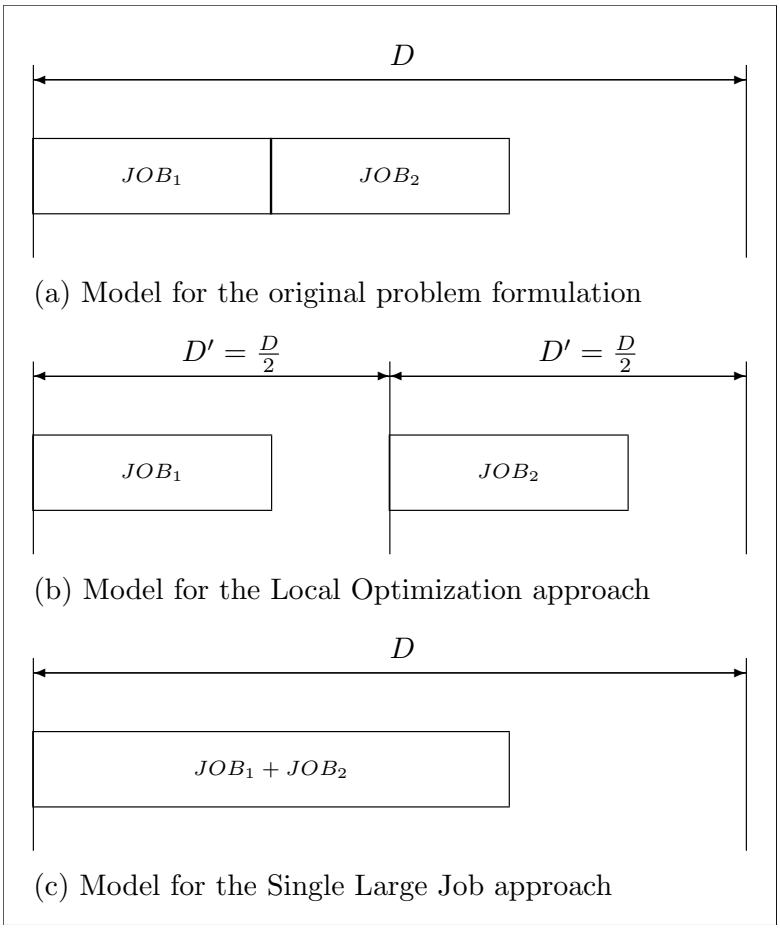

Figure 5. Illustration of models used for (a) original problem formulation, (b) Local Optimization and (c) Single Large Job approach

5.2.5 Approaches to Maximize LOC

We explored exhaustive search and semi-exhaustive search to find the optimal LOC. To minimize computational time, we make us of our first observation, which states a valid checkpoint assignment is a checkpoint assignment that does not violate the global deadline when no errors occur. In our exhaustive search, we only explore valid checkpoint assignments. During the development of the exhaustive search method, we learned that:

- Some checkpoint assignments are equivalent;
- Some checkpoint assignments do not contribute in finding the maximal LoC;

Based in these two observations, we developed a semi-exhaustive search method.

5.2.6 Experiments

The purpose of the experiments presented in this section is as follows. First, we show that both approaches, which directly apply the solution for a single job, i.e. Local Optimization and Single Large Job, in the general case, do not obtain the maximal LoC for a set of jobs. Second, we show the effectiveness of the proposed heuristic, i.e. Semi-Exhaustive Search, by showing that, in most cases, it finds the optimal checkpoint assignment and the maximal LoC. Third, we show the efficiency of the Semi-Exhaustive Search by comparing the computation time of this approach against the computation time of the Exhaustive Search approach.

We present results for the following two experiments:

- we compare the LoC obtained from the four approaches: Local Optimization, Single Large Job, Exhaustive Search, and Semi-Exhaustive Search;
- we compare the computation time of the Exhaustive Search approach against the proposed Semi-Exhaustive Search method.

Table 1. Input scenarios

Scenario	m	T	τ	D	P_T
A	2	1000	10	2800	0.99999
B	2	1000	10	2600	0.99999
C	3	1000	10	3900	0.99999
D	4	1000	10	5200	0.99999
E	5	1000	10	6500	0.99999

Table 2. Comparison of computation time for Exhaustive Search (ES) and Semi-Exhaustive Search (SES) approach.

Scenario	T_{ES}	$T_{S\,ES}$	Reduction ratio $\alpha = T_{ES} / T_{S\,ES}$
A	1965 ms	1162 ms	1.691
B	1396 ms	1023 ms	1.365
C	9131 ms	1906 ms	4.791
D	623710 ms	37918 ms	16.449
E	41380904 ms	2321786 ms	17.823

5.2.7 Conclusions

We addressed the following problem: given a set of jobs which employ RRC, and a global deadline, optimize RRC per job, i.e. obtain the optimal checkpoint assignment, with the goal to maximize the LoC with respect to the given global deadline. The contribution of this paper is three-fold.

First, we showed that the solution from our previous work, [19], cannot be directly applied as a solution to this problem. We have reviewed two approaches: Local Optimization and Single Large Job. Both approaches aim to solve this problem by directly applying the solution shown in [19]. For both approaches we demonstrated the disability to find the optimal checkpoint assignment that results in the maximal LoC.

Second, we have discussed the Exhaustive Search method, which is always able to find the optimal checkpoint assignment that results in the maximal LoC. The previous work, [19], is taken into account and a new expression for evaluation of LoC is derived. The new expression is able to calculate the LoC for a given checkpoint assignment. The method finds the optimal checkpoint assignment by exploring all valid checkpoint assignments.

Third, due to the fact that the Exhaustive Search method is computationally intensive and extremely time consuming, we have derived a method, i.e. Semi-Exhaustive Search, to maximize the LoC in significantly shorter time. We have shown experiments where we observe significant reduction in time when comparing the Semi-Exhaustive Search against the Exhaustive Search approach, and further show that the Semi-Exhaustive Search is able to obtain the same results as the Exhaustive Search approach, i.e. the optimal checkpoint assignment that results in the maximal LoC.

Further, we want to point out two important observations that come out from this work. First, even though we are targeting a relatively simple problem, i.e. identifying the optimal checkpoint assignment that results in the maximal LoC with respect to a

given global deadline for multiple jobs which have the same processing time, it turns out that this problem is not trivial and a previously developed method for identifying the optimal number of checkpoints for a single job is not sufficient and cannot be re-used in order to obtain the solution for the problem of multiple jobs. Out of this we conclude that by relaxing the assumption that all jobs have the same processing time would make the problem of identifying the optimal checkpoint assignment that maximizes the LoC with respect to a given global deadline even more complex, and as such we would address it in future work. Second, even though we target multiple jobs with the same processing time, the results have shown that the optimal checkpoint assignment may be asymmetrical, i.e. even though all the jobs are having the same processing time, not all the jobs use the same number of checkpoints. Important to note is that in both this work and our previous work we have considered the widely adopted equidistant checkpointing scheme, i.e. all the executions segments of a job are of the same size. However, unanswered is the question if the equidistant checkpointing scheme can always achieve the maximal LoC. This is an open problem and it will be addressed in our future work.

6 Summary

In this deliverable we have reported on the activities performed in T3.1 within the BASTION project. The activities on fault modeling, the use of LBIST methodologies for fault management, and quality of error handling are summarized below.

We studied different aging faults, and focused on NBTI-based faults, and simulation tools were developed for evaluating aging CMOS circuits. Several monitors for these were investigated to monitor the results of these faults (D1.1 & D2.1). Also NFFs were studied, and after the market survey, the attention was directed to IRFs. Simulation tools were developed to evaluate the susceptibility of digital CMOS circuits for IRFs. Several monitors for these were investigated to monitor the results of these faults (D1.1 & D2.1).

Design partitioning strategies have been analyzed to improve the diagnostic resolution of LBIST solutions for a better error localization and error handling. In addition LBIST debugging approaches have been evaluated considering the trade-off between accuracy and test execution effort.

To classify and handle faults during operation, we are developing a general methodology to handle and manage faults based on health map. Fault management is distributed across system levels from HW to operating system, which operates being aware of the current status of system integrity (health).

We addressed the following problem: given a set of jobs which employ RRC, and a global deadline, optimize RRC per job, i.e. obtain the optimal checkpoint assignment, with the goal to maximize the LoC with respect to the given global deadline. The contribution of this paper is three-fold. We found previous work cannot be directly applied as a solution to this problem. We discussed an exhaustive search method, which finds the optimal checkpoint assignment but is computationally intensive and extremely time consuming. We derived a method, i.e. Semi-Exhaustive Search, to maximize the LoC in significantly shorter time. We have shown experiments where we observe significant reduction in time when comparing the Semi-Exhaustive Search against the Exhaustive Search approach, and further show that the Semi-Exhaustive Search is able to obtain the same results as the Exhaustive Search approach, i.e. the optimal checkpoint assignment that results in the maximal LoC. Further, we want to point out two important observations that come out from this work. First, even though we are targeting a relatively simple problem, it turns out that this problem is not trivial and a previously developed method for identifying the optimal number of checkpoints for a single job is not sufficient and cannot be re-used. Second, even though we target multiple jobs with the same processing time, the results have shown that the optimal checkpoint assignment may be asymmetrical, i.e. even though all the jobs are having the same processing time, not all the jobs use the same number of checkpoints.

7 References

[1] H. G. KERKHOFF and H. EBRAHIMI, "INVESTIGATION OF INTERMITTENT FAULTS IN DIGITAL CMOS CIRCUITS," *JOURNAL OF CIRCUITS, SYSTEMS AND COMPUTERS,* 2015.

[2] J. LiU, Real-time systems, Prentice Hall PTR, 2000.

[3] C. Aubrun, D. Simon and Y. Q. Song, Co-design Approaches for Dependable Networked Control Systems, ISTE Wiley, 2010.

[4] C. Constantinescu, "Trends and challenges in vlsi circuit reliability," *IEEE Micro,* vol. 23, no. 4, p. 14–19, 2003.

[5] S. Hareland, J. Maiz, M. Alavi, K. Mistry, S. Walsta and C. Dai, "Impact of cmos process scaling and soi on the soft error rates of logic processes," in *IEEE Symposium on VLSI Technology, Digest of Technical Papers*, Kyoto, 2001.

[6] R. Baumann, "The impact of technology scaling on soft error rate performance and limits to the efficacy of error correction," in *International Electron Devices Meeting (IEDM'02)*, 2002.

[7] V. Chandra and R. Aitken, "Impact of technology and voltage scaling on the soft error susceptibility in nanoscale cmos," in *IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems (DFTVS'08)*, 2008.

[8] P. Shivakumar, M. Kistler, S. Keckler and D. Burger, "Modeling the effect of technology trends on the soft error rate of combinational logic," in *IEEE International Conference on Dependable Systems and Networks (DSN 2002)*, 2002.

[9] I. Koren and C. M. Krishna, Fault-Tolerant Systems, Morgan Kaufmann Publishers Inc., 2007.

[10] Y. Zhang and K. Chakrabarty, "Fault recovery based on checkpointing for hard real-time embedded systems," in *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, 2003.

[11] V. Izosimov, P. Pop, P. Eles and Z. Peng, "Scheduling of fault-tolerant embedded systems with soft and hard timing constraints," in *in Proceedings of the conference on Design, automation and test in Europe*, 2008.

[12] S. Punnekkat, A. Burns and R. Davis, "Analysis of checkpointing for real-time systems," *Real-Time Systems,* vol. 20, no. 1, p. 83–102, 2001.

[13] D. Nikolov, U. Ingelsson, E. Larsson and V. Singh, "Estimating error-probability and its application for optimizing roll-back recovery with checkpointing," in *Fifth IEEE International Symposium on Electronic Design, Test and Application*, 2010.

[14] D. Nikolov, M. Väyrynen, U. Ingelsson, E. Larsson and V. Singh, "Optimizing fault tolerance for multi-processor system-on-chip," in *Design and test technology for dependable systems-on-chip*, 2011, p. 66–91.

[15] M. Väyrynen, V. Singh and E. Larsson, "Fault-tolerant average execu- tion time optimization for general-purpose multi-processor system-on- chips," in *in Proceedings of the Conference on Design, Automation and Test in Europe*, 2009.

[16] A. Ziv and J. Bruck, "Analysis of checkpointing schemes with task duplication," *IEEE Transactions on Computers,* vol. 47, no. 2, p. 222–227, 1998.

[17] S. Nakagawa, S. Fukumoto and N. Ishii, "Optimal checkpointing inter- vals of

three error detection schemes by a double modular redundancy," *Mathematical and computer modelling,* vol. 38, no. 11, p. 1357–1363, 2003.

[18] K. Shin, T. Lin and Y. Lee, "Optimal checkpointing of real-time tasks," *IEEE Transactions on Computers,* vol. 100, no. 11, p. 1328–1341, 1987.

[19] D. Nikolov, U. Ingelsson, V. Singh and E. Larsson, "Evaluation of level of confidence and optimization of roll-back recovery with checkpointing for real-time systems," *Microelectronics Reliability,* vol. 54, no. 5, p. 1022–1049, 2014.

[20] M. Moras, J. Martin-Martinez, V. Velayudhan, R. Rodriguez, M. Nafria, X. Aymerich and E. Simoen, "Negative bias temperature instabilities in pMOSFETS: Ultrafast characterization and modelling," in *Conference on Electron Devices (CDE)*, 2015.

[21] "ATA/ATAPI Command Set (ATA8-ACS)," AT Attachment 8, ANSI INCITS, September 6, 2008.

[22] V. Agarwal, C. Bhattacharyya, T. Niranjan and S. Susarla, "Discovering Rules from Disk Events for Predicting Hard Drive Failures," in *International Conference on Machine Learning and Applications (ICMLA '09)*, 2009.

[23] P. Bardell, W. H. McKenney and J. Savir, Built-In Test for VLSI: Pseudorandom techniques, New York: John Wiley and Sons, 1987.

[24] J. Rajski, J. Tyszer, M. Kassab, N. Mukherjee, R. Thompson, K. H. Tsai, A. Hertwig, N. Tamarapalli, G. Mrugalski, G. Eide and J. Qian, "Embedded deterministic test for low cost manufacturing test," in *International Test Conference (ITC'02)*, 2002.

[25] J. K. Kim and B. K. Kim, "robabilistic schedulability analysis of harmonic multi-task systems with dual-modular temporal redundancy," *Real-Time Systems,* vol. 26, p. 199–222, 2004.

[26] S. W. Kwak and J. M. Yang, "Probabilistic optimisation of checkpoint intervals for real-time multi-tasks," *International Journal of Systems Science,* vol. 44, no. 4, p. 595–603, 2013.

[27] S. Kwak, B. Choi and B. Kim, "An optimal checkpointing-strategy for real-time control systems under transient faults," *IEEE Transactions on Reliability,* vol. 50, no. 3, p. 293–301, 2001.

[28] Mentor Graphics, "Tessent Shell Reference Manual," *Software Version 2015.3.*

[29] Synopsys, "DFT Compiler User Guide," *Version J-2014.09-SP5.*

[30] Mentor Graphics, "Tessent IJTAG User's Manual," *Software Version 2015.3.*

[31] W.-T. Cheng, M. Sharma, T. Rinderknecht, L. Lai and C. Hill, "Signature based diagnosis for logic BIST," in *International Test Conference (ITC'06)*, 2006.