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 Hierarchical Test and Fault Management Methodology and Field Learning (Deliverable D3.3)

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

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

Organisation name of lead contractor for this deliverable: Lund University

Revision 2.5

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 D3.3 described in contract 619871.

Table of Revisions

Version	Date	Description and reason	Author	Affected sections
1.0	June 3, 2016	Structure created	E. Larsson	Contents
1.5	July 22, 2016	Added section on hierarchical design and test	M. Beck, P. Engelke	Section 4
2.0	August 26, 2016	Section 6 added	M. Sonza Reorda, M. Jenihhin, J. Raik	Section 6
2.1	Sept 7, 2016	Fault Management sections added	A. Jutman, S. Devadze, K. Shibin	Section 5, Section 2.2, Section 3.2
2.2	Oct 18, 2016	Polishing	E. Larsson	All
2.3	Dec 16, 2016	Finalizing	E. Larsson	All
2.4	Dec 22, 2016	Update of Backgro-und, SOTA, Sect. 5	A. Jutman, E. Larsson	Sect. 2, 3, 5, Abbreviations
2.4a	Dec 23, 2016	Preparing for submission	A. Jutman	All sections
2.5	Jan 4, 2017	Updating KPIs	A. Jutman	Conclusions

Author, Beneficiary

Artur Jutman, Testonica Lab
Piet Engelke, Infineon
Matthias Beck, Infineon
Erik Larsson, University of Lund
Maksim Jenihhin, Tallinn University of Technology
Jaan Raik, Tallinn University of Technology
Matteo Sonza Reorda, Politecnico di Torino

Executive Summary

This document reports on the work in BASTION on hierarchical design and test, fault management, fault handling procedures, and a novel rejuvenation technique to mitigate in-field ageing faults.
The work is part of Task 3.3, where Testonica, Infineon, University of Lund, Tallinn University of Technology and Politecnico di Torino have contributed.

The work makes use of results from T3.1, T3.2, and T2.1 (fault classification, instrument design and network topologies), and the overall objective is to address the top-level management schemes to collect information from instruments, classify errors, perform appropriate action (e.g. ageing fault isolation/relaxation) and eventually create reports for NFF study. In particular, work has been performed to handle the increasing complexity with techniques for hierarchical design and test by partitioning the designs into smaller partitions. For fault management, a mathematical framework analyzing the effect on non-equidistant check-pointing. The results show a possibility to improve reliability. We also describe a general fault handling methodology and present a realistic fault management scenario. For rejuvenation, preliminary results gathered on a small but representative processor show that the method can achieve an improvement in the maximum delay increase over a 10 years period of 43%: this result seems promising, as it goes in the direction of allowing an extended circuit life time, with very limited impact on performance and no changes in the circuit structure.

The document is organized as follows. First we discuss background and the state-of-the art. Next, new contributions are described and the deliverable is concluded with a summary of the obtained results, and a list of references.

List of Abbreviations

AET	- Average Execution Time
ATPG	- Automatic Test Pattern Generation
BTI	- Bias Temperature Instability
BIST	- Built-In Self-Test
CC	- Clustered Checkpointing
CMOS	- Complementary Metal-Oxide Semiconductor
CPU	- Central Processing Unit, also Processor
CUT	- Circuit Under Test
D	- Deadline
DfT	- Design for Test
EDT	- Embedded Deterministic Test
EM	- Electro-migration
EQC	- Equidistant Checkpointing
EXS	- exhaustive search
FM	- Fault Manager
FPU	- Floating point unit
FP7	- European Union's 7^{th} 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
IO	- Input/Output
IP	- Intellectual Property
IRF	- Intermittent Resistive Faults
IT	- Information Technology
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
PBTI	- Positive Bias Temperature Instability
PDL	- Procedural Description Language

PLL	- Phase-Locked Loop
PMOS	- p-type Metal Oxide Semiconductor
PRPG	- Pseudo-Random Pattern Generator
RRC	- Rollback Recovery with Checkpointing
RM	- Rate-Monotonic
RSN	- Reconfigurable Scan Networks
RTL	- Register Transfer Level
RTS	- Real-Time System
SDF	- Standard Delay Format
SoC	- System on Chip
STUMPS	- Self-Test Using Multiple-input signature register and Parallel Shift-register sequence generator
TAP	- Test Access Port
TDDB	- Time Dependent Dielectric Breakdown
t.u.	- time unit
WCET	- Worst-Case Execution Time

Table of Contents

1 Introduction

This deliverable reports result performed by BASTION partners on hierarchical test and fault management methodology and field learning.

1.1 Structure of the Document

This report is structured as follows. In Section 2, we give the background on hierarchical design and test, fault management, and rejuvenation. In Section 3, we detail state of the art on hierarchical design and test, fault management, and rejuvenation. In Section 4 we describe our contributions on hierarchical design and test, in Section 5 we detail our contributions in respect to fault management, and in Section 6 we detail the contributions on rejuvenation. Section 7 summarizes the deliverable.

2 Background

In this section we give background information on hierarchical design and test (Section 2.1), fault management (Section 2.2) and rejuvenation (Section 2.3).

2.1 Hierarchical Design and Test

In this section, we discuss design partitioning, pattern retargeting, and test coverage.

2.1.1 Design Partitioning

For a hierarchical test the overall design has to be partitioned into multiple smaller pieces. Each partition needs to be isolated with dedicated design measures to allow a stand-alone test of the isolated hierarchy independent of the surrounding design modules. There are different aspects to be considered for the partitioning. The way how the design partitions are chosen has not only an impact on the testability and error handling capabilities, but affects also typical design parameters like area, timing and routability. In addition, the selection of the partitions has a direct impact on the design process regarding tool runtimes, memory requirements and setup effort. For the design partitioning strategy all the different aspects need to be considered and a trade-off fulfilling as many requirements as possible has to be found.

2.1.2 Pattern Retargeting

A hierarchical test approach consists of two major steps. At first the test content for the used test instruments needs to be defined and the module level test patterns need to be generated for the chosen design partitions. The functionality and test coverage of these module-level tests is verified considering the later-on design integration scheme. The second step is to transfer and map the module-level tests to the next level of hierarchy until the chip-level is reached. For IJTAG based test instruments this pattern retargeting is based on the ICL and PDL descriptions. For other test patterns like scan the pattern retargeting task mainly consists of mapping the used module level pins to the upper hierarchy. In case the number of available top-level pins is not sufficient to access all test partitions in parallel, a top-level test strategy is required to schedule the test execution. This test plan needs to be supported by related design measures multiplexing the different modules-level pins to the available test pads.

2.1.3 Test Coverage

When using a hierarchical test approach the tests are developed on module-level or for the stand-alone test instruments. On this level not only the tests are prepared, but also the test coverage is checked and debugged. When analyzing the test coverage on module level the impact of the later-on chip-level integration needs to be considered. Even if the modules are isolated and should be independent from the later-on surrounding system, the isolation itself needs to be taken into account for the test coverage evaluation.

For judging the overall test quality, the test coverage number for the complete design has to be looked at. Therefore, the calculated coverages of the modules need to be combined to get the overall chip-level value.

2.2 Fault Management

In this section we detail background in respect to fault management. A fault management solution needs to enable correct operation in presence of faults. Recent computer systems are increasingly susceptible to soft errors, which can cause incorrect operation. To maintain correct operation in the presence of soft errors, it is important to employ fault tolerance techniques to detect and recover from soft errors. However, these techniques often introduce a time overhead. For a vast group of computer systems, correct operation is defined as producing the correct response within given time constraints (deadlines). These computer systems are known as real-time systems (RTS). Employing fault tolerance techniques in RTS may result in deadline violations due to the introduced time overhead. Therefore, it is necessary to optimize the fault tolerance technique, such that the probability of meeting the deadlines is maximized. To measure the probability that a job competes before a given deadline, the statistical concept Level of Confidence (LoC) can be used. Thus, for RTS, it is important to optimize the fault tolerance technique with the goal to maximize the LoC with respect to a given deadline. Roll-back Recovery with Checkpointing (RRC) is well-known technique that copes with soft errors.

In RRC, a job is duplicated and simultaneously executed on two processing nodes. During the execution, a number of checkpoints (intermediate states of job's execution) are taken from both processing nodes and the checkpoints are compared against each other. If the checkpoints match, that is an indication that no errors have occurred since the previous checkpoint and therefore, the current checkpoints are saved as a safe state from which the job can resume its execution. If the checkpoints mismatch, that is an indication that errors have occurred in at least one of the processing nodes and therefore, the job is restarted from the latest saved checkpoint. While RRC enables correct operation in the presence of soft errors, it introduces time overhead due to taking, comparing and saving/loading checkpoints. The number of checkpoints used in RRC affects the completion time, i.e. the time required for the job to complete. If no errors occur, using less checkpoints reduces the completion time (completion time increases linearly with the number of checkpoints). However, if errors occur, using more checkpoints reduces the completion time due to the lower re-execution penalty (portion of a job between two successive checkpoints). This shows that the number of checkpoints should be carefully selected such that RRC is optimized. For RRC, different optimization goals exist.

Fault Management is also based on the ability of the system to collect low-level health status information about its building blocks that can be provided by various checkers, sensors and monitors. Rapid emergence of embedded instrumentation as an industrial paradigm and adoption of respective IEEE 1687

standard [1] by key players of semiconductor industry opens up new horizons in developing efficient test, debug and health monitoring frameworks. The IEEE Std 1687 also shortly called IJTAG has been initially started as an initiative to standardize access to on-chip embedded instrumentation, like monitors, sensors and checkers as well as DFT (Design-for-Testability) infrastructure, various BIST (Built-In Self-Test) and trace data collection solutions for system and software debug [2]. The IJTAG concept embraces also the paradigm of Reconfigurable Scan Networks (RSN) [3] and has become a very attractive industrial solution for both scan-based manufacturing test and system debug [2], [4], [5]. An architectural extension to IJTAG for system health-monitoring and Fault Management (FM) has been presented in BASTION D2.3. Instrument synchronization, calibration and triggering approached have been introduced in D2.1. The Health Map composition together with respective fault classification scheme, and initial fault handling scenario have been proposed in D3.1. In this deliverable we describe general fault handling methodology and present a realistic Fault Management scenario.

2.3 Rejuvenation

In this section, the background on rejuvenation is detailed. Lifetime reliability is a key challenge in current nanoscale semiconductor design and manufacturing processes. One of the most critical downsides of technology scaling beyond the 65nm node is the non-determinism of the devices' electrical parameters caused by time-dependent variations [6] in the operating characteristics of the device (*aging*). Bias Temperature Instability (BTI), and Hot Carrier Injection (HCI), which are identified as two essential sources of time-dependent variations [7], result in degradation of the oxide, thus causing a shift of the Threshold Voltage (V_{TH}) over time. In terms of magnitude, BTI has become the most prominent effect. It manifests in two distinct forms, depending on the type of transistor involved: Negative BTI (NBTI), which affects pMOS transistors, and its counterpart Positive BTI (PBTI), which affects nMOS devices. In current technologies, the impact of PBTI is much lower than NBTI. Therefore, we specifically focused first on the *Negative Bias Temperature Instability (NBTI)* phenomenon [8]. It is worth mentioning that the importance of PBTI is expected to increase, particularly with the adoption of high-k, Hafnium-based dielectrics in the gate-oxide for leakage reduction. [9].

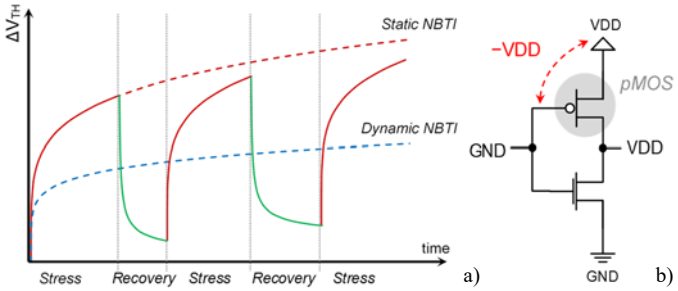

Figure 2.1 (a) Illustration of NBTI stress and recovery phases; (b) CMOS invertor gate under NBTI stress.

NBTI is defined as the effect that occurs when a pMOS transistor is negatively biased. The effect manifests itself as an increase of the pMOS transistor threshold voltage $|V_{THp}|$ over time (see Figure 2.1). This leads to drive current reduction and noise increase, which in turn causes an increase in the device delay. NBTI's impact on the long-term stability of functional logic is expressed through the incapability of storing a correct value in memory elements such as flip-flops. This effect is due to the

de-synchronization between clock distribution and signal propagation through the logic paths of a circuit. Therefore, after several years of circuit operation, the NBTI-induced delays may cause, first, intermittent faults and, ultimately, permanent functional failures in the circuit. [10].

Different from state-of-the-art, in our work a novel approach (based on hierarchical modeling) is used to calculate the gate and path delay degradations caused by NBTI aging. Moreover, we propose a method to generate suitable stimuli, able to slow down the typical NBTI effects, thus increasing the circuit life time. To the best of our knowledge, this is the first time that evolutionary algorithms are applied to the task of rejuvenation programs generation. The proposed methodology does not require redesign and can be applied to an existing circuit even already deployed in the field, exploiting, if necessary, the existing design-for-testability instruments.

3 State of the Art

In this section we describe state of the art in respect to hierarchical design and test (Section 3.1), fault management (Section 3.2) and rejuvenation (Section 3.3).

3.1 Hierarchical Design and Test

Hierarchical test is a well-established technique for core-based SoCs. In essence, a SoC is a composition of a set of cores implementing various functionalities that are integrated by adding some glue logic. Typically, designers of SoCs will try to maximize the reuse of existing cores to speed up the development process and to meet time-to-market targets. Consequently, cores are mostly adapted from previous design projects or are sourced from external vendors. Depending on their origin, cores are available on different levels of hardware description. In [11] three types of cores are differentiated: soft cores implemented on RTL, firm cores described as a gate-level netlist, and hard cores provided as a technology-dependent layout.

The mixture of soft, firm, and hard cores integrated in a SoC heavily influences the test architecture of that device. Hard cores, for example, require an isolation wrapper (e.g. IEEE1500) to be placed at the physical boundary of the core. This facilitates test access, and enables reuse of test patterns that were provided by the creator of the core. Thus, as already mentioned, for core-based SoCs, i.e. devices containing hard cores, hierarchical test is a natural choice. Numerous publications on test access, test generation, and test optimization for core-based SoCs are available. As a starting point refer to [11] [12] [13].

For a SoC that integrates soft and firm cores, however, the implementation of isolation wrappers is optional. From the perspective of test, these devices may simply be treated as a monolithic flat netlist. According to conventional wisdom, ATPG will actually produce the best test pattern sets when operating on such a flat top-level netlist. Recently it has been demonstrated in [14], that hierarchical test is beneficial for these monolithic SoCs as well.

Compared to core-based SoCs there is an additional degree of freedom when implementing hierarchical test in SoCs containing soft and firm cores. For soft cores the location of the isolation wrapper is no longer fixed at the physical boundary of a core. Rather the location of the wrapper may be optimized to reduce area, test time, test data volume, and to improve the at-speed test coverage at the core's interface (see [15]).

The problem of optimizing core wrappers, however, cannot be solved locally at core-level only. Rather it is tightly linked to the overall partitioning of the SoC, see [14]. As hierarchical test is now being extended beyond the scope of a single IC, the

partitioning of a SoC has to respect the additional requirements of embedded instrumentation and in-field test as well. In Chapter 4, we present a survey of all requirements affecting the partitioning of SoCs including those for embedded instrumentation and in-field test.

3.2 Fault Management

In this section, we focus on the usage of RRC as a major component in a fault management solution. Most of the research work on RRC focuses on average execution time (AET). Ziv et al. have presented a technique to analyze the AET of four different checkpointing schemes based on task duplication [16]. In a previous work, we have derived an expression to calculate the optimal number of checkpoints that results in the minimal AET [17]. Nakagawa et al. have analyzed the AET for three different checkpointing schemes based on double modular redundancy and derived analytical expressions for optimal checkpoint intervals [18]. Common assumption in these studies is that the checkpoints are evenly distributed throughout the execution of the job (equidistant checkpointing). Analyses on AET when checkpoints are not evenly distributed (nonequidistant checkpointing) are presented in [19] [20] [21]. Ziv et al. have proposed an on-line algorithm for checkpoint placement with the goal to minimize the AET, and they have shown that the proposed algorithm results in a lower AET when compared against an equidistant checkpointing scheme where fixed checkpointing intervals are used [21]. Shin et al. have analyzed the AET for a basic model (perfect error coverage) and an extended model (imperfect error coverage) [20]. They have shown that for the basic model equidistant checkpointing intervals minimize the AET. However, that is not necessarily the case for the extended model, where they minimize the AET under a reliability constraint. While achieving minimal AET is important, minimal AET does not provide any guarantees that deadlines in RTS are met.

Since for RTS it is crucial to meet the deadlines, some studies have analyzed RRC with respect to worst case execution time (WCET). Zhang et al. have proposed schedulability tests for RTS where RRC is used, assuming that the number of faults is bounded to a fixed number [22]. However, it is difficult to predict the number of errors that can occur within an interval of time. When RRC is employed in RTS, the number of checkpoints used affects the probability to meet the deadlines. To measure the probability that a job completes before a given deadline, in a previous work, we have used the statistical metric LoC [23]. In our previous work, we have derived an expression to calculate the LoC with respect to a given deadline, and proposed a method to obtain the optimal number of checkpoints that results in the maximal LoC [23]. Furthermore, we showed that the number of checkpoints that minimizes the AET results in a much lower LoC, with respect to a given deadline, compared to the maximal LoC that can be achieved, [23]. Kwak et al. derived a reliability equation over a mission time, i.e. the probability that the deadline is met, for a single real-time control task while using Markov models [24]. In another study, Kwak et al. discuss multiple real-time tasks and derive an explicit formula of the probability that all tasks are successfully completed with a given set of checkpoint intervals [25]. Using Markov model, Kwak et al. calculate the probability of task completion against faults that occur in a Poisson process for a checkpoint scheme [26]. The common assumption in all these studies is that the checkpoints are evenly distributed throughout the execution of the job, i.e. equidistant checkpointing scheme. Zhang et al. have presented an adaptive checkpointing scheme to achieve fault tolerance and power reduction in embedded RTS [27]. In [27], the authors showed that by using the

proposed adaptive checkpointing scheme, which in a general case can be considered as non-equidistant checkpointing scheme, the likelihood of timely task completion in the presence of faults is increased. Still, the results are obtained by simulation, and no mathematical framework is presented to calculate the probability to meet a given deadline when the checkpoints are not evenly distributed.

Another aspect of BASTION's fault management strategy relies on IEEE 1687 compliant instrumentation and scan networks as a health and diagnostic data collection infrastructure. IEEE 1687 has recently become a hot research topic, with rapidly growing amount of studies being published these days, hence facilitating the standard's wide industrial adoption. The research hype has been recently facilitated by a dedicated set of IEEE 1687 network benchmarks [28], [29] enabling experimental research and comparison of results across different teams. This benchmark set was developed by the BASTION consortium.

The research wave around IJTAG is led by studies addressing security aspects of test access infrastructure. A formal framework to verify whether an unauthorized access to target network segments is possible has been presented in [30]. Available access management methods either protect the access port [31], [32], [33], or particular instruments by protecting SIB access [34], [35], [36], [37]. The most advanced methods rely on cryptographic codes [32], [36] or LFSR [37]. Test and diagnostics of IJTAG and RSN infrastructure has also received a due attention (see [38] , [39], [40]).

Expected large-scale adoption of IEEE 1687 standard and respective infrastructure by chip vendors created an important opportunity to reuse IJTAG in the field. An extension to IJTAG for system health-monitoring and Fault Management (FM) has been proposed in [41] and [42] and further elaborated in [43], [44], [45], [46], [47], [48]. Recent works focusing on implementation challenges of on-chip IJTAG retargeting engines [49] and on-the-fly retargeting framework [50] also consider instrumentation reuse in the field. Reliability and fault tolerance of IJTAG networks during online FM operation has been detailed in [51].

3.3 Rejuvenation

Previous works appearing in the literature address the NBTI problem both for memories [52], [53] and for functional logic. Usually, to mitigate the impact of NBTI on the circuit's lifetime these approaches adopt *redesign* strategies, *voltage and frequency scaling* and *internal node control* guided by monitoring attributes or design structure analysis. The work in [54] proposes a *redesign approach* for functional logic *based on a transistor sizing technique* to mitigate NBTI-induced delay; the method requires the knowledge about the critical gates and paths to which it should be applied. Otherwise, this technique will result in an unacceptable area overhead and excessive power consumption. In [55] the authors present a method for *NBTI-aware synthesis* by characterizing the delay of every gate in a standard cell library as a function of the input signal probability (P_z). It demonstrates an average of 10% area recovery for 65nm technology under the pessimistic assumption that *all pMOS transistors in the design are under constant static NBTI stress*. Since the calculation process in [56] and [55] is based on electrical SPICE simulations, allowing the derivation of aging curves for each logic component, it is also prohibitively time consuming. This is due to an extremely large number of stress recovery cycles that have to be computed. There are works, e.g., [57], that propose an approach for temporarily hiding NBTI-induced aging by *changing voltage and frequency* of the circuit.

Approaches to analyze the efficiency of *controlling input signal probability* for mitigating NBTI at circuit level were proposed in [58], [59]. Works in [60], [61] propose to exploit the idle time of processors and unused bits in source operands [62]. A relevant approach for processor circuit aging reduction is presented in [63], where authors propose to replace the default NOP with a special "maximum aging reduction NOP instruction" that, while having no effect on the program state, minimizes the NBTI effect. The results show that this method can extend circuit lifetime by an average of 37%, with performance, power, and area overhead within 1%.

4 Hierarchical Design and Test

In this section, we will detail our contribution on hierarchical design and test in respect to design partitioning (Section 4.1), pattern retargeting (Section 4.2), and test coverage (Section 4.3).

4.1 Design Partitioning

For the hierarchical test the overall design is broken down into smaller parts that can be tested independently. The partitioning strategy of the design cannot focus only on the test and debug requirements, but needs to consider also all other design and implementation aspects. Usually the partitions that are chosen for the hierarchical test match the implementation hierarchies. In theory test and implementation hierarchies could differ, but this would significantly complicate the design process.

There are many aspects to be considered for the design partitioning. The main considerations are discussed in the next sections.

4.1.1 Number of Partitions

Before concentrating on the strategy how to partition the design, the first question is how many partitions should be used. There are different reasons to partition the overall chip-level design. One reason is the intended hierarchical test allowing the test of some components stand-alone independent from the surrounding system. Here the overall test strategy and requirements are important, e.g. are there some critical modules that need to be periodically checked in the system. Another motivation for the design partitioning is dealing with circuit complexity. With increasing design sizes today's implementation tools come to their limits from the run time point of view and from the required IT infrastructure e.g. the needed amount of memory. For huge designs it is no longer feasible to perform a flat top-down implementation approach considering the complete design in one step. Instead a modular approach is required. First the partitioned modules are implemented before they are integrated into the chip-level design.

Even if with an increased number of design partitions the implementation complexity issues can be solved and a fine granular test and debugging is possible, there are some drawbacks with the hierarchical implementation and test. The more partitions are used the higher is the setup effort for the configuration of the implementation and test tools. For each partition design and timing constraints need to be applied. For each partition also the intended functionality needs to be verified and for the test appropriate test patterns or test sequences need to be generated. The number of design partitions does not only affect the setup effort, but also influences the implementation results. For the physical implementation optimizations can no longer be performed over the partition boundaries. This might lead to some area or timing penalties at the module boundaries. For the structural test the scan chain lengths might differ for the different modules and there might be some test coverage issues introduced by the module level test isolation. In general, the hierarchical design and test approach might lead to some non-optimal solutions at the module boundaries.

As a result, the number of design partitions has to be a trade-off between the desired or required partition needs and the related setup and implementation drawbacks. A reasonable number of partitions might be in the range of e.g. ten separate design modules. All the partitions should have comparable sizes to benefit most from the complexity reduction.

4.1.2 Interfaces

One main aspect for the design partitioning are the interfaces of the chosen design partitions. Depending on where and how the functionality is split into separate design partitions, the resulting interfaces between the hierarchical modules contain more or less interface signals. From the testability point of view, it is desirable to minimize the number of interface signals. Each interface pin requires some kind of test isolation [64]. The isolation effort and the number of scan wrapper cells used for the module isolation increase with the number of interface pins. The number of required scan wrapper cells determines the length or amount of the isolation wrapper scan chains. Especially for the interface test on top-level it is beneficial to have only a small amount of wrapper cells per module, as this simplifies the top-level scan configuration (see Section 4.2.3.5). In case dedicated isolation cells are used the related area overhead is another reason to minimize the number of interface pins.

From the design implementation point of view, a limited number of module interface pins is also helpful as this reduces the top-level routing between the design partitions. On the other hand, the module interfaces cannot only be defined based on test and implementation demands. For the module-level verification it is required to have interfaces that are reasonable from the functional point of view. Splitting tightly linked functionality over the hierarchy boundaries would complicate the module verification and debugging.

Another interface aspect is the amount of combinational logic related to the interface pins. For the test and its related module isolation it is preferable to have registered input and output pins. This eases sharing of functional registers for the isolation cells and reduces the module-level test coverage impact caused by the isolation wrappers (see also Section 4.3.2). Even if registered module pins also significantly ease meeting the top-level timing constraints, they are not always desired from the architectural point of view. The output register of one partition followed by the input register of the next partition build a kind of pipeline stage requiring an extra clock cycle to propagate data through the interface. This extra latency is not always acceptable for timing critical interfaces as it might reduce the overall system performance.

4.1.3 Clock domains

Another important aspect for the design partitioning strategy is the number of functional clock domains per partition. Ideally each hierarchical module should use only one functional clock frequency. While different asynchronous clock frequencies should be prevented wherever possible, using one master clock and synchronous divided clocks per hierarchy might be still acceptable without major drawbacks.

One reason for minimizing the number of clocks is the local clock control logic required in each design partition. The hierarchical test approach requires for both LBIST and for the productive scan test patterns that the generated test clock sequences are self-contained within the tested module. At the boundary of the test partition usually a (slow) always running test clock is provided. This clock is used as the source for the scan shift operation and the slow-speed capture for static fault models. For the delay test additional (fast) at-speed clock sources are required, which are usually provided by on-chip PLLs. Derived from the provided clock source the clock control logic provides the desired sequence of clock pulses in the capture phase. For the delay test the clock control logic also needs to handle the clock switching from the test clock used during scan shift to the at-speed clock source used during scan capture. Such a clock control block is required for each (asynchronous) clock

domain. Therefore, the area overhead and implementation effort increases with the number of clocks used in a partition.

Another reason for minimizing the number of clocks per test partition is the handling of timing exceptions. In synchronous designs usually all data needs to propagate through the combinational logic within one clock cycle before it is latched into the next register stage. For some exceptionally slow paths it might be sufficient that the data is valid after two or more clock periods. Such paths are considered as a so-called “multi-cycle paths”. For so-called “false paths” there might be no timing constraints at all. For a LBIST targeting delay defects these timing exceptions cause problems. For the detection of any timing related defects the clock used in the LBIST capture phase has to be an at-speed clock with the functional clock frequency. Transitions exercised on these exceptional timing paths would result in an unpredictable value being captured at the receiving registers as they are operated faster than expected. For a MISR that is typically used to compress the test responses of a LBIST operation, it is essential that only predictable values are captured. A single unknown value compressed into the MISR would corrupt the final signature and prevent the LBIST operation. Therefore, all timing exception paths require special handling to prevent unpredictable data from reaching the MISR. For a LBIST based on pseudo-random test stimuli these timing paths cannot be blocked by constraints in the patterns. Instead the blocking needs to be done in hardware. The paths with unreliable timing could be blocked e.g. by multiplexers selecting a different data source in LBIST mode. The handling of these timing exceptions is one of the major efforts for implementing a LBIST targeting delay defects.

Also for the (deterministic) production test patterns these timing exceptions have some impact. They do not prevent testing, but if on-chip test data compression techniques are used, unknown values reduce the effective compression rate. As most of these timing exceptions are usually defined between different (synchronous) clock domains, a partitioning minimizing the clock domains per test hierarchy helps to reduce the number of timing exception within a test hierarchy.

4.1.4 Power Domains and Test Power Consumption

Power domains are another aspect to be considered for the design partitioning. It is recommended to minimize the number of power domains per test hierarchy. When using very few or just one power domain per test partition, this can help dealing with excessive test power consumption.

Especially for scan-based structural tests the power consumption can be significantly higher than in functional operation mode. Clock gating cells, which are used to save power in functional mode, have to be transparent during the scan shift phase. This means while shifting data through the scan chains each scan register is clocked. For the LBIST operation there is also a very high switching activity for the data on the scan chains as usually pseudo-random stimuli are used. A comparable scan switching activity is also seen for normal pattern-based tests using compressed scan test data. This extraordinary high switching activity can lead to power related instabilities or even prevent the intended test execution.

The hierarchical test approach provides new possibilities to address the high test activity. Instead of operating all test partitions in parallel, a sequential test approach can reduce the overall chip-level test power consumption. While certain test hierarchies are tested the other partitions can be kept idle. Especially when the partitions are separated based on power domains, it is possible to completely switch off power domains that are not required for a certain test scenario. Such a power

domain based test can reduce the overall test power consumption, but it does not solve local power issues e.g. voltage drops caused by too high switching activity within a test partition.

Another advantage of using a partitioning based on power domains is that the required power isolation might be reused for the test. At the boundary of a power domain dedicated power isolations cells are required to ensure a known output values when the power domain is switched off. These isolation cells could be also beneficial for the hierarchical test providing some module isolation capabilities.

4.1.5 Error Localization and Error Handling

For the in-system test which is performed especially in safety critical applications it is often not sufficient to detect a failure, but it is also necessary to know which component has failed. This basic error localization is required for judging the severity of the detected error and for initiating the appropriate error handling actions. The hierarchical test by itself directly provides the simple diagnostic information which test partition has failed. The error localization capabilities increase with the number of test partitions, but for the reasons described in Section 4.1.1 it is not recommended to split the design into a large number of very small partitions. Measures to further improve the error localization within a test partition have been discussed in [65].

Even if the number of test partitions is limited, the error handling capabilities can be improved when they are already considered during the design partitioning. One approach is to partition the design by clearly separating safety critical components from less critical modules. This helps to considerer the safety risk for the error handling in case a test is failing in the system. In addition, this partitioning allows having more and stricter tests for the safety critical modules. Another approach is to partition the design based on functional importance. Based on the failing component it might be acceptable to switch off the defective part accepting some performance loss or limited functionality. When the error handling strategy contains fallback mechanisms like rescheduling of tasks or the replacement of faulty components by spare parts, it is essential that the test partitioning allows the identification of the defective modules.

4.1.6 Optimization Possibilities

The design partitioning should consider potential optimizations possibilities. Often designs contain some functional modules that are used multiple times. In this case it is advisable to partition the design in a way that such a functional module is mapped to its own design hierarchy which is then instantiated multiple times. Using the same design partition multiple times in the overall chip has several benefits.

First of all, the design effort is reduced as the module has to be implemented and verified only once. The same is true for the test related tasks like LBIST fault simulation, scan test pattern generation and verification. For the scan test patterns there is another significant advantage if the same test hierarchy is instantiated multiple times. In this case each instance requires exactly the same test stimuli. Therefore, the input data can be shared between the instances reducing the scan data volume.

4.1.7 Other Partitioning Aspects

There are some other aspects to be taken into account for the circuit partitioning. One consideration is IP reuse. In some cases, one IP block might be used for multiple products. In this case it is beneficial to use a dedicated design partition for this module. The advantage is that this block needs to be implemented only once and can

be directly reused as is in the next product. The same is true for the test data. Module-level test patterns or test sequences can be reused without changes in a later design. This significantly reduces the generation and verification effort.

Another point to be considered for the partitioning is the organizational design project setup. The development of complex designs is often spread over different development sites. Different design teams provide different components for the overall product. This distributed development implies some natural partitioning based on how the work is split.

4.1.8 Summary

There are many different aspects to be considered for the design partitioning (see Figure 4.4.1). In some cases, the different partitioning goals are conflicting. For some of the parameters also the design implementation and test partitioning goals might be contradicting. Therefore, there is not one optimal solution to partition the design. Instead a trade-off is required taking into account all the different aspects and trying to achieve the goals for most of them while not violating any constraints in an unacceptable way.

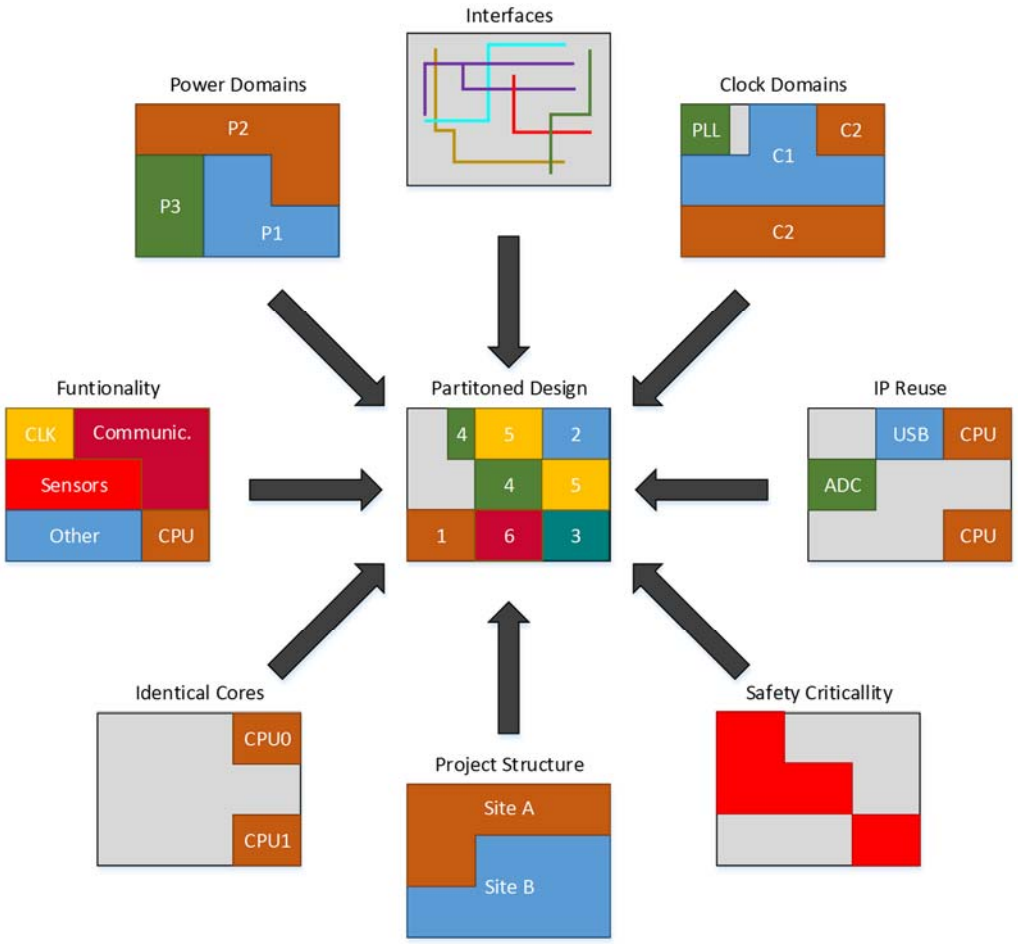

Figure 4.4.1: Design partitioning aspects

For some partitioning decisions the impact on the implementation is not always immediately visible. If there are some unforeseen implementation problems (e.g. the timing cannot be met, routing is not possible due to massive congestions), it might be required to modify some partitions in the design phase to solve the issue.

To allow a flexible partitioning a script environment has been developed that automates the design partitioning on RTL level. After assigning the functional modules to a certain partition the tool generates the RTL descriptions for the partition entities and creates the required connections.

Today it is still a manual decision how to partition the design. In future it would be desirable to automate this step or to have at least a tool supported decision. Attributes for the functional modules could contain information about the partition aspects (e.g. used clock domains, power domains). Based on some weighting functions and some constraints (e.g. maximum number of partitions) a reasonable partition scheme could be generated.

4.2 Pattern Retargeting

Pattern retargeting describes the task of propagating the test content from the embedded test instruments up the hierarchies to the level where the test should be executed. This is usually the chip-level, but pattern retargeting could also be used to propagate tests further to the board-level or even system-level.

For pattern retargeting different kinds of tests need to be considered. In Section 4.2.1 the retargeting of test sequences of IJTAG controlled test instruments is described. Here a special focus is on a dynamic retargeting approach. Another typical area of application for pattern retargeting is the structural scan based test for digital logic. The general mapping of module-level scan test patterns to higher hierarchy levels is discussed in Section 4.2.2. A related special topic is the top-level scan test strategy addressed in Section 4.2.3.

4.2.1 Test control sequences (IJTAG) and Dynamic Retargeting

Pattern retargeting for IJTAG-based test instruments is based on the two standardized descriptions languages “Instrument Connectivity Language” (ICL) and “Procedural Description Language” (PDL). The ICL contains the structural information about the implemented test features, the used pins and their connection. The PDL describes the test procedures supported by a test instrument.

The pattern retargeting consists of two steps. First the ICL network for the top-level needs to be generated or extracted. If the IJTAG infrastructure has been inserted by an IJTAG tool, usually this tool also generates the top-level ICL description. In case the IJTAG connections have been implemented manually as part of the design process the top-level ICL needs to be extracted. The extraction is based on structural tracing of the test signals in the design. Based on the ICL containing the information how the test instruments are connected into the design the pattern retargeting can be performed. The test sequences defined in the PDL need to be mapped from the instrument-level through the IJTAG network to the top-level.

Usually during the retargeting process just the available procedures of the PDL are called using some parameter values if required. The result is the generated test sequence that can be applied from the tester. In some cases, the generation of these onetime generated “static” test sequences is not sufficient. Instead some “dynamic” retargeting is required to actively react on some test results. This means the test sequence cannot be generated upfront prior to the test, but needs to be adjusted dynamically during the test execution.

An example for a test requiring such a dynamic retargeting is the debugging of LBIST failures. The test sequence for the initial pass/fail LBIST operation can be retargeted from the PDL as described above. For this static go/no-go test the LBIST can be set up using only few parameters (e.g. the pattern count). Once this standard

LBIST has been executed and a failure has been detected by a non-matching final MISR signature the debug process for identifying the root cause can be started.
The first debug phase needs to identify the failing LBIST patterns. For this purpose still a static test could be used. Instead of running the complete LBIST and checking the MISR signature at the very end, for debugging each LBIST pattern could be executed stand-alone. This requires that for each pattern the pseudo-random pattern generator (PRPG) can be initialized with the pattern-specific seed. The same is necessary for the MISR which needs to be initialized with the golden signature that would have resulted from the fault-free LBIST run up to the current pattern. By executing each LBIST pattern separately and by checking each resulting MISR signature, all failing LBIST patterns can be identified.

Alternatively, the first failing LBIST pattern could be identified by a dynamic procedure using a binary search algorithm. Instead of executing the complete LBIST the MISR signature would be checked after half of the patterns. Depending on this MISR signature it can be concluded if the first fail occurs in the first or second half of the complete LBIST operation. Based on this test result the LBIST would be run for the failing half again checking the MISR value after half of the remaining patterns. This search can be continued until the first failing LBIST pattern is identified. As the next step of this search algorithm is based on the previous test result, the pattern retargeting process providing the required test sequences needs to be configured dynamically. This means the overall debug sequence cannot be generated upfront, but needs to be created when executing the test. As different devices usually would show different fails, the required debug sequence would be device-specific.

While the identification of the failing patterns could be done using a static or dynamic pattern retargeting approach, the next step of the debug process always requires dynamic retargeting. After the failing LBIST patterns have been identified, additional debug information is required for finding the root cause of the failing behavior. One approach is to stream out the uncompressed data of the failing patterns. This means the failing LBIST pattern is executed by loading the scan chains from the PRPG as usual, but after the capture phase the scan architecture is reconfigured forming long traditional scan chains and the data is shifted out through available top-level pins bypassing the MISR. With this approach the internal registers capturing the wrong data can be found without any uncertainty allowing an analysis of the root cause by traditional scan diagnosis methods. As it is unknown in advance for which LBIST patterns these kind of special diagnosis sequences are required, these patterns need to be dynamically created. This means the retargeting process calling the debug procedure described in the PDL can only be started after the failing LBIST patterns have been identified. It is usually not feasible to store the debug sequences for all LBIST patterns upfront in the tester memory as the data volume of the uncompressed shift-out values would be very large.

4.2.2 ATPG Test Patterns

Scan pattern retargeting translates the module-level test patterns to the chip-level design for the later-on execution at the tester. The scan pattern retargeting does not involve any ATPG. In its simplest form it just maps the patterns from module-level to chip-level pins.

Pattern retargeting needs to be considered already during the initial test pattern generation on module-level. The module-level test patterns have to be independent from the design integration. Therefore, an isolation wrapper is required at the module boundary breaking the data paths from and to the surrounding system [64]. While this

pin isolation has to be available for all functional data pins, there are some test control signals that have to be directly controllable without any isolation. Such non-isolated test control signals are the test clocks and the scan enable signal (used for switching between scan shift and functional capture). In addition, there are usually some test mode pins for configuring the different test modes. These pins either control the test modes directly or provide access to some serial test data registers. After the test mode is configured in an initialization phase these pins are usually kept constant while the test is executed.

Due to the limited availability of package pins the test control signals are usually shared for the chip-level test pattern retargeting. This means all test partitions that should be tested in parallel are using the same scan control signals resulting in a synchronized test execution. A shared scan enable signal leads to a common scan shift phase even for the case of different scan chain lengths used in different test partitions. For the modules with shorter scan chains some over-shift cycles are added during the pattern retargeting process to balance the shift phase between different test partitions. To minimize these over-shift cycles and to improve the overall efficiency of the retargeted scan patterns the scan chains of parallel test modules should be balanced using comparable lengths. Even if individual scan control pins would be available, today's retargeting tools could not make use of them and they would still align the module level patterns during the retargeting step.

Similar tool limitations exist for the used scan clocks. Today's ATPG tools allow only a fixed clocking sequence for the module-level capture phases. This means even if the module-level scan clock would be directly controllable by a chip-level pin, the ATPG tool would not support different clocking schemes for different capture phases. Therefore, already for the stuck-at pattern generation usually a module-internal clock control circuit is required to enable different clocking schemes for different patterns.

For the delay test the situation is even more complex. The capture clocking scheme is usually configured by scan flip-flops. For the module-level ATPG run these configuration registers need to be part of the module-level scan chains. It is not possible to do some dynamic clock switching based on configuration registers outside of the test partition. All configuration bits need to be part of the module-level pattern set.

The pattern retargeting process for a hierarchical scan test flow consists mainly of mapping the scan test data used at the module-level boundary to the chip-levels pins. At first a chip-level test setup sequence needs to be provided setting all the used test partitions into the desired test mode. In addition, this initialization sequence needs to multiplex the module-level scan chains to chip-level pads. Before converting the module-level test patterns to the chip-level, the correct circuit initialization has to be verified. This is done by checking the structural connection of module-level-scan pins to the chip-level pads and by verifying if module level pin constraints assumed during the module-level pattern generation are met. Afterwards the module-level test data is translated to the chip-level. The patterns of the different active test partitions are aligned during this process as described before. For the pattern retargeting process only the module-level test patterns and the chip-level netlist describing how the test partitions are integrated into the chip are required. A complete netlist description of the tested partitions is not needed. Therefore, simplified abstraction models can be used for the modules to reduce the memory consumption of the retargeting process. After the scan patterns have been retargeted they need to be verified.

The pattern verification and debugging is usually based on pattern simulation taking into account the SDF timing information. The goal for the hierarchical scan test approach is to perform most of the pattern verification already on module-level. This does not only reduce the tool runtimes and memory requirements, but this approach also allows performing the pattern verification task earlier with shorter iteration times. In addition, module-level debugging is in general easier due to the reduced design complexity. The module-level pattern simulations can be performed and debugged as usual. There are no restrictions and limitations. On chip-level a full verification is only required for the patterns testing the logic between the test partitions. In addition, a quick check of the module-level patterns that were retargeted to the chip-level is needed. The main purpose of this check is to verify the retargeting step and the correctness of assumptions used during module-level ATPG. Therefore, the verification concentrates on aspects not visible already on module-level, e.g. chip-level test setup sequence, configuration of chip-level clock sources and the connections including timing from the module pins to the chip-level pads. The module-internal capture does not need to be verified completely again.

4.2.3 Top-Level Scan Test Strategy

For a traditional non-hierarchical scan test the definition of the scan test architecture is usually not very complicated. The key parameter is the number of pins available for the test. This number is usually defined by the given design package. In some cases, the number of used test pins is artificially limited to increase the number of devices tested in parallel at the tester. Based on the number of available test pins the number of scan channels can be defined. As usually some on-chip test data compression is used, the number of internal scan chains is by a factor higher than the number of external scan channels determined by the used compression ratio. This means after defining a reasonable compression rate, the number of internal scan chains can be directly derived from the given test pins. When using a flat top-level scan insertion approach the implementation tool ensures balanced scan chains all having roughly the same length. The length of the resulting scan chains can be estimated early in the design phase by dividing the number of expected registers by the number of planned scan chains. For the test pattern generation nothing special needs to be considered. Only one single ATPG run is required (per fault model) to generate the scan test patterns for the complete design. When using test data compression the stimuli provided through the external scan channels can be mapped to any internal scan chain. The ATPG tool takes care about this data distribution and ensures a good utilization of the test interface bandwidth.

For a hierarchical test approach the planning of the scan test architecture and the test scheduling is more complex. Here two different test types need to be considered. At first the module-level test patterns need to be retargeted to the top-level. In addition, a test pattern needs to be generated on top-level for testing the logic and interconnects between the different test partitions. When planning the scan test architecture for the test partitions not only the number of available top-level pins needs to be considered, but also the number of test partitions. Each partition requires at least one scan channel. This means for a design with ten test partitions at least 20 scan test data pins (10 scan channel inputs and 10 scan channel outputs) are required for testing all hierarchies in parallel. For certain products this might already exceed the number of available package pins. Therefore, a parallel test would not be possible and the partitions could only be tested (in groups) sequentially. As a consequence, the top-level test schedule already needs to be considered when planning the module-level

scan test architecture. This is completely different from the traditional flat scan test approach. In that approach always the complete design can be tested in parallel. Even if only very few pins are available, it would be possible to increase the test data compression rate and/or scan chain length to enable the parallel test.

Different test strategies for the retargeting of the module-level test patterns are described in the next sections.

4.2.3.1 Parallel Pattern Retargeting Approach

The parallel test approach combines the module-level test patterns of all test partitions and retargets them into a single common top-level test. All the modules are tested in parallel. This retargeting approach requires that the scan pins of all partitions are accessible from the top-level in one common test mode. For the planning of the module-level scan test architectures the available top-level pins need to be assigned and distributed to the different test partitions. As stated before each partition requires at least one scan channel. As described in Section 4.2.2 the lengths of the scan chains in the partitions should be comparable to prevent inefficiencies when the pattern retargeting process aligns the shift operation for the partitions. Therefore, the pin assignment for the modules has to consider the number of scan registers in the partitions to get comparable scan chains lengths. As long as the number of available top-level pins is sufficiently large to allow multiple scan channels being used for each test partition, a reasonable scan chain balancing between the modules should be possible. If the expected scan chain lengths differ too much, some fine tuning is possible by varying the compression rate between test partitions.

The parallel test scenario for an example design with five test partitions of comparable size and ten usable scan channels on the top-level is shown in Figure 4.4.2. All five partitions are tested at the same time each using two scan channels.

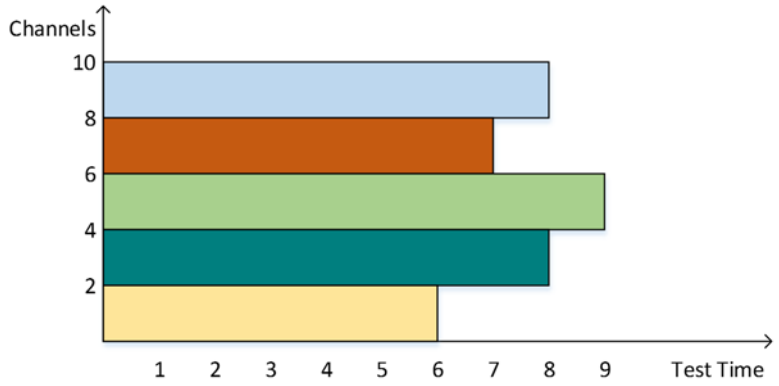

Figure 4.4.2: Parallel test approach

The main advantage of this test strategy is the simple setup und design implementation. Only a single pattern retargeting task needs to be performed (per fault model). As there is only a single scan test retargeting test mode, there is no need for implementing some multiplexing logic for switching between different test modes. Another positive aspect is the comparatively small number of scan channels used per partition. This eases the reuse of the IP in other designs as there is only a small pin requirement for the top-level integration.

The main drawback of this approach is the large number of required top-level pins. Not only due to the scan chain balancing aspect this approach is only recommended if for each partition several scan channels can be used. Using scan test data compression with a single data input channel often leads to some test coverage

loss even for reasonable compression rates. To circumvent these data compression limitations and to ensure the maximum possible test coverage, it is advisable to have the possibility to use more than one scan input channel per test partition (see also Section 4.2.3.4).

The assignment of top-level pins to dedicated test partitions can lead to some other inefficiency in case the partitions vary significantly in the test pattern count. If the number of test patterns for partition A is lower than that for partition B, top-level pins dedicated to A cannot be reassigned to B without changing the test configuration (see also Section 4.2.3.4).

Another potential drawback of the parallel test is the test power consumption. As all partitions are simultaneously active during the scan test this could lead to power related instabilities as described in Section 4.1.4.

Positive	Negative
Single test configuration	Many top-level pins required
Simple design integration (no muxes)	Scan chain balancing difficult
Few module pins allowing easy IP reuse	Could be critical regarding test power
	Potential single channel coverage loss
	Potential bandwidth inefficiencies

Table 4.1: Summary parallel retargeting approach

4.2.3.2 Sequential Pattern Retargeting Approach

In the sequential test approach the test partitions are tested one after the other through the same top-level pin interface. The big advantage is that the different test partitions do not interact. As the test of each partition is completely independent of the other modules, there are no inefficiencies caused by different scan chain lengths or pattern counts. Therefore, it is possible to reuse existing blocks without aligning the top-level scan test concept between different modules. The module-level scan architecture can be defined in the same way as described for the flat non-hierarchical scan test approach.

The sequential test approach usually does not involve any test time penalty compared to the parallel test. As more top-level pins can be used for the test of one partition, more and therefore shorter scan chains can be used. This leads to comparable overall test times. The test time might be even reduced by preventing the inefficiencies described above. Figure 4.4.3 shows the sequential test approach for the example introduced in Section 4.2.3.1. The partitions would be tested one after the other, but as all five partitions could be operated using all ten scan channels the number of scan chains could be five times higher and therefore five times shorter using the same compression ratio. The overall test time is less compared to Figure 4.4.2 as the tester bandwidth can be fully utilized.

The overall scan test times are comparable or even less as long as the test time is dominated by the scan shift times. The sequential test requires test partitions of a certain size. If the scan chains would become extremely short, the efficiency of the test would be reduced due to a fixed number of configurations cycles required for every pattern (e.g. for configuring some compactor masking). In addition, a very high number of scan chains for a small test partition might lead to routing issues.

Another advantage of the sequential test approach is the possibility to reduce the scan test power consumption, by keeping the non-tested partition idle. If the partitioning is taking into account different power domains as proposed in Section 4.1.4 the non-tested domains might be switched off completely.

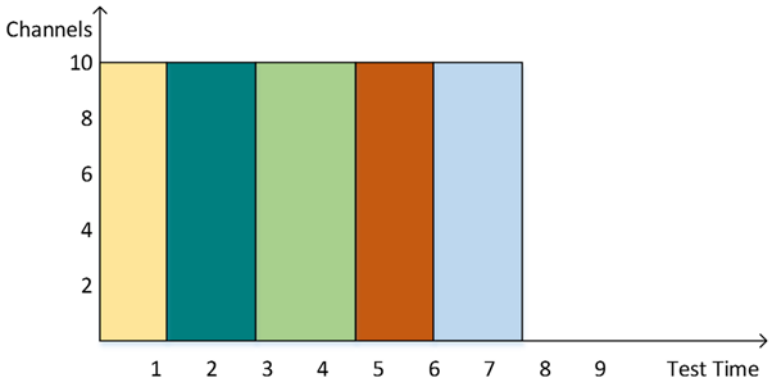

Figure 4.4.3: Sequential test approach

The main drawback of this test strategy is the need of a separate test mode for each sequentially tested partition due to the required pin multiplexing of the scan output channels. As a consequence, each sequential test also requires a separate pattern retargeting run considering the individual test mode entry sequence.

Positive	**Negative**
No dependencies between partitions causing inefficiencies	Requires short chains to compensate test time
Combination of non-aligned modules	Multiple patterns and test modes
Allows test power reduction	

Table 4.2**: Summary sequential** retargeting approach

4.2.3.3 Mixed Parallel/Sequential Pattern Retargeting Approach

In some cases, it might be beneficial to use a mixed parallel/sequential test approach. In this approach several groups of parallel tested partitions are tested sequentially as shown in Figure 4.4.4.

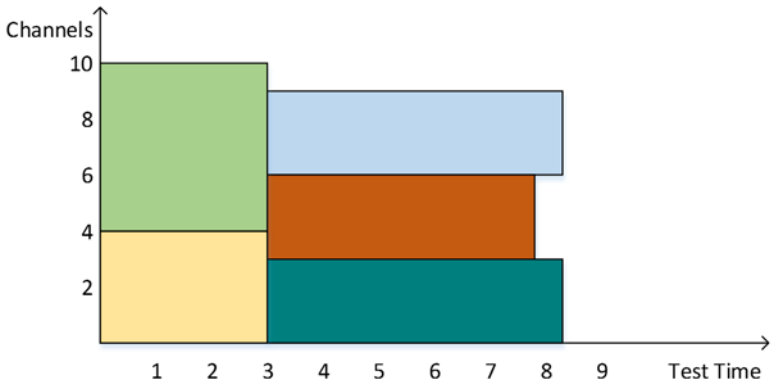

Figure 4.4.4: Mixed parallel/sequential approach – scenario A

For the mixed test approach the scan architecture of the test partitions is basically determined by the available top-level pins. When using the mixed approach there is much more flexibility for the module level scan planning. There are many scan configurations with different scan channel numbers and different scan chain lengths for each module that could still be combined to perform a reasonable top-level test. In the examples of Figure 4.4.4 and Figure 4.4.5 the top right (light blue) test

partition can be implemented with three or four scan channels without having any impact on the overall test time.

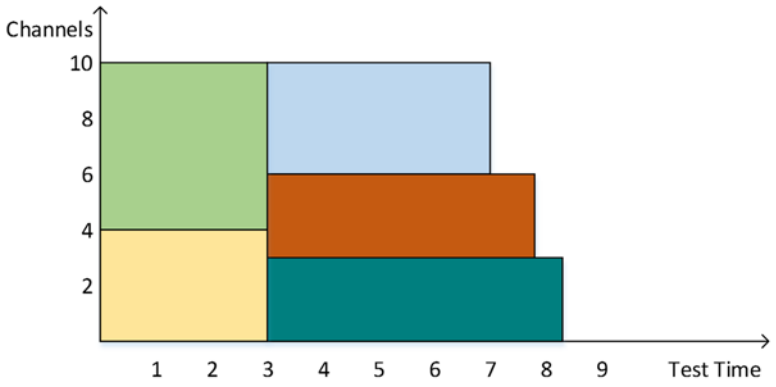

Figure 4.4.5: Mixed parallel/sequential approach – scenario B

The mixed parallel/sequential approach is an option, if there are not enough top-level pins available for the parallel test and still the partitions are too small for a pure sequential test. In addition, this approach helps to deal with IP blocks that come already with a fixed scan architecture not fitting into the parallel or sequential top-level concept.

The main drawback of this approach is the complex top-level multiplexing for the scan channel outputs. The required test pattern counts for the modules can only be estimated before running the final test pattern generation. Thus the top-level test scheduling and the test mode planning has to be based on assumptions or it needs to be done late in the design phase.

In summary the mixed parallel/sequential test scheduling approach combines the advantages and drawbacks of the pure parallel and sequential test.

4.2.3.4 Flexible Test Configuration

In the previous sections fixed scan architectures have been assumed for the modules. In some cases, it might be beneficial to implement multiple scan architectures in the modules, as this allows a more flexible test scheduling on the top-level. There are different reasons for such a flexible approach. If a module is intended as an IP block to be used in multiple products, the implementation of different scan configurations would ease the product-specific test integration. In some cases, different pin configurations are used for different test insertions e.g. for wafer test and package test. Also here the module-level flexibility would help to deal with the different top-level test constraints. Even if the top-level pin-interface is known upfront and does not change, there are still scenarios for which flexible scan architectures are beneficial.

As stated in the parallel test Section 4.2.3.1 the test with a single scan input channel might lead to some test coverage loss when using on-chip data compression. The implementation of a multi-mode scan configuration can solve the issue by applying a mixed parallel/sequential test scheme as described in the previous section. First a parallel test could be performed by testing all the test partitions at the same time using a small number of pins per module. For modules facing insufficient coverage due to limited compression an additional test phase could be added. This time a second scan test mode is used that dedicates more pins to the partition leading to a relaxed compression rate. Due to the larger number of pins used for the module it is no longer possible to access all modules at the same time.

A similar approach could be used for optimizing the utilization of the external test interface. As shown in Figure 4.4.2 the number of test patterns differs for the test partitions. Once a module is tested completely the related top-level scan pins could be used to enhance or speed-up the test of the remaining partitions. This means at first only a subset of *n* patterns is retargeted, wherein *n* is the number of patterns of the module with the lowest pattern count. Afterwards the remaining test partitions are reconfigured making use of the freed pins. Then the next patterns are retargeted. The example shown in Figure 4.4.6 reconfigures the scan access only once (the freed pins of the yellow partition at the bottom are used for green partition in the middle). For further optimizations the test access could have been reconfigured again after the next partition is tested completely. This retargeting process can be quite complex for designs with many test partitions. Thus some kind of automation would be required to simplify the scheduling of the pins.

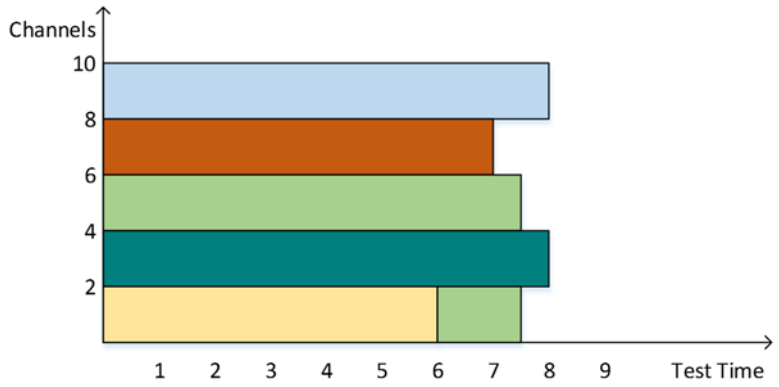

Figure 4.4.6: Reconfigured parallel test

The prerequisite for flexible top-level test scheduling is the implementation of multiple scan configurations on module-level. In general, it is possible to implement different test modes with different numbers of scan pins independent of the used test data compression method. For the EDT [66] compression used in the experiments and analyses of this work there is one special mode allowing to operate the compression hardware with a flexible number of scan channels. This significantly reduces the area and effort for implementing a module supporting multiple scan configurations. The main drawback of this special solution is that at least two scan input channels have to be available. This could be an issue for a pin-limited parallel test approach.

Another scan flexibility that can be used on module-level is the number of internal scan chains varying the compression rate. In this case the module-level test interface remains constant, but the number and resulting lengths of the scan chains can be varied for different configurations. Such an approach can be used as an alternative solution to handle the potential test coverage loss due to compression limits. Faults not detected due to an overly aggressive compression rate might be detectable after reconfiguring the scan architecture to a second mode with a lower compression rate. These add-on patterns would be of course more costly regarding test time and tester memory requirements due to the increased scan chain lengths.

Another application of using a different scan chain length in different modes is LBIST. As the LBIST is typically using pseudo-random test stimuli, the number of scan chains is not limited by test data compression rates. To reduce the LBIST execution time it is desirable to use more and therefore shorter scan chains compared to the ATPG based test approach.

4.2.3.5 Top-Level Test

In addition to the retargeting of the module-level test patterns also the test of the logic and interconnects between the different test partitions needs to be considered for the top-level. One problem for the initial planning of the top-level scan test architecture is the unpredictable number of wrapper cells. While there are early estimations for the number of functional registers per module it is usually unclear how many of these registers have to be part of wrapper scan chains used for module isolation. For the test of the top-level logic and module interfaces it is required to operate all wrapper chains in parallel assuming that there is interaction between most of the test partitions. In addition, potential scan chains for remaining top-level logic outside the test partitions need to be considered. Because of the necessary parallel test approach all wrapper chains and the additional top-level scan chains should be balanced to enable an efficient test. This complicates the early planning due to the unknown wrapper chain lengths. Therefore, the initial top-level test concept has to be based on vague estimations and might need to be adjusted after the initial implementation of the test partitions.

Another aspect of the top-level test concept is the required routing. Usually also the top-level test requires some on-chip test data compression to deal with test data volume and test time. The required compression logic could be placed as glue logic on the top-level between the design partitions (see Figure 4.4.7). To avoid or to reduce glue logic on top-level the compression module could be also placed in one of the design partitions.

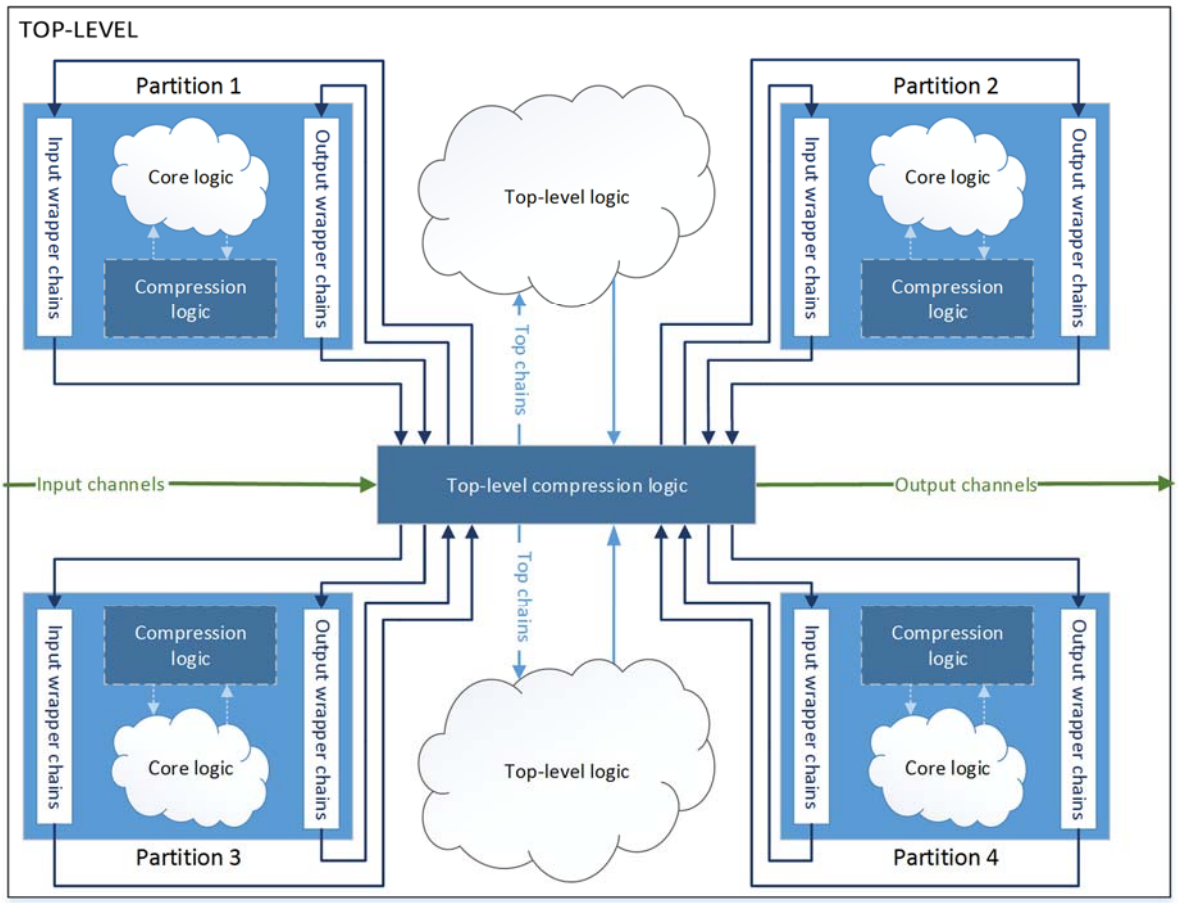

Figure 4.4.7: Example of top-level test concept

No matter where this top-level compression hardware is placed the routing could be difficult as all module-level wrapper chains and all potential top-level scan chains from all over the chip need to be connected to it.

To relax top-level routing the number of top-level chains should be limited to a reasonable number. For this purpose, it might be required to have different wrapper scan chain configurations for the module-level test and the top-level test. For the module-level test the wrapper chains should be balanced with the core scan chains. Based on the compression ratio used for the test partitions this can lead to a significant number of (short) wrapper chains for the module test. As in this mode the wrapper chains are connected locally to the compression logic within the test partitions routing is usually no issue. For the top-level test the number of wrapper chains could be reduced to ease the top-level routing. The implementation of such a dedicated second wrapper scan chain configuration allows the planning of the top-level test strategy completely independent of any constraints the module-level test.

4.3 Test Coverage

When using a hierarchical test approach there are some test coverage related topics that need to be considered especially for the generation of scan test patterns. The Section 4.3.1 deals with test coverage analysis on module-level. In Section 4.3.2 the test coverage impact of the test isolation wrapper is discussed in more detail. The calculation of the overall test coverage number based on the module-level tests is described in Section 4.3.3.

4.3.1 Module-level Coverage Analysis

To ensure the overall product test quality it is essential to check the test coverage already on module-level. In case the test coverage for a module does not reach the expected goals the root cause of the untested faults needs to be analyzed and measures need to be taken to improve testability.

Such a module-level coverage analysis is required no matter if the top-level test approach is flat or hierarchical. For the flat approach some trial synthesis and trial ATPG is done before providing a functional module to the top-level integrator. The patterns generated during this experimental ATPG run cannot be used later-on for the top-level, but the coverage of the block can be evaluated and easily debugged assuming an “ideal” design integration of the module.

For a hierarchical test approach the module-level coverage analysis is more complicated. For the module-level test the implemented scan isolation wrapper is in isolation mode. Therefore, the access to all gates “outside” of the wrapper is blocked and the faults in this logic are not testable in this test mode. These faults will be targeted in the later-on top-level ATPG run which tests the interfaces between the test partitions. Targeting all faults in the module the obtained test coverage of an ATPG run may appear to be too low. This complicates the coverage analysis as it is unclear if the observed coverage loss is caused by testability issues of the core logic or if this is just an artificial side effect of the isolation wrapper. Unfortunately, there is no easy way to consider only the faults “inside” of the isolation wrapper for the coverage analysis. Especially if functional registers are shared for the isolation and if there is no dedicated naming convention or hierarchy used for the wrapper, there is no strict boundary between the core logic to be tested and the gates accessible only in the top-level test mode.

One way to deal with is problem is to setup a dummy ATPG run configuring the module in the test mode for testing the interfaces. Even if this ATPG setup and the

generated patterns can never be used for the productive test, it helps for debugging the module-level coverage by determining which faults will be detected by the later-on top-level test. After having generated the module-level test patterns the list of faults detected in this artificial top-level test can be imported to calculate the overall test coverage for the module considering both test modes. Based on this overall test coverage the testability of the module can be analyzed and improved if required.

An example for the module-level test coverage is shown in Figure 4.4.8. For a module of an industrial design two different netlist versions are compared. At first the module is implemented without any module isolation. Here ATPG can be performed without any restrictions reaching a test coverage of 99.54%. Afterwards the same design is used, but during the scan insertion step wrapper chains for the module isolation are inserted. For this netlist version two different test modes need to be considered. In the internal test mode targeting the core-internal fault the stuck-at test coverage achieved by ATPG is only 98.20%. After running a second ATPG run in external test mode targeting the interface faults the combined test coverage is 99.55%. This means at the end the stuck-at test coverage is comparable for both netlist versions. So, the insertion of module isolation wrappers for the hierarchical test does not impact the overall stuck-at test coverage. For the delay test the situation might be different as discussed in the next section.

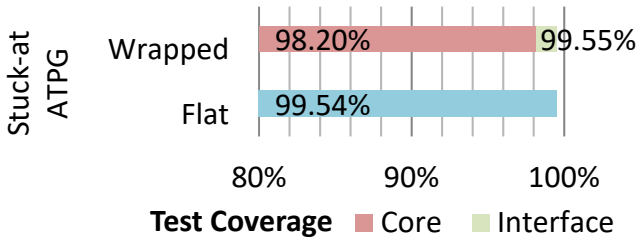

Figure 4.4.8: Test coverage comparison of wrapped and non-wrapped module

4.3.2 Test Coverage Impact of Isolation Wrappers

Besides the difficulties to determine the test coverage on module-level, there might be a test coverage impact caused by the isolation wrapper. When using dedicated wrapper cells for the module isolation, the delay test checking the interface timing between different test partitions becomes difficult.

When using shared wrapper cells it is still possible to test the functional interface path with the appropriate frequency (see Figure 4.4.9). The functional output register of module A is launching a transition, which is then captured into the functional input register of module B. This test is executed as part of the top-level test in which the wrapper chains of the modules are configured in external test mode.

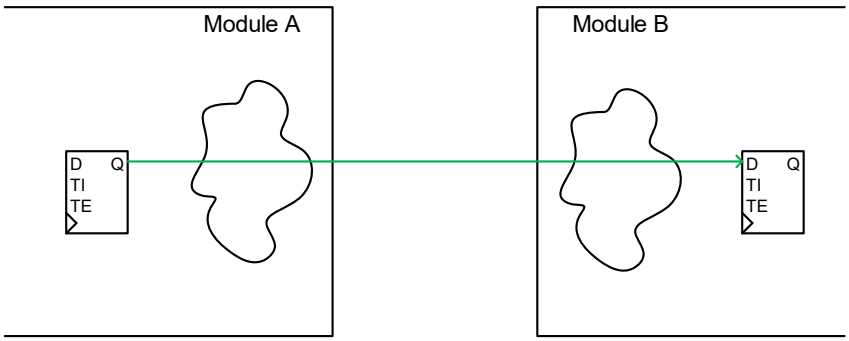

Figure 4.4.9: Interface test using shared wrapper cells (external test mode)

When adding dedicated isolation wrapper cells at the module boundary the functional interface paths can no longer be tested as a whole. In this case the interface will be tested in two steps. First the logic between the functional input or output register and the dedicated wrapper cell is tested as part of the module-level test (see Figure 4.4.10).

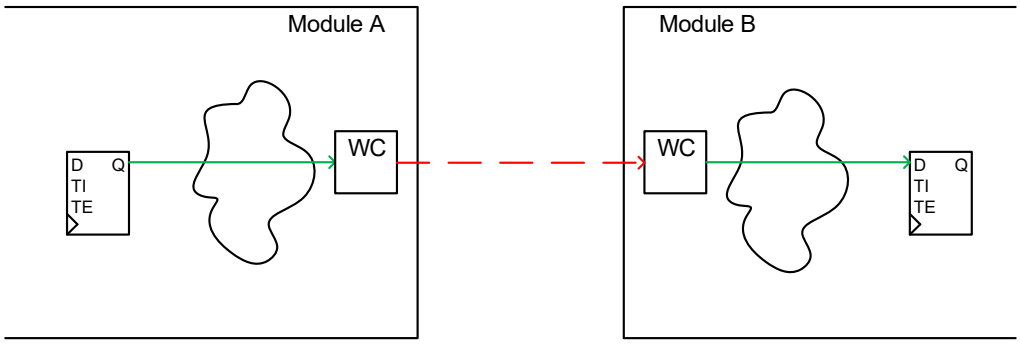

Figure 4.4.10: Interface test using dedicated wrapper cells (internal test mode)

Afterwards the interconnection between the modules is tested as part of the top-level test using the dedicated wrapper cells as launch and capture point (see Figure 4.4.11).

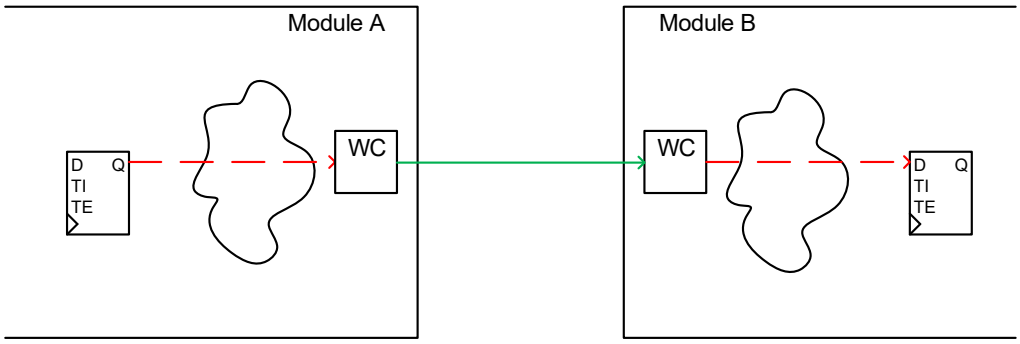

Figure 4.4.11: Interface test using dedicated wrapper cells (external test mode)

With this test approach it is still possible to check the connectivity between the modules, but the functional timing cannot be verified. As the functional path is split into three individual segments, the test would be too relaxed when using the functional clock and target frequency for the wrapper cells. In theory increasing the test frequency for the interface paths segments could be a solution to ensure the test quality, but in most cases this would not be feasible. First of all, this approach would require to increase the test frequency for the module IO path segments without over-

testing the core logic. Using a dedicated clock with a special timing for the wrapper cells might be an option, but adjusting the frequency would be difficult when providing the at-speed clock pulses from an on-chip clock source like a PLL. However, the main problem of such a test would be that the timing to be checked could be different for each interface path. Based on where the isolation cells are inserted the split of the functional path into an input and output segment could vary for each module connection. This would require adapting the test frequency individually to each segment making it practically impossible to check the functional performance of each path.

This limitation of the at-speed interface test would already occur if only one of the two modules would use a dedicated wrapper cell breaking the functional connection path. Therefore, the usage of dedicated wrapper cells should be prevented where possible or the design partitioning should ensure module interfaces with non-critical timing which does not need to be checked. In case dedicated wrapper cells are used for timing relevant connections they might be tested by some functional interface tests.

4.3.3 Overall Fault Coverage Calculation

To judge the chip-level test quality of a hierarchical test it is not sufficient to have the module-level test coverage numbers for each partition, but it is required to calculate the overall test coverage for the complete design.

The test pattern generation for the individual test partitions is done in separate ATPG runs. In addition, different test modes are used for the module-level tests and the top-level interface test. Therefore, the overall test coverage cannot be determined directly in a single ATPG run. For the calculation of the overall test coverage the results from the individual ATPG runs need to be combined. To transfer the test coverage information from one ATPG run to another without losing any information the complete fault list containing the fault detection status of each individual fault can be written. By merging the fault lists of all performed pattern generations, the overall fault detection status can be derived and the complete test coverage can be calculated. Some faults might be contained in multiple faults lists, e.g. faults located in the module-level interface logic “outside” the isolation wrapper. These faults are classified as “untestable” in the fault lists of the module-level ATPG runs. The same faults are usually “detected” in the fault list of the top-level interface test. When merging the fault lists, a fault has to be considered as “detected” when it was detected at least in one of the ATPG runs. Some faults might be contained only in a single fault list, e.g. module internal faults when using simplified greybox models for the top-level ATPG run. For these faults the detection status can be taken directly from the fault list where they occur. When merging the fault lists of the test partitions and the top-level the instance names used in the module-level fault lists need to be prepended by the top-level instance name of the instantiated modules.

4.4 Case Study Open Source Quad-Core Processor

The hierarchical design and test approach has been evaluated as a case study for an open source quad-core processor. The test case design contains four identical CPU cores and some additional logic on the top-level.

Based on this architecture an obvious partitioning is to divide the design into five parts. The CPU core is handled as one design and test partition which is then instantiated four times together with the remaining logic on the top-level. Using the CPU core as one partition provides several benefits. For the test it is sufficient to

generate the test patterns only once and then to reuse these patterns for all four instances. In addition, the test stimuli can be shared between the instances reducing the overall test data volume. Using identical CPU core implementations also reduces the design effort as the design optimization and verification needs to be done only for one CPU core. Nevertheless, there is a drawback of this partitioning approach. The RTL code of the CPU core used for this study is very generic. Many features of the CPU core can be enabled and configured using a large number of configuration inputs. When the CPU core is implemented stand-alone not taking into account the top-level configuration the complete functionality of the core needs to be implemented even if it is never used after integration into the design. When using a flat implementation or a different partitioning unused logic could be optimized away considering tied pins at the CPU boundary. This optimization is not possible with the chosen partitioning. In addition, these configuration inputs of the CPU core need be isolated for the hierarchical test. Even if an input signal would be tied on top-level, wrapper cells are required for the module isolation. This does not only result in longer wrapper scan chains, but also leads to test patterns generated for logic that is functionally not used. On the other hand, the isolated CPU core and its generated patterns could be used without any changes in other designs which use a different top-level configuration.
As the test case design uses only one power domain and one clock frequency both parameters do not need to be considered for the partitioning strategy. Nevertheless, the chosen partitioning would allow addressing test power issues by testing modules sequentially.

For each CPU core several DFT features have been implemented as part of the case study. Besides the module isolation and on-chip test data compression hardware an integrated LBIST solution enables the self-test operated at-speed to check the performance within the system. As the LBIST can be executed individually for each CPU core it is directly visible which core has failed in case of a defect. This provides some basic error localization capabilities. The diagnosis resolution could be enhanced using the features described in [65].

A simplified overview of the implemented DFT core is shown in Figure 4.4.12.

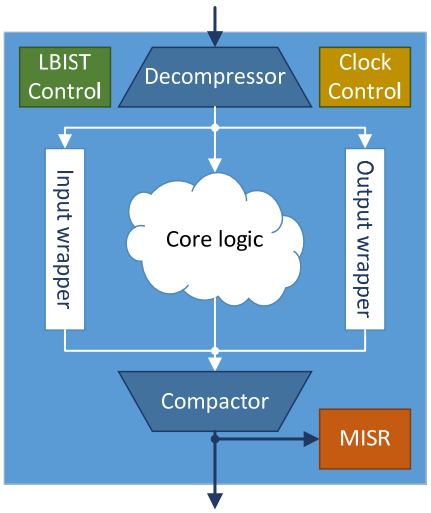

Figure 4.4.12: CPU core with implemented DFT features

The overall top level test implementation looks as shown in Figure 4.4.7. Test patterns are first generated for the stand-alone CPU core. These core-level test patterns are then retargeted to the top-level, testing all four cores in parallel. Additional top-level patterns are generated to test the top-level logic and the interfaces between the CPU cores. The resulting test coverage numbers and pattern counts are shown in Table 4.3.

	CPU (stand-alone)		**Overall design**	
	Coverage	**# Patterns**	**Coverage**	**# Patterns**
Stuck-at	92.38%	1705	97.99%	345 + 1705 = 2050
Transition	84.51%	2695	87.93%	980 + 2695 = 3675

Table 4.3: Test coverage and pattern count for stand-alone CPU and overall design

5 Fault management

In this section, we first describe methods of reliability analysis and optimization. Then, we detail a general fault handling methodology and present a realistic Fault Management scenario both relying on the BASTION extensions to IEEE 1687 and the proposed fault classification scheme.

5.1 Reliability Analysis and Optimization

This sub-section is organized as follows. First, we provide a mathematical framework to evaluate the Level of Confidence (LoC) for a given distribution of checkpoints. Next, we outline an Exhaustive Search approach to obtain the optimal distribution of a given number of checkpoints that results in the maximal LoC. To overcome the complexity of the Exhaustive Search approach, we developed a heuristic, i.e. Clustered Checkpointing, which distributes a given number of checkpoints such that the LoC is maximized. Finally, we summarize with some results that evaluate the ability of the heuristic to find the optimal solution.

5.1.1 Evaluation of Level of Confidence

In this section, we show how to calculate the LoC with respect to a given deadline for a given distribution of n_c checkpoints. The distribution of the n_c checkpoints is represented by a vector $\bar{n}_c$ of size n_c, where the *i*-th element (ES_i) in the vector represents the size of the *i*-th execution segment, i.e. $\bar{n}_c = [ES_1, ES_2 ... ES_{n_c}]$. The LoC is calculated as a sum of probabilities that a job completes at time instances lower than or equal to the given deadline.

For a given $\bar{n}_c$, the re-execution of different execution segments has different impact on the completion time due to the different re-execution cost, i.e. the sum of the size of the execution segment ES_i, and the checkpointing overhead τ. Thus, the completion time can be expressed as a discrete variable ${}^{\bar{n}_c}t_{\bar{k}}$ (see Eq. (1)), where $\bar{k}$ denotes a re-execution vector. The re-execution vector $\bar{k}$ represents a vector of size n_c where each element k_i indicates the number of re-executions of the i-th execution segment.

$$ {}^{\bar{n}_c}t_{\bar{k}} = T + n_c \times \tau + \sum_{i=1}^{n_c} (ES_i + \tau) \times k_i \qquad (1) $$

For each discrete value of ${}^{\bar{n}_c}t_{\bar{k}}$ there exists a probability that a job completes at that given instance in time. We denote the probability that a job completes at ${}^{\bar{n}_c}t_{\bar{k}}$ with $p_{\bar{n}_c}(t_{\bar{k}})$. Next, we elaborate how to calculate $p_{\bar{n}_c}(t_{\bar{k}})$. When a job completes at time ${}^{\bar{n}_c}t_{\bar{k}}$, it means that each execution segment ES_i has been executed exactly $k_i + 1$ times, i.e. k_i erroneous executions and a single successful execution.

Since we assume that occurrence of errors is an independent event and that the probability P_T is given, we can calculate the probability of a successful execution segment. Given P_T, the probability that no errors occur in one processing node during the execution of ES_i is evaluated as $P_T^{\frac{ES_i}{T}}$. We denote with P_{ε_i} the probability of

successful execution of ES_i, i.e. the probability that no errors occur in both processing nodes during the execution of the *i*-th execution segment. We calculate P_{ε_i} as:

$$P_{\varepsilon_i} = P_T^{\frac{ES_i}{T}} \times P_T^{\frac{ES_i}{T}} = P_T^{\frac{2\times ES_i}{T}} \quad (2)$$

Given Eq. (2), the probability of an erroneous execution of ES_i is calculated as $1-P_{\varepsilon_i}$. Finally, the probability that a job completes at ${}^{\bar{n}_c}t_{\bar{k}}$ is calculated with the following expression:

$$p_{\bar{n}_c}(\bar{k}) = \prod_{i=1}^{n_c} P_{\varepsilon_i}\left(1-P_{\varepsilon_i}\right)^{k_i} = P_T^2 \prod_{i=1}^{n_c}\left(1-P_{\varepsilon_i}\right)^{k_i} \quad (3)$$

Eq. (3) represents the probability distribution function $p_{\bar{n}_c}(\bar{k})$, and it calculates the probability that each of the execution segments ES_i has been executed once successfully and k_i times erroneously. To compute the LoC with respect to a given deadline it is required to sum the terms $p_{\bar{n}_c}(\bar{k})$ for all ${}^{\bar{n}_c}t_{\bar{k}}$ which are lower or equal to the deadline. We denote with $\Lambda_{\bar{n}_c}(D)$ the LoC with respect to a given deadline D for a given distribution $\bar{n}_c$ and it is calculated with the following expression:

$$\Lambda_{\bar{n}_c}(D) = \sum_{\bar{k}}^{{}^{\bar{n}_c}t_{\bar{k}}\le D} p_{\bar{n}_c}(\bar{k}) \quad (4)$$

For a given $\bar{n}_c$, only some re-execution vectors $\bar{k}$ produce ${}^{\bar{n}_c}t_{\bar{k}} \le D$. We refer to these re-execution vectors as valid re-execution vectors. To obtain the set of valid re-execution vectors we use the function $f(\bar{k},i)$ (see Eq. (5) and Eq. (6)).

For a given valid re-execution vector, $f(\bar{k},i)$ produces a new valid re-execution vector, if such vector exists, or it returns an empty set otherwise. The function requires two input parameters: a valid re-execution vector ($\bar{k}$) and an update index (*i*). For the given parameters, $f(\bar{k},i)$ first computes a tentative re-execution vector $\bar{k}^N$ by incrementing k_i by one and setting to zero all k_j where $j > i$ (see Eq. 5). If $\bar{k}^N$ is a valid re-execution vector and $i > 0$, then $f(\bar{k},i)$ returns $\bar{k}^N$. Otherwise, $f(\bar{k},i)$ proceeds with a recursive call $f(\bar{k},i-1)$. If the function is invoked with $i = 0$, $f(\bar{k},i)$ returns an empty set, meaning that all valid re-execution vectors have been identified. To identify the entire set of valid re-execution vectors, $f(\bar{k},i)$ is iteratively called with these arguments: (1) the most recently obtained valid re-execution vector (the initial valid re-execution vector is $\bar{k} = [0,0...0]$) and (2) the number of checkpoints n_c is passed as the update index.

$$\bar{k}^N = [k_1, k_2 ... k_{i-1}, k_i + 1, 0, ...0] \quad (5)$$

$$f(\bar{k},i)=\begin{cases}\bar{k}^N, \text{iff } \bar{k}^N \text{ is } valid \text{ and } i \geq 1\\ f(\bar{k},i-1), \text{iff } \bar{k}^N \text{ is not } valid \text{ and } i \geq 1\\ \varnothing, \text{iff } i=0\end{cases} \quad (6)$$

Important to note is that for all $\bar{n}_c$ which are a permutation of a given $\bar{n}_c^{\Diamond}$, $\Lambda_{\bar{n}_c}(D)$ is always the same as $\Lambda_{\bar{n}_c^{\Diamond}}(D)$. This follows from the commutative property of addition and multiplication. The commutative property of addition and multiplication ensure that for a given ${}^{\bar{n}_c}t_{\bar{k}}$ there exists a corresponding ${}^{\bar{n}_c^{\Diamond}}t_{\bar{k}^{\Diamond}}$ and $p_{\bar{n}_c}(\bar{k}) = p_{\bar{n}_c^{\Diamond}}(\bar{k}^{\Diamond})$, where $\bar{n}_c^{\Diamond}$ and $\bar{k}^{\Diamond}$ are the same permutation (re-ordering) of $\bar{n}_c$ and $\bar{k}$. For a given $\bar{n}_c$ there exist $n_c!$ different permutations and thus, $n_c!$ different $\bar{n}_c^{\Diamond}$ (assuming that all execution segments in $\bar{n}_c$ are of different size).

Additional consideration should be taken for distributions $\bar{n}_c$ where a number of execution segments have the same size. In such case, many different re-execution vectors $\bar{k}$ would provide the same completion time and $p_{\bar{n}_c}(\bar{k})$ will be the same for all those vectors, making the evaluation of $\Lambda_{\bar{n}_c}(D)$ very inefficient. To avoid this, we make use of a reduced format of $\bar{n}_c$, denoted with $\tilde{n}_c = [ES_1^{m_1}, ES_2^{m_2} ... ES_q^{m_q}]$, where $ES_i^{m_i}$ is used to indicate that there are m_i execution segments of size ES_i. One observes that the size of $\tilde{n}_c$ is q, which is lower than n_c (the size of $\bar{n}_c$) as long as two or more execution segments in $\bar{n}_c$ have the same size. Similarly, we can use the reduced format of $\bar{k}$, i.e. $\tilde{k}$ to represent the re-execution vectors. Each element k_i in $\tilde{k}$ denotes the total number of re-executions of an execution segment of size ES_i (any of the m_i different execution segments). The advantage with this is that every $\tilde{k}$ corresponds to a unique instance of the completion time. The completion time is calculated as:

$${}^{\tilde{n}_c}t_{\tilde{k}} = T + n_c \times \tau + \sum_{i=1}^{q}(ES_i + \tau) \times k_i \quad (7)$$

The probability that a job completes at ${}^{\tilde{n}_c}t_{\tilde{k}}$ is calculated as:

$$p_{\tilde{n}_c}(\tilde{k}) = \prod_{i=1}^{q}\binom{m_i + k_i + 1}{k_i} P_{\varepsilon_i}\left(1 - P_{\varepsilon_i}\right)^{k_i} \quad (8)$$

Eq. (8) calculates the probability that each execution segment of size ES_i has been executed once successfully and k_i times erroneously. However, since there are m_i different execution segments of size ES_i, and since the k_i re-executions may occur as a result of an error in any of the m_i execution segments, there are a total $\binom{m_i + k_i + 1}{k_i}$ different combinations. Therefore, this term is included in Eq. (8).

Finally, to compute the LoC, it is required to obtain all valid $\tilde{k}$. To achieve this, the function $f(\bar{k}, i)$ can be used, where instead of the re-execution vector $\bar{k}$, we use $\tilde{k}$ as the input argument and q (the size of $\tilde{k}$) is passed as the update index.

5.1.2 Exhaustive Search

In previous section, we showed how to evaluate the LoC with respect to a given deadline for a given distribution of n_c checkpoints. However, the problem we aim to solve is to obtain the optimal distribution, for a given number of checkpoints n_c, which results in the maximal LoC. As one way to obtain the optimal distribution of n_c checkpoints is to evaluate the LoC for all possible distributions of the given n_c checkpoints, in this section, we review an exhaustive search method that finds the optimal distribution which results in the maximal LoC. Further, we show that it is possible to speed up this method by avoiding exploration of distributions which are permutation of each other.

While a trivial exhaustive search approach can always find the optimal solution, the major drawback with any trivial exhaustive search algorithm is the complexity. For this particular problem, the complexity is directly proportional to the number of possible distributions of n_c checkpoints, and this number increases rapidly with n_c. However, in the previous section, we showed that the LoC for any distribution which is a permutation of a given distribution $\bar{n}_c$ will be the same as the LoC obtained for $\bar{n}_c$. This implies that there is no need to explore all possible distributions to find the optimal distribution that results in the maximal LoC. For that reason, in this section, we discuss a method, namely Exhaustive Search (EXS), that finds the optimal distribution by only evaluating the LoC for distributions $\bar{n}_c$ which are not permutations of each other. In particular, we show how to obtain these distributions, and we determine the number of different distributions that need to be explored with this method. Furthermore, we compare the complexity of the EXS method with a trivial exhaustive search method that explores all possible distributions of a given number of checkpoints n_c.

The complexity of the trivial exhaustive search method is directly proportional to the number of all possible distributions of n_c checkpoints in a job with a processing time T, and this number is evaluated with the following equation:

$$\binom{T-1}{n_c-1} = \frac{(T-1)!}{(T-n_c)!(n_c-1)!} \tag{9}$$

Assuming that a checkpoint can be taken at any integer time unit, for a job with processing time T, expressed in time units, there are T different points where a checkpoint can be taken. Since one checkpoint must be taken at the end of the job, i.e. at time unit T, the rest of the $n_c - 1$ checkpoints can be taken at any of the remaining $T-1$ time units. Therefore, the number of combinations of $n_c - 1$ checkpoints over $T-1$ time points represents the amount of all possible distributions of n_c checkpoints as it is expressed with Eq. (9).

In , we report the number of all possible distributions of n_c checkpoints in a job with a processing time T = 1000 t.u. at various instances of n_c. As can be seen from Table I, the number of possible distributions grows rapidly with n_c. While the trivial exhaustive search approach explores all possible distributions of n_c checkpoints, the EXS method does not need to evaluate the LoC for all distributions, and instead, it should only explore distributions which are not permutations of each other. To achieve this goal, it is sufficient to explore distributions $\overline{n}_c$ which satisfy the following condition:

$$ES_i \leq ES_j \quad \forall\, i \leq j \tag{10}$$

An important consequence that follows from the condition expressed in Eq. (10) is that the size of the shortest execution segment in a distribution $\overline{n}_c$, i.e. ES_1, can never be larger than $\left\lfloor \frac{T}{n_c} \right\rfloor$, where T represents the processing time of the job and n_c represents the number of checkpoints that are to be distributed. Next, we elaborate how to obtain all those distributions that satisfy the condition in Eq. (10).

To obtain the different distributions $\overline{n}_c$, we use the function $next(\overline{n}_c, i)$, given in Eq. (13), which for a given distribution $\overline{n}_c$ and an update index *i*, generates the next distribution to be explored. The function $next(\overline{n}_c, i)$ first computes a tentative distribution $\overline{n}_c^N$ (Eq. (12)) by incrementing the size of the *i*-th execution segments by

n_c	$\binom{T-1}{n_c-1}$
2	999
3	498501
4	165668499
5	41251456251
6	8209039793949
7	1359964259197551
8	192920644197595449
9	23922159880501835676
10	2634095604619702128324

Table 5.1 Number of possible distributions of checkpoints in a job with a processing time $T = 1000$ **time units**

one ($ES_i = ES_i + 1$), and assigning the same incremented value to all execution segments with index in the range $[i+1, n_c - 1]$. The size of the last execution segment, i.e. ES_{n_c}, is computed such that the sum of all the execution segments is the same as the given processing time of the job T (see Eq. (11)). If the tentative distribution $\overline{n}_c^N$ satisfies the condition in Eq. (10), then the function returns $\overline{n}_c^N$ as output. Otherwise, the function steps in a recursive call by decrementing the update index *i*, i.e.

$next(\overline{n}_c, i-1)$. If the function is called with an update index $i = 0$, the function returns an empty set, meaning that $\overline{n}_c$ is the last distribution to be explored.

$$ES_{n_c} = T - \sum_{j=0}^{i-1} ES_j - (n_c - i) \times (ES_i + 1) \tag{11}$$

$$\overline{n}_c^N = [ES_1, ES_2 ... ES_{i-1}, ES_i + 1, ES_i + 1, ... ES_i + 1, ES_{n_c}] \tag{12}$$

$$next(\overline{n}_c, i) = \begin{cases} n_c^N, \text{iff Eq. (10) holds} \\ next(\overline{n}_c, i-1), \text{iff Eq. (10) does not hold} \\ \varnothing, \text{iff } i = 0 \end{cases} \tag{13}$$

When the function is used, the function is always invoked with the current distribution $\overline{n}_c$ that is being explored and $n_c - 1$ as an update index. The initial distribution is constructed by assigning the value of 1 (the shortest size of an execution segment) to all execution segments except for the last one. The size of the last execution segment is computed such that the sum of all execution segments is equal to the processing time of the job *T*. In other words, the initial distribution is presented with the following vector $\overline{n}_c = [1,1,...1, T - n_c + 1]$.

By using this function, the number of distributions which are explored with the EXS method is reduced when compared to the number of all possible distributions given with Eq. (9). Next, we elaborate how to compute $r_{n_c}[T]$, the number of distributions of n_c checkpoints in a job with a processing time *T*, which are explored with the EXS method.

We denote with $r_{n_c}^i[T]$ the number of distributions of n_c checkpoints in a job with a processing time *T*, where the size of the first execution segment, i.e. ES_1, is fixed to a value *i*. Due to the consequence which followed from Eq. (10), the size of the first execution segment cannot be larger than $\left\lfloor \frac{T}{n_c} \right\rfloor$. Hence, we compute $r_{n_c}[T]$ with the following expression:

$$r_{n_c}[T] = \sum_{i=1}^{\left\lfloor \frac{T}{n_c} \right\rfloor} r_{n_c}^i[T] \tag{14}$$

When the size of the first execution segment in a distribution $\overline{n}_c$ is fixed to a value *i*, $r_{n_c}^i[T]$ can be obtained by counting how many distributions of $n_c - 1$ checkpoints in a job with a processing time $T - i$ exist, such that the shortest execution segment among the $n_c - 1$ execution segments has to be larger or equal to *i*. Observe that when distributing $n_c - 1$ checkpoints in a job with a processing time $T - i$, the shortest execution segment cannot be larger than $\left\lfloor \frac{T-i}{n_c - 1} \right\rfloor$ (the consequence of Eq. (10)). Given this, the following recurrence equation can be used to calculate $r_{n_c}^i[T]$:

$$r_{n_c}^{i}[T] = \sum_{j=i}^{\left\lfloor \frac{T-i}{n_c-1} \right\rfloor} r_{n_c-1}^{j}[T-i] \tag{15}$$

The initial condition of the recurrence equation given with Eq. (15) is expressed as:

$$r_3^i[T] = \left\lfloor \frac{T-i}{2} \right\rfloor - i + 1 \tag{16}$$

Namely, the expression in Eq. (16) is derived from the fact that there exists only a single distribution of two checkpoints in a job with a processing time *T* such that the size of the first execution segment is fixed to a given value. When the size of the first execution segment in a distribution of two checkpoints is fixed to a value *i*, the size of the second execution segment is directly calculated by subtracting the value *i* from the processing time of the job. Following the recurrence equation, Eq. (15), we get:

$$r_3^i[T] = \sum_{j=i}^{\left\lfloor \frac{T-i}{2} \right\rfloor} r_2^j[T-i] = \sum_{j=i}^{\left\lfloor \frac{T-i}{2} \right\rfloor} 1 = \left\lfloor \frac{T-i}{2} \right\rfloor - i + 1 \tag{17}$$

In Table II, we show $r_{n_c}[T]$, obtained by using Eq. (14) and Eq. (15), for different values of n_c and a job with a processing time T = 1000 t.u. To show the reduced complexity of the EXS method over the trivial exhaustive search approach, we compare the results presented in and . As shown in and , the complexity of the EXS method is significantly lower than the complexity of the trivial exhaustive search approach. For example, we see that for n_c = 10, the number of all possible distributions is $\approx 2 \times 10^{21}$ (observe for n_c = 10). However, by excluding the distributions, which are permutation of each other, the number of different distributions that are explored by the EXS method is $\approx 1 \times 10^{15}$ (observe for n_c = 10). From this example, it should be evident that by excluding the redundant distributions the complexity of the EXS method is reduced tremendously in comparison to the trivial exhaustive search. Still, the complexity of the EXS method is very high. For that reason, we developed the Clustered Checkpointing (CC) method that aims to find the distribution that maximizes the LoC for a given n_c.

n_c	$r_{n_c}[T]$
3	83333
4	6965278
5	350697875
6	11835956777
7	287302124354
8	5274078114658
9	76037051194142
10	886745696653253

Table 5.2 Number of distribution of nc checkpoints, explored with the EXS method, for a job with a processing time T=1000 t.u.

5.1.3 Clustered Checkpointing

In this section, we present the Clustered Checkpointing (CC) method which distributes a given number of checkpoints such that the LoC is maximized. The CC heuristic explores only distributions $\overline{n}_c$ which are made out of clusters of execution segments, where all the execution segments that belong to the same cluster have the same size. The number of clusters is an input argument to the CC method, along with the number of checkpoints. Provided as an input argument, the number of clusters limits the search space of the CC method to only search for distributions $\overline{n}_c$ where the maximum number of clusters is kept lower than or equal to the provided input. Setting the number of clusters to lower values can drastically reduce the search space in comparison to the EXS method.

Given the following inputs: a processing time of a job *T*, a deadline *D*, a checkpointing overhead τ, a probability P_T that no errors occur in a processing node within an interval of length T, a number of checkpoints n_c, and a number of clusters *c*, the CC method outputs the distribution $\overline{n}_c^*$ which provides the highest LoC among all distributions $\overline{n}_c$ which are explored with this method. The distribution $\overline{n}_c^*$ which provides the highest LoC may contain at most *c* clusters. The reason for this is because the CC method is allowed to explore distributions where the maximum number of clusters is equal to *c*. However, it is possible that $\overline{n}_c^*$ contains less than *c* clusters. This is achieved by allowing clusters that do not contain any execution segments. Next, we detail the method.

For ease of reference, let us introduce some notations that will be used to explain the CC method. Given the number of clusters *c*, we use the notation k_i, where $i \leq c$, to indicate the number of execution segments that belong to the i-th cluster. Since all execution segments that belong to one cluster have the same size, we introduce the notation C_i, where $i \leq c$, to represent the size of the execution segments that belong to the *i*-th cluster. The set of values that can be assigned to k_i and C_i is constrained by the following expressions:

$$\sum_{i=1}^{c} k_i = n_c \tag{18}$$

$$\sum_{i=1}^{c} k_i \times C_i = T \tag{19}$$

The constraint given with the expression in Eq. (18) enforces a limit on the values that can be assigned to k_i, i.e. the total number of executions segments in all *c* clusters must be equal to the number of checkpoints n_c (when n_c checkpoints are taken a total of n_c execution segments exist). The constraint given with Eq. (19) limits the size of the execution segments that belong to the *c* clusters, as the sum of all the execution segments in all clusters must be equal to the processing time of the job *T*. Observe that for a given set of values k_i, that satisfy Eq. (18), there are multiple alternatives to assign values to C_i that satisfy Eq. (19). In other words, provided that distributing the n_c execution segments among the *c* clusters is decided, i.e. $k_i \setminus i \in [1, c]$ is known, multiple assignments to $C_i \setminus i \in [1, c]$ that satisfy Eq. (19) are

possible. This may cause exploration of distributions $\overline{n}_c$ that are permutation of each other. However, we discussed in the previous section, that to avoid exploration of distributions $\overline{n}_c$ which are permutation of each other it is sufficient to force the constraint that the executions segments should be ordered in ascending order based on their size (see Eq. (10)). Therefore, for the CC method, we add the following set of constraints:

$$C_i < C_j, \forall\ i < j \text{ s.t. } k_i \times k_j > 0 \setminus i, j \in [1, c-2] \tag{20}$$

$$C_{c-2} \leq C_{c-1} \tag{21}$$

The constraint in Eq. (20), ensures that any two clusters *i* and *j* among the first $c-2$ clusters, which have at least one execution segment ($k_i \times k_j > 0$), contain execution segments of different size where $C_i < C_j$ if $i < j$. Observe that for two clusters *i* and *j*, $C_i = C_j$ indicates that the two clusters should be merged into one cluster. The constraint in Eq. (21) allows merging the clusters $c-2$ and $c-1$. The reason for this is that while the CC method assigns predetermined values $C_i \setminus i \in [1, c-2]$ that satisfy Eq. (20), the values for C_{c-1} and C_c are calculated (see Eq. (25) and Eq. (27)). Next, we elaborate how the CC method explores the search space.

The CC method explores the search space in a systematic manner. It starts the exploration with a distribution $\overline{n}_c^0$ that is constructed as follows. For the first $c-2$ clusters, $k_i = 1 \setminus i \in [1, c-2]$ and $C_i = i \setminus i \in [1, c-2]$. The values for k_{c-1}, C_{c-1}, k_c and C_c are calculated with the following expressions:

$$\hat{T} = T - \sum_{i}^{c-2} k_i \times C_i \tag{22}$$

$$\hat{n}_c = n_c - \sum_{i=1}^{c-2} k_i \tag{23}$$

$$k_{c-1} = \hat{n}_c - \left(\hat{T} \text{ modulo } \hat{n}_c\right) \tag{24}$$

$$C_{c-1} = \left\lfloor \frac{\hat{T}}{\hat{n}_c} \right\rfloor \tag{25}$$

$$k_c = \hat{T} \text{ modulo } \hat{n}_c \tag{26}$$

$$C_c = \left\lceil \frac{\hat{T}}{\hat{n}_c} \right\rceil \tag{27}$$

The expression in Eq. (22), $\hat{T}$, represents the remaining processing time that has to be distributed among the last two clusters, while the expression in Eq. (23), $\hat{n}_c$, represents the remaining number of checkpoints (executions segments) that need to be distributed among the last two clusters. Equidistant distribution is applied to obtain the size of the execution segments that belong to the last two clusters. Observe that in case $\hat{T}$ is a multiple of $\hat{n}_c$, the cluster $c-1$ contains $k_{c-1} = \hat{n}_c$ execution segments, while the last cluster does not contain any execution segments, i.e. $k_c = 0$. In such case, the initial distribution $\overline{n}_c$ consists of $c-1$ clusters (the last cluster is empty).

To do the exploration in a systematic manner, we construct a function $nextCC(\overline{n}_c, q)$ which for a given distribution $\overline{n}_c$ generates the next distribution to be

explored (if such distribution exists). A distribution $\overline{n}_c$ is defined with the information about the clusters, i.e. k_i and C_i, where $i \in [1, c]$. To obtain the next distribution, $nextCC(\overline{n}_c, q)$ requires an input argument q which is referred to as the update index. The update index is used to indicate which cluster should be updated and assigned with predetermined values. Note that only the first $c-2$ can be updated, while the last two clusters are calculated (k_i and C_i for the last two clusters are calculated according to Eqs. (24–26)). Therefore, q can only accept values in the range $[1, c-2]$. The last two clusters are treated differently than the other clusters. These two clusters co-exist together, i.e. $k_{c-1} \times k_c > 0$, only if $\hat{T}$ (see Eq. (22)) is not a multiple of $\hat{n}_c$ (see Eq. (23)), and in such case the difference $C_c - C_{c-1}$ is exactly one. This follows from the fact that the size of the execution segments, which belong to the last two clusters, is obtained by equidistantly distributing the remaining $\hat{n}_c$ checkpoints over the remaining processing time $\hat{T}$. As a consequence of the equidistant distribution, even if the number of clusters c is equal to n_c, the CC method explores fewer distributions than the EXS method. Namely, each distribution $\overline{n}_c$, examined with the CC method, consisting of exactly $c = n_c$ clusters (for such case a cluster maps to an execution segment, i.e. $C_i \equiv ES_i$) will be constrained to $C_c - C_{c-1} = 1$, which is not the case for the EXS method, where the only constrain on the last two execution segments is that $ES_{n_c} - ES_{n_c-1} > 0$ (follows from Eq. (10)). Next, we detail how the function $nextCC(\overline{n}_c, q)$ generates the next distribution $\overline{n}_c$ to be explored.

The function $nextCC(\overline{n}_c, q)$ is recursive. If $nextCC(\overline{n}_c, q)$ is called with an update index q = 0, the function returns an empty set, indicating that the current distribution $\overline{n}_c$ is the last distribution found in the search space. For the given update index q, $nextCC(\overline{n}_c, q)$ tries to update the q-th cluster as long as $k_q > 0$. If the updates to the q-th cluster are not possible or if $k_q = 0$, the function recursively calls itself with an update index $q-1$. If $k_q > 0$, the function tries to update the q-th cluster. Two types of updates are possible: (1) update the size of the execution segments in the q-th cluster, i.e. C_q, and (2) update the number of execution segments that belong to q-th cluster, i.e. k_q. Each of the two updates involves updating all clusters j where $j > q$. This domino-effect follows from the constraints provided in Eqs. (20–21). First, $nextCC(\overline{n}_c, q)$ tries to update C_q. Only if the update of C_q is not successful, the function tries to update k_q. To update C_q, $nextCC(\overline{n}_c, q)$ performs the following steps. First, it assigns a new value to C_q, i.e. $C_q = C_q + 1$. Second, for all clusters $j \in [q+1, c-2]$, k_j is set to 1, and the size of the execution segments of each of the clusters $j \in [q+1, c-2]$ is evaluated as $C_j = C_{j-1} + 1$. Next, k_{c-1}, C_{c-1}, k_c and C_c are calculated according to Eqs. (24–26), and then the constraint in Eq. (21) is evaluated. If the constraint is not violated, then the update of C_q is successful, and the next distribution is provided with the new values of C_i and k_i, where $i \in [1, c]$. However, if the constraint is violated, it means that it is not possible to assign the new value for

C_q since it breaks the ascending order of the clusters (Eqs. (20–21)). The new value of C_q may still be possible if some clusters are allowed to have zero execution segments. Therefore, the update of C_q proceeds iteratively. Starting with $j = q + 1$, in each step, (1) k_j is set to zero, (2) $C_j = C_{j-1}$, (3) k_{c-1}, C_{c-1}, k_c and C_c are calculated according to Eqs. (24–26), and (4) the constraint in Eq. (21) is evaluated. If the constraint is not violated, the iteration stops and the update of C_q is rendered successful. However, if the constraint is violated, the iteration proceeds with the next step, by incrementing *j*. Finally, if $j > c - 2$, it means that in none of the iteration steps the constraint was satisfied and thereby, the update of C_q is not successful. In such case, the function $nextCC(\overline{n}_c, q)$ proceeds by trying to update k_q.

To update k_q, the function first checks the following condition:

$$n_c - \sum_{i=1}^{q} k_i > 1 \tag{28}$$

The condition in Eq. (28) checks if the remaining number of checkpoints (execution segments) that need to be distributed among the rest of the clusters, i.e. all clusters *j* where $j \in [q+1, c]$, is greater than one. Only if this condition is satisfied, it is possible to increase the number of execution segments that belong to the cluster *q*. Observe that comparing the remaining number of checkpoints with one, means that after increasing k_q it is possible that only one checkpoint (execution segment) remains. In such case, $k_j = 0$ for all $j \in [q+1, c-2], k_{c-1} = 1$, and $k_c = 0$. If the condition in Eq. (28) is not satisfied, it means that the update of k_q is not successful, and therefore, the function $nextCC(\overline{n}_c, q)$ recursively calls itself with an update index $q - 1$. However, if the condition in Eq. (28) is satisfied, then $k_q = k_q + 1$ and $C_q = C_{q-1}$. Note that the latest modifications, i.e. k_q and C_q, still do not represent the next distribution to be explored. To obtain the next distribution, after modifying k_q and C_q, the function $nextCC(\overline{n}_c, q)$ recursively calls itself with the same update index *q*, to ensure that C_q is greater than C_{q-1}. Observe that the function always first tries to update C_q. Important to note is that if *q* = 1, i.e. the updates refer to the first cluster, and the condition in Eq. (28) is satisfied, then C_1 is set to zero, $k_1 = k_1 + 1$, and the function $nextCC(\overline{n}_c, q)$ recursively calls itself with the same update index. This means that if the next distribution is obtained by increasing the number of execution segments in the first cluster, then the size of all the execution segments in the first cluster must be set to one.

So far, we discussed the function $nextCC(\overline{n}_c, q)$ and how it generates the next distribution provided an arbitrary update index *q* and a current distribution $\overline{n}_c$. However, when the CC method uses the function $nextCC(\overline{n}_c, q)$, to explore the search space, it always invokes the function with an update index $q = c - 2$ and the current distribution $\overline{n}_c$. Starting with the initial distribution $\overline{n}_c^0$, the CC method computes the LoC for $\overline{n}_c^0$ and keeps it as a reference point. Next, it repeatedly invokes the function $nextCC(\overline{n}_c, q)$ with $q = c - 2$, until $nextCC(\overline{n}_c, q)$ returns an empty set. For each

distribution generated by the function $nextCC(\overline{n}_c, q)$, the CC method evaluates the LoC and compares it with the reference point. If the LoC is higher than the reference point, then the reference point is updated and the distribution that has resulted with the higher LoC is recorded. Finally, the CC method outputs the distribution $\overline{n}_c^*$ that provides the highest LoC among all the distributions $\overline{n}_c$ that have been explored with this method. Important to note is that, the set of distributions $\overline{n}_c$ explored by the CC method for $c = x$, is superset of the distributions $\overline{n}_c$ explored for $c < x$. The CC method becomes equivalent to the EXS method, when $c = n_c + 1$.

Next, we discuss the complexity of the CC method. Note that by complexity, here, we refer to the size of the search space, i.e. the number of distributions $\overline{n}_c$ that are explored. The complexity of the CC method increases with the number of clusters c. The lowest complexity is achieved when the number of clusters is set to three. In such case, the number of distributions that are explored is limited with the following expression $\left\lfloor \frac{T}{n_c} \right\rfloor \times (n_c - 1)$. For the case c = 3, k_i and C_i are assigned with predetermined values only for the first cluster, while k_i and C_i are calculated for the other two clusters using Eqs. (24–26). When generating the distributions for c = 3, the function $nextCC(\overline{n}_c, q)$ is invoked with q = 1 and therefore, only updates of k_1 and C_1 are considered. Observe that if neither of the updates are possible, the function $nextCC(\overline{n}_c, q)$ would recursively call itself with an update index q = 0, which in turn would return an empty set and stop the search. Assuming that the first cluster already contains a number of execution segments k_1, the maximum value of C_1 is determined from the constraint given with Eq. (21). Note that for c = 3, C_2 is calculated with Eq. (25). Hence, the maximum size of C_1 is determined as follows:

$$C_1 \le \frac{T - k_1 \times C_1}{n_c - k_1} \Leftrightarrow C_1 \le \frac{T}{n_c} \tag{29}$$

The expression in Eq. (29) shows that for any value of k_1, C_1 can take values in the range $\left[1, \frac{T}{n_c}\right]$. On the other hand, if updates of C_1 are not possible, i.e. $C_1 = \frac{T}{n_c}$, updates of k_1 are only possible if the condition in Eq. (28) is satisfied. This condition provides the upper bound for k_1, and this bound is evaluated as $n_c - 1$. Therefore, the total number of distributions for c = 3 is evaluated as $\left\lfloor \frac{T}{n_c} \right\rfloor \times (n_c - 1)$.

The complexity of the CC method rapidly increases by increasing the number of clusters c. The reason for this is as follows. First, increasing the number of clusters c, increases the number of different cases to assign how many execution segments should belong to a given cluster, i.e. number of cases to assign k_i, where $i \in [1, c-2]$. Furthermore, the number of execution segments that are to be distributed among the first $c-2$ clusters may vary from one to $n_c - 1$. Second, once k_i is decided for each cluster $i \in [1, c-2]$, the number of cases to assign C_i, that satisfy the condition given

with Eq. (20), grows rapidly and therefore, the number of explored distributions grows (reaching the same complexity as EXS for $c = n_c + 1$) . However, the highlight is that the CC method finds the same distribution $\overline{n}_c^*$ as the EXS method, for sufficiently low values of *c*.

5.1.4 Summary

The objective of the experiments is multi-fold. First, evaluate to what extent the proposed heuristic (the CC method) is able to obtain the optimal solution (exhaustive search (EXS)). Second, evaluate which of the two schemes, i.e. equidistant (EQC) and non-equidistant checkpointing, provides the highest LoC. Third, evaluate how many clusters are needed for the CC method, to find a solution that approaches the optimal solution obtained by the EXS method. Finally, evaluate how the complexity of the CC method increases by increasing the number of clusters. We present results for the scenarios given in . Each scenario is defined with: a processing time *T*, a deadline *D*, a checkpointing overhead τ , and a probability P_T that no errors occur in a processing node within an interval of length *T*.

Scenario	T	D	τ	P_T
A	100 t.u.	150 t.u.	2 t.u.	0.99999
B	1000 t.u.	1300 t.u.	20 t.u.	0.99999
C	1000 t.u.	1300 t.u.	10 t.u.	0.99999

Table 5.3 Input scenarios

In --, for scenario A, B, and C respectively, we show the LoC (along with $\overline{n}_c$) obtained from the three methods, i.e. EQC, CC and EXS, for different values of n_c. Note that, for this experiment, the results presented for the CC method are obtained while assuming that the number of clusters *c*, which is an input argument to the CC method, is set to three. Observe that the range of n_c values is different for each scenario, i.e. $n_c \in [2, 25]$ in , $n_c \in [2, 15]$ in and $n_c \in [2,30]$ in . The upper bound of n_c, in all scenarios, is obtained from the condition that the best case execution time, i.e. time required for the job to complete when no errors occur $T + n_c \times \tau$, should not violate the deadline. Hence, for n_c values above the upper bound $\Lambda_{\overline{n}_c}(D) = 0$. Thus, we only compute $\Lambda_{\overline{n}_c}(D)$ for n_c values lower than the upper bound.

Since the values $\Lambda_{\overline{n}_c}(D)$ are very close to 1, the difference $\overline{\Lambda}_{\overline{n}_c}(D) = 1 - \Lambda_{\overline{n}_c}(D)$ results in numbers that are more convenient to present by using scientific notation. Therefore, in Tables IV–VI, we present the values $\overline{\Lambda}_{\overline{n}_c}(D)$. One observes that lower values for $\overline{\Lambda}_{\overline{n}_c}(D)$ are better (the LoC is higher). Furthermore, in Tables --, for each n_c, we present the distribution (in reduced format) $\widetilde{n}_c$ which provides the reported LoC for each method. For each n_c value in --, we highlight the highest LoC achieved (marked with bold). Next, we discuss the results.

As can be seen from –, comparing the results from the EXS method with the results from the CC method (with three clusters) shows that the CC method in most of the presented cases is able to find the optimal $\overline{n}_c^*$. Note that the EXS method

guarantees the optimal solution, since it explores all possible distributions $\overline{n}_c$. The advantage of the CC method, where only three clusters are used, is that it finds the solution in significantly shorter time than the EXS method. The reason that no results are reported for the EXS method for n_c values larger than five, in –, is that EXS is very time-consuming which is due to the large number of distributions $\overline{n}_c$ that need to be explored. Given that the average time to compute $\Lambda_{\overline{n}_c}(D)$ for a given $\overline{n}_c$ is 10µs, and that the number of different $\overline{n}_c$ (excluding $\overline{n}_c$ which are permutation of each other) for $n_c = 6$ is larger than 10^{11} (see for $n_c = 6$), it would roughly take 280 hours to obtain the result. In contrast, for the same example, the CC method, that only uses three clusters, explores less than 10^3 different $\overline{n}_c$ and thus, produces the result in less than 10ms. The CC method is a good heuristic since it finds, in most cases, the optimal $\overline{n}_c^*$ in substantially shorter time than the EXS method.

Comparing the results from the CC method with the results from the EQC method shows that the CC method is always able to find a distribution $\overline{n}_c$ which results in an LoC that is higher or at least equal to the LoC obtained from the EQC method (observe –). The EQC method distributes the checkpoints "evenly". However, due to the limitation of integer sized execution segments, when *T* is not a multiple of n_c, the EQC method reports the LoC for a distribution $\overline{n}_c$ which consists of two clusters. Observe that in , the EQC provides the same LoC for all $n_c \in [2,15]$. The reason for this is because distributing the checkpoints evenly, for any $n_c \in [2,15]$, results in execution segments of size such that even a single re-execution violates the deadline. However, for the same scenario, i.e. Scenario B, for any $n_c \in [2,13]$, the CC method, with three clusters, finds a distribution $\overline{n}_c$ which results in a higher LoC than the LoC obtained from the EQC method. Important to note, is that for n_c values which are close to the upper bound, both EQC and CC achieve the same LoC, although $\overline{n}_c$ is not same (observe $n_c \in [24,25]$ for , $n_c \in [14,15]$ for and $n_c \in [29,30]$ for). For such n_c values, no re-execution of execution segments of any size is possible without violating the deadline, and therefore all $\overline{n}_c$ produce the same LoC. The major observation is that for all scenarios in , the maximal LoC is achieved when the checkpoints are not evenly distributed. For example, in the maximal LoC is achieved for $n_c = 13$ while using $\widetilde{n}_c = [6^7, 9^2, 10^4]$, i.e. 7 execution segments of size 6 t.u., 2 execution segments of size 9 t.u., and 4 execution segments of size 10 t.u. The results indicate that non-equidistant checkpointing can improve the LoC when compared against EQC. An important implication is that by using non-equidistant checkpointing the LoC can be improved in addition to achieving a shorter best case execution time, i.e. completion time when no errors occur $T + n_c \times \tau$. For example, in , the maximal LoC for EQC is achieved for $n_c = 25$, but higher LoC can be achieved for $n_c \in [7,24]$ when non-equidistant checkpointing is used (the highest LoC is achieved for $n_c = 15$ with the CC method). The same observation follows from –. Important to note is that besides the scenarios presented in , we made experiments on different scenarios by varying *T*, τ, P_T, and *D*. For all experiments, we observed the same trend as shown

in –, i.e. non-equidistant checkpointing provides higher LoC than EQC. However, we present only the results for the scenarios given in .

n_c	EQC	Non-Equidistant Checkpointing		n_c	EQC	Non-Equidistant Checkpointing	
		EXS	CC			EXS	CC
2	1.99999e-5 $\tilde{n}_c=[50^2]$	**1.12000e-5** $\tilde{n}_c=[44^1,56^1]$	**1.12000e-5** $\tilde{n}_c=[44^1,56^1]$	14	1.63343e-15 $\tilde{n}_c=[7^{12},8^2]$	**1.57695e-15** $\tilde{n}_c=[4^3,8^{11}]$	**1.57695e-15** $\tilde{n}_c=[4^3,8^{11}]$
3	2.66678e-10 $\tilde{n}_c=[33^2,34^1]$	**2.55998e-10** $\tilde{n}_c=[20^1,40^2]$	**2.55998e-10** $\tilde{n}_c=[20^1,40^2]$	15	1.61335e-15 $\tilde{n}_c=[6^5,7^{10}]$	**1.57695e-15** $\tilde{n}_c=[3^4,8^{11}]$	**1.57695e-15** $\tilde{n}_c=[3^4,8^{11}]$
4	2.49998e-10 $\tilde{n}_c=[25^4]$	**2.09559e-10** $\tilde{n}_c=[19^2,31^2]$	**2.09559e-10** $\tilde{n}_c=[19^2,31^2]$	16	**1.59510e-15** $\tilde{n}_c=[6^{12},7^4]$	**1.59510e-15** $\tilde{n}_c=[6^{12},7^4]$	**1.59510e-15** $\tilde{n}_c=[6^{12},7^4]$
5	2.39998e-10 $\tilde{n}_c=[20^5]$	**1.12000e-10** $\tilde{n}_c=[18^4,28^1]$	**1.12000e-10** $\tilde{n}_c=[18^4,28^1]$	17	**1.57863e-15** $\tilde{n}_c=[5^2,6^{15}]$	**1.57863e-15** $\tilde{n}_c=[5^2,6^{15}]$	**1.57863e-15** $\tilde{n}_c=[5^2,6^{15}]$
6	**2.07477e-15** $\tilde{n}_c=[16^2,17^4]$	**2.07477e-15** $\tilde{n}_c=[16^2,17^4]$	**2.07477e-15** $\tilde{n}_c=[16^2,17^4]$	18	1.75199e-10 $\tilde{n}_c=[5^8,6^{10}]$	**7.60007e-11** $\tilde{n}_c=[5^{16},10^2]$	**7.60007e-11** $\tilde{n}_c=[5^{16},10^2]$
7	1.95991e-15 $\tilde{n}_c=[14^5,15^2]$	**1.95161e-15** $\tilde{n}_c=[7^1,15^3,16^3]$	**1.95161e-15** $\tilde{n}_c=[7^1,15^3,16^3]$	19	2.10599e-10 $\tilde{n}_c=[5^{14},6^5]$	**1.36000e-10** $\tilde{n}_c=[4^{15},10^4]$	**1.36000e-10** $\tilde{n}_c=[4^{15},10^4]$
8	1.87599e-15 $\tilde{n}_c=[12^4,13^4]$	**1.83839e-15** $\tilde{n}_c=[7^2,14^4,15^2]$	**1.83839e-15** $\tilde{n}_c=[7^2,14^4,15^2]$	20	2.09999e-10 $\tilde{n}_c=[5^{20}]$	**1.84319e-10** $\tilde{n}_c=[3^{12},8^8]$	**1.84319e-10** $\tilde{n}_c=[3^{12},8^8]$
9	1.81113e-15 $\tilde{n}_c=[11^8,12^1]$	**1.70906e-15** $\tilde{n}_c=[6^3,13^2,14^4]$	**1.70906e-15** $\tilde{n}_c=[6^3,13^2,14^4]$	21	2.09599e-10 $\tilde{n}_c=[4^5,5^{16}]$	**2.07479e-10** $\tilde{n}_c=[2^6,5^2,6^{13}]$	**2.07479e-10** $\tilde{n}_c=[2^6,5^2,6^{13}]$
10	1.75999e-15 $\tilde{n}_c=[10^{10}]$	**1.55958e-15** $\tilde{n}_c=[8^6,13^4]$	**1.55958e-15** $\tilde{n}_c=[8^6,13^4]$	22	1.20000e-5 $\tilde{n}_c=[4^{10},5^{12}]$	**3.20015e-6** $\tilde{n}_c=[4^{21},16^1]$	**3.20015e-6** $\tilde{n}_c=[4^{21},16^1]$
11	1.71943e-15 $\tilde{n}_c=[9^{10},10^1]$	**1.58463e-15** $\tilde{n}_c=[5^4,8^1,12^6]$	1.58908e-15 $\tilde{n}_c=[5^4,11^4,12^3]$	23	1.99999e-5 $\tilde{n}_c=[4^{15},5^8]$	**1.12000e-5** $\tilde{n}_c=[2^{22},56^1]$	**1.12000e-5** $\tilde{n}_c=[2^{22},56^1]$
12	1.68642e-15 $\tilde{n}_c=[8^8,9^4]$	**1.53599e-15** $\tilde{n}_c=[5^4,10^8]$	**1.53599e-15** $\tilde{n}_c=[5^4,10^8]$	24	**1.99999e-5** $\tilde{n}_c=[4^{20},5^4]$	**1.99999e-5** $\tilde{n}_c=[4^{23},8^1]$	**1.99999e-5** $\tilde{n}_c=[4^{23},8^1]$
13	1.65807e-15 $\tilde{n}_c=[7^4,8^9]$	**1.53236e-15** $\tilde{n}_c=[6^7,9^2,10^4]$	**1.53236e-15** $\tilde{n}_c=[6^7,9^2,10^4]$	25	**1.99999e-5** $\tilde{n}_c=[4^{25}]$	**1.99999e-5** $\tilde{n}_c=[4^{25}]$	**1.99999e-5** $\tilde{n}_c=[4^{25}]$

Table 5.4 Comparison of $\overline{\Lambda}_{\tilde{n}_c}(D)$ for different checkpointing schemes at different n_c values for Scenario A*

***Note:** Since $\overline{\Lambda}_{\tilde{n}_c}(D)=1-\Lambda_{\tilde{n}_c}(D)$, lower values for $\overline{\Lambda}_{\tilde{n}_c}(D)$) indicate higher LoC. For each n_c , for each method, $\overline{\Lambda}_{\tilde{n}_c}(D)$ along with the respective distribution $\tilde{n}_c$ (in reduced format) is reported. The highest LoC achieved among the tree methods is marked with bold.

n_c	EQC	Non-Equidistant Checkpointing		n_c	EQC	Non-Equidistant Checkpointing	
		EXS	CC			EXS	CC
2	1.99999e-5 $\tilde{n}_c = [500^2]$	**1.51999e-5** $\tilde{n}_c = [240^1, 760^1]$	**1.51999e-5** $\tilde{n}_c = [240^1, 760^1]$	9	1.99999e-5 $\tilde{n}_c = [111^8, 112^1]$	$\varnothing$	**4.00015e-6** $\tilde{n}_c = [100^8, 200^1]$
3	1.99999e-5 $\tilde{n}_c = [333^2, 334^1]$	**1.12000e-5** $\tilde{n}_c = [220^2, 560^1]$	**1.12000e-5** $\tilde{n}_c = [220^2, 560^1]$	10	1.99999e-5 $\tilde{n}_c = [100^{10}]$	$\varnothing$	**5.60012e-6** $\tilde{n}_c = [80^9, 280^1]$
4	1.99999e-5 $\tilde{n}_c = [250^4]$	**8.00010e-6** $\tilde{n}_c = [200^3, 400^1]$	**8.00010e-6** $\tilde{n}_c = [200^3, 400^1]$	11	1.99999e-5 $\tilde{n}_c = [90^1, 91^{10}]$	$\varnothing$	**8.00008e-6** $\tilde{n}_c = [60^{10}, 400^1]$
5	1.99999e-5 $\tilde{n}_c = [200^5]$	**5.60014e-6** $\tilde{n}_c = [180^4, 280^1]$	**5.60014e-6** $\tilde{n}_c = [180^4, 280^1]$	12	1.99999e-5 $\tilde{n}_c = [83^8, 84^4]$	$\varnothing$	**1.12000e-5** $\tilde{n}_c = [40^{11}, 560^1]$
6	1.99999e-5 $\tilde{n}_c = [166^2, 167^4]$	$\varnothing$	**4.00016e-6** $\tilde{n}_c = [160^5, 200^1]$	13	1.99999e-5 $\tilde{n}_c = [76^1, 77^{12}]$	$\varnothing$	**1.51999e-5** $\tilde{n}_c = [20^{12}, 760^1]$
7	1.99999e-5 $\tilde{n}_c = [142^1, 143^6]$	$\varnothing$	**3.20017e-6** $\tilde{n}_c = [140^6, 160^1]$	14	**1.99999e-5** $\tilde{n}_c = [71^8, 72^6]$	$\varnothing$	**1.99999e-5** $\tilde{n}_c = [71^{13}, 77^1]$
8	1.99999e-5 $\tilde{n}_c = [125^8]$	$\varnothing$	**3.20017e-6** $\tilde{n}_c = [120^7, 160^1]$	15	**1.99999e-5** $\tilde{n}_c = [66^5, 67^{10}]$	$\varnothing$	**1.99999e-5** $\tilde{n}_c = [66^{14}, 76^1]$

Table 5.5 Comparison of $\overline{\Lambda}_{\tilde{n}_c}(D)$ for different checkpointing schemes at different n_c values for Scenario B*

***Note:** Since $\overline{\Lambda}_{\tilde{n}_c}(D) = 1 - \Lambda_{\tilde{n}_c}(D)$, lower values for $\overline{\Lambda}_{\tilde{n}_c}(D)$) indicate higher LoC. For each n_c, for each method, $\overline{\Lambda}_{\tilde{n}_c}(D)$ along with the respective distribution $\tilde{n}_c$ (in reduced format) is reported. The highest LoC achieved among the tree methods is marked with bold.

n_c	EQC	Non-Equidistant Checkpointing		n_c	EQC	Non-Equidistant Checkpointing	
		EXS	CC			EXS	CC
2	1.99999e-5 $\tilde{n}_c=[500^2]$	**1.45999e-5** $\tilde{n}_c=[270^1,730^1]$	**1.45999e-5** $\tilde{n}_c=[270^1,730^1]$	17	2.11764e-10 $\tilde{n}_c=[58^3,59^{14}]$	∅	**4.80010e-11** $\tilde{n}_c=[55^{16},120^1]$
3	1.99999e-5 $\tilde{n}_c=[333^2,334^1]$	**9.60008e-6** $\tilde{n}_c=[260^2,480^1]$	**9.60008e-6** $\tilde{n}_c=[260^2,480^1]$	18	2.11111e-10 $\tilde{n}_c=[55^8,56^{10}]$	∅	**7.60007e-11** $\tilde{n}_c=[50^{16},100^2]$
4	**2.49998e-10** $\tilde{n}_c=[250^4]$	**2.49998e-10** $\tilde{n}_c=[250^4]$	**2.49998e-10** $\tilde{n}_c=[250^4]$	19	2.10526e-10 $\tilde{n}_c=[52^7,53^{12}]$	∅	**1.01547e-10** $\tilde{n}_c=[45^{16},93^2,94^1]$
5	2.39998e-10 $\tilde{n}_c=[200^5]$	**2.36515e-10** $\tilde{n}_c=[115^1,221^3,222^1]$	**2.36515e-10** $\tilde{n}_c=[115^1,221^3,222^1]$	20	2.09999e-10 $\tilde{n}_c=[50^{20}]$	∅	**1.24560e-10** $\tilde{n}_c=[40^{16},90^4]$
6	2.33332e-10 $\tilde{n}_c=[166^2,167^4]$	∅	**2.08146e-10** $\tilde{n}_c=[110^3,223^2,224^1]$	21	2.09524e-10 $\tilde{n}_c=[47^8,48^{13}]$	∅	**1.52395e-10** $\tilde{n}_c=[35^{15},79^5,80^1]$
7	2.28570e-10 $\tilde{n}_c=[142^1,143^6]$	∅	**1.87146e-10** $\tilde{n}_c=[105^4,193^2,194^1]$	22	2.09091e-10 $\tilde{n}_c=[45^{12},46^{10}]$	∅	**1.77848e-10** $\tilde{n}_c=[30^{13},67^2,68^7]$
8	2.24999e-10 $\tilde{n}_c=[125^8]$	∅	**1.44000e-10** $\tilde{n}_c=[100^6,200^2]$	23	2.08696e-10 $\tilde{n}_c=[43^{12},44^{11}]$	∅	**1.96153e-10** $\tilde{n}_c=[25^{10},57^4,58^9]$
9	2.22221e-10 $\tilde{n}_c=[111^8,112^1]$	∅	**1.22777e-10** $\tilde{n}_c=[95^7,167^1,168^1]$	24	2.08333e-10 $\tilde{n}_c=[41^8,42^{16}]$	∅	**2.05724e-10** $\tilde{n}_c=[20^6,48^2,49^{16}]$
10	2.19999e-10 $\tilde{n}_c=[100^{10}]$	∅	**7.60008e-11** $\tilde{n}_c=[90^9,190^1]$	25	**2.07999e-10** $\tilde{n}_c=[40^{25}]$	∅	**2.07999e-10** $\tilde{n}_c=[40^{25}]$
11	2.18181e-10 $\tilde{n}_c=[90^1,91^{10}]$	∅	**6.00009e-11** $\tilde{n}_c=[85^{10},150^1]$	26	1.99999e-5 $\tilde{n}_c=[38^{14},39^{12}]$	∅	**5.00012e-6** $\tilde{n}_c=[30^{25},250^1]$
12	2.16666e-10 $\tilde{n}_c=[83^8,84^4]$	∅	**4.80011e-11** $\tilde{n}_c=[80^{11},120^1]$	27	1.99999e-5 $\tilde{n}_c=[37^{26},38^1]$	∅	**9.60005e-6** $\tilde{n}_c=[20^{26},480^1]$
13	2.15383e-10 $\tilde{n}_c=[76^1,77^{12}]$	∅	**4.00011e-11** $\tilde{n}_c=[75^{12},100^1]$	28	1.99999e-5 $\tilde{n}_c=[35^8,36^{20}]$	∅	**1.45999e-5** $\tilde{n}_c=[10^{27},730^1]$
14	2.14285e-10 $\tilde{n}_c=[71^8,72^6]$	∅	**3.60012e-11** $\tilde{n}_c=[70^{13},90^1]$	29	**1.99999e-5** $\tilde{n}_c=[34^{15},35^{14}]$	∅	**1.99999e-5** $\tilde{n}_c=[34^{28},48^1]$
15	2.13333e-10 $\tilde{n}_c=[66^5,67^{10}]$	∅	**3.60011e-11** $\tilde{n}_c=[65^{14},90^1]$	30	**1.99999e-5** $\tilde{n}_c=[33^{20},34^{10}]$	∅	**1.99999e-5** $\tilde{n}_c=[33^{29},43^1]$
16	2.12500e-10 $\tilde{n}_c=[62^8,63^8]$	∅	**4.00011e-11** $\tilde{n}_c=[60^{15},100^1]$	/			

Table 5.6 Comparison of $\overline{\Lambda}_{\tilde{n}_c}(D)$ for different checkpointing schemes at different n_c values for Scenario C*

***Note:** Since $\overline{\Lambda}_{\tilde{n}_c}(D)=1-\Lambda_{\tilde{n}_c}(D)$, lower values for $\overline{\Lambda}_{\tilde{n}_c}(D)$) indicate higher LoC. For each n_c, for each method, $\overline{\Lambda}_{\tilde{n}_c}(D)$ along with the respective distribution $\tilde{n}_c$ (in reduced format) is reported. The highest LoC achieved among the tree methods is marked with bold.

Next, we examine how many clusters are sufficient for the CC method, such that it finds a solution that approaches the optimal one. –, respective to each of the scenarios in , show the result of the CC method when the number of clusters c varies from three to five. Note that since the CC method requires $c \le n_c+1$, for those cases that violate

this constraint no results are presented (marked with "$\varnothing$" in –). For example, it is not possible to run the CC method with arguments c = 5 and n_c = 2. Since it is possible that the CC method with c_1 clusters finds the same solution as with c_2 clusters, where $c_1 > c_2$, in –, we use the symbol "$\leftarrow$" to indicate that the solutions are identical. As can be seen from –, in most cases, the CC method does not need more than three clusters to find a solution that approaches the optimal one, i.e. the CC method with c = 4 and c = 5 provide the same solution as with c = 3 (therefore, marked with "$\leftarrow$"). The only exception is visible in , where for n_c = 11, the CC method requires four clusters to find the optimal solution (compare this result with the result for the EXS method in). From the results presented in –, we conclude that the CC method finds a solution very close to the optimal, if not optimal, by only using three or four clusters. Since for low number of clusters, the complexity of the CC method is significantly lower than the complexity of the EXS method, once again, we show that the CC method is an efficient and effective heuristic.

n_c	Clustered Checkpointing			n_c	Clustered Checkpointing		
	$c = 3$	$c = 4$	$c = 5$		$c = 3$	$c = 4$	$c = 5$
2	**1.12000e-5** $\widetilde{n}_c = [44^1, 56^1]$	$\varnothing$	$\varnothing$	14	**1.57695e-15** $\widetilde{n}_c = [4^3, 8^{11}]$	$\leftarrow$	$\leftarrow$
3	**2.55998e-10** $\widetilde{n}_c = [20^1, 40^2]$	$\leftarrow$	$\varnothing$	15	**1.57695e-15** $\widetilde{n}_c = [3^4, 8^{11}]$	$\leftarrow$	$\leftarrow$
4	**2.09559e-10** $\widetilde{n}_c = [19^2, 31^2]$	$\leftarrow$	$\leftarrow$	16	**1.59510e-15** $\widetilde{n}_c = [6^{12}, 7^4]$	$\leftarrow$	$\leftarrow$
5	**1.12000e-10** $\widetilde{n}_c = [18^4, 28^1]$	$\leftarrow$	$\leftarrow$	17	**1.57863e-15** $\widetilde{n}_c = [5^2, 6^{15}]$	$\leftarrow$	$\leftarrow$
6	**2.07477e-15** $\widetilde{n}_c = [16^2, 17^4]$	$\leftarrow$	$\leftarrow$	18	**7.60007e-11** $\widetilde{n}_c = [5^{16}, 10^2]$	$\leftarrow$	$\leftarrow$
7	**1.95161e-15** $\widetilde{n}_c = [7^1, 15^3, 16^3]$	$\leftarrow$	$\leftarrow$	19	**1.36000e-10** $\widetilde{n}_c = [4^{15}, 10^4]$	$\leftarrow$	$\leftarrow$
8	**1.83839e-15** $\widetilde{n}_c = [7^2, 14^4, 15^2]$	$\leftarrow$	$\leftarrow$	20	**1.84319e-10** $\widetilde{n}_c = [3^{12}, 8^8]$	$\leftarrow$	$\leftarrow$
9	**1.70906e-15** $\widetilde{n}_c = [6^3, 13^2, 14^4]$	$\leftarrow$	$\leftarrow$	21	**2.07479e-10** $\widetilde{n}_c = [2^6, 5^2, 6^{13}]$	$\leftarrow$	$\leftarrow$
10	**1.55958e-15** $\widetilde{n}_c = [8^6, 13^4]$	$\leftarrow$	$\leftarrow$	22	**3.20015e-6** $\widetilde{n}_c = [4^{21}, 16^1]$	$\leftarrow$	$\leftarrow$
11	1.58908e-15 $\widetilde{n}_c = [5^4, 11^4, 12^3]$	**1.58463e-15** $\widetilde{n}_c = [5^4, 8^1, 12^6]$	$\leftarrow$	23	**1.12000e-5** $\widetilde{n}_c = [2^{22}, 56^1]$	$\leftarrow$	$\leftarrow$
12	**1.53599e-15** $\widetilde{n}_c = [5^4, 10^8]$	$\leftarrow$	$\leftarrow$	24	**1.99999e-5** $\widetilde{n}_c = [4^{23}, 8^1]$	$\leftarrow$	$\leftarrow$
13	**1.53236e-15** $\widetilde{n}_c = [6^7, 9^2, 10^4]$	$\leftarrow$	$\leftarrow$	25	**1.99999e-5** $\widetilde{n}_c = [4^{25}]$	$\leftarrow$	$\leftarrow$

Table 5.7 Comparison of $\overline{\Lambda}_{\overline{n}_c}(D)$ for the CC method with varying number of clusters $c \in [3,5]$, at different n_c values, for Scenario A

n_c	Clustered Checkpointing			n_c	Clustered Checkpointing		
	$c=3$	$c=4$	$c=5$		$c=3$	$c=4$	$c=5$
2	**1.51999e-5** $\tilde{n}_c=[240^1,760^1]$	$\varnothing$	$\varnothing$	9	**4.00015e-6** $\tilde{n}_c=[100^8,200^1]$	←	←
3	**1.12000e-5** $\tilde{n}_c=[220^2,560^1]$	←	$\varnothing$	10	**5.60012e-6** $\tilde{n}_c=[80^9,280^1]$	←	←
4	**8.00010e-6** $\tilde{n}_c=[200^3,400^1]$	←	←	11	**8.00008e-6** $\tilde{n}_c=[60^{10},400^1]$	←	←
5	**5.60014e-6** $\tilde{n}_c=[180^4,280^1]$	←	←	12	**1.12000e-5** $\tilde{n}_c=[40^{11},560^1]$	←	←
6	**4.00016e-6** $\tilde{n}_c=[160^5,200^1]$	←	←	13	**1.51999e-5** $\tilde{n}_c=[20^{12},760^1]$	←	←
7	**3.20017e-6** $\tilde{n}_c=[140^6,160^1]$	←	←	14	**1.99999e-5** $\tilde{n}_c=[71^{13},77^1]$	←	←
8	**3.20017e-6** $\tilde{n}_c=[120^7,160^1]$	←	←	15	**1.99999e-5** $\tilde{n}_c=[66^{14},76^1]$	←	←

Table 5.8 Comparison of $\overline{\Lambda}_{\bar{n}_c}(D)$ for the CC method with varying number of clusters $c \in [3,5]$, at different n_c values, for Scenario B

n_c	Clustered Checkpointing			n_c	Clustered Checkpointing		
	$c=3$	$c=4$	$c=5$		$c=3$	$c=4$	$c=5$
2	**1.45999e-5** $\tilde{n}_c=[270^1,730^1]$	∅	∅	17	**4.80010e-11** $\tilde{n}_c=[55^{16},120^1]$	←	←
3	**9.60008e-6** $\tilde{n}_c=[260^2,480^1]$	←	∅	18	**7.60007e-11** $\tilde{n}_c=[50^{16},100^2]$	←	←
4	**2.49998e-10** $\tilde{n}_c=[250^4]$	←	←	19	**1.01547e-10** $\tilde{n}_c=[45^{16},93^2,94^1]$	←	←
5	**2.36515e-10** $\tilde{n}_c=[115^1,221^3,222^1]$	←	←	20	**1.24560e-10** $\tilde{n}_c=[40^{16},90^4]$	←	←
6	**2.08146e-10** $\tilde{n}_c=[110^3,223^2,224^1]$	←	←	21	**1.52395e-10** $\tilde{n}_c=[35^{15},79^5,80^1]$	←	←
7	**1.87146e-10** $\tilde{n}_c=[105^4,193^2,194^1]$	←	←	22	**1.77848e-10** $\tilde{n}_c=[30^{13},67^2,68^7]$	←	←
8	**1.44000e-10** $\tilde{n}_c=[100^6,200^2]$	←	←	23	**1.96153e-10** $\tilde{n}_c=[25^{10},57^4,58^9]$	←	←
9	**1.22777e-10** $\tilde{n}_c=[95^7,167^1,168^1]$	←	←	24	**2.05724e-10** $\tilde{n}_c=[20^6,48^2,49^{16}]$	←	←
10	**7.60008e-11** $\tilde{n}_c=[90^9,190^1]$	←	←	25	**2.07999e-10** $\tilde{n}_c=[40^{25}]$	←	←
11	**6.00009e-11** $\tilde{n}_c=[85^{10},150^1]$	←	←	26	**5.00012e-6** $\tilde{n}_c=[30^{25},250^1]$	←	←
12	**4.80011e-11** $\tilde{n}_c=[80^{11},120^1]$	←	←	27	**9.60005e-6** $\tilde{n}_c=[20^{26},480^1]$	←	←
13	**4.00011e-11** $\tilde{n}_c=[75^{12},100^1]$	←	←	28	**1.45999e-5** $\tilde{n}_c=[10^{27},730^1]$	←	←
14	**3.60012e-11** $\tilde{n}_c=[70^{13},90^1]$	←	←	29	**1.99999e-5** $\tilde{n}_c=[34^{28},48^1]$	←	←
15	**3.60011e-11** $\tilde{n}_c=[65^{14},90^1]$	←	←	30	**1.99999e-5** $\tilde{n}_c=[33^{29},43^1]$	←	←
16	**4.00011e-11** $\tilde{n}_c=[60^{15},100^1]$	←	←	/			

Table 5.9 Comparison of $\overline{\Lambda}_{\bar{n}_c}(D)$ for the CC method with varying number of clusters $c \in [3,5]$, at different n_c values, for Scenario C

Finally, with the results of the following experiment, we show how the complexity of the CC method increases by increasing the number of clusters *c*. The complexity of the CC method depends on the following parameters: the number of checkpoints n_c, the number of clusters *c*, and the processing time of the job *T*. In and , for *T* = 100 and T = 1000 respectively, for $n_c \in [2,25]$, we show the number of distributions $\overline{n}_c$ explored by the CC method for various number of clusters $c \in [3,5]$. As can be seen from and , the complexity of the CC method increases rapidly by increasing the

number of clusters. For the case of T = 100 t.u. (), increasing the number of clusters from c = 3 to c = 4, on average, increases the number of explored distributions by a factor of 29, while increasing the number of clusters from c = 3 to c = 5, on average, increases the number of explored distributions by a factor of 414. For example, for n_c = 10, the CC method explores: 84 distributions for c = 3, 2790 distributions for c = 4 (33 times more than for c = 3), and 42095 distributions for c = 5 (501 times more than for c = 3). The increase is even more evident for the case of T = 1000 t.u., where increasing the number of clusters from c = 3 to c = 4, on average, increases the number of explored distributions by a factor of 344, while increasing the number of clusters from c = 3 to c = 5, on average, increases the number of explored distributions by a factor of 63475.

n_c	Clustered Checkpointing			n_c	Clustered Checkpointing		
	$c = 3$	$c = 4$	$c = 5$		$c = 3$	$c = 4$	$c = 5$
2	50	∅	∅	14	91	2837	43151
3	66	834	∅	15	84	2841	43779
4	75	1411	7171	16	90	2845	44227
5	80	1803	14149	17	80	2863	44241
6	80	2097	20507	18	85	2800	44343
7	84	2297	25800	19	90	2791	43908
8	84	2473	30299	20	95	2879	43829
9	88	2573	33796	21	80	2749	43201
10	90	2686	36811	22	84	2718	42578
11	90	2709	38952	23	88	2674	41643
12	88	2760	40933	24	92	2698	41117
13	84	2790	42095	25	96	2731	40307

Table 5.10 Number of distributions explored by the CC method with varying number of clusters $c \in [3,5]$, at different n_c values, for T = 100 t.u.

n_c	Clustered Checkpointing			n_c	Clustered Checkpointing		
	$c = 3$	$c = 4$	$c = 5$		$c = 3$	$c = 4$	$c = 5$
2	500	∅	∅	14	923	343439	63492695
3	666	83334	∅	15	924	350572	66536434
4	750	145336	6965446	16	930	356856	69351150
5	800	190503	15157744	17	928	361845	71870844
6	830	224697	23074067	18	935	366744	74231622
7	852	251131	30284096	19	936	370953	76359523
8	875	272565	36756775	20	950	375573	78383716
9	888	289671	42508054	21	940	378045	80164600
10	900	304399	47662911	22	945	381458	81890868
11	900	316264	52247333	23	946	384400	83428392
12	913	326804	56399776	24	943	387451	84921115
13	912	335670	60101201	25	960	389768	86249061

Table 5.11Number of distributions explored by the CC method with varying number of clusters $c \in [3,5]$, at different n_c values, for T = 1000 t.u.

5.2 Fault Handling Methodology

When a fault occurs in a complex SoC working under the control of an OS, it is necessary that the latter becomes aware of the fault as quickly as possible. The OS must then take actions to isolate and mitigate the effects of the fault.

There are several ways available to detect a fault in a resource of the system: such as OS-controlled temporal and modular redundancy of task execution, application-side data and program flow integrity checks, various Power-On Self-Test (POST) checks and others. In the following, we concentrate on the faults, which are detected on-line (during normal operation of the system) by the built-in instruments.

For situation when a fault is detected, there must be a method for passing the relevant information up to the OS level where the reaction to the fault will be taken. In BASTION D2.3 we described an architectural extension to the IJTAG, which is suitable for system health data collection and emergency signaling, hence facilitating the Fault Management (FM) functions. FM is further facilitated by the results of the fault classification based on the categories described in D3.1, the information from instruments as well as the accumulated fault statistics stored in the Health Map.

For a generic case of a fault being detected by an instrument, the flowchart with four actors (Instrument, Instrument Manager, Fault Manager and Operating System) is shown in Figure 5.1. In the following subsections we describe the process in more detail.

5.2.1 Error Classification

Ability to classify errors, malfunctions and faults is an important basis for health map management, effective system recovery and fault management. We propose HW architectures and SW support for the on-line in the field fault classification and health map maintenance. In the process of physical degradation of the system, the observed fault progresses from intermittent towards a permanent one.

In D3.1 we classify the faults by several properties (persistence, severity, criticality, diagnostic granularity and location) according to their severity levels and their contribution to the permanent malfunction of system's components and modules. The classification has a strong relation to fault management processes and the architecture of the Health Map. Taking into account the information from the health map, the FM 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.

5.2.2 Fault detection

Whenever a fault is detected by an instrument, the information about this event is quickly passed to the IM by asynchronous propagation of the Fault and Corrected flags (see D2.3). In response, IM sends an interrupt request to a CPU which is executing the OS kernel, which, in turn invokes the FM to service this interrupt.

5.2.3 Instrument Manager watchdog

It is possible that due to the fault, the CPU or the OS is not reacting to the interrupt request. For this case, IM may employ a watchdog which monitors the ability of the system to respond to the fault occurrence event. It is set when an interrupt is sent and should be reset by the fault management Interrupt Service Routine (ISR), indicating the reaction from the system. It is assumed that if the system does not respond to the interrupt request of IM, the fault must have occurred in a

critical location and the only way to resume the operation is to reset system. After the reset, the system may check if the source of the reset was the fault management watchdog.

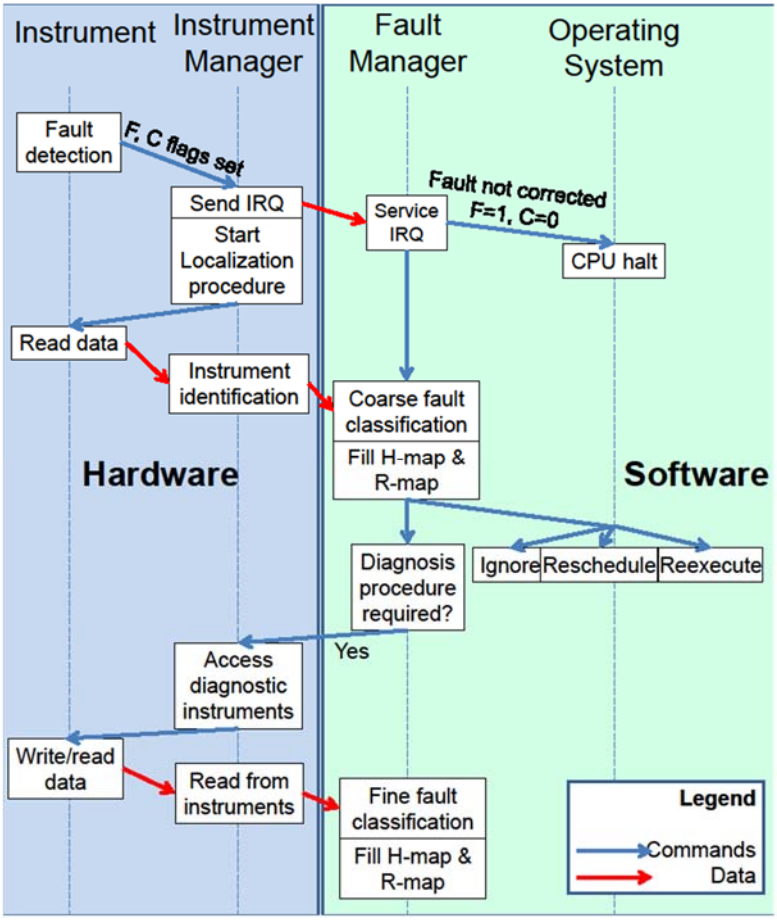

Figure 5.1. Fault handling flowchart

5.2.4 Fault localization

Concurrently with sending the interrupt request, IM starts the instrument localization procedure. During this procedure, IM will subsequently open the hierarchical IJTAG network segments with the Fault flag set (see D2.3). When an instrument will be reached, IM can report the location of the fault as the position in the IJTAG network.

5.2.5 Coarse-grained fault classification

Based on the information about which instrument has raised the Fault flag, the FM can perform coarse classification of the fault. Since the detailed diagnostic information about the fault is not available at this time, the diagnostic granularity is limited by the granularity of the instrument location (e.g. it is attached to a CPU or FPU, or some particular checker) and the fault location is of the first type. However, in some cases (as in the example in the next section) the criticality of the resource can be determined. In any case, the Health and the Resource Maps should be updated with the obtained information.

5.2.6 System response

Based on the information derived in the previous step, the OS may need to take actions to mitigate the effects of the fault on the functional operation of the system. The fault can be ignored if it does not affect the operation or the task was not critical. Alternatively, the task can be rescheduled to another resource or re-executed on the same one.

5.2.7 Diagnosis

Depending on the outcome of the coarse classification step, the FM system may decide to get more detailed diagnosis information and for this the diagnostic procedure is launched. In case some information must be sent or received to/from the diagnostic instruments (such as Built-In Self Test (BIST) or other Design for Test (DfT) hardware), the IJTAG network may be used. IM is then instructed to exchange data with the instrument.

5.2.8 Fine-grained fault classification

If some additional information is acquired as a result of the diagnostic procedure, FM can perform fine fault classification and update both Health and Resource Maps. The Health Map composition together with respective fault classification scheme, and initial fault handling scenario have been detailed in D3.1.

5.3 Example scenario: processor cache

To demonstrate our proposed fault handling flow in action, we devised an example scenario of fault occurrence and its handling in a cache memory of a CPU core. In this example, the CPU is one of several similar cores inside a larger SoC, it has L1 and L2 caches which contain error detection and correction logic in their controllers. This logic acts as an instrument for the FMI purposes and the instruments of all caches are connected to the IJTAG network. The L2 cache can be dynamically resized in case some of the locations become faulty [16]. At the moment when the fault occurs, the CPU is executing a task without strict deadlines.

According to the system policy, when a soft error occurs, the cache is invalidated, but not disabled. However, when the fault is classified as permanent, the faulty part of the cache is disabled.

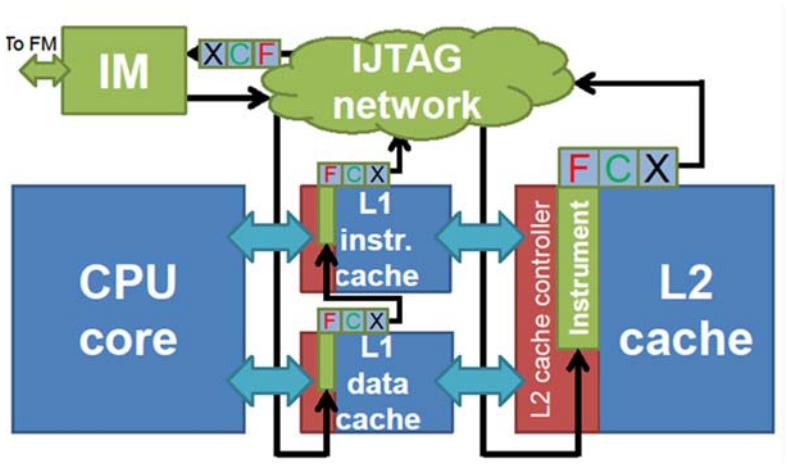

Figure 5.2. Processor with caches connected to IJTAG instrumentation network

The fault is detected, but not recovered by the ECC checker inside the cache controller. The checker is also an instrument in the IJTAG network of the SoC and supports the asynchronous Fault and Correct flags (see Figure 5.2). As soon as the fault is detected, the flags are set and the fault handling mechanism is engaged. As a result, the following series of events unfolds:

1. ECC checker in the CPU L2 cache controller detects an unrecoverable error and sets the flags to F=1 and C=0
2. The state of flags reaches the IM via the asynchronous propagation network. IM immediately sends a dedicated interrupt request to the CPU and simultaneously starts the instrument localization procedure. During this procedure, IM subsequently opens the hierarchical levels of IJTAG network until it reaches the instrument which raised the Fault flag.
3. CPU is interrupted by request from IM and the FM ISR is called to process the information. Since the error is not corrected, execution of the application code must be halted. CPU then waits within the FM ISR until the instrument localization data arrives.
4. Instrument has been localized by IM and the information is passed to FM. The instrument in question is the ECC checker in L2 cache.
5. Having the information, FM can now coarsely classify the occurred fault. It identifies the resource criticality as high (cache integrity is essential for correct program execution). According to the fault statistics, this module has already suffered from faults in the past and as a result, decision is made to disable it and perform the diagnostic procedure. The L2 cache is marked faulty in the Resource Map. Routinely, the fault statistics is updated too.
6. Since the task does not have strict deadlines, it is re-executed on the same CPU core, but with the disabled cache.
7. FM launches the diagnostic procedure. Cache controller has its own BIST, which can be used for acquiring detailed information. FM communicates with IM to access the BIST through the IJTAG network.
8. BIST has produced the results which show that one half of the L2 cache is still operational. This data is transported to FM.
9. FM updates the fault statistics for the faulty half of the L2 cache and stores a permanent fault with maximum severity bbbas the current state of that module. Resource Map is updated with the information about the reduced capacity of L2 cache in that CPU core. Thus, fine-grained classification is performed.

6 Rejuvenation

In this Section we describe a method developed by the BASTION partners, facing faults produced by aging effects via a technique, which is called *rejuvenation*. This technique aims at slowing down the aging effect by applying suitable input stimuli. A method is described, to generate such a stimuli. Some preliminary experimental results are reported.

6.1 Introduction

A common understanding of the NBTI process in the scientific community is still a hot research subject. Nowadays, two widely accepted theories coexist, namely the *reaction-diffusion model* (R-D) [8] and the *trapping/detrapping model* (T-D) [67]. In this work we will rely on the first one. Generally, NBTI includes *stress* and *recovery* phases (see Fig. 2.1a). The stress phase occurs when a pMOS transistor is in a negatively biased condition, i.e., VGS = −VDD (see Fig. 1b for an example of NBTI in a CMOS invertor gate). However, when the biased voltage is removed, i.e., VGS = 0, the pMOS transistor is in the recovery phase and the NBTI effect is partially reversed. The V_{THp} will still increase over time, however in case of sufficient lengths of recovery phases, the aging process may be slowed down considerably. It has been shown that NBTI depends on many factors [68], but its strongest correlation is with the *signal probability* P_z (input duty cycle). The signal probability $P_z(x_i)$ for a logic gate's input x_i is defined as the ratio of time during which the input signal x_i is set to logic 0.

We propose a novel approach to mitigate NBTI in processor circuits using *rejuvenation* of pMOS transistors along NBTI-critical paths with dedicated programs. The method incorporates hierarchical fast, yet accurate identification of NBTI-critical paths at gate level and the rejuvenation Assembler programs generation using an Evolutionary Algorithm. In this work we exploit a general-purpose evolutionary toolkit called *µGP* [69], [70] and find a suitable fitness function using an open source hardware analysis framework called *zamiaCAD* [71]. The advantage of such flow lies in its flexibility for solving the dependencies of impacts by individual gates to the most critical NBTI-induced path delay by using evolutionary optimization processes. The advantage of such flow is its ability and flexibility in solving the dependencies of impacts by individual gates to the most critical NBTI-induced path delay using the evolutionary optimization process. The generated rejuvenation programs are applied at predefined periods in order to drive pMOS transistors to the recovery phase and thus to extend the reliable lifetime of any processor implemented in nanoelectronic technology.

Sub-Section 1.1describes the method for fast, yet accurate hierarchical modeling of NBTI-induced delays. Sub-Section 6.3introduces a flow for evolutionary generation of rejuvenation programs. Section 6.4 presents experimental results on an open-source MIPS processor Plasma and Section 1.1 draws some conclusions.

6.2 Hierarchical Modelling of NBTI-Induced Delays

In [72] we have proposed an approach for fast, yet accurate modeling of NBTI-induced delays. For the formulated task of rejuvenation programs generation for processor designs, the approach can be summarized by the following steps:

- *Step A* (at the transistor level): Obtain a technology and environment dependent curve for voltage threshold shift as a function of gate input signal probability $\Delta V_{THp}(P_z)$.

- *Step B* (at the gate level): Obtain technology and environment dependent curves for degradation of gate delays as a function of voltage threshold shift $\Delta t(\Delta V_{THp})$ for each gate type in the netlist (INV, 2NAND, 2NOR). This implies that SPICE electrical simulations of the individual gates have to be performed only once.
- *Step C* (at logic paths): Identify NBTI-critical paths at the gate level. This step involves: a) simulation for signal probabilities; b) static timing analysis for nominal path delays and NBTI-induced path delays; c) dedicated algorithm for selecting the paths.

Note, that a preprocessing step flattens complex gates into NAND, NOR and INV stages (e.g. an AND gate is represented by a NAND gate followed by an inverter gate).

6.2.1 Modelling NBTI-induced V_{THp} shift

In the NBTI effect analysis, we rely on a reaction-diffusion (R-D) mechanism based predictive model for dynamic NBTI presented in [73], [74] and verified with an industrial 65-nm technology as stated in 0. The proposed model predicts the long term threshold voltage V_{THp} degradation due to NBTI at a time t > 1000s and is proven to be independent of the frequency at high frequencies [73]. It captures the dependence of NBTI on a gate input signal probability P_z (probability that pMOS transistor is under stress) in addition to its dependence on other key process and design parameters as presented in 0:

$$|\Delta V_{THp}| \approx \left(\frac{n^2 K_v^2 P_z C t}{\xi_1^2 t_{ox}^2 (1-P_z)}\right)^n \qquad (1)$$

where $C = T_o^{-1} exp(-E_a / kT)$, E_a is the activation energy of hydrogen species, k is the Boltzmann's constant, T – temperature, ξ_1 and T_o are technology dependent constants, and K_v is expressed by following Equation (2) [73]:

$$K_v = \left(\frac{q\, t_{ox}}{\varepsilon_{ox}}\right)^3 K_1^2\, C_{ox}\left(V_{gs} - V_{th}\right)\sqrt{C}\, \exp\left(\frac{2E_{ox}}{E_{o1}}\right) \qquad (2)$$

where q is the electron charge, $C_{ox} = \varepsilon_{ox} / t_{ox}$ is the oxide capacitance per unit area, and in the strong inversion region the vertical electrical field is given by $E_{ox} = (V_{gs} - V_{th}) / t_{ox}$; ε_{ox} K_1 and E_{01} are technology dependent constants.

Equation (1) is valid only for dynamic NBTI, since if P_z reaches the value 1, the value of ΔV_{THp} becomes infinite and the formula is incorrect. Therefore, the upper limit of ΔV_{THp} is defined by Equation (3), which models only static NBTI [73]:

$$|\Delta V_{THp}| = (K_v^2 t)^n \qquad (3)$$

The values of the involved technology and environmental parameters in Equation (1) can be summarized by a parameter γ, in the following form:

$$|\Delta V_{THp}| = \gamma \left(\frac{P_z}{1-P_z}\right)^n \qquad (4)$$

Equation (4) represents a mathematically convenient function of threshold voltage V_{THp} degradation dependence on the signal probability for input signal $P_z(x_i)$ of pMOS transistor, where n = 1/6 is a variety of the dominant diffusion species (H or H_2) expressed by the time exponent parameter and γ = 0.0904 is a parameter that embraces the selected technology and environmental variables. In Figure 6.1, the corresponding dependence is illustrated for PTM 65 nm technology [75] after 10 years of NBTI-induced degradation at constant temperature T=400K and the supply voltage Vdd = 1.1V.

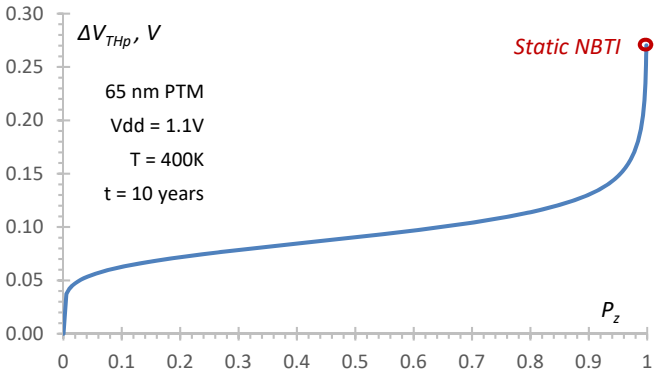

Figure 6.1 Threshold voltage shift ΔVTHp as a mathematically convenient function of signal probability Pz

Alternative technologies, adjusted silicon measurements data or environmental variables may result in variations of the function shape and, thus, changes of the parameters' values. Equation (4) allows fast computation of NBTI-induced V_{THp} shift, as it depends only on a signal probability at the inputs of the considered logic gate. The threshold voltage shift values serve as an input for modeling the NBTI-induced gate delays.

6.2.2 Modeling NBTI-induced gate delay degradation

Further, we calculated the gate delay t_{Pzi} taking into account NBTI degradation provided by Equation (4):

$$\Delta t_{gate} = \lambda \cdot \Delta V_{THp}(x_i) + \mu \cdot \Delta V_{THp}(x_i)^2 \qquad (5)$$

where Δt_{gate} is the gate output delay increase (in percentage) compared to the nominal gate delay, $\Delta V_{THp}(x_i)$ is the change of V_{THp} for pMOS transistor at the gate input x_i, while λ and μ are technology dependent constants. For example, in our experiments λ and μ are set to 1.63 and 5.3 for the Inverter gate (see Figure 6.2). The maximum and average deviation values of the fitting function from the SPICE results were 4.22% and 1.19%, correspondingly [72].

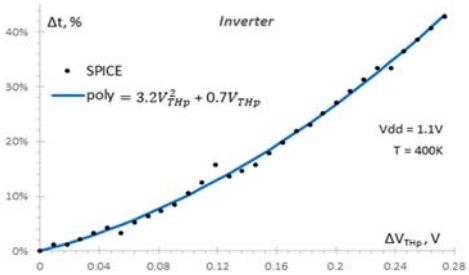

Figure 6.2 Dependence of the gate output delay Δt on the ΔVTHp in an Inverter gate. Results of SPICE simulations (blue curve) and of the proposed mathematically convenient function (black dashed curve).

In case of many-input gates embracing pMOS transistors networks, both the physical location of each pMOS transistor relative to the output node and the combination of gate inputs 1→0 transitions have an impact on the level of gate delay degradation. In our approach, each combination of gate input values for each gate is modelled by different values of constants λ and μ in Equation (5). Note, that our NBTI-induced path delay modelling approach models the gate-output load capacitance influenced by the circuit structure, e.g. the fan-out size and the length of wires (available at the layout floor planning phase) only for the nominal delay

calculation. As a simplification id does not explicitly model it to calculate the NBTI-induced delays. Our previous experiments [76] demonstrate that this simplification yields acceptable NBTI-induced path delay estimation with inaccuracy lower than 2% when compared to SPICE simulations.

6.2.3 NBTI-critical path identification

In the following, a method for fast calculation of the NBTI-induced delay degradation at paths of a gate-level circuit is proposed. We use the following notations:

- $d(G_k)$ is the nominal delay of the fresh gate G_k without considering aging, i.e. its delay at time zero;
- $\tau(G_{k,i})$ is the increase in the delay of the gate G_k from the i-th input to the output of the gate caused by NBTI-induced aging;
- $t(G_{k,i})$ is the total delay of the gate G_k from the i-th input to the output of the gate caused by NBTI-induced aging,

$t(G_{k,i}) = d(G_k) + \tau(G_{k,i})$;

- $t(G_k)$ is the total maximum delay of the gate G_k over all its inputs m_k, when taking into account NBTI-induced aging.

$t(G_k) = \max \{t(G_{k,1}), t(G_{k,2}), \ldots, t(G_{k,m_k})\}$;

- $D(G_k)$ is the delay calculated for the slowest signal path in the cone $C_{IN}(G_k)$ based on the values of $t(G_{k,i})$ for all gates on this path, $D(G_k) = \max \{(D(G_i) + t(G_{k,i})) \mid G_i \in IN(G_k)\}$,

  where $IN(G_k)$ is the set of input gates of G_k , and $t(G_{k,i})$ is the total delay of the gate G_k from the output of the gate G_i caused by aging.

Consider a combinational circuit as a network of gates where all the gates have numbers which show the ranking of gates in a partial order such that:

(1) all the input gates are numbered in an arbitrary order

(2) all other gates may get their numbers only if all their predecessor gates have already got their numbers.

We present *Algorithm 1* for calculating $D(G_k)$, which is based on processing the gate-level netlist, gate by gate, from inputs to outputs. The method calculates the maximal degraded path delay values $D(G_k)$ for all the gates of the circuit based on the estimates of $t(G_k)$. Here, NG is the number of gates in the circuit and PI is the set of gates connected to primary inputs.

Algorithm 1. NBTI-aware static timing analysis

FOR all gates G_k, k = 1,2, …, NG:

$$t'(G_{k,i}) = \begin{cases} t(G_{k,i}), & x_i = 0 \\ d(G_k), & x_i \neq 0 \end{cases}$$

IF $G_k \in$ PI, **THEN**

$D(G_k) = t'(G_{k,0})$

ELSE

$D(G_k) = \max \{D(G_i) + t'(G_{k,i}) \mid G_i \in IN(G_k)\}$

END IF

END FOR

As a result, fast identification of NBTI-critical paths can be performed at the gate level.

6.3 Evolutionary Generation of Rejuvenation Programs

Natural evolution is based on random variations: some are rejected while others preserved according to quite objective evaluations. Only changes that are beneficial to the individuals are likely to spread into subsequent generations. Darwin called this principle *natural selection*, a quite simple process where random variations "afford materials" [77]. The field of Evolutionary Computation (EC) originates in the 1960s when several researchers ― independently ― tried to replicate such a characteristic to tackle difficult problems [78].

Most of the EC jargon mimics the precise terminology of biology: evolution proceeds through discrete steps called *generations*; a single candidate solution is an *individual*; the set of all candidate solutions that exists at a particular time represents the *population*. EC algorithms rely on the idea of promoting variations that produce small, yet quantifiable, differences in the goodness of the solution.

In CAD, evolutionary heuristics started to be seen as alternatives to classic approaches in the 1990s for several well-known NP-hard problems [79]. Optimizing test programs for rejuvenating a given circuit is the paradigmatic application for EC. Such an optimizer needs to be able to solve dependencies caused by impacts of individual gates and capable to obtain a cost-effective global solution with respect to all NBTI-critical paths.

The approach proposed in this work was implemented coupling two open-source tools: an evolutionary toolkit, and a hardware design and analysis framework.

µGP (also spelled *MicroGP*) is a general-purpose evolutionary toolkit developed at Politecnico di Torino [80] [70]. It allows a high degree of customization of evolutionary operators, stop criteria, and algorithm parameters. Internally it represents candidate solutions as multi-graphs, where each node roughly corresponds to a locus of the genome. Differently from most EAs, loci can be occupied by alleles with different characteristics, e.g. integer, float or fixed values, and the probability of appearance of each allele can be tuned. The internal representation allows efficiently handling problems whose solutions are as simple as bit signals and as complex as full-pledged assembly programs. µGP implements a quite large variety of genetic operators that can handle the specific characteristics of the individuals' genome.

zamiaCAD is a scalable hardware design and analysis framework [71] [81] [82]. Its front-end includes a parser and an elaboration engine that both support full VHDL-2002 standard specification and a set of VHDL-2008 extensions. On the back-end side, the framework allows design simulation, static analysis and other applications for debug. zamiaCAD has an Eclipse IDE plug-in based graphical user interface for advanced design entry and navigation.

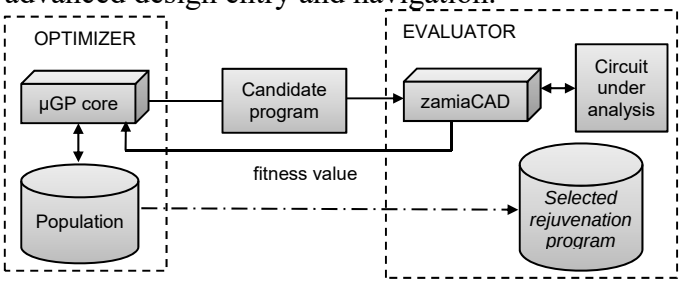

Figure 6.3 A general flow of the evolutionary system developed in this work

The general flow is presented in Figure 6.3. It is composed of two main parts: an OPTIMIZER, that is, µGP, and an *EVALUATOR* based on the zamiaCAD framework. The optimizer generates candidate solutions (i.e., *individuals*) cultivating a population of programs. The assembly language of the microprocessor is described in XML file

that enable μGP to map its internal multigraphs into syntactically correct and sensible programs. In the experiments reported here, μGP was allowed to generate programs using 47 different instructions, including conditional branches. The optimizer determined the length of the program, the sequence of instructions, the optimal values for their operand, either registers or immediate, and the targets of the conditional branches. The adopted XML file forbade backward jumps as they could create endless loops.

The usefulness of each program vector is evaluated by simulating the NBTI-degraded path delays for the timing-delay critical paths identified in the NBTI-analysis process. The longest NBTI-degraded path delay value is reported back to the μGP core in form of *fitness value*. Optionally, the designer can increase the efficiency of the rejuvenation program generation procedure by providing the OPTIMIZER with details on the design application or structure. The first case is relevant when the processor is to be used in an embedded system with a very specific program. I.e. the expected workload is (partially) known. The second case is relevant if a particular part (a module) of the processor is more important for rejuvenation. Such details can be modeled as a *secondary fitness value.* In the proposed flow, if e.g. the ALU core is the primary goal for rejuvenation, the secondary fitness value can count the number of ALU operations inside the programs. This value is a mere *hint* for the evolutionary algorithm, as the region of the search space more promising for rejuvenating the ALU are the programs that do use the ALU. μGP uses it to discriminate two solutions only when the first elements, that is the rejuvenation efficacies, are the same. μGP uses the fitness along with structural information to select the more promising individuals as parents in the next generation. Eventually, the best individual found during the evolution process, i.e. the one with the highest fitness, is selected. μGP evolves the population until a *steady state* condition is detected, that is, no improvement is recorded for a given number of generations.

6.4 Experimental Results

The proposed approach is demonstrated on a MIPS processor design Plasma [83] described in RTL VHDL. The detailed analysis of the NBTI-induced delays and rejuvenation was performed for the Arithmetic Logic Unit core ALU (32 bit). The gate level implementation was synthesized with Synopsys Design Compiler. It utilized 1,002 basic gates (INV, 2-input NAND and 2-input NOR), where 138 gates are along the longest path at time-zero (pre-NBTI stress).

The experiments to assess evolutionary generation of Assembler programs for rejuvenation of NBTI-impacted processors are summarized in Table 6.1. First, 8 representative workloads were selected (listed in row 1):

- *NOR, XOR* are two representative workload Assembler programs exercising these two logical operations correspondingly with a large set of random operands; *ADD, MULT* are such representative workloads for these two mathematical operations correspondingly. Note, a set of similar workloads for other *logic and mathematical operations* were also exercised in our experiments and produced results similar to these four representatives.
- *mix* is an Assembler program with a deterministic combination of the CPU mathematical instructions for a set of random operands; *NOP* is a workload consisting of NOP instructions only. Intuitively, it keeps a significant part of the ALU core unexercised and therefore under NBTI stress.
- *rand1* and *rand2* are two representatives of a set of random Assembler programs (256 in total). Interestingly, the distribution of random workloads, with regard to their NBTI-induced path delay impact, was very wide. Here *rand1* is the workload

with the minimal NBTI-induced path delay and *rand2* is a representative of a random workload with an average NBTI-induced path delay.

The Plasma processor was simulated with the workloads and the gate input signal probabilities in the ALU core were obtained. Further, the NBTI-induced path delays were calculated as described in Sub-Section 6.2. The estimated impact of a 10-year stress NBTI with the corresponding workloads on the longest path is presented in row 2 of Table 6.1.

Table 6.1 Rejuvenation analysis for the ALU core as a part of Plasma.

Workloads		*NOR*	*XOR*	*ADD*	*MULT*	*mix*	*NOP*	*rand1*	*rand2*
Δt by NBTI		10.08	10.13	21.06	18.04	18.52	27.74	8.69	18.37
a) Universal Rejuvenation									
Δt^R after rejuvenation for the given overhead (%)	0.001	10.09	10.09	16.64	13.53	13.86	17.79	8.34	13.72
	0.01	10.09	10.1	15.7	12.59	12.86	15.67	8.29	12.72
	0.1	10.1	10.1	14.68	11.59	11.8	13.41	8.26	11.64
	1.0	10.1	10.11	13.74	10.54	10.63	11.03	8.31	10.42
Rej. gain	0.1	-0.02	0.03	6.38	6.46	6.72	14.33	0.43	6.73
b) *Universal Rejuvenation (ALU functions preferred)*									
Δt^R after 0.1% rej.		9.80	10.08	14.38	11.31	12.63	11.88	14.53	8.40
Rejuvenation gain		0.28	0.05	6.69	6.74	6.66	6.64	13.21	0.29
c) *Specific Rejuvenation (targeting workloads)*									
Δt^R after 0.1% rej.		9.8	10.1	14.48	11.4	11.76	13.57	8.22	11.66
Rejuvenation gain		0.28	0.03	6.58	6.65	6.75	14.17	0.47	6.71
d) *Rejuvenation with the selected low-aging Random program out of 256*									
Δt^R after 0.1% rej.		9.79	10.08	15.51	12.43	12.84	15.07	8.69	12.84
Rejuvenation gain		0.28	0.05	5.55	5.62	5.52	12.67	0.00	5.53
e) *Rejuvenation with an average Random program out of 256*									
Δt^R after 0.1% rej.		10.08	10.13	21.04	18.04	18.36	20.39	8.73	18.36
Rejuvenation gain		0.00	0.00	0.03	0.00	0.00	7.35	-0.03	0.01

Experiment a). A *universal* rejuvenation program was generated using the single fitness value (see Sub-Section 6.3). Its impact on rejuvenation of the stressed ALU core was analyzed for a set of allowed execution overheads (0.001%, 0.01%, 0.1% and 1%). For example, the 0.1% execution overhead of the proposed rejuvenation stimuli results in 14.68% path delay increase for the *ADD* workload versus 21.06% path delay increase without rejuvenation. The absolute Rejuvenation gain is therefore 6.38% or relatively the delay is deduced by 1.43 times, that can be translated into extra several years of reliable operation of the processor in the field.

Experiment b). Different from *Experiment a)*, the OPTIMIZER was configured to use in addition a secondary fitness value to prefer in the rejuvenation program the instructions implemented in the ALU core. The rejuvenation gain is slightly better than in *Experiment a)*.

Experiment c). Here, instead of the universal one, there were generated rejuvenation programs that target specific workloads of Plasma, i.e. the workloads *NOR*, *XOR*, etc. Note, the negative rejuvenation gain in *Experiment a)* is caused inability of the universal rejuvenation program to influence a set of extreme signal probabilities by the *NOR* workload. Both assisted rejuvenation program generation *Experiments b)* and *c)* were able to find a better solution in this case.

Experiments d) and *e)* represent attempts to rejuvenate the ALU core with random Assembler programs. Here even the random program with the lowest NBTI impact selected out of 256 random programs demonstrates a weaker rejuvenation capability compared to the programs generated with the proposed flow.

As a reference a standalone gate-level ALU core extracted from the Plasma processor is analyzed. demonstrates results of rejuvenation stimuli generation (binary vectors) with the proposed flow for a set of workloads as described in 0. Given the direct access to the primary inputs, such standalone circuit has a better controllability

of gate inputs. First, higher NBTI-induced path delays can be demonstrated with particular workloads and, second, a higher rejuvenation gain can be achieved compared to the ALU core as an integrated part of the processor circuit.

Table 6.2 Rejuvenation of a standalone ALU core (32-bit) of Plasma.

Workloads		*NOR*	*OR*	*AND*	rand	highNBTI	lowNBTI
Δt by NBTI (%)		23.88	33.33	24.56	12.35	54.80	9.17
Δt^R after rej. for given overhead (%)	0.001	18.39	25.42	18.86	12.35	38.56	9.17
	0.01	15.76	21.51	16	12.35	30.53	9.17
	0.1	14.56	19.35	14.42	12.34	26.09	9.17
	1.0	13.45	17.05	13.56	12.31	21.8	9.17
Rej. gain	0.1	9.32	13.98	10.14	0.01	28.71	0

The evolutionary generation of one rejuvenation program, either universal in *Experiments a)* and *b)* or each of the specific ones in *Experiment c)*, took 21 hours on a moderate Win 64bit workstation. This time includes iterative execution (approximately 20K times) of the evolutionary algorithm with the design simulation and NBTI-induced path delay calculation calls. The analysis in *Experiments d)* and *e)* takes 3.5 seconds. Here, we used the internal simulator of zamiaCAD (about 10 times slower than the state-of the-art commercial simulators).

The proposed rejuvenation approach **supports the major BASTION key performance indicator "KPI4: *Extend the product's lifetime 3-5 times*"**. The actual

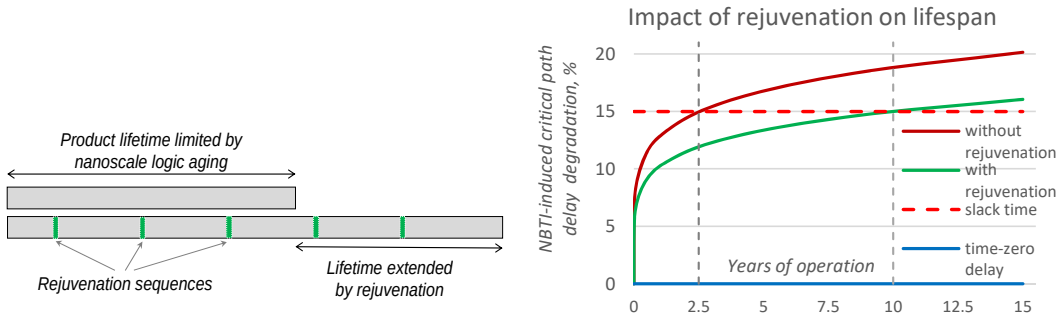

Figure 6.4 An illustrative schedule for rejuvenation stimuli execution and the resulting impact

impact on the circuit life time strongly depends on the application scenario.

As an example, we reported in Figure 6.4 a scenario, where rejuvenation sequences are applied periodically: in this way we experience some performance degradation, while improving the life time. The chart on the right-hand side illustrates a scenario with the slack time set to 15% of the most critical path's delay at time zero (i.e. the fresh circuit). The NBTI-induced delay at the longest path is estimated to reach 15% after 2.5 years and 19% after 10 years of operation for a given workload without rejuvenation. At the same time, the delay is still 15% after 10 years if the rejuvenation procedure is applied. A similar rejuvenation impact is demonstrated by the simulation-based preliminary experimental results in Table 6.1, e.g. for the *mix* workload with 0.11% execution overhead. The reader should note that a reduction of the NBTI-induced delay by just several percent can significantly extend reliable lifetime of nanoscale logic, e.g., the discussed illustrative example in Figure 6.4 provides an extension in 4 times, i.e. from 2.5 to 10 years.

At the same time, the proposed approach contributes to another **major BASTION key performance indicator "KPI1: *Improvement of the efficiency of***

aging fault detection and relaxation by at least 30%". The efficiency of aging relaxation is addressed by the novel approach targeted to generation and application of specific rejuvenation stimuli. It can be observed from our experiments that design exploitation workloads based on random stimuli, while unrealistic in practice, result in the lowest NBTI-induced degradation. This observation can motivate an approach to rejuvenate the designs with random stimuli. We have compared the efficiency of such rejuvenation with rejuvenation by the specifically generated rejuvenation stimuli. Consider the most realistic workloads *ADD*, *MULT*, *mix* highlighted by blue in Table 6.1. The NBTI-induced degradation (row 2) has significantly better relaxation with the dedicated rejuvenation stimuli (row 6, Universal rejuvenation with 0.1% overhead) compared to the average random stimuli (the second last row). Therefore, the preliminary experimental results demonstrate potential for the rejuvenation (as a part of relaxation) efficiency improvement by up to 33%.

6.5 Conclusions

In the frame of the BASTION project we developed an approach for NBTI mitigation in processor designs by rejuvenation of degraded pMOS transistors along NBTI-critical paths. The method incorporates hierarchical fast, yet accurate modelling of NBTI-induced delays at transistor, gate and path levels for generation of rejuvenation Assembler programs using an evolutionary flow. These programs are to be applied further as an execution overhead to drive those pMOS transistors to the recovery phase, which are the most critical for the NBTI-induced path delay in processors.

The experimental results demonstrate the feasibility of rejuvenation with dedicated Assembler programs and efficiency of their evolutionary generation providing for significant reduction of NBTI-induced path delays, i.e. in 1.4 times on average in case of small execution overheads (e.g. 0.1% or less). This can be translated into several extra years of reliable operation of the processor in the field. Notably, the universal rejuvenation programs are efficient independently from the exercised workloads. The proposed flow allows configurations to specifically address particular expected workloads (e.g. in embedded systems) or assisting the generation process with implementation based “hints” that in our experiments resulted only in minor improvements.

7 Conclusions

In this deliverable we have reported on the activities performed in T3.3 within the BASTION project. The following summarizes the deliverable.

In the hierarchical test and design section first different design partitioning strategies have been discussed concluding that there is not one optimal solution. Instead a trade-off is required taking into account all the different test and design aspects. After describing the general pattern retargeting process different top-level test strategies for hierarchical scan tests have been presented and compared in detail. Afterwards the impact of the hierarchical test on the test coverage has been analyzed. While the hierarchical test approach does not reduce the test coverage in general, there might be some impact for the delay test when using dedicated wrapper cells for the module isolation. Finally, the overall hierarchical test flow has been evaluated in small case study.

When it comes to fault management, RRC is an efficient method to cope with soft errors. In this research, we showed that by distributing the checkpoints in a non-equidistant manner it is possible to improve the LoC in comparison to equidistant

checkpointing. The non-equidistant distribution that provides the highest LoC is made out of clusters, where all execution segments that belong in the same cluster have the same size. The number of clusters is relatively low, which allows the proposed heuristic, i.e. Clustered Checkpointing, to find the same distribution as the exhaustive search approach, which always finds the optimal distribution, in much shorter time.
We presented a method to slow down the aging effects which have been observed in many recent semiconductor technologies. The key idea is that these effects can be partly compensated by applying suitable stimuli to the circuit during idle periods: in the case of a processor, this results in executing some additional instructions when possible, forcing some well defined values on the inputs of the most critical units of the processor from the point of view of delay. Some preliminary results we gathered show that this approach can slow down the aging effects by about 40%, thus correspondingly increasing the typical circuit life time.

Two BASTION KPIs have been addressed by this deliverable, which is:

KPI1: *Improvement of the efficiency of aging fault detection and relaxation by at least 30%.* The efficiency of *aging relaxation* is addressed by a novel approach targeted to generation and application of specific rejuvenation stimuli. The preliminary experimental results demonstrate potential for the rejuvenation efficiency improvement by up to 33%. See Section 6.4 for details. The contribution of this deliverable in terms of *fault detection efficiency* by fast alarming of the OS as well as minimization of *circuit performance impact* caused by the instrumentation and data collection infrastructure is given in Section 5. Other aspects of KPI1 are covered in D3.2, where different aspects of KPI1 (efficiency) are also explained.

KPI4: *Extend the product's lifetime 3-5 times.* We have proposed a novel NBTI-mitigation approach for rejuvenation of processors with dedicated Assembler programs targeted at lifetime extension and published preliminary experimental results. See Section 6.4 for details.

8 Bibliography

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

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

[3] R. Baranowski, M. A. Kochte and H.-J. Wunderlich, "Modeling, Verification and Pattern Generation for Reconfigurable Scan Networks," in *IEEE International Test Conference (ITC'2012)*, Anaheim, USA, 2012.

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

[5] M. Keim and T. Waayers et al., "Industrial Application of IEEE P1687 for an Automotive Product," in *Euromicro Conf. on Digital System Design (DSD'2013)*, 2013.

[6] S. Hamdioui, D. Gizopoulos, G. Guido, M. Nicolaidis, A. Grasset and P. Bonnot, "Reliability Challenges of Real-Time Systems in Forthcoming Technology Nodes," in *ACM/IEEE Conference on Design, Automation and Test in Europe*, 2013.

[7] S. Mahapatra, D. Saha, D. Varghese and P. B. Kumar, "On the generation and recovery of interface traps in MOSFETs subjected to NBTI, FN, and HCI stress," *IEEE Trans. Electron. Dev.,* vol. 53, no. 7, pp. 1583-1592, 2006.

[8] M. A. Alam and S. Mahapatra, "A comprehensive model of PMOS NBTI degradation," *Microelectronics Reliability,* vol. 45, no. 1, pp. 71-81, 2005.

[9] S. Kumar, S. Kim and S. Sapatnekar, "Adaptive techniques for overcoming performance degradation due to aging in digital circuits," in *Asia and South Pacific Design Automation Conference*, 2009.

[10] T. Grasser and B. Kaczer, "Negative bias temperature instability: Recoverable versus permanent degradation," in *European Solid State Device Research Conference*, 2007.

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

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

[13] S. K. Goel, K. Chiu, E. J. Marinissen, T. Nguyen and S. Oostdijk, "Test Infrastructure Design for the the Nexperia Home Platform PNX8550 System Chip," in *Proceedings of the conference on Design, automation and test in Europe*, 2005.

[14] A. Sehgal, J. Fitzgerald and J. Rearick, "Test Cost Reduction for the AMD Athlon Processor using Test Partitioning," in *IEEE International Test Conference (ITC'07)*, 2007.

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

[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] 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.

[18] 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.

[19] 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.

[20] 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.

[21] A. Bruck and J. Ziv, "An on-line algorithm for checkpoint placement," *IEEE Transaction on Computers,* vol. 46, no. 9, pp. 976-985, 1997.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] J-M Yang and S.W. Kwak, "A checkpoint scheme with task duplication considering transient and permanent faults," in *IEEE International conference on Industrial Engineering and Engineering Management*, 2010.

[27] Y. Zhang and K. Chakrabary, "Energy-aware adaptive checkpointing in embedded real-time systems," in *Design, Automation and Test in Europe*, 2003.

[28] A. Tsertov, A. Jutman, S. Devadze, M. Sonza-Reorda, E. Larsson, F. Ghani Zadegan, R. Cantoro, M. Montazeri and R. Krenz-Baath, "A Suite of IEEE 1687 Benchmark Networks," in *International Test Conference (ITC'2016)*, Fort Worth, USA, Nov 2016.

[29] "An online collection of ITC'2016 IEEE 1687 Benchmark Networks [Online]," 2016. [Online]. Available: www.fp7-bastion.eu/1687.

[30] M. A. Kochte, R. Baranowski, M. Sauer, B. Becker and H.-J. Wunderlich, "Formal Verification of Secure Reconfigurable Scan Network Infrastructure," in *European Test Symposium (ETS'2016)*, Amsterdam, The Netherlands, May 23-26, 2016.

[31] M. Beck and P. Engelke, "On Safety and Security Aspects of IEEE 1687 Networks," in *Test Standards Application Workshop (TESTA'2016)*, Amsterdam, The Netherlands, May 26-27, 2016.

[32] C. Clark, "Anti-tamper JTAG TAP design enables DRM to JTAG registers and

P1687 on-chip instruments," in *Proc. IEEE Int. Symp. on Hardware-Oriented Security and Trust (HOST)*, June 2010.

[33] R. Baranowski, M. A. Kochte and H.-J. Wunderlich, "Access Port Protection for Reconfigurable Scan Networks," *J. Electronic Testing: Theory and Applications,* vol. 30, no. 6, pp. 711-723, 2014.

[34] J. Dworak and A. Crouch, "Don't Forget to Lock your SIB: Hiding Instruments using P1687," in *Proc. IEEE International Test Conference (ITC)*, 2013.

[35] A. Zygmontowicz, J. Dworak, A. Crouch and J. Potter, "Making it Harder to Unlock an LSIB: Honeytraps and Misdirection in a P1687 Network," in *Proc. Design, Automation and Test in Europe Conf.*, Mar. 2014.

[36] R. Baranowski, M. A. Kochte and H.-J. Wunderlich, "Fine-grained access management in reconfigurable scan networks," *IEEE Trans. on CAD of Integrated Circuits and Systems,* vol. 34, no. 6, pp. 937-946, June 2015.

[37] H. Liu and V. Agrawal, "Securing IEEE 1687-2014 Standard Instrumentation Access by LFSR Key," in *Proc.IEEE Asian Test Symp.(ATS'2015)*, 2015.

[38] M. Schaal, "Test of reconfigurable scan-networks," Master's thesis, Universität Stuttgart, Germany, Dec. 2013.

[39] R. Cantoro, M. Montazeri, M. Sonza-Reorda, F. Ghani Zadegan and E. Larsson, "On the Testability of IEEE 1687 Networks," in *Asian Test Symposium (ATS)*, 2015.

[40] R. Cantoro, M. Montazeri, M. Sonza-Reorda, F. Ghani Zadegan and E. Larsson, "On the Diagnostic Analysis of IEEE 1687 Networks," in *Proc. European Test Symposium (ETS)*, Amsterdam, The Netherlands, May 23-26, 2016.

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

[42] E. Larsson and K. Shibin, "Fault management in an IEEE P1687 (IJTAG) environment," in *Proc. IEEE Int'l Symp. Design and Diagnostics of Electronic Circuits Systems (DDECS)*, April 2012, embedded tutorial.

[43] 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.

[44] 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.

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

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

[47] G. Ali, A. Badawy and H. Kerkhoff, "Online Management of Temperature Health Monitors using the IEEE 1687 Standard," in *presentation at Test Standards Application Workshop (TESTA)*, Amsterdam, The Netherlands, May 26-27, 2016.

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

[49] A. Ibrahim and H. Kerkhoff, "Analysis and Design of an On-Chip Retargeting Engine for IEEE 1687 Networks," in *Proc. European Test Symp. (ETS)*, Amsterdam, The Netherlands, May 23-26, 2016.

[50] M. Portolan, "A Novel Test Generation and Application Flow for Functional Access to IEEE 1687 instruments," in *Proc. European Test Symp. (ETS)*, Amsterdam, The Netherlands, May 23-26, 2016.

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

[52] C. Ferri, D. Papagiannopoulou, R. I. Bahar and A. Calimera, "NBTI-Aware Data Allocation Strategies for Scratchpad Memory Based Embedded Systems," in *IEEE 12th Latin American Test Workshop*, 2011.

[53] L. M. F. Ahmed, "ReliablCache Design with On-Chip Monitoring of NBTI Degradation in SRAM Cells using BIST," in *28th IEEE VLSI Test Symposium*, 2010.

[54] S. Khan and S. Hamdioui, "Modeling and Mitigating NBTI in Nanoscale Circuits," in *17th IEEE International On-Line Testing Symposium*, 2011.

[55] S. V. Kumar, C. H. Kim and S. S. Sapatnekar, "NBTI-aware synthesis of digital circuits," in *Design Automation Conference*, 2007.

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

[57] A. Tiwari and J. Torrellas, "Facelift: Hiding and slowing down aging in multicores," in *Int. Symposium on Microarchitecture*, 2008.

[58] F. Firouzi, S. Kiamehr and M. Tahoori, "A linear programming approach for minimum NBTI vector selection," in *IEEE Great Lakes Symposium on VLSI*, 2011.

[59] Y. Wang, X. Chen, W. Wang, V. Balakrishnan, Y. Cao, Y. Xie and H. Yang, "On the efficacy of input Vector Control to mitigate NBTI effects and leakage power," in *Quality Electronic Design Int'l Symp.*, 2009.

[60] J. Abella, X. Vera and e. al., "Penelope: The NBTI-aware processor," in *40th IEEE/ACM Int. Symposium on Microarchitecture*, 2007.

[61] L. Li, Y. Zhang, J. Yang and J. Zhao, "Proactive nbti mitigation for busy functional units in out-of-order microprocessors," in *ACM/IEEE Conference on Design, Automation and Test in Europe*, 2010.

[62] X. Fu, T. Li and J. Fortes, "NBTI tolerant microarchitecture design in the presence of process variation," in *Int. Symposium on Microarchitecture*, 2008.

[63] F. Firouzi, S. Kiamehr and M. Tahoori, "NBTI mitigation by optimized NOP assignment and insertion," in *ACM/IEEE Conference on Design, Automation and Test in Europe*, 2012.

[64] BASTION, "Report on methods for IJTAG network adaptation and optimization for error detection and diagnosis (Deliverable D3.2)".

[65] BASTION, "Report on Error Classification and Corresponding Error Handling

(Deliverable D3.1)".

[66] 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.

[67] Y. Cao, J. Velamala, K. Sutaria, M.-W. Chen, J. Ahlbin, I. Sanchez Esqueda, M. Bajura and M. Fritze, "Cross-Layer Modeling and Simulation of Circuit Reliability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 33, no. 1, pp. 8-23, 2014.

[68] H. Kukner, S. Khan, P. Weckx, P. Raghavan, S. Hamdioui, B. Kaczer, F. Catthoor, L. V. d. Perre, R. Lauwereins and G. Groeseneken, "Comparison of reaction-diffusion and atomistic trap-based BTI models for logic gates," *IEEE Transactions on Device and Materials Reliability,* vol. 14, no. 1, pp. 173-183, 2014.

[69] G. Squillero, "MicroGP—An Evolutionary Assembly Program Generator," in *Genetic Programming and Evolvable Machines*, 2005.

[70] E. Sanchez, M. Schillaci and G. Squillero, Evolutionary Optimization: the μGP toolkit, Springer, 2011.

[71] "zamiaCAD framework web page," [Online]. Available: http://zamiaCAD.sf.net.

[72] M. Jenihhin, G. Squillero, T. Copetti, V. Tihhomirov, S. Kostin, M. Gaudesi, F. Vargas, J. Raik, M. S. Reorda, L. B. Poehls, R. Ubar and G. Medeiros, "Identification and Rejuvenation of NBTI-Critical Logic Paths in Nanoscale Circuits," *Journal of Electronic Testing-Theory and Applications (JETTA),* vol. 32, no. 3, pp. 273-289, 2016.

[73] S. Bhardwaj, W. Wang, R. Vattikonda, Y. Cao and S. Vrudhula, "Predictive modeling of the NBTI effect for reliable design," in *IEEE Custom Integr. Circuits Conf.*, 2006.

[74] W. Wang, V. Reddy, A. Krishnan, R. Vattikonda, S. Krishnan and Y. Cao, "Compact Modeling and Simulation of Circuit Reliability for 65nm CMOS Technology," *IEEE Trans. on Device and Materials Reliability,* vol. 7, no. 4, pp. 509-517, 2007.

[75] W. Zhao and Y. Cao, "Predictive Technology Model for Nano-CMOS Design Exploration," *Journal on Emerging Technologies in Computing Systems,* vol. 3, no. 1, 2007.

[76] S.Kostin, J. Raik, R.Ubar, M. Jenihhin, T. Copetti, F. Vargas and L. Bolzani, "SPICE-Inspired Fast Gate-Level Computation of NBTI-induced Delays in Nanoscale Logic," in *18th IEEE DDECS*, 2015.

[77] C. Darwin, On the Origin of the Species by Means of Natural Selection: Or, The Preservation of Favoured Races in the Struggle for Life, John Murray, 1859.

[78] A. S. J. Eiben, Introduction to Evolutionary Computing, Springer , 2015.

[79] G. Squillero, "Artificial evolution in computer aided design: from the optimization of parameters to the creation of assembly programs," *Computing,* vol. 93, pp. 93-103, 2011.

[80] G. Squillero, "MicroGP—An Evolutionary Assembly Program Generator," in *Genetic Programming and Evolvable Machines*, 2005.

[81] A. Tšepurov, G. Bartsch, R. Dorsch, M. Jenihhin, J. Raik and V. Tihhomirov, "A Scalable Model Based RTL Framework zamiaCAD for Static Analysis," in *IFIP/IEEE Int. Conf. on Very Large Scale Integration*, 2012.

[82] M. Jenihhin, A. Tsepurov, V. Tihhomirov, J. Raik, H. Hantson, R. Ubar, G. Bartsch, J. Escobar and H.-D. Wuttke, "Automated Design Error Localization in RTL Designs," *IEEE Design & Test,* vol. 31, no. 1, pp. 83-92, 2014.

[83] "Open Cores Plasma CPU project," [Online]. Available: http://opencores.org/project,plasma.

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

[85] 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.

[86] 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.

[87] 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.

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

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

[90] 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.

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

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

[93] 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.

[94] 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.

[95] 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.

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

[97] 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.

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

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

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

[101] 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.

[102] 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.

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

[104] 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.

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