Laboratoire
Informatique
Robotique
Microélectronique
Montpellier

Université de Montpellier

Habilitation à Diriger les Recherches

Test and Diagnosis of Integrated Circuits

Candidate:
Alberto Bosio

Jury:
Lorena Anghel
Daniel Chillet
Said Hamdioui
Paolo Montuschi
Patrick Girard
Bruno Rouzeyre

Vendredi 3 Avril 2015

Contents

List of Figures

List of Tables

Chapter 1

Summary

1.1 Curriculum Vitae

1.1.1 Personal Information

- **Name:** Alberto BOSIO
- **Date and Place of Birth:** 3 March 1977, Fossano (Italy)
- **Nationality:** Italian
- **Personal address:** 683, Avenue du Pére Soulas, 34090 Montpellier, France
- **Current position:** Maître de conférences CNU 61
- **Work address:** LIRMM, UMR 5506, 161 rue Ada, 34392 Montpellier Cedex 5, France
- **Work phone:** 04 67 41 85 76
- **Fax:** 04 67 41 85 00
- **Email:** alberto.bosio@lirmm.fr

1.1.2 Cursus

- **From september 2007:** "Maître de conférences" at LIRMM/Université de Montpellier 2
- **2006-2007:** Post-Doc at LIRMM
- **2005-2006:** Post-Doc at Politecnico di Torino, Computer Science Department

- **2003-2005:** Ph.D. degree in Computer Engineering from Politecnico di Torino. Title: "Dependable Architectures for Safety-Critical Applications"
- **2003:** National Engineering habilitation exam, Turin, Italy
- **2002:** Master degree in Computer Engineering from Politecnico di Torino. Title: "Automatic March Test Generator"
- **1996:** High School degree (Computer Science Technical Institute), Fossano, Italy

1.1.3 Research Interests

- Memory and Microprocessor Testing
- Digital Testing and Design for Testability
- ATPG, Fault Simulation
- Diagnosis
- Power Aware Test
- Test of Low Power Devices

1.1.4 Summary of the hot points

- Co-author of 1 book, 11 international journal papers, more than 50 papers and presentations in international conferences and 2 patents
- Co-supervisor of 8 PhD students and 20 master students
- Participation to overall 7 research contracts
- Web chair of the TTTC, ETS
- Web chair of: BTW'10, BTW'11, BTW'12, BTW'14, ETS'13
- Publication chair of: SIES'08, DTIS'14, DTIS'15, ETS'15, ETS'16
- Member of the program committee of several international conferences and reviewers for several journals

1.2 Summary of scientific activities

1.2.1 Preamble

I have carried out all my studies in Italy. Moreover, after a PhD in Computer Engineering in the domain of the test and verification of digital systems at the Politecnico di Torino (Italy), I pursued 1 year of post-doc research at the same institute still in the same domain. Thanks to cooperations between the Politecnico di Torino and the LIRMM of Montpellier, I had the opportunity to obtain a post-doc position in France. There, in addition to the topics which were already familiar to me, I had the occasion to explore new fields in the test topic: the diagnosis and the power-aware test. My scientific results, students I have supervised, my teaching activities, and in general my research projects directly reflects my career path.

1.2.2 Current research activities

Research activities I carried on after my nomination as Maître de conférences deal with three main axes: the fault diagnosis, the power-aware test and the test of low-power devices. In the next subsections I will introduce each research axe, the details are given in the Chapter 2 Section 2.1.

1.2.2.1 Fault Diagnosis

The ever-increasing growth of the semiconductor market results in an increasing complexity of digital circuits. Smaller, faster, cheaper and low-power consumption are the main challenges in semiconductor industry. The reduction of transistor size and the latest packaging technology (i.e., System-On-a-Chip, System-In-Package, Trough Silicon Via 3D Integrated Circuits) allows the semiconductor industry to satisfy the latest challenges. Although producing such advanced circuits can benefit users, the manufacturing process is becoming finer and denser, making chips more prone to defects. Physical defects like shorts and opens will occur during any single step of the fabrication process. These defects can be randomly caused by contaminations or due to systematic process-design interaction [1]. In modern deep submicron technologies, systematic defects are becoming more likely to appear than random defects. This is caused by the reduced chip sizes, the use of new complex process technologies, new materials and the increasing number of vias and contacts [2]. Today, systematic defects appear not only in the cell interconnection, but also inside the cell itself (intra-cell defect).

The test is one of the most critical tasks in the semiconductor production process. It is not only necessary to seek for fault free devices but it plays a key role in analyzing defects in the manufacturing process as well. The feedbacks derived from the test process are the only way to analyze and isolate many of the defects in today's processes enabling to obtain a fast and efficient yield ramp-up. Fault Diagnosis plays a crucial role in this scenario, since test can only provide information on the system behavior (good/no good). Fault diagnosis starts from the test response with the aim to locate the faulty part of the circuit and then identify which is the source of the observed failures. Unraveling the location and cause of the defect helps to improve both the circuit design and the manufacturing process, thus leading to a lower cost, an improved yield and a shorter time-to-market.

The main contributions of this work focus on the development of efficient diagnosis approach targeting digital circuits.

1.2.2.2 Power-Aware Test

Nowadays, electronic products present various issues that become more important with CMOS technology scaling. High operation speed and high frequency are mandatory requests. On the other hand, power consumption is one of the most significant constraints due to large diffusion of portable devices. These needs influence not only the design of devices, but also the choice of appropriate test schemes that have to deal with production yield, test quality and test cost.

Testing for performance, required to catch timing or delay faults, is therefore mandatory, and it is often implemented through at-speed scan testing for logic circuits. At-speed scan testing consists of using a rated (nominal) system clock period between launch and capture for each delay test pattern, while a longer clock period is normally used for scan shifting (load and unload cycles).

In order to test for transition delay faults, two different schemes are used in practice during at-speed scan testing: Launch-off-Shift (LOS) [12] and Launch-off-Capture (LOC) [13].

Although at-speed scan testing is mandatory for high- quality delay fault testing, its applicability is severely challenged by test-induced yield loss, which may occur when a good chip is declared as faulty during at-speed scan testing [15]. Both schemes (LOS and LOC) may suffe from this problem, whose the major cause is Power Supply Noise (PSN), i.e., IR- drop and Ldi/dt events, caused by excessive switching activity (leading to excessive power consumption) during the launch- to-capture cycle [16] of delay testing

schemes. In order to deal with this problem, dedicated techniques to reduce the risk of artificial yield loss induced by excessive PSN during at-speed scan testing have been proposed in the literature [17]. These techniques are mainly based on test pattern modification or power-aware Design-for-Testability (DfT).

Despite the fact that reduction of test power is mandatory to minimize the risk of yield loss, some experimental results have proved that too much test power reduction might lead to test escape and reliability problems because of the under-stress of the circuit during test [18]. So, in order to avoid any yield loss and test escape due to power issues during test, test power has to map the power consumed during functional mode. To this purpose, the knowledge of functional power for a given CUT is required and can be exploited as a reference for defining the power consumption (upper and lower) limits during power-aware delay test pattern generation for LOS or LOC.

The main contributions of this work is the development of meaningful test solutions respecting the power-constraints.

1.2.2.3 Test of Low-Power Devices

System-on-chips (SOCs) have found their application in every latest hand-held consumer devices, such as smart phones, PDAs, digital cameras and other mobile applications. This is because SOCs designed with deep-submicrometer technologies can integrate all components and functions that historically were placed on a printed-circuit board. This trend allows SOCs to embrace a variety of IP cores, such as embedded processors, MPEG encoders/decoders, DSPs, embedded memories, etc.

The need of power efficient electronic devices leads to implement dedicated structures to reduce as much as possible the power consumption. These needs influence not only the design of devices, but also the choice of appropriate test schemes or eventually design the novel DfT structures that have to deal with production yield, test quality and test cost (ITRS).

The overall SoC power consumption has two contributors: (i) dynamic and (ii) static power. While the dynamic power is strictly related to the running application, the static power is only dependent on the leakage currents. As shown in [24], static power consumption is becoming the most important contributor to the overall power consumption especially for the latest technologies (i.e., under 65nm). In order to reduce the static power consumption the so-called power-gating techniques have been proposed so far. The power-gating relies on splitting the SoC into different blocks called power islands.

Each power island can be switched on/off independently of the others. Using this technique, if one or more power islands are not used by the running application, they are switched off, in order to reduce the static power consumption.

The main contributions of this work is the analysis of the impact of defects in the power-gating circuitry and in the development of efficient test solutions.

1.2.3 Students

After my Ph.D. I have co-supervised 8 Ph.D. students and 20 master students. Table 1.1 summarizes Ph.D. students in chronological order. Master students are not listed in this table for the sake of simplicity. Section 2.2 of Chapter 2 will describe the work of each student in detail.

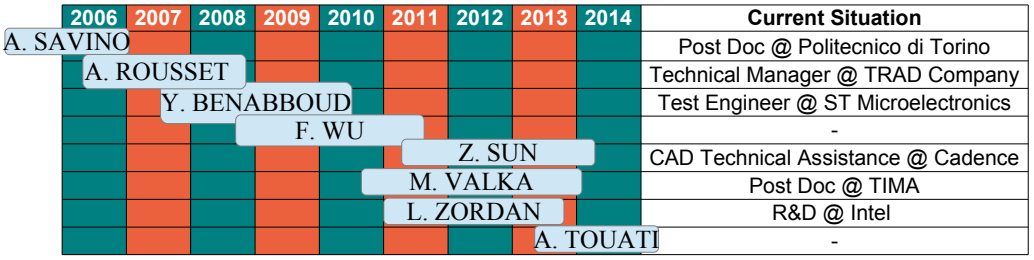


TABLE 1.1: Students Summary.

1.2.4 Research Contracts

Starting from my Ph.D., I have cooperated to national- and european-funded research contracts. The following list summarizes each contract. Details are given in Chapter 2 Section 2.4

- 2005-2008: Participation to NanoTEST project (European MEDEA+ 2A702) that has the ambition to create a breakthrough in methods and flows used by the test technologies by considering the test in the whole value chain from Design to Application. A strong consortium composed of European Semiconductor industries, Academics and Small and Medium Enterprises has grouped their competences to successfully address this challenge
- 2007-2009: Participation to the contract with STM Crolle ("Bourse Cifre")
- 2008-2011: Participation to TOETS project (European CATRENE CT302) that is the continuation of the NanoTEST project

- 2011-2013: Participation to the contract with INTEL that focuses on the test of low-power SRAMs
- 2011-2014: Participation to the contract with STE ("Bourse Cifre")
- 2011-2014: Participation to the contract with STE Grenoble ("Bourse Cifre")
- 2013-2016: Participation to LIA LAFISI (CNRS and Université de Montpellier 2 funded international laboratory) that has the following main objectives:
  - Synergize research efforts on all aspects of test and fault tolerance of hardware-software integrated systems, from the circuit to system level.
  - Promote joint education of students to research careers and set up advanced course programs, summer schools, or research internships.
  - Facilitate diffusion of a scientific culture of high quality in the field of hardware-software integrated systems.
  - Set up the necessary means for the valorization and technological transfer of research results obtained in the framework of the LIA.

1.2.5 Teaching activities

Starting from my Ph.D. studies, I have performed various teaching activities at Politecnico di Torino, University of Montpellier 2 and the USTH, students of private institutions. Table 1.2 summarizes my pedagogical activities. The column dedicated to the time spent for each subject is classified in 3 set: lectures (i.e., "Cours Magistraux" and "Travaux Dirigés"), and practical work (i.e., "Travaux Pratique").

Course	**Subject**	**Period**	**Hours**		**Location**
			Lectures	**Practical**	
Computer Science	C language	2003/04	0	25	Politecnico di Torino
		2004/05	0	25	
	C, C+ languages	from 2007/08 until now	108	114	Université de Montpellier 2
Digital System Testing	Memory Test	2004/05	12	0	Politecnico di Torino
	ATPG, Fault Simulation	from 2009/10 until now	42	16	Polytech' Montpellier
Computer Network	TCP/IP	from 2007/08 until now	75	0	Université de Montpellier 2
Digital System Design	Synthesis and VHDL language	from 2007/08 until now	30	162	Université de Montpellier 2
	Synthesis	from 2011/12 until now	30	45	University of Science and Technology of Hanoi
Advanced MPSoC Architecture	Concurrent Programming	from 2007/08 until now	105	42	Université de Montpellier 2
	Concurrent Programming and Fault Tolerance	from 2011/12 until now	30	45	University of Science and Technology of Hanoi

Table 1.2: Teaching Activities Summary.

1.2.6 Dissemination of knowledge

- International Symposium on Industrial Embedded Systems (SIES): Publication Chair (2008)
- Design and Test of Integrated Circuits (DTIS): Publication Chair (2014 and 2015)
- European Test Symposium (ETS): Publication Chair (ETS'15)
- South European Test Seminar (SETS): General Chair (2014)
- "Testing, Reliability, and Fault-Tolerance" Track Chair of ISVLSI'15
- Program Committee: ISQED (from 2009), DDECS (from 2011), ETS (from 2013)
- Web Master of the "Test Technology Technical Council" (tab.computer.org/tttc) of IEEE Computer Society (from 2012)
- Web Master of the "European Test Symposium" general home (http://www.ieee-ets.org/) from 2012
- Web Master of ETS'13
- Web Master of BTW'10, BTW'11, BTW'12 and BTW'14
- Reviewer for the following main Journals: IEEE Transaction on VLSI, IEEE Transaction on Computer, Journal of Electronic Testing - Theory and Applications, IET Computers & Digital Techniques
- Reviewer for the following main conferences: DAC, ITC, VTS, ETS, ATS
- Paper on IEEE Transaction on Computer, awarded by a movie published at "Computing Now" (http://www.computer.org/portal/web/computingnow/1211/whatsnew/tc)

1.2.7 Publications

Table 1.3 summarizes the number of my publications for each year, classified by patent, book, journal papers (with review process), papers published in official proceedings (coming from conferences or workshops with review process). The complete list of publications is given in Chapter 4.

Year	Patent	Book	Journal	Prooceding			Total
				Invited	Conference	Workshop	
2005	0	0	0	0	2	0	2
2006	0	0	0	1	6	1	8
2007	0	0	1	0	4	1	6
2008	0	0	2	0	4	1	7
2009	0	1	1	0	5	0	7
2010	0	0	2	0	5	0	7
2011	0	0	0	2	8	0	10
2012	0	0	1	1	5	0	6
2013	0	0	2	0	4	0	6
2014	2	0	2	0	4	1	10
Total	2	1	11	55			69

Table 1.3: Publications Summary.

Chapter 2

Details

2.1 Research activities

This section summarizes the research activities I carried out starting from my master thesis in 2002.

Research activities at Politecnico di Torino

Since 2002, I worked in the area of test of digital systems and dependability for safety-critical applications at the Politecnico di Torino (Italy), in cooperation with Prof. Paolo Prinetto's research group. My research activity mainly focused on the definition of new methodologies and the implementation of tools able to improve the development of highly dependable systems, at different levels: for basic digital components, for systems on chip, up to microprocessor-based systems.

My research activity, developed during 1 year of master thesis, plus 3 years of PhD and 1 years of PostDoc focused on test and functional verification of digital systems and . In particular I worked on the following topics:

- Definition of functional fault models for memories RAM and automatic generation of test sequences (Section 2.1.1);
- Definition of functional test generation methodology for RISC microprocessor cores (Section 2.1.2);
- Definition of a design environment to verify the system core-wrapper 1500-Compliance with the purpose to assure that the component can be successfully integrated in a SoC (Section 2.1.3).

Research activities at LIRMM

Starting from November 2006 I started working at LIRMM with the group of Patrick Girard and Serge Pravossoudovitch still working on the test topic but now focusing on diagnosis, power-aware test and test of low-power devices. In the following I will detail the main contributions.

The diagnosis is the process of isolating possible sources of observed failures in a defective circuit. Today, manufacturing defects appear not only in the cell interconnection, but also inside the cell itself (intra-cell defect). State of the art diagnosis approaches can identify the defect location at gate level (i.e, one or more standard cells and/or interconnections can be provided as possible defect location). Some approaches have been developed to target the intra-cell defects. In this work, we propose an intra-cell diagnosis method based on the "Effect-Cause" paradigm aiming at locating the root cause of the observed failures inside a logic cell. It is based on the Critical Path Tracing (CPT) here applied at transistor level. The main characteristic of our approach is that it exploits the analysis of the faulty behavior induced by the actual defect. In other word, we locate the defect by simply analyzing the effect induced by the defect itself. The advantage is the fact that we are defect independent (i.e., we do not have to explicitly consider the type and the size of the defect). Moreover, since the complexity of a single cell in terms of transistor number is low, the proposed intra-cell diagnosis approach requires a negligible computational time. The efficiency of the proposed approach has been evaluated by means of experimental results carried out on both simulations-based and industrial silicon data case studies. This activity is described in Section 2.1.4.

High power consumption during test may lead to yield loss and premature aging. In particular, excessive peak power during at-speed delay fault testing represents an important issue. In the literature, several techniques have been proposed to reduce peak power consumption during at-speed LOC or LOS delay testing. On the other hand, some experiments have proved that too much test power reduction might lead to test escape and reliability problems. So, in order to avoid any yield loss and test escape due to power issues during test, test power has to map the power consumed during functional mode. In literature, some techniques have been proposed to apply test vectors that mimic functional operation from the switching activity point of view. In this work, we propose a novel flow to determine the functional power to be used as test power (upper and lower) limits during at-speed delay testing. This activity is described in Section 2.1.5.

In low-power SRAMs, power gating mechanisms are commonly used to reduce static power consumption. When the SRAM is not accessed for a long period, such mechanisms allow shutting off one or more memory blocks (core-cell array, address decoder, I/O

logic, etc), thus reducing leakage currents. In order to guarantee static power reduction in low-power SRAMs, reliable operation of power gating mechanisms must be ensured by adequate test techniques. In this work, we first present a detailed analysis based on electrical simulations to identify faulty behaviors caused by realistic defects that may affect power gating mechanisms embedded in low-power SRAMs. Based on this analysis, we present an efficient test solution targeting detection of observed faulty behaviors. This activity is described in Section 2.1.6.

2.1.1 Memory Test (Master and Ph. D. Thesis)

Embedded memories are the densest components within a System on Chip (SoC), accounting for up to 90% of its real size. Today's technologies allow the design and manufacturing of memory cores with many I/O ports, and multi-port RAM core generators are commonly available in many ASIC vendors library. To have an idea of today's SoC complexity, it is enough to consider that typically more than 30 embedded memories are placed on a single chip, they are scattered around the device rather than concentrated in one location, they all have different types, sizes, and access protocols and timing, and they can even be doubly embedded inside embedded cores. Many activities have been devoted to Built-In Self-Test of embedded memories. In this context, the research activity focused then on the definition of test algorithms for memories. Among the different types of algorithms proposed to test random access memories (RAM), March Tests have proven to be faster, simpler, regularly structured and linear in complexity. A March Test consists of a sequence of March Elements, each composed by a sequence of basic read/write operations to be performed on each cell of the memory, in either ascending or descending order, before proceeding to the next memory cell. The majority of the published March Tests has been manually generated; unfortunately, the continuous evolution of the memory technology requires the constant introduction of new classes of faults, such as dynamic and linked faults, and makes the task of hand-writing test algorithms harder and not always leading to optimal results. Although some researchers published hand-made March Tests able to deal with new fault models, the problem of a comprehensive methodology to automatically generate March Tests for SRAMs, addressing both classic and new fault models, is still an open issue. This work proposed a new approach to the automatic generation of March Tests. The formal model adopted to represent memory faulty behaviors allows the definition of a general methodology to deal with both the most important classes of memory faults (Dynamic, Static, and also linked faults), and even new user-defined fault models. Experimental results show that the new automatically generated March Tests can reduce the test complexity, and

therefore the test time, compared to the well-known state of the art in memory testing of about 15%.

This work and its extension are published in [W1, C1, C2, C4, C5, C6, C8, C15, C26, C28, J1, J2, J3] referenced in the Publication list of chapter 4.

2.1.2 Microprocessor Test (Ph. D. Thesis and Post-Doc at Politecnico di Torino)

In order to guarantee very high performance, a large number of SoC designs are built around embedded RISC microprocessor or digital signal processor (DSP) cores. Nevertheless, due to the high complexity and the limited accessibility of their internal logic, embedded microprocessor cores often introduce testability issues. As an example, Scan Chains insertion is often avoided not to impact the processor performance and therefore structural testing of a microprocessor core is usually unfeasible. Several alternative approaches have been developed in the last years. The most successful are called Software-Based Self-Test (SBST) and are based on the idea of testing the microprocessor by having it executing a given and predetermined list of instructions. The list of executed instructions constitutes the test program. Test programs are usually stored either in a dedicated ROM or loaded into a RAM by an external system (i.e. ATE or the Operating System). The main advantages are that no extra area is required and the test is executed by the microprocessor itself (at-speed test). Several methodologies have been proposed to generate the test programs. The empirical approach requires neither a microprocessor nor a fault model. This approach is fairly easy to apply but lacks of a formal way to compute the fault coverage or the quality of the test program. Other researchers propose the use of pseudo-random sequences of instructions, where both operands and/or instructions could be randomized, and or deterministic sequences of instructions. The deterministic approach demonstrated to be better than the pseudo-random one both in the final coverage of functional faults and the overall test time (and therefore number of instructions in the program). Despite their efficiency, many of the suggested approaches lack of focus on the more complex functional blocks, like the pipelining interlock mechanism or the cache hierarchy. In the test of a microprocessor core used in critical applications none of these blocks can be ignored. This research activities led to a test generation methodology designed to build a non-concurrent on-line test of a RISC microprocessor core; the proposed solution allows to generate very complex test programs based on a deterministic SBST approach and to execute them under very tight timing constraints. The presented case study is a Motorola PowerPC 603 Microprocessor core for which, to our knowledge, no SBST solution has ever been published.

The proposed methodology is defined in four steps:

- Identification of all the functional components of the target microprocessor;
- Identification of the fault models for each component;
- Collection of the state of the art about the test strategies available for each component;
- Development of the test programs.

This work and its extension are published in [C3, I1, J7] referenced in the Publication list of chapter 4.

2.1.3 Functional Verification of IEEE std. 1500 Compliance(Ph. D. Thesis and Post-Doc at Politecnico di Torino)

The race to market high-volume quality products demands a shorter design-to-manufacturing cycle, forcing System-on- Chip (SoC) designers to strongly rely on Intellectual Property (IP) cores from multiple sources. The shorter time-to-volume requires faster silicon bring-up with a high degree of diagnosability. This means being able to isolate each embedded core during test and debug activities. The adoption of adequate test and diagnosis strategies is therefore a major challenge in modern SoCs production. The IEEE Standard Testability Method for Embedded Core-Based Integrated Circuits (IEEE std. 1500) addresses the specific challenges that come with testing deeply embedded reusable cores supplied by different providers, who often use different hardware description levels and mixed technologies. It defines a comprehensive set of guidelines for building the core test infrastructure. It includes:

- A Core Test Wrapper: a wrapper placed around the boundaries of the core that allows accessing its testing functionalities using a standard interface and protocol. The wrapper is completely transparent when the core is not in test mode;
- An Information Model: a formal description of the IEEE std. 1500 functionalities implemented by the Core Test Wrapper. The standard supports many functionalities, some mandatory and some optional. The Information Model is the bridge between core providers and core users and facilitates the automation of test data transfer and reuse between these two entities. The Information Model is described using the IEEE std. 1450.6 Core Test Language (CTL) and includes:
  - The set of wrapper's signals;

- The wrapper communication protocol;
- Information about test patterns.

Some literature presents solutions to build SoCs with IEEE std. 1500 testability features; nevertheless, by analyzing the standard, it is clear that implementing a fully compliant core is not a trivial task. The need to support as wide a range of embedded core test applications as possible has led to a very flexible solution. The IEEE std. 1500 specifies a mandatory minimal set of hardware support. However, a designer can extend the test infrastructure by creating virtually unlimited sets of registers and instruction extensions. An IEEE std. 1500 compliant design is therefore exposed to a range of possible design errors that require to be early identified and fixed. A comprehensive approach to thoroughly verify the functionality of IEEE std. 1500 wrappers and wrapped cores in a SoC environment is therefore mandatory. The problem of verifying the compliance of an IP core to the IEEE std. 1500 has been poorly addressed in literature. The main drawbacks of the existing solutions are that, for each core, the verification module must be configured by hand and that the verification strategy, i.e. the order used to verify each aspect (rule) of the standard, is fixed. Moreover, the authors verify the SoC and wrapper functionalities without systematically addressing every single aspect (rule) of the standard. To overcome these problems, we present a verification framework based on the use of the UML language, designed to systematically address the verification of the standard. Besides providing the actual implementation of the framework, this work focuses on the definition of an abstract model of the standard enabling core providers andor integrators to build their custom verification environments. Starting from the abstract model proposed in, it is possible to build a verification environment for the IEEE std. 1500. In particular we will show how the functionalities provided by Specman EliteTM(Cadence), a commercial functional verification EDA can be used to implement such a verification environment.

This work and its extension are published in [C9, W2, J4] referenced in the Publication list of chapter 4.

2.1.4 Diagnosis (from the Post-Doc at LIRMM)

The ever-increasing growth of the semiconductor market results in an increasing complexity of digital circuits. Smaller, faster, cheaper and low-power consumption are the main challenges in semiconductor industry. The reduction of transistor size and the latest packaging technology (i.e., System-On-a-Chip, System-In-Package, Trough Silicon

Via 3D Integrated Circuits) allows the semiconductor industry to satisfy the latest challenges. Although producing such advanced circuits can benefit users, the manufacturing process is becoming finer and denser, making chips more and more prone to defects.

Physical defects like shorts and opens will occur during each step of the fabrication process. These defects can be randomly caused by contaminations or due to systematic process-design interaction [1]. In modern deep submicron technologies, systematic defects are becoming more likely to appear than random defects. This is caused by the reduced chip sizes, the use of new complex process technologies, new materials and the increasing number of vias and contacts [2]. Today, systematic defects appear not only in the cell interconnection, but also inside the cell itself (intra-cell defect). In the literature, existing works prove that these defects can escape classical test solutions.

The test is one of the most critical tasks in the semiconductor production process. It is not only necessary to seek for fault free devices but it plays a key role in analyzing defects in the manufacturing process as well. The feedbacks derived from the test process are the only way to analyze and isolate many of the defects in today's processes enabling to obtain a fast and efficient yield ramp-up.

Fault Diagnosis plays a crucial role in this scenario, since test can only provide information on the system behavior (good/no good). Fault diagnosis starts from the test response with the aim to locate the faulty part of the circuit and then identify which is the source of the observed failures. Unraveling the location and cause of the defect helps to improve both the circuit design and the manufacturing process, thus leading to a lower cost, an improved yield and a shorter time-to-market. State of the art fault diagnosis approaches can identify the defect location at gate level (i.e, one or more standard cells and/or inter-connections can be provided as possible defect location) [30]. The fault diagnosis results (i.e., possible defect locations) are then further used in defect analysis, where the circuit is physically examined to determine the mechanism of the failure. Physical Failure Analysis (PFA) is physical analysis that corresponds to the physical identification of the defect. It mainly consists in selective de-layering and cross-sectioning of a die. PFA is not only crucial and time-consuming but also destructive and irreversible. Therefore, a preliminary diagnosis procedure is mandatory to correctly guide the PFA to eventually save time and increase success rate.

As previously discussed, state of the art fault diagnosis approaches are able to locate the possible defect at gate level (i.e., inter-cell). Therefore, in the case of circuit affected by intra-cell defects, results of inter-cell fault diagnosis may cause problems and impact the efficient of the PFA. So that, PFA may take more time to identify the actual defects. Moreover, in the worst case, PFA may fail (i.e., it does not identify the root cause of

the observed error) and it can destroy the circuit. The solution is to develop an efficient diagnosis approach able to target intra-cell defects.

To the best of our knowledge, commercial tools only target inter-cell fault diagnosis, while in the literature some research works already addressed the intra-cell fault diagnosis problem [10, 11, 31]. The approach of [11] is based on the use of a defect dictionary. The dictionary is created by means of defect injection campaign at transistor level. During diagnosis, the observed failures are used to search in to the defect dictionary the most suitable defect location and type. However, the need of pre-computed defect dictionary for each cell and defect type makes this approach highly complex. Moreover, if the injected defect is not accurate enough it can lead to erroneous results during diagnosis.

The approach of [31] is based on the use of a fault dictionary. The main difference w.r.t. [11] is that the dictionary is created exploiting a switch-level simulation (i.e., the transistors are considered as switches). Thus, instead of injecting defects, only fault models are injected. The advantage is the reduced simulation time compared to [11]. Two types of fault are considered, the stuck-at and dominant bridging fault. These faults are modeled at switch-level in order to be simulated and to create the related fault dictionary. However, defects leading to delay faults are not targeted. Moreover, to include delay faults or other types of faults a switch-level model of them has to be defined.

The approach of [10] proposes to convert transistor level netlist into an equivalent gate level netlist. Then classical inter-cell fault diagnosis tools can be applied. The main drawback of this approach is that the set of transformation rules depends on targeted defect types.

Due to the drawbacks and limitations of previous works on intra-cell diagnosis, it is necessary to develop an efficient and accurate intra-cell diagnosis solution to ensure the PFA success rate. In this work we propose a new intra-cell diagnosis approach able to provide accurate defect localization in order to improve the efficiency of PFA and to eventually save cost and time. The main characteristic of our approach is that it exploits the analysis of the faulty behavior induced by the actual defect. In other word, we locate the defect by simply analyzing the effect induced by the defect itself. Thus, conversely to previous work on intra-cell diagnosis [11], there is no need to characterize the library to create a defect dictionary. The faulty behavior analysis is performed by applying the Critical Path Tracing (CPT) at transistor level. Compared to [10, 31] there is no need to pre-compute any fault dictionary and to convert the netlist.

2.1.4.1 From Defects To Fault Models

In this section, we present the main causes of physical defects and how these defects are usually modeled at transistor-level domain. Then, we discuss about the faulty behavior induced by defects and how the fault models represent them. This analysis is important in order to give the basics on how the proposed approach is defect independent while targeting transistor-level circuit descriptions. A physical defect can be caused by several phenomena such as metal line broken/deformed, contact or via broken/deformed. These phenomena will lead to either an unexpected connection between two or more nets (short) or a missing connection on one net (open). A net, at transistor-level domain, correspond to different elements: polysilicon (the transistor gate terminal), active (the transistor drain and source terminals), metal line (interconnections between transistors), contact (connection trough active and metal level 1) or via (connections trough metal levels). Usually, these defects are modeled at transistor-level domain as: (i) an unexpected connection between two nets associated to a specific resistance value (resistive-short), (ii) an unexpected resistance value on a given net (resistive-open). Depending on the considered resistance value, the effect induced by the defect can vary. Thus, the choice of meaningful resistance values is crucial to have an accurate model of physical defects. Existing works on intra-cell diagnosis rely on this way to model defects to create the defect dictionary.

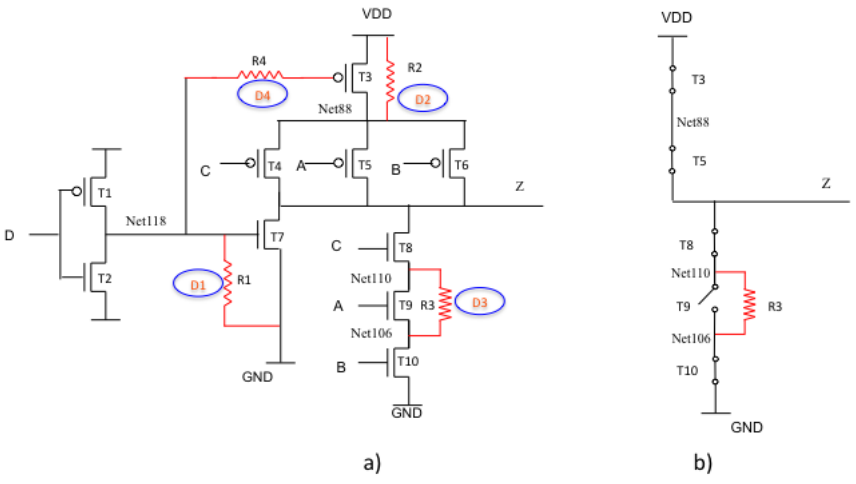


FIGURE 2.1: Physical defects modeling at transistor-level domain b) Equivalent circuit when the stimuli "0111" are applied.

Figure 2.1.a) shows an example of physical defects and the related model at transistor-level domain for a complex gate composed of four primary inputs (A, B, C and D) and one primary output (Z). In the transistor-level netlist four defects are highlighted in red.

Please note that the four defects are only an example, the proposed approach can target any possible defects.

Defect D1 models an unexpected connection between net118 and ground. The behavior of this defect depends on the resistance value R1. If R1 is lower than a threshold value RT, then the Vgs (voltage level between gate and source terminal of transistor T7) is always lower than Vth of T1. Thus T1 remains switched off (faulty behavior). If R1 is grater than RT, then Vgs depends on the voltage level of Net118 (correct behavior). From a logical point of view the first case (when R1 <RT) is equivalent to have Net118 stuck at logic value '0'. A similar consideration can be done for defect D2. Depending on R2 value, it impacts the voltage level of net88. From a logical point of view it is equivalent to have net88 stuck at logic value '1'.

Defect D3 models an unexpected connection between net110 and net106. The behavior of this defect depends on the resistance value R3 and also on the applied stimuli (i.e., it can activated or not). For example Figure 2.1.b) gives the circuit when the stimuli "0111" are applied to inputs ABCD respectively. Transistors T8 and T10 are switched-on while T9 is switched-off and output Z is set to logic '1'. However, due to the presence of this defect, the output is now connected to the ground through R3. Thus the defect is activated. Once the defect is activated we analyze the R3 value to determine its behavior. If R3 is lower than Rmin then, the output will be set to logic value '0'. In this case, from a logic point of view the Net106 force its logic value '0' to the Net110 and then the output is changed. On the other hand, if R3 is greater than Rmin but lower than Rmax (Rmin <R3 <Rmax) then, output Z will be affected by an undesired transition (from logic '1' to '0') due to the long discharge time. Finally, if R3 is grater than Rmax then output Z will take the correct logic value and the circuit behavior is faulty free. Note that the value of Rmin and Rmax depends on the circuit technology.

Finally, defect D4 affects the net Net118. For certain values of R4 the signal propagation trough gate of transistor T4 is delayed. To summarize, the analyzed faulty behavior induced by the four defects of Figure 2.1.a) are:

- D1, D2: the faulty behavior results in a net always set to a given logic value (either '1' or '0');
- D3: the faulty behavior results in a net set to a given logic value (in the example logic '0') depending on the input configuration or in a signal propagation delay affecting primary output Z;
- D4: the faulty behavior results in net where the signal propagation is affected by a given delay.

Even if we target the transistor-level domain, we exploit the knowledge of the analyzed faulty behavior to be defect independent. Thus, we can avoid to explicitly considering the resistance value. We will show in the next section that the proposed intra-cell approach identifies the possible locations of a defect. Usually faulty behaviors induce by defects are represented by means of fault models. From above the example we can list the exploited fault models:

- Stuck-at fault model [28]: the logic value of a given net appears to be stuck at a constant logic value ('0' or '1'), referred as stuck-at-0 or stuck-at-1 respectively (e.g., defects D1 and D2 in Figure 2.1).
- Dominant Bridging fault model [28]: this fault involves two nets called aggressor and victim. The logic value of the victim is set to the logic value of the aggressor (e.g., defect D3 in Figure 2.1).
- Delay fault model [28]: the transition from a given logic value V to the opposite logic value !V is delayed. Two types of delay faults are defined: slow-to-rise transition fault model (slow transition from logic '0' to logic '1') and slow-to-fall transition fault model (slow transition from logic '1' to logic '0').

Since the proposed diagnosis approach provides possible defect locations, for each of them we will associate one or more fault models according to the observed faulty behavior.

2.1.4.2 Intra-Cell Diagnosis Flow

In this section, we present the whole intra-cell logic diagnosis approach. It is able to locate the root cause of observed failures inside a logic cell. Since the proposed approach works at transistor-level, it cannot be applied to the whole circuit due to its complexity (i.e., billions of transistors). However, it can be easily applied to a single target logic cell (i.e., up to fifty transistors) identified by a logic-level diagnosis tool.

Figure 2.2 sketches the overall diagnosis flow. First of all, the **test** applies test patterns to the DUT (Device Under Test) to distinguish between the correct circuit behavior and the faulty circuit behavior caused by defects. These defects induce failing output responses for one or more input test patterns. Input test patterns leading to observed faulty behavior are called failing test patterns and stored in to a file called datalog. Input patterns for which no faulty behavior is observed are called passing test patterns. Then, the **inter-cell fault diagnosis** exploits datalog information to determine a list of suspected logic cells (i.e., candidates). Any available commercial diagnosis tool can

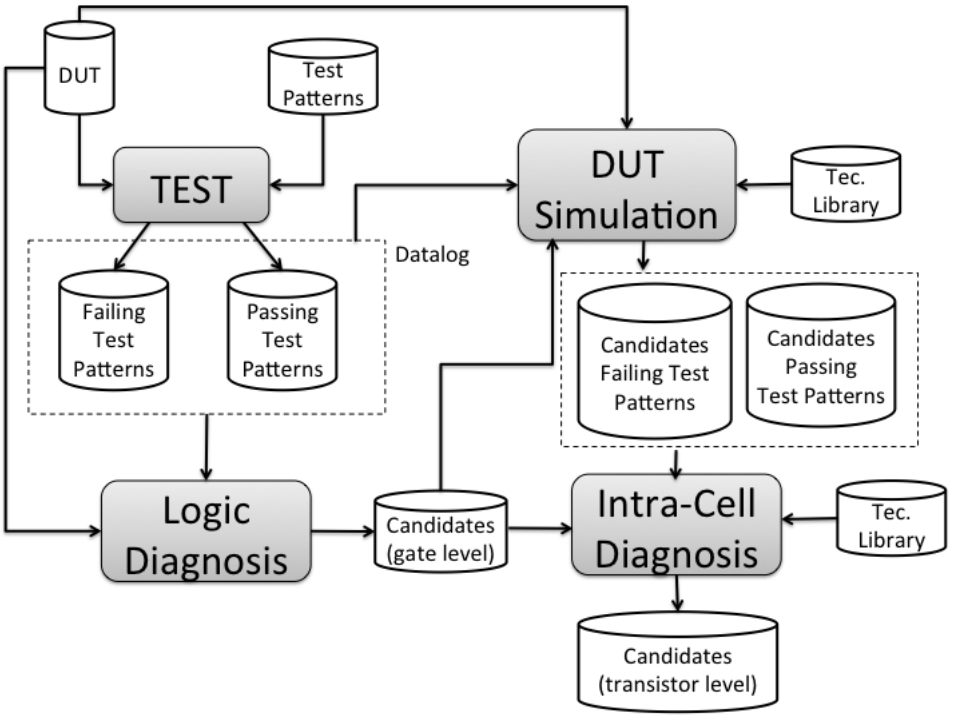


FIGURE 2.2: Overall diagnosis flow.

be adopted. For each suspected cell, we have to know logical values applied to it when failing and passing test patterns are applied to the DUT (i.e., DUT simulation). **DUT simulation** aims at determining the local set of failing/passing patterns for each suspected logic cell reported by inter-cell diagnosis. Finally, the **intra-cell diagnosis** is executed for each Suspected Gate (SG) and the pre-determined local failing/passing test patterns set (*lfp* and *lpp*). The intra-cell diagnosis result is a list of candidates at transistor level. For each suspected net a set of fault models able to explain observed failures is associated.

2.1.4.3 DUT Simulation

The intra-cell diagnosis is applied on a single candidate identified as the Suspected Gate (SG). The preliminary step of the proposed intra-cell diagnosis approach aims at determining the local failing/passing patterns defined as *lfp* and *lpp* respectively.

Figure 2.3 shows the SG located in the circuit. When a failing test pattern fp is applied to the circuit PIs, the fault affecting the SG is sensitized and its effect is then propagated to at least one circuit PO. This is guaranteed by the fact that during the test a failure is observed when the fp is applied to the circuit. Since the intra-cell diagnosis is applied to the SG only, we have to known the logical values of the SG inputs (called local patterns) when the fp is applied to circuit PIs. Thus, a logic simulation of the fp is required to get

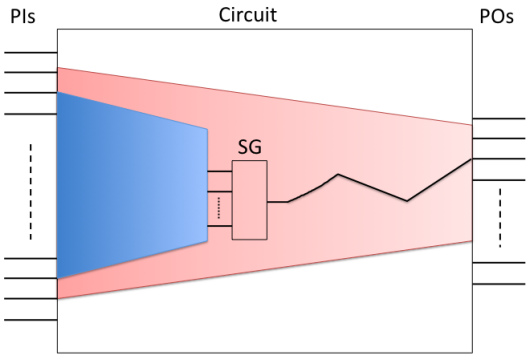


FIGURE 2.3: Local Failing/Passing test patterns.

those values. This simulation has to be performed for each given failing test pattern. In a similar way, we have to know which are the logical values applied to the SG in the case of passing test patterns. A test pattern does not detect any fault for two reasons: (i) the fault is not sensitized (ii) the fault is sensitized but is effect is not observed because the fault effect cannot be propagated to reach the primary outputs. For the passing test pattern, we have to discriminate the reason why the fault located in the SG has not been detected (i.e., discriminate between (i) and (ii)). Thus, we have to verify if the fault effect can be propagated or not. If yes, then the pattern could be considered as a local passing test pattern because any failures should be observed during the test.

However, one more consideration must be done. In section 2.1.4.1 we described the faulty behavior considered in this work. Some of them depend only on the local gate input values (i.e., stuck-at and bridging faults). Some others depend not only on the local gate input values but also on the previous local values (i.e., delay faults). Taking in to account this consideration, we can now classify a given lpp as follows:

- If we assume that the defect affecting the suspected gate SG leads to a static faulty behavior (i.e., stuck-at, bridging) then, lpp is classified as local passing pattern;
- If we assume that the defect affecting the suspected gate SG leads to a dynamic faulty behavior (i.e., delay) then, lpp can not be classified as local passing test pattern, because we do not consider the previous pattern.

At this stage of the diagnosis flow, we must consider as valid both the above assumption. Therefore, we will store the lfp and lpp in to different data structure associated to the static faulty behavior and to the dynamic faulty behavior respectively. Finally, for the case of defects leading to a dynamic faulty behavior, a sequence of test patterns has to be applied to detect the defects. It may be possible that the same local pattern could be

declared falling and passing. For this case, we knew that the defect affecting the circuit is a dynamic type so that only delay fault will be targeted thus discarding both stuck-at and bridging faults. The analysis performed to determine local failing and passing test patterns leads to the taxonomy shown in Figure 2.4.

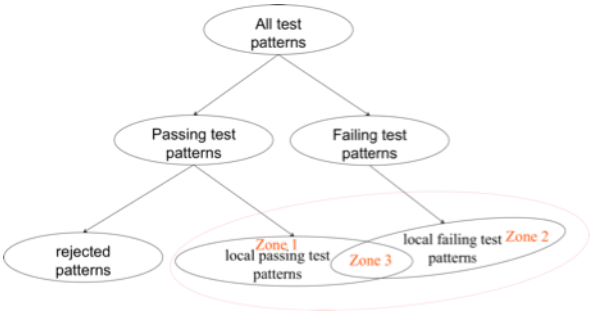


FIGURE 2.4: failing and passing local test patterns taxonomy.

After the test, patterns can be classified into two categories: failing and passing test patterns.

- **Definition 1**: Failing test patterns are used to determine local failing test patterns. Zone 2 in Figure 2.4 shows this type of test patterns.
- **Definition 2**: Passing test patterns are used to determine local passing test patterns. The passing test patterns illustrated in zone 1 of Figure 2.4.
- **Definition 3**: When at least one local test pattern is declared at the same time failing and passing ($lfp \cap lpp \neq \emptyset$) as shown in the zone 3 of Figure 2.4, it means that the defect affecting the circuit leads to a dynamic faulty behavior. In this case, we can discard the static faulty behavior (i.e, static and bridging) to be the root cause of observed failures.
- **Definition 4**: If $lfp \cap lpp = \emptyset$, then defect affecting the circuit can lead either to a dynamic or static faulty behavior. In this case, we must consider both static and dynamic faulty behavior can be to be the root cause of observed failures.

In the next section we will describe in detail the applied intra-cell diagnosis approach and how the information extracted during DUT simulation are exploited.

2.1.4.4 Effect-Cause Intra-cell diagnosis algorithm

In this section we present the proposed intra-cell diagnosis approach. It is based on the "Effect-Cause" paradigm and it exploits the Critical Path Tracing (CPT) algorithm here applied at transistor level.

The proposed intra-cell diagnosis approach requires two main inputs: the SG description at transistor level and the local set of failing/passing test patterns. Figure 2.5 gives the pseudo code of the intra-cell diagnosis procedure.

```
INTRA-CELL DIAGNOSIS(SG, lfp, lpp)
1  GSL = GDSL = all SG nets and transistor terminals
2  GBSL = all SG couple of nets
3  for each lfp
4        do
5           FAULT-FREE-SIMULATION(SG, lfp)
6           CPT(SG, lfp, CSL, BSL, DSL)
7           GSL = GSL ∩ CSL
8           GBSL = GBSL ∩ CBSL
9           GDSL = GDSL ∩ CDSL
10
11 for each lpp
12       do
13          FAULT-FREE-SIMULATION(SG, lpp)
14          CPT(SG, lpp, CVL, CBVL)
15          GSL = GSL − CVL
16          GBSL = GBSL − CVSL
17
18 FAULT-ALLOCATION(GSL, GBSL, GDSL)
```

FIGURE 2.5: Intra-cell diagnosis pseudo code.

The procedure is divided in two blocks identified by the two "for loop" statements. The first one targets the lfp. For each lfp, a fault-free simulation is performed by using a switch-level simulation. In the switch-level simulation, the transistors (i.e., nMOS and pMOS) behave as on-off switches.

Since the simulated netlist is composed of few transistors the required simulation time is negligible. The fault-free simulation is mandatory to determine the logic value of each net. Then, Critical Patch Tracing (CPT) is executed starting from the SG output. It traces back internal nets to reach the SG inputs. Each traced net is set as critical. A net is critical if the inversion of its logical value causes the inversion of the output value. Each critical net is marked as a suspect. A suspect can be the root cause of observed errors or simply a net that propagate the fault effect. Each suspect is stored in a list with its logic value. The logic value is kept in the list because it will be used during the fault model allocation. The result of the CPT is the Suspect List (SL). The SL is defined as follows:

$$SL = \{(net_0, LV_0), (net_1, LV_1), ..., (net_n - 1, LV_n - 1)\} \tag{2.1}$$

where:

- net_i: is the critical net (transistor-level interconnection nets and transistor terminals).
- $LV_i = \{0, 1, U\}$: is the logic value of the critical net. U is the unknown value.

As stated before, a given suspect can simply propagate the fault effect (i.e., it is not the root cause of observed errors). This case can happen for two reasons:

1. The suspect net belongs to the propagation path of the actual faulty net. In this case, the suspect is logically equivalent to the actual faulty net.
2. The suspect net is indeed the victim of a bridging fault. In this case, another net (i.e. the aggressor) "forces" a faulty value on the victim.

The second case is more complex than the first one from the diagnosis point of view. Here, we have to verify if a bridging fault is possible. For this reason, after the CPT a second list of suspects is created. The so-called Bridging Suspect List (BSL) contains all the possible couples "Victim/Aggressor" that can be involved in a bridging fault. The victim belongs to the actual SL, while the aggressor can be any net having an inverted logic value w.r.t. the logic value of the victim. The BSL is defined as follows:

$$BSL = \{(V_0/A_0), (V_1/A_1), ..., (V_m - 1/A_m - 1)\} \tag{2.2}$$

where:

- $V_i \in SL$: is the victim.
- $A_i = (Net_j, LV_j)$: is the aggressor. It can be any net of the gate having a logic value opposite to the logic value of the victim. Thus $LV_j = !LV_i$.

So far (equations 2.1 and 2.2), we defined as critical nets for which the inversion of their logical value causes the inversion of the output. With these two lists, we are able to identify any static faults (see Section 2.1.4.1). To complete our analysis, we have also to consider the case of dynamic faults affecting the analyzed gate and thus the circuit (see Section 2.1.4.1). As for the equation 2.1 we look for critical nets, where a critical net can be either a transistor terminal or an interconnection net. We define those nets as critical delay nets. Critical delay nets are added in a list called Delay Suspect List (DSL) defined as follows:

$$DSL = (Net_0, Net_1, ...Net_n - 1) \tag{2.3}$$

where:

- Net_i is the critical net (transistor-level interconnection nets and transistor terminals).

Please note that in this case, the logical value of a critical delay net is not stored in the list because it is not used during the fault model allocation. Basically we do not distinguish between a slow-to-rise/slow-to-fall delay fault. The three lists are stored in a so-called **Current Suspect List** (CSL), the **Current Bridging Suspect List** (CBSL) and the **Current Delay Suspect List** (CDSL). The assumption of this work is the presence of only one defect on a given circuit. Thus, the root cause of observed errors has to be present in all lists provided by the CPT application. For this reason, we update the global suspect lists by performing an intersection (line 7, 8 and 9 in Figure 2.5). The intersection between two suspects lists SL_a and SL_b is defined as follows:

$$SN = (net_i, LV_i) \in SL_a \cap SL_b if (SN \in SL_a) and (S_N \in SL_b) \tag{2.4}$$

This definition removes a net from the suspect list if the net is traced with different logic values (e.g., once with '0' and another with '1'). This is coherent with the stuck-at fault model. If a net is affected by a stuck-at fault, its value must be always the same during the failing test patterns application. Otherwise, the net cannot be affected by a stuck-at fault. Thus it is removed from the suspect list.

The intersection between two bridging suspect lists elements $BSLE_a$ and $BSLE_b$ is defined as follows:

$$BSLE_a \cap BSLE_b = (net_i, LV_i \cap LV_m)/(net_j, LV_j \cap LV_n) if (net_i = net_m) and (net_j = net_n) \tag{2.5}$$

The intersection between logic values is defined in the Figure 2.6

The above leads to keep a couple Victim/Aggressor (V/A) even if it appears in two lists with different logic values. Conversely to the case of a stuck-at fault, this case can occur if a strong dominant bridging fault is the root cause of observed failures.

The intersection between two delay suspects lists is similar to the one for SL (suspects lists). The difference is that there is no logic value associated to the delay candidates. The intersection between two delay suspects DSL_a and DSL_b is defined as follows:

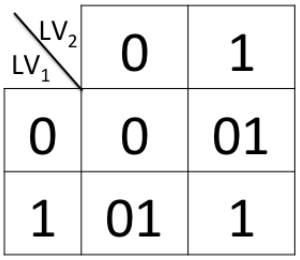


FIGURE 2.6: Logic values intersection.

$$DSN = (net_i) \in DSL_a \cap DSL_b if (DSN \in DSL_a) and (DSN \in DSL_b) \tag{2.6}$$

The DSN is kept if and only if it belongs to all DSLs. Please note that the logic value is not stored in the DSL (equation 2.6), thus, we don't need to consider the logic value of candidates during the DSLs intersection.

The result of the intersection is stored in the so-called **Global Suspect List)** (GSL), the **Global Bridging Suspect List** (GBSL) and the **Global Delay Suspect List** (GDSL).

The second block of the intra-cell diagnosis procedure aims at applying the CPT for each lpp to vindicate the suspected elements. The main concept behind this block has been already exploited for the inter-cell diagnosis. This step is applied for the GSL and GBSL list but not for the GDSL. This is because we can determine local passing pattern only for the case of static fault, thus we can vindicate only for GSL and GBSL. As for the first block, we create two suspect lists. The first one contains critical nets called Vindicate List (VL). In this case each critical net is vindicated to be the root cause of the observed failure. The second list contains all possible couple V/A that can be vindicated to be involved in a bridging fault called Bridging Vindicate List (BVL). The victim belongs to the actual VL, while the aggressor can be any net having an opposite logic value w.r.t. to the logic value of the victim.

We compute the difference between the vindicate lists and the suspect lists (lines 15 and 16 of Figure 2.5). Owing to the knowledge of the actual passing test patterns it is possible to narrow down the actual list of suspects.

The difference between a suspect list SL and a vindicate list VL is defined as follows:

$$SN = (net_i, LV_i) \in (SL - VL) if (SN \in SL) and (SN \ni VL) \tag{2.7}$$

In the same way the difference between a bridging suspect list BSL and a bridging vindicate list BVL is defined as follows:

$$SBN = (net_i, LV_i)/(net_j n LV_j) \in (BSL - BVL) if (SN \in SL) and (SN \ni VL) \quad (2.8)$$

Finally, the last step of the intra-cell diagnosis is the fault model allocation. Basically for each suspect we exploit the stored logic value to associate a fault model. Once again, three types of fault models are considered: (i) the stuck-at fault, (ii) the dominant bridging fault and (iii) the delay fault. During the fault model allocation, it could be happen that one or several suspected lists are empty. This means that the root cause of observed errors cannot be the fault model corresponding to the empty suspected list. For example, when $lfp \cap lpp \neq$ only dynamic faults are possible. So that, after suspected lists construction for lfp and lpp, GSL and GBSL will be empty and we consider only faults in the GDSL.

2.1.4.5 Experimental Results

The proposed intra-cell diagnosis procedure has been implemented in C++. It has been validated by means of simulations and actual silicon data. The simulation-based validation exploits a defect injection campaign. Physical defects can take forms of missing or extra materials and they are often modeled by open- and short-circuits as presented in. Defects are thus injected into the transistor-level netlist of a given gate of a DUT. Then, by using a spice simulator, the faulty gate is simulated in order to determine its truth table. The truth table is then used as library model, so that the whole faulty circuit is simulated at gate level to emulate the test phase. Observed failures are stored in the failure file (i.e., datalog). Injected defects lead to stuck-at, bridging and delay faults. The silicon-based validation has been carried out on STM circuits declared faulty during the production test.

Simulation-based validation

We used two circuits named A and B to perform the simulation-based validation. The circuits correspond to actual STM products and have been synthesized with a STM 90nm technology. Table 2.1 shows the characteristics of the circuits in terms of gates, FFs and scan chains.

For each logic cell in the targeted technology library, we randomly injected one defect at time and we perform spice simulation to characterize the behavior of the faulty logic

Circuit	#Gate	#FlipFlop	#ScanChain
A	258	30	1
B	69,8804	56,373	25

TABLE 2.1: Circuit Characteristics.

cell. Injected defects lead to have 3 types of faulty behaviors. The 30% of them lead to stuck-at faults, the 30% lead to bridging faults and the remaining 40% lead to delay faults.

Then, we randomly select one gate in the target circuit and we replace it by the faulty gate. We perform the simulation and we store observed failures, if any, in the datalog. For each datalog we then run a commercial logic diagnosis tool to identify our suspected gate.

The test patterns applied for the experiments have been generated exploiting a commercial ATPG tool. The test sets target transition fault models and the test length is 25 and 500 patterns for circuit A and B respectively. After the DUT simulation step, we got in average 3 local failing patterns and 6 local passing patterns.

Table 2.2, Table 2.3 and Table 2.4 show the achieved intra-cell diagnosis results for stuck-at, bridging and delay faults respectively. The first column reports the suspected gate name (given by the logic diagnosis tool). The second and third column report the number of input and output of the gate. Column 4 shows the gate complexity in terms of internal gate nets. The fifth column gives the actual injected in terms of location and type. Finally, column 6 shows the result of the proposed intra-cell diagnosis approach. First of all the results show the accuracy of the proposed approach because in all the cases, the injected defect has been correctly identified. The second comment is related to the resolution. Basically the resolution depends on the injected defect. For the case of defects leading to stuck-at faults (Table 2.2), two out of five cases have only one stuck-at fault reported and one case has two equivalent faults identified. For the other two cases, the proposed intra-cell diagnosis method pointed out 3 equivalent stuck-at faults. Please note that the diagnosis results contain also some bridging and delay faults. The average size of suspect stuck-at and bridging faults list is 7. For the case of defects leading to bridging faults (Table 2.3), four cases have only one fault identified and the other one case has 2 equivalent faults reported. Moreover, the proposed approach leads to an empty list of suspected stuck-at and delay. This happens because the behavior of a bridging fault cannot be modeled by a stuck-at fault (in most of the cases). Finally, concerning the resolution of defects leading to delay faults, we have than the best result identifies two suspects, while the worst result has 5 suspects. For example, for the first case AO7NHVTX1, the injected fault is delay fault affecting N2D (the drain of the

transistor N2). Intra-cell diagnosis has identified the transistor N2 and P5 as suspects. When a transistor is identified as suspect, all of the three terminals of this transistor are suspected. The proposed approach leads to an empty list of suspected stuck-at and bridging. This happens because the behavior of a delay fault cannot be modeled by a stuck-at and bridging fault (in most of the cases). To conclude, the CPU time required for the intra-cell diagnosis is lower than 1sec. The low execution time and the small number of suspect candidates will save time during FA procedure.

SG	Input	Output	Complexity	Injected Fault	Results
AO7SVTX1	3	1	6	N16 Sa1	N16 Sa1; Input A Sa0
NR3ASVTX1	3	1	7	N022 Sa0	N022 Sa0; N029, Input A Sa1
AO6CHVTX4	3	1	8	N113 Sa0	Input C, N147 Sa1; N113 Sa0
AO8DHVTX1	4	1	9	Input A Sa1	Input A Sa1
AO5NHVTX1	3	1	9	N71 Sa0	N71 Sa0

TABLE 2.2: Stuck-at-Faults Results.

SG	Input	Output	Complexity	Injected Fault	Results
AO7SVTX1	3	1	6	Z-Gc	Z-Gc
AO7NHVTX1	3	1	7	N50-Gc	N50-Gc
AO6CHVTX4	3	1	8	N113-N109	N113-N109, N113-N125
AO5NHVTX1	3	1	9	N55-A	N55-A
AO9SVTX1	5	1	10	N22-N31	N22-N31s

TABLE 2.3: Bridging-Faults Results.

SG	Input	Output	Complexity	Injected Fault	Results
AO7SVTX1	3	1	6	N2D	N2, P5
AO8DHVTX1	4	1	9	Net118	Z, Net118, D
AO5NHVTX1	3	1	9	N0D	N0, N1, P7, Net55, Z
AO9SVTX1	5	1	10	P4S	Z, Net9, P4

TABLE 2.4: Delay-Faults Results.

In the last part of the simulation-based validation, we run an extensive set of experiments. For each targeted library cell we randomly select in the circuit (circuit B in Table 2.1) 100 cell instances. For each cell instance we inject 10 random defects for a total of 1000 diagnosis run. The test set is the same than the one used for the previous experiments. Table 2.5 reports the select library cell name, the input output number and the complexity in the first four columns. Last column gives the average resolution expressed as the average number of candidates provided by the proposed approach. The select cells have different complexity from 6 up to 24 transistors and different number

of input. The main objective of this experiment was to validate the efficiency of the approach on more complex cells. In this sense we achieved good results since in average we have about 1 candidate. However, two cases lead to have about 4 candidates. For these cases the reason why the resolution is worst than the others is the few number of inputs (i.e., only two) that corresponds to a limited set of local patterns (i.e., up to four).

SG	Input	Output	Complexity	Resolution
AO7SVTX1	3	1	6	1.5
AO7NHVTX1	3	1	6	2
NR3ASVTX1	3	1	7	1.8
AO6CHVTX4	3	1	8	2.1
AO8DHVTX1	4	1	9	1.7
AO5NHVTX1	3	1	9	1.4
AO9SVTX1	5	1	10	1.2
AN2BHVTX8	2	1	18	4.1
MUX21HVTX6	3	1	24	1.03
ND4ABCHVTX8	4	1	23	1.1
EOHVTX6	4	1	14	1.3
OR4ABCDHVTX4	4	1	14	1.3

TABLE 2.5: Experimental Results.

Silicon-based validation

To finally prove the efficiency of the proposed approach, we used some actual faulty circuits to highlight the effectiveness of the proposed intra-cell diagnosis approach. All the experiments are carried out on STM actual circuits. Table 2.6 shows the characteristics of circuits in terms of technology node, gates, FFs, scan chains and test patterns number. Test patterns target stuck-at, transition and bridging faults. After the DUT simulation step we got a number of local passing patterns varying from 5 to 7 depending on the circuit and the applied test set. The number of local failing patterns varies from 2 to 4.

Circuit	Technology	#Gate	#FlipFlop	#ScanChain	#Patterns
H	90nm	698,804	56,373	25	500
M	90nm	896,417	60,006	219	1,055
C	55nm	1,995,419	183,868	43	1,000

TABLE 2.6: Circuit Characteristics.

Circuit H

Three faulty cases of circuit H were used as case study for the proposed intra-cell diagnosis approach. Table 2.7 shows the logic diagnosis, intra-cell diagnosis results and the actual defect location with defect mechanism identified after physical analysis.

Sample	Logic Diag	Intra Diag	Actual defect
H1	U2890/Z (sa01)	A-Z, A-B, A-C	A-Z
H2	U280/Z (sa0), U280/C (sa1),U280/D (sa1)	Net61 (sa0), A-Net61	Net61 metal1 bridging with gnd
H3	U280/A (str)	N0S (open)	N0S metal 1 open

TABLE 2.7: Logic diagnosis vs intra-cell diagnosis vs actual defect.

The first column gives different circuits names and, the second column presents the logic diagnosis results. The third column shows the intra-cell diagnosis results and the last column gives the actual defect location and the defect mechanism. Logic diagnosis gives one logic cell as candidate. Please note that for the case of H2, the logic diagnosis provides 3 candidates. However, the 3 candidates refer to the same cell "U280". Targeting these logic cells, intra-cell diagnosis is applied. For all the cases, the proposed intra-cell diagnosis gives the actual defect. The resolution is quiet good. One case has only one candidate. For the other two cases, there are two and three equivalent faults obtained.

For the sample H1, intra-cell diagnosis was applied to the given suspect gate U2890. After intra-cell diagnosis, 3 bridges inside the gate were identified as suspects (Table 2.7). **i) Bridge between Input A and output Z, ii) bridge between Input A and Input B and iii) bridge between Input A and Input C**. Input A is always the aggressor for every case. During Physical Failure Analysis (PFA), a FIB cross section is performed to verify the intra-cell diagnosis result. Figure 2.7 gives the photo showing the bridge defect identified by PFA. The actual defect is the **bridge between input A and output Z**. This result proves the effectiveness of our intra-cell diagnosis approach and the proposed diagnosis flow.

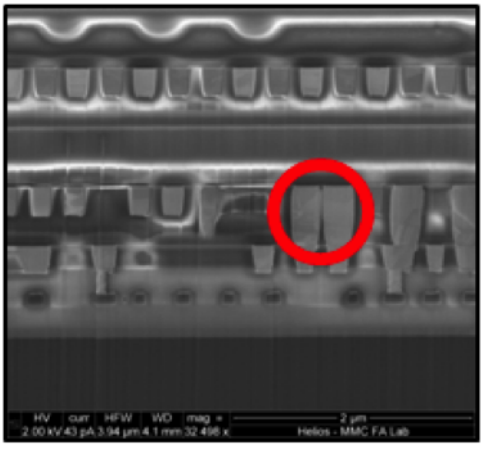


FIGURE 2.7: Physical failure analysis result.

For the sample H2, logic diagnosis identifies three nets leading to one suspect cell (Table 2.7). On the suspect gate the intra-cell diagnosis gives 2 suspects: **Net61 stuck-at 0**

and bridge between Net61 and Input A. The actual defect is the net61 shorted to GND and thus showing a behavior like a stuck at 0. For the case H3, logic diagnosis tool gives one gate as candidate associated to the slow to rise transition fault (StR) at the input A. In this case intra-cell diagnosis gives just one suspect the source terminal of transistor N0 that is the actual defect. Once again, this proves the efficiency and accuracy of our approach.

Circuit M

For circuit M, logic diagnosis identifies a stuck-at 0 fault at the output of the gate I552 (AO7HVTX1). The intra-cell diagnosis flow is then applied to this gate. Two open defects have been identified as suspects. Figure 2.8.a) depicts the netlist at transistor level of the suspect gate AO7HVTX1 and shows the two identified open defect in red.

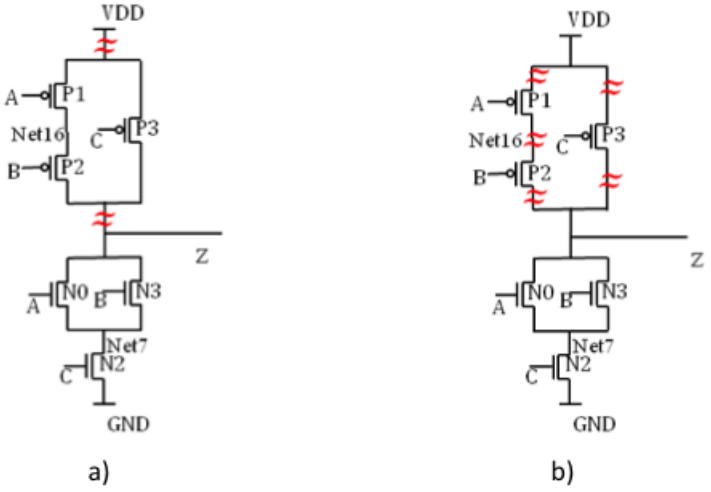


FIGURE 2.8: Physical failure analysis result.

To verify the intra-cell diagnosis result, we performed the PFA. In this case, the PFA identified a multiple open defect (5 contacts are deformed and missing). Figure 2.8.b) reports in red the 5 open defects on the transistor level netlist of the suspect gate.

Since our approach is based on the single defect assumption, the result of the intra-cell diagnosis is two open defects equivalent to the real multiple defects. Even if the diagnosis is not correct, it is very interesting to note that the suspect location identified by our approach include the actual defect position. That means that even in the presence of a multiple defect, the intra-cell diagnosis can provide useful information to correctly guide the PFA.

Circuit C

For circuit C, logic diagnosis gives a composite stuck-at fault model 01 at the output of gate U32362 as suspect. Then, the intra-cell diagnosis is thus applied to this gate. The intra-cell diagnosis flow provides an empty list of suspects. This result means that the actual defect is not inside a gate (i.e., it is an inter-cell defect). Thanks to this result, the PFA is applied to search an inter-cell defect. The analysis has been carried out closest nets to the suspect gate. Layout of these nets is shown in Figure 2.9 (left picture). The PFA successfully identified the actual defect as a bridging fault as shown in Figure 2.9 (right picture). In this case, the result of the intra-cell diagnosis avoids wasting time in investigating for a defect inside the suspect gate.

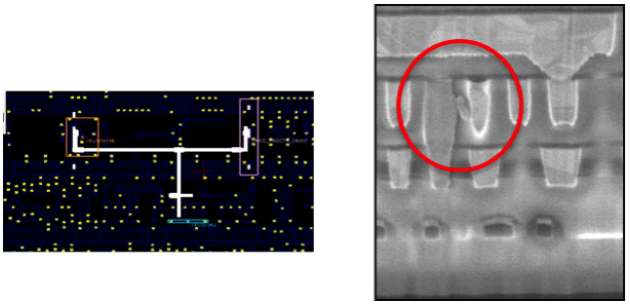


FIGURE 2.9: Layout view of suspects (left) and actual defect (right).

The second case of failure concerning the circuit C has been exploited to compare our approach w.r.t. the defect- and fault- dictionary based approaches. For this experiment we built a dictionary only for the cell identified by the inter-cell diagnosis (i.e., because the dictionary was not created during the design of the circuit). To create the dictionaries we applied the serial simulation algorithm (i.e., we injected one defect/fault at a time) that has a complexity corresponding to $O(n^2)$ per pattern, where n is the number of defects/faults. The complexity is dominated by the bridging defect/faults for which we have to consider all the possible combination of two nets. On the other hand, the proposed approach requires two simulations per pattern (the first is the fault free simulation and the second one corresponds to the CPT application). Thus, our approach has a complexity of $O(1)$. All the approaches report one candidate that was actually a short between two nets as reported in Figure 2.10. This experiment is important because it shows that our approach can be precise as the one based on defect- and fault-dictionary. Moreover, for this case we do not have the pre-computed dictionary because at the time were the circuit was designed no dictionary were computed. Thus, it was more time consuming to build the dictionary even for only one cell than simply apply our approach.

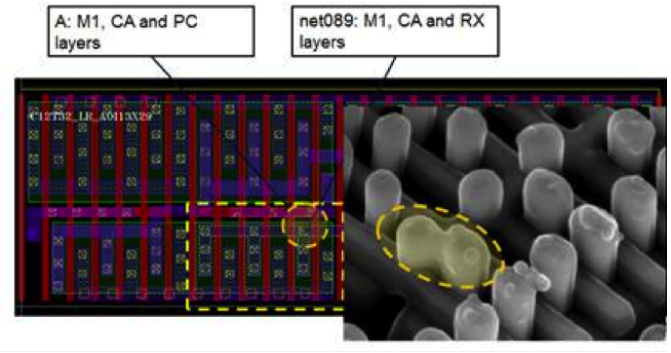


FIGURE 2.10: Layout view of suspects (left) and actual defect (right).

2.1.4.6 Conclusions

In this work, we have presented an intra-cell diagnosis approach. The proposed approach is based on the Critical Path Tracing here applied at transistor level. The CPT exploits the knowledge of the faulty behavior induced by a defect so that the proposed intra-cell diagnosis is not depended on a given defect. The diagnosis approach takes into consideration 3 fault models such as stuck-at, bridging and delay to represent the faulty behaviors induced by the physical defects (i.e., short and open). The proposed intra-cell diagnosis approach has been validated by means of simulation and on actual silicon data. It leads to a precise localization of the root cause of observed errors. Experimental results on actual cases show that the intra-cell diagnosis generally gives meaningful information further exploited by PFA engineers. This information allows saving times in searching the root cause of a faulty device.

This work and its extension are published in [W3, C10, C11, C12, C13, C14, C17, C18, C19, C21, C25, C35, C42, J5, J10] referenced in the Publication list of chapter 4.

2.1.5 Power-Aware Test

Nowadays, electronic products present various issues that become more important with CMOS technology scaling. High operation speed and high frequency are mandatory requests. On the other hand, power consumption is one of the most significant constraints due to large diffusion of portable devices. These needs influence not only the design of devices, but also the choice of appropriate test schemes that have to deal with production yield, test quality and test cost.

Testing for performance, required to catch timing or delay faults, is therefore mandatory, and it is often implemented through at-speed scan testing for logic circuits. At-speed scan testing consists of using a rated (nominal) system clock period between launch

and capture for each delay test pattern, while a longer clock period is normally used for scan shifting (load and unload cycles). In order to test for transition delay faults, two different schemes are used in practice during at-speed scan testing: Launch-off-Shift (LOS) and Launch-off-Capture (LOC) [18].

Although at-speed scan testing is mandatory for high-quality delay fault testing, its applicability is severely challenged by test-induced yield loss, which may occur when a good chip is declared as faulty during at-speed scan testing [15]. Both schemes (LOS and LOC) may suffer from this problem, whose the major cause is Power Supply Noise (PSN), i.e., IR-drop and Ldi/dt events, caused by excessive switching activity (leading to excessive power consumption) during the launch-to-capture cycle [16] of delay testing schemes. In order to deal with this problem, dedicated techniques to reduce the risk of artificial yield loss induced by excessive PSN during at-speed scan testing have been proposed in the literature [18]. These techniques are mainly based on test pattern modification or power-aware Design-for-Testability (DfT).

Despite the fact that reduction of test power is mandatory to minimize the risk of yield loss, some experimental results have proved that too much test power reduction might lead to test escape and reliability problems because of the under-stress of the circuit during test [18]. So, in order to avoid any yield loss and test escape due to power issues during test, test power has to map the power consumed during functional mode. To this purpose, the knowledge of functional power for a given CUT is required and may be used as a reference for defining the power consumption (upper and lower) limits during power-aware delay test pattern generation for LOS or LOC.

A solution has been proposed in [21] and [22], where authors introduce a new process to generate test vectors that mimic functional operation from the switching activity point of view. The process consists of shifting-in a test vector (at low speed) and then applying several successive at-speed clock cycles before capturing the test response. These tests are based on a LOC scheme with the insertion of multiple functional cycles between launch and capture operations. The idea behind this process is that after a certain number of clock cycles, the CUT reaches a pseudo-functional state (i.e., a state similar to a functional state) which insures that test power will mimic functional power, thus preventing any possible yield loss or test escape due to power issues.

In this work, our goal is to evaluate the accuracy of the above-mentioned solution presented in [21] and [22]. We propose a methodology in which several power estimation flows are built to compare the power consumption of pseudo-functional patterns (used in [21] and [22]) and LOS patterns (used in a conventional LOS scheme with a single launch-to-capture cycle) with the power consumption of actual functional patterns. For

this purpose, we have used the framework presented in [23] that deals with stimuli generation for design validation. In this framework, functional patterns are generated to maximize the switching activity of a given design, which can be used to determine the test power limits during at-speed delay testing. The proposed methodology has been validated on the Intel MC8051 microcontroller synthesized in a 65nm industrial technology. Experimental results show that the peak power consumption of pseudo-functional patterns obtained as in [21] and [22] is about 9% lower than the maximum functional peak power consumption of the MC8051 microcontroller. The output of this comparative study is a new flow to determine the functional power to be used as test power limits during at-speed scan delay fault testing.

2.1.5.1 The Proposed Methodology

The methodology proposed in this paper mainly relies on the framework presented in [23]. The goal is still to generate functional stimuli (i.e., test programs) for design validation purpose that maximize power consumption of the design. In the remainder of the paper, we refer to such stimuli as functional patterns.

These stimuli are here used for defining test power limits during the application of at-speed delay test patterns. Then, power limits are first compared with the power consumed during the application of the LOS test patterns. We focus on LOS since it is the test scheme providing a higher transition fault coverage but also a higher power consumption compared to LOC [18]. In the remainder of the paper, we refer to such stimuli as LOS patterns.

Next, we compare power consumption limits obtained with functional patterns with those obtained from patterns generated as described in [21] and [22]. In the remainder of the paper, we refer to such stimuli as pseudo-functional patterns. In order to generate pseudo-functional patterns, we use a LOS-based ATPG program in which many clock cycles are considered between launch and capture.

In the next subsections we detail how the functional stimuli are generated and what are the metrics used to perform the power analysis.

Test program generation

In the literature, main research works focus on the generation of methodologies oriented to minimize power consumption of both hardware and software. Conversely, as described in [23] and [32] the authors consider the mirror problem, that is, how to generate test programs oriented to maximize power consumption of the considered processor. In [23] the authors propose a methodology that deals with stimuli generation for design

validation. Specifically, the main objective of the test program is not just verifying the circuit functionality, but rather, the robustness of the whole system by applying test programs that maximize power consumption. More intuitively, increasing the power consumption can be seen as a kind of self-burn-in process, where the functionalities of the circuit are tested on extreme power conditions. In fact, exploiting similar approaches, in [32] authors propose a methodology for the automatic generation of stress programs to be used during the reliability characterization process of microprocessors.

In this work, we exploit the methodology proposed in [32], in order to generate power-aware test programs. The proposed approach exploits the logic simulation of a RTL description of the processor core, integrated in the generation loop. Supported by an evolutionary optimization tool, introduced in [33], syntactically correct assembly programs are generated, forming a population of programs; then, a high-level logic simulator evaluates every test program providing to the evolutionary optimizer a feedback value representing the test program goodness. This value, also known as fitness value, is computed by measuring the switching activity of the RTL signals that correspond to flip-flops in the synthesized version of the processor core. The evolutionary optimizer improves test programs by mimicking the Darwinian concepts of evolution. The higher the fitness value, the better the test program is. The generation process iterates up to reaching a steady condition, and then, the processor RTL description is substituted with the gate-level one, starting a new generation phase that improves the initial results.

Evaluation Framework

Power consumption has been evaluated during the application of functional patterns, LOS patterns and pseudo-functional patterns [21] and [22].

The power analysis has been done considering two power metrics: cycle average power and peak power consumption. Both cycle average and peak power consumption are measured by means of WSA (Weighted Switching Activity).

In general, cycle average refers to the average of instantaneous power consumed by the device during a specified time window. Peak power refers to the highest power value measured during the same time window. The definition of the time window varies according to the stimuli applied to the device. In the following, we define for each type of applied stimuli the time window and its meaning.

Functional patterns

The applied functional patterns correspond to a test program executed by the microprocessor as described above. Before the test program execution, the microprocessor has to

execute a small piece of Operating System (OS) to ensure the correct initialization of the microprocessor itself.

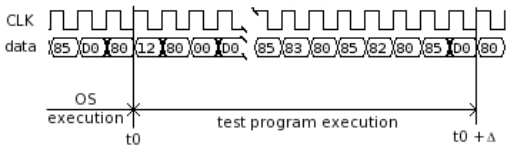


FIGURE 2.11: Functional patterns time window.

Figure 2.11 depicts a snapshot of a test program execution waveform. For the sake of readability only the clock signal and the I/O data are reported. After the OS execution, the test program execution starts at time t_0 and ends at time $t_0 + \Delta$. For each test program we consider the time window $(t_0, t_0 + \Delta)$ as the period to estimate the cycle average and the peak power.

LOS patterns

Power consumption can be evaluated during the launch and capture cycles of the LOS testing schemes. Launch power is the power consumed during the launch cycle (also called "launch-to-capture" or "test" cycle), which is performed with an at-speed clock cycle. Capture power is the power consumed right after the capture edge, during a time interval also equal to the rated clock period of each experimented circuit. Figure 2.12 shows the time windows in which these power measures are made.

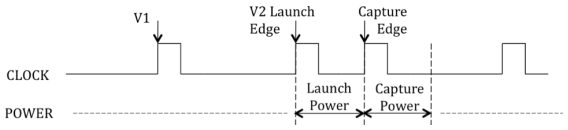


FIGURE 2.12: LOS patterns time window.

In [18], it has been proved that launch power is greater than capture power. For this reason, in this paper we only consider launch cycle as the time window to estimate the cycle average and the peak power. Both values are evaluated for the whole set of LOS test patterns. Then, the most power-consuming pattern concerning cycle average and peak power is considered.

Pseudo-functional patterns

Pseudo-functional patterns are based on a LOS test scheme modified by adding n capture cycles. Since in [21] and [22] the number of capture cycles necessary to reach a pseudo-functional state is not specified, we generate several pseudo-functional patterns having $n = \{5, 10, 50, 100\}$.

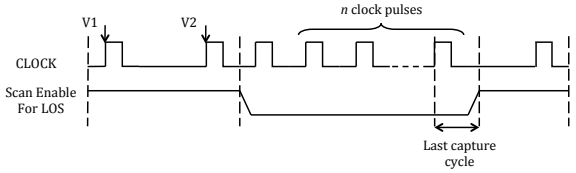


FIGURE 2.13: Pseudo-functional patterns time window.

Figure 2.13 shows the *"n capture method"*. This method considers the power consumed in the last capture cycle as a good approximation of the real functional power consumption. Thus, the time window used to measure both the cycle average and the peak power corresponds to the last capture cycle. The calculation of the average and peak power values is repeated for the whole set of test vectors generated by the ATPG tool. Since we are interested only in the critical values of the power, we will take into account only the most power-consuming pseudo-functional patterns, in terms of cycle average and peak power.

2.1.5.2 Case Study

The adopted case study is the Intel microcontroller MCS-51 (MC8051) synthesized under a 65nm industrial technology library. MC8051 represents a classical Harvard architecture non-pipelined CISC architecture, with 8-bit ALU and 8-bit registers. Table 2.8 gives the details of the microcontroller synthesis in terms of number of primary inputs/outputs, flip-flops, logical gates and number of transition faults. During the synthesis one scan chain has been added.

#Primary Inputs	65
#Primary Outputs	94
#Scan Chains	1
#FFs	578
#Logical Gates	9,451
#Transition Faults	37,752

TABLE 2.8: MC8051 Synthesis.

The MC8051 is embedded into a SoC [34] composed of three cores: (i) the MC8051 microcontroller, (ii) a 64Kx8 bit SRAM memory and (iii) a 16x16 parallel multiplier.

During its mission mode, the microcontroller core reads the program to be executed from an external RAM memory, it communicates with the outside through its parallel port and it drives the multiplier for arithmetic computations. The SoC structure is shown in Figure 2.14.

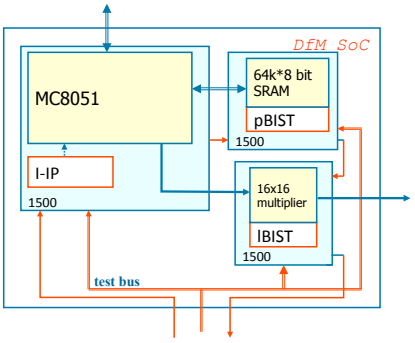


FIGURE 2.14: SoC Architecture.

In order to estimate power consumption (in terms of both cycle average and peak power) for Functional, LOS and Pseudo-functional patterns three different flows have been implemented.

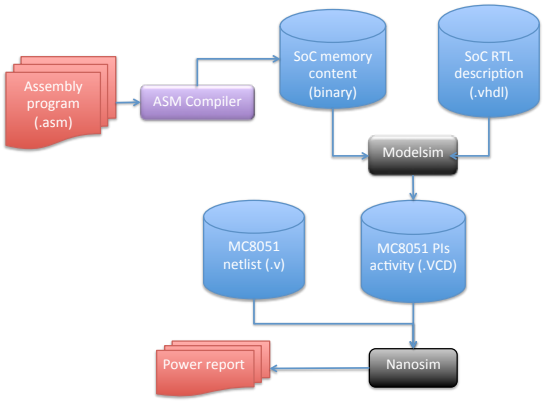


FIGURE 2.15: Functional patterns power estimation flow.

Figure 2.15 shows the implemented flow to estimate power consumption for the functional patterns. Each functional pattern corresponds to a generated test program. The test program is compiled in order to obtain the memory content to be loaded into the SoC. The SoC with the test program loaded in memory is simulated using ModelSim®. The switching information related to the MC8051 primary inputs is stored in a VCD file. So, in the resulting VCD file we have the stimuli applied to the microcontroller when the target test program is executed. The last step of the flow actually estimates the power consumed during the test program execution. Since our goal is to evaluate the power consumed by the microcontroller, we simulate only the MC8051 microcontroller gate level netlist by applying the stimuli stored in the VCD. The tool used to perform such analysis is NanoSim®. The result is the power report containing both cycle average and peak power evaluated during the time window $(t_0, t_0 + \Delta)$.

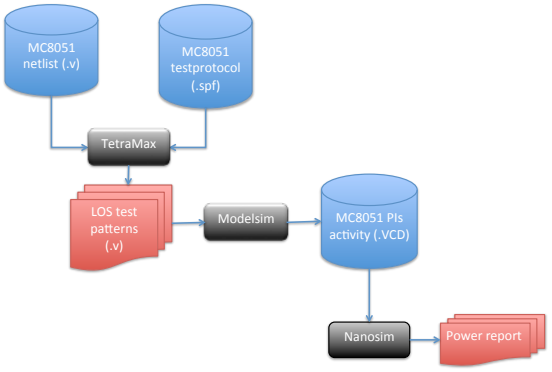


FIGURE 2.16: LOS patterns power estimation flow.

Figure 2.16 shows the implemented flow to estimate power consumption for the LOS patterns. The LOS pattern set is generated using TetraMAX®. The required inputs are the MC8051 gate level netlist and the test protocol file (.spf) in which the timing to be applied during the LOS test scheme is described. Note that we used the default setting and random filling options for the ATPG. Static and dynamic compactions were used during test set generation. The number of generated LOS patterns is 1405 and the achieved fault coverage is 79%. The generated LOS test patterns set is then simulated using ModelSim®. The switching information for the patterns is stored in a VCD file. Finally, the VCD file is used to perform the power analyses with NanoSim®. The result is the power report containing both cycle average and peak power evaluated during the time window (the launch-to-capture cycle).

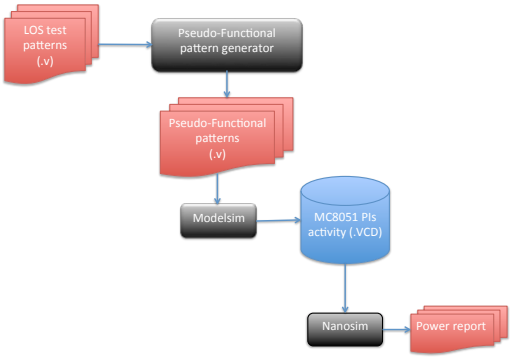


FIGURE 2.17: Pseudo-functional patterns power estimation flow.

Figure 2.17 shows the implemented flow to estimate power consumption for the Pseudo-functional patterns. Pseudo-functional patterns are based on a LOS test scheme modified by adding n captures cycles, where $n = \{5, 10, 50, 100\}$. Each pseudo-functional pattern is then simulated using ModelSim®. The switching information for the patterns is stored in a VCD file. Finally, the VCD file is used to perform the power analyses with

NanoSim®. The result is the power report containing both cycle average and peak power evaluated during the time window.

Experimental results

This section decscribes the experimental results. All the experiments were carried our on an Intel Xeon@3.16GHz with 8GB of RAM.

Experiments have been done on a MC8051 description synthesized with an industrial 65nm technology, considering a power supply voltage of 1.2V. Only dynamic switching power due to switching capacitances has been considered (short-circuit power and leakage power consumptions were neglected). This is a valid assumption for the considered technology with the purpose of comparing between Functional, LOS and Pseudo-functional patterns. Similarly, although in deeper technologies the impact of leakage power on the overall power consumption is greater than in the 65nm technology, we may assert that it will not affect the main conclusions of our comparison between Functional, LOS and Pseudo-functional patterns. This forecast is based on the fact that we expect that a higher leakage power will possibly impact the three sets of patterns in the same manner (as the circuit and numbers of scan FFs are the same, a similar level of leakage currents can be predicted).

The power consumption measured during the time window $(t_0, t_0 + \Delta)$ for the functional patterns is reported in Table 2.9. Results are expressed in milliWatt. In our experiments we generated 30 different functional patterns (i.e., test programs). The second column ("Clock cycles") reports the length of each test program in terms of the number of clock cycles that corresponds to the size of the time window. The value reported for each functional pattern in the "Peak" column represents the highest peak power measured over the considered time window $(t_0, t_0 + \Delta)$. The value reported for each functional pattern in the "Average" column represents the Average power measured over the considered time window $(t_0, t_0 + \Delta)$. Note that all measurements represent the overall circuit power consumption, including both logic power (power in the combinational logic) and sequential power (power in scan flip-flops). Clock power (power in the clock tree) is not included.

Functional patterns considered in Table 2.9 have been developed recurring to two different approaches. Both strategies guarantee a comprehensive excitation of the processor core, though they exploit different generation schemes. The first 25 test programs, labeled *test program 01, test program 02, etc.*, were developed by hand, targeting the stuck-at fault coverage of the processor core. These programs achieved about 95% fault coverage on the targeted fault model. Interestingly, the highest peak value was obtained by *test program 25*, which repeatedly sweeps the addressable memory for exciting the

Functional Patterns	Clock cycles	Peak (mW)	Average (mW)
test program 01	4349	106.25	3.56
test program 02	4721	106.86	3.88
test program 03	5697	106.58	4.11
test program 04	6433	106.24	4.50
test program 05	6561	106.21	4.58
test program 06	8613	107.80	5.44
test program 07	12185	107.51	1.30
test program 08	14732	107.09	1.34
test program 09	162567	106.97	7.01
test program 10	19973	106.03	7.20
test program 11	20415	108.49	5.14
test program 12	10851	107.04	4.79
test program 13	23851	111.29	7.27
test program 14	27056	107.46	2.23
test program 15	26980	107.50	2.22
test program 16	42463	106.24	1.08
test program 17	42943	107.64	6.53
test program 18	52113	105.84	8.45
test program 19	53835	107.35	8.02
test program 20	71947	109.02	7.69
test program 21	79892	107.51	2.37
test program 22	94349	106.11	3.39
test program 23	277617	110.27	8.34
test program 24	1048261	106.14	12.61
test program 25	1050438	119.10	9.41
program stress alu	553	107.96	1.30
program stress fsm	421	106.73	0.79
program stress mem	609	108.48	1.68
program stress uP	391	106.79	1.02
program stress ram	407	107.78	1.00

TABLE 2.9: Functional Patterns Power Estimation.

program counter circuitry. On the other hand, patterns labeled *programs stress alu, fsm, etc.*, were automatically generated. In this case, functional programs stress mainly the single part of the processor core highlighted in the program name.

Patterns	Peak (mW)	Average (mW)
LOS	146.93	14.09
LOS 5	108.78	12.15
LOS 10	108.89	12.19
LOS 50	108.62	11.97
LOS 100	108.73	11.63

TABLE 2.10: LOS And Pseudo-Functional Patterns Power Estimation.

Table 2.10 reports the power consumption for LOS and Pseudo-functional patterns. In

the first column ("Patterns") we report LOS patterns and the set of pseudo-functional patterns denoted by the number of capture cycles (5, 10, 50, 100). Results are expressed in milliWatt. The value reported for each functional pattern in the "Peak" column represents the highest peak power measured during the launch-to-capture cycle for LOS patterns. The peak power of the pseudo-functional patterns is measured during the last capture cycle. The value reported for each pattern in the "Peak" column represents the highest peak power measured over the complete set of patterns. The value reported for each pattern in the "Average" column represents the Average power measured during the launch-to-capture cycle for LOS patterns. For the pseudo-functional patterns the average is measured during the last capture cycle. As for peak values, the average power is measured over the complete set of patterns. Note that similarly to the experiments carried out on functional patterns, all measurements represent the overall circuit power consumption, including both logic power (power in the combinational logic) and sequential power (power in scan flip-flops). Clock power (power in the clock tree) is not included.

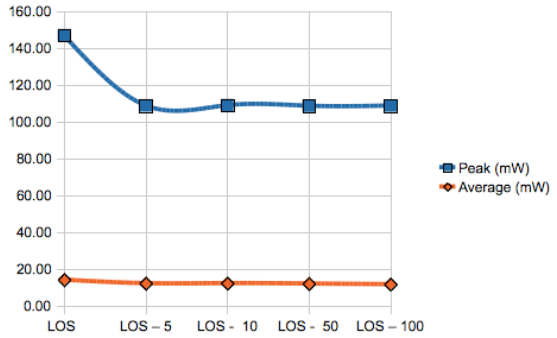


FIGURE 2.18: Behavior of LOS and pseudo-functional patterns.

The results in Table 2.10 are further elaborated in the graph of Figure 2.18 where both Peak power and cycle Average power are reported. As predicted in [21] and [22] both peak power and average power reach a steady state value after a certain number of capture cycles. In our experiments, the peak power converges at a stable value after 10 capture cycles, while the average power converges after 5 capture cycles. The peak power convergence value is lower than that during the launch-to-capture cycle for LOS patterns. The reduction of peak power achieved by applying more than 10 capture cycles is about 26% compared to the peak power during the launch-to-capture cycle for LOS patterns. The same conclusions can be drawn for the average power. In this case, the reduction is about 17% compared to the cycle average power during the launch-to-capture cycle for LOS patterns. The trend depicted in Figure 2.18 confirms the data given in [21] and [22].

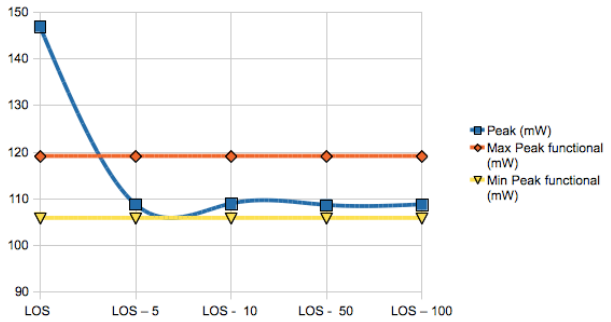


FIGURE 2.19: Behavior of Functional and Pseudo-functional patterns.

Figure 2.19 shows a comparison between the functional peak power and the peak power obtained with LOS and Pseudo-functional patterns. In the figure is reported the two bounds (upper and lower) representing respectively the maximum and the minimum functional peak power. From the analysis of data reported in Figure 2.19 we can make the following observations:

- As reported in the literature [18], the power consumed during test application is higher than the power consumed during functional mode. The gap between LOS peak power (the peak power consumed by LOS patterns during the launch-to-capture cycle) and the maximum functional peak power is about 23.37%. The gap between LOS peak power and the minimum functional peak power is 38.82%.
- The comparison between the functional peak power and the pseudo-functional peak power shows that the latter is not a very accurate prediction of the actual functional peak power (at least in our case study). The pseudo-functional peak power is about 9% lower than the maximum functional peak power. This is also confirmed by the fact that it is really close to the minimum functional peak power (only 2.8% lower). Thus, only considering pseudo-functional peak power for defining test power limits may lead to test escapes due to under stress.

2.1.5.3 Conclusions

In this work we proposed a novel flow to estimate functional power to be used as power limits during at-speed delay fault test. A first (predictable) conclusion shown by this study is that power consumed during test is higher than power consumed during functional mode. A second (unpredicted) conclusion of this study is that the peak power consumption of pseudo-functional patterns is not so accurate compared to the real functional peak power. Thus, as a final conclusion we have shown that a procedure to generate meaningful functional patterns to estimate the functional power is mandatory to avoid test escape and yield loss.

This work and its extension are published in [W4, C23, C24, C27, C29, C30, C31, C34, C38, C39, C46, I2, I4, J6, J8, J9] referenced in the Publication list of chapter 4.

2.1.6 Test of Low Power Devices

With the growing demand of high performance, multi-functional and hand-held devices, power consumption has emerged as a major design concern. Simultaneously, technology scaling is shrinking device features as well as lowering transistor threshold voltage, which is associated with an exponential increase in subthreshold leakage current. Therefore, power consumption due to leakage currents (i.e. static power) has become a major contributor to the total power consumption in CMOS circuits [35].

Despite increasing static power consumption, aggressive device scaling in each technology generation allows high integration density. Nowadays, system-on-chips (SOCs) designed using deep-submicrometer technologies can integrate all components and functions that historically were placed on a printed-circuit board. Among all different IP cores included in a single SOC, embedded memories are the densest components, accounting for more than 90% of the overall chip area [24]. Due to such high density, memory devices are arising as the main contributor to the overall SOC static power consumption, which requires the development of appropriate mechanisms to reduce static power consumption in memories.

Since SRAMs are still dominant in most SOCs, techniques to reduce static power consumption of such devices have been proposed in all stages of design process. At transistor level, dynamic control of the transistor gate-source and substrate-source back bias has been exploited to reduce leakage during standby periods [36]. At architectural level, a mechanism based on so-called canary cells has been proposed to dynamically reduce the voltage supplied to the core-cell array [37]. Another approach, widely used in industry, is based on embedded power gating mechanisms and voltage regulation systems, which allow significant static power savings by lowering the voltage supplied to SRAM memory blocks (e.g. the core-cell array) that are not in use [38]. Even though such devices offer substantial benefits in reducing static power of SRAMs, their adoption in practice depends on the availability of test methods, such that their correct operation can be ensured in the field. This is because the dense structure of SRAMs designed with deep-submicrometer technologies prompts all memory blocks to be extremely vulnerable to physical defects, which may lead to faulty behaviors during functional operation [25].

Due to the complex nature of SRAM internal behavior, identification of faulty behaviors and development of efficient test solutions are non-trivial tasks. Hence, in order to minimize test development effort and produce more efficient and dedicated memory

tests, we can rely on information provided by electrical simulations. In this context, the analysis of the memory layout allows determining realistic defects (e.g. resistive-opens and resistive-bridges) within memory blocks, and electrical simulations are then performed to determine those defects that may lead to faulty behaviors. Finally, test solutions are generated based on results of such failure analysis. Several studies have been performed, according to this approach, to understand failure mechanisms and to develop efficient test solutions for SRAMs, notably [25, 39]. However, only few works, notably [26], target test of devices that are specific to low-power SRAMs, such as power gating mechanisms. Existing test techniques for power gating mechanisms are based on dedicated design-for-test (DFT) solutions [26]. Nevertheless the efficiency of such solutions, none of them is viable to characterize, from the functional point of view, the faulty behaviors of power gating mechanisms. Thus, existing solutions cannot be used to mitigate the presence of defects. Moreover, if the DFT cannot be applied due to specific design constraints, another test strategy must be developed.

To overcome such limitations of state-of-the art SRAM test methods, in this paper we focus on functional test solutions targeting power gating mechanisms. In this work, we present extensive electrical simulation results, using a commercial low-power SRAM as case-study, to show the impact of realistic defects that may affect the correct functioning of power gating mechanisms. Finally, we present an efficient test solution targeting sensitization and detection of faulty behaviors identified in previous failure analysis.

2.1.6.1 Low-Power Sram: Architecture And Functioning

Figure 2.20 depicts a conceptual view of the low-power SRAM used as case study in this work. It is a commercial single-port word-oriented SRAM, designed by Intel with a 40nm low-power process technology. Such SRAM has a core-cell array divided in N *core-cell blocks* (CBs), where each CB is connected to an *I/O block* (IOB), as shown in Figure 2.20.

Each pair $CB_i/IOB_i, i \in [0, N-1]$, is referred to as a memory element. As illustrated in Figure 2.21, a CB is composed of core-cells connected to a subset of m *bit lines* (BLs) and n *word lines* (WLs), whereas an IOB consists of a multiplexer of BL pairs and an I/O circuitry (a write driver and a sense amplifier). The multiplexer controls the access of such m BLs to the I/O circuitry. Before every operation, pre-charge circuits are turned on to maintain all BL pairs, write driver input signals (WD and WDB) and sense amplifier input signals (SA and SAB) at V_DD level. In this case, control signals $\overline{BLeq_{WD}}$, $\overline{BLeq_{SA}}$ and all signals $\overline{BLeq_y}, y \in [0, m-1]$, are set to logic '0'. When an operation is being performed, pre-charge circuits are turned off and a single BL pair

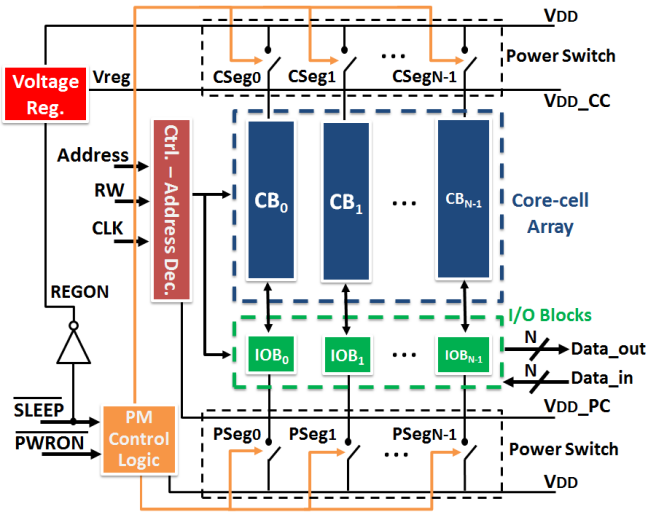


FIGURE 2.20: Low-power SRAM architecture.

accesses the I/O circuitry. If a BL pair BL_y/BLB_y is accessing the I/O circuitry, then SEL_y is set to logic '1' and SEL_z is set to logic '0', $\forall y \neq z, y, z \in [0, m-1]$. In this case, BL_y is connected to nodes *WD* and *SA* and BLB_y is connected to nodes *WDB* and *SAB*. During a write operation, signal *Write_EN* activates the write driver, whereas signal *SAON* activates the sense amplifier during a read operation.

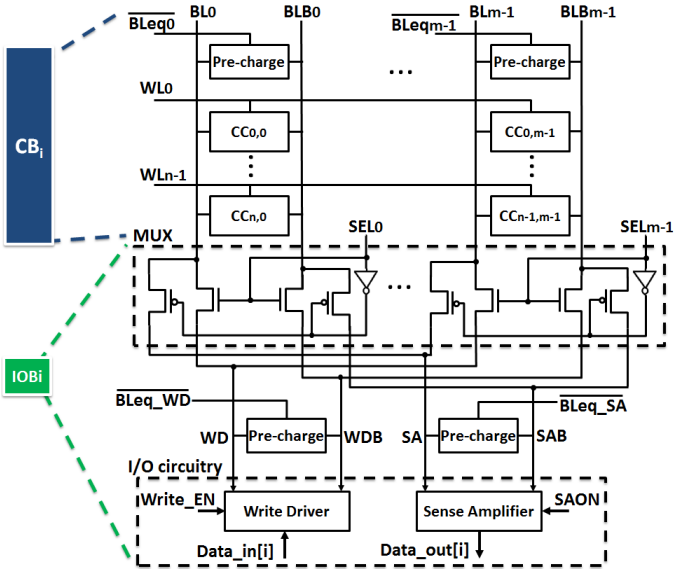


FIGURE 2.21: SRAM memory element.

Each memory element provides access to one core-cell of the array when executing read and write operations. As the result of a read, the data stored in the accessed core-cell is returned in memory element output *Data_out[i]*, while the data to be written in the accessed core-cell, during a write, is specified through input *Data_in[i]*. The SRAM input *Data_in* and output *Data_out* group signals *Data_in[i]* and *Data_out[i]*, respectively, of

all memory elements (i.e. *Data_in* and *Data_out* have N bits each). Hence, the number N of memory elements corresponds to the SRAM word width.

As shown in Figure 2.20, this low-power SRAM also embeds power gating mechanisms, which are implemented using power switch blocks, to control the voltage supplied to the core-cell array and the peripheral circuitry (IOBs, control block and address decoder). In next subsection, we provide a detailed description of the power gating architecture.

Power Gating Architecture

The power switches (PSs) connected to the core-cell array and the peripheral circuitry are structured in N segments [26], as shown in Figure 2.20, with N equals to the number of memory elements of the SRAM. The power gating architecture described in this work is a classical one, thus widely adopted in industry for both SRAMs and cores. This is because its segmented structure allows redundancy and ensures that all structures are always powered uniformly as described in [26]. Figure 2.22 illustrates the structure of each segment $CSeg_i$ and $PSeg_i, i \in [0, N-1]$, of the PS connected to the core-cell array and the peripheral circuitry, respectively.

Each segment $CSeg_i$ is composed of four PMOS transistors in parallel, as shown in Figure 2.22.a. Such transistors, indexed from *P_CC0* to *P_CC3*, have a dedicated control signal connected to the gate terminal (signals *Ctrl_CC0* to *Ctrl_CC3*). This allows transistors in segments $CSeg_i$ to be controlled individually. Segments $PSeg_i$ are composed of two PMOS transistors in parallel, *P_PC0* and *P_PC1*, as shown in Figure 2.22.b. Such transistors also have dedicated control signals connected to the gate terminal (signals *Ctrl_PC0* and *Ctrl_PC1*), which allows transistors in segments $PSeg_i$ to be controlled individually as well.

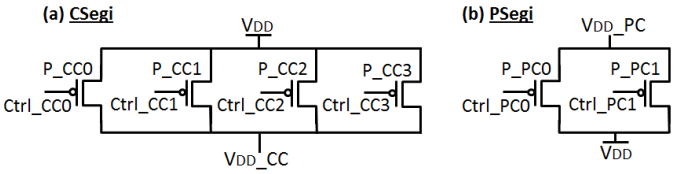


FIGURE 2.22: Segment of the power switch connected to (a) the core-cell array and (b) the peripheral circuitry.

Such power gating mechanisms, along with a voltage regulation system (highlighted in red in Figure 2.20), enable power modes on the SRAM by varying the voltage applied to the core-cell array and peripheral circuitry internal supply lines (V_{DD}_CC and V_{DD}_PC in Figure 2.20, respectively).

Power Modes

For the low-power SRAM studied in this work, three power modes can be distinguished: 1) *active*, 2) *deep-sleep*, and 3) *power-off*.

In *active* (ACT) mode, PSs connected to both core-cell array and peripheral circuitry are activated (i.e. the gate of all PMOS transistors in all segments is at 0V). This allows supply lines V_{DD}_CC and V_{DD}_PC to be driven by the main supply rail, which provides V_{DD} voltage level. Hence, both core-cell array and peripheral circuitry are powered at V_{DD}, which allows operations to be performed. Note that the voltage regulator is switched off (i.e. signal *REGON* is set to logic '0') in ACT mode.

In both *deep-sleep* (DS) and *power-off* (PO) modes, PSs connected to both core-cell array and peripheral circuitry are deactivated (i.e. the gate of all PMOS transistors in all segments is at V_{DD} level). Hence, lines V_{DD}_CC and V_{DD}_PC are no longer connected to the main supply rail, and, consequently, no operation is allowed to be performed.

In DS mode, the voltage regulator is switched on (i.e. signal *REGON* is set to logic '1') to generate a fixed voltage level (*Vreg* in Figure 2.20), lower than V_{DD}, to be provided to the core-cell array through V_{DD}_CC, whereas V_{DD}_PC discharges to 0V. The voltage *Vreg*, which must be high enough to guarantee data retention in all core-cells of the array, drastically reduces static power consumption due to current leakage.

In PO mode, the embedded voltage regulator is switched off, hence both supply lines V_{DD}_CC and V_{DD}_PC discharge to 0V. Core-cells are no longer able to retain any data in PO mode.

All control signals driving PSs, which are required to set the SRAM into different power modes [9-10], are generated by the *power mode control* (PM control) logic. It is important to note that the PM control logic is always supplied at V_{DD} level (refer to Figure 2.20), such that the SRAM is able to switch among power modes in any time. Such circuit is described in detail in the next subsection.

Figure 2.23 illustrates the structure of the PM control logic. It is important to note that, in Figure 2.23, resistances *Df1* to *Df6* (highlighted in red) are resistive-open defects that do not occur in a defect-free PM control logic. Such defects will be studied in the following sections.

The PM control logic generates control signals driving PSs according to inputs $\overline{SLEEP}$ and $\overline{PWRON}$. Such inputs allow configuring the SRAM power mode as follows:

1. ACT mode: $\overline{SLEEP}$ and $\overline{PWRON}$ must be set to logic '1' and logic '0', respectively.

2. DS mode: $\overline{SLEEP}$ must be set to logic '0', regardless of $\overline{PWRON}$.

3. PO mode: both inputs must be set to logic '1'.

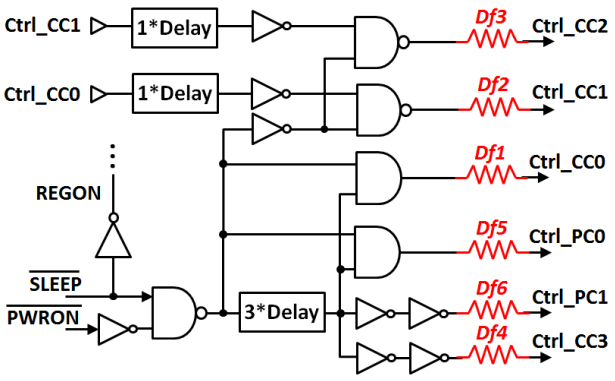


FIGURE 2.23: Power mode control logic.

The most important role of the PM control logic is to activate transistors into PS segments in the correct order when the SRAM is switched from DS or PO mode to ACT mode, which is referred to as *wake up* (WU) phase. A specific activation order exists because transistors that compose PS segments have different drive strength *DRS*, such that DRS_{P_CC0}<DRS_{P_CC1}<DRS_{P_CC2}<DRS_{P_CC3} and DRS_{P_PC0}<DRS_{P_PC1}. During WU phase, transistors into PS segments must be activated in a cascade fashion w.r.t the increasing order of drive strength, which allows the voltage at lines V_{DD}_CC and V_{DD}_PC to rise slowly towards V_DD. Consequently, when high drive strength transistors *P_CC3* and *P_PC1* are activated, the voltage level at lines V_{DD}_CC and V_{DD}_PC is higher than 90% of V_DD, which avoids large peak currents that are generated when V_DD line is connected directly to a line at low voltage through high drive strength transistors. Such cascade WU technique is commonly used in commercial VLSI circuits (e.g. an SRAM) to reduce the overall power noise on the system, during WU phase.

The PM control logic ensures the expected activation order by controlling the fall transition of signals connected to the gate of transistors into PS segments during WU phase, which is described as follows. The WU phase is started by the WU command, i.e. by configuring the SRAM in ACT mode. Consequently, signals *Ctrl_CC0* and *Ctrl_PC0* are immediately set to logic '0', letting PMOS transistors controlled by such signals to be turned on. Next, signals *Ctrl_CC1* and *Ctrl_CC2* are set to logic '0' after one and two delay units from the beginning of WU phase, respectively. In Figure 2.23, we can verify that the fall transition of signal *Ctrl_CC1* follows the fall transition of *Ctrl_CC0*, while the fall transition of signal *Ctrl_CC2* follows the fall transition of *Ctrl_CC1*. Finally, the WU phase terminates when signals *Ctrl_CC3* and *Ctrl_PC1* are pulled down, which occurs after three delay units from the WU command.

The duration of the WU phase, i.e. the time interval between the WU command and the instant in time when all transistors into all PS segments are activated, is referred to as *WU time*. Note that read and write operations are not allowed to be performed during this time interval.

2.1.6.2 Resistive-Open Defects In The Power Mode Control Logic

In this section, we present an in-depth study, based on electrical simulations, to characterize the SRAM behavior in presence of resistive-open defects shown in Figure 2.23. As shown in Figure 2.23, defects *Df1* to *Df4* affect control signals driving PSs connected to the core-cell array, whereas *Df5* and *Df6* affect control signals driving PSs connected to the peripheral circuitry.

Note that we did not study defects affecting internal nodes of the PM control logic. This is because the presented PM control logic is an Intel specific circuit, hence, defects inside such circuit may lead to impacts that may not occur in other memories (e.g. an SRAM developed by another producer), with a different implementation of the PM control logic. On the other hand, defects affecting control signals of PSs have a similar impact on every SRAM that embeds power gating mechanisms. Thus, presented results are valid for any SRAM that embeds such facilities.

Experimental Setup

All electrical simulations have been performed using an extracted Spice model corresponding to the SRAM described in Section 2.1.6.1 reference memory block with 64 memory elements and 4K addresses (i.e. each address provides access to 64 core-cells) has been considered, organized as a core-cell array of 6-transistors core-cells composed of 512 BLs and 512 WLs.

Each simulation has been performed in presence of a single defect, since the occurrence of multiple defects has low probability in a small circuit such as the PM control logic, and consisted of the following main steps:

1. Transition from ACT to DS or PO mode (the SRAM is configured in ACT mode at the beginning of all simulations).
2. Transition to ACT mode (i.e. WU phase).
3. Execution of operation sequences (OSs), after WU time.

Such steps have been chosen to sensitize potential issues that may be caused by malfunctions of PSs. Step one allows verifying whether or not PSs are correctly deactivated

during a switch to DS or PO mode, whereas step two allows verifying the correct activation order of transistors into PS segments, during WU phase. Finally, step three allows verifying if the SRAM remains functional following WU phase.

In our experiments, the whole set of *process corner*, *voltage* and *temperature* (PVT) conditions (selected according to the SRAM specifications) have been examined when analyzing each defect. Hence, for each defect, electrical simulations have been performed by varying the following parameters:

- **Process corner**: slow, typical, fast, fast NMOS/slow PMOS and slow NMOS/fast PMOS.
- **Supply voltage** V_{DD}: 0.9V, 1.0V and 1.1V.
- **Temperature**: -30C, 25C and 125C.

Nominal values of clock cycle ($Tcyc_{nom}$) and the WU time (WUT_{nom}) used on the experiments have also been extracted from the SRAM specifications, as follows:

- $Tcyc_{nom}$: 4ns, 3ns and 2.5ns for supply voltage equals to 0.9V, 1.0V and 1.1V, respectively.
- WUT_{nom}: 185ns, 150ns and 125ns for supply voltage equals to 0.9V, 1.0V and 1.1V, respectively. The same values have been used for WU phase from both DS and PO modes.

Finally, it is important to mention that resistance values of defects have been chosen from a few Ωs up to several MΩ in order to provide a complete view of the studied phenomena.

Summary of Experimental Results

Experimental results have shown that all studied defects induce a delay on the fall transition of signals controlling PS segments, during WU phase. In presence of *Df1*, *Df2*, *Df3* and *Df5*, we observed that such delay may break the activation order of transistors into PS segments, which provokes large peak currents during WU phase. Simulation results have shown that the worst-case impact is caused by *Df5*. Such defect may induce a peak current that is up to 22 times higher than the expected maximum current. This worst-case situation occurs during WU phase from DS mode, with PVT = fast, 1.1V, -30C.

Despite the induced peak currents, experiments have shown that *Df1*, *Df2*, *Df3* and *Df5* do not impact the activation of high drive strength transistors *P_CC3* and *P_PC1*,

during WU phase. Hence, supply lines V_{DD}_CC and V_{DD}_PC remain fully charged and stable at V_DD when OSs are performed after WUT_{nom}. Consequently, no *faulty behavior* (FB) has been observed during the execution of such operations. A FB corresponds to an observed memory behavior that deviates from the expected one, when a given a set of operations is executed in the memory.

On the other hand, it has been observed that *Df4* and *Df6* induce a delay on the activation of high drive strength transistors *P_CC3* and *P_PC1*, respectively, during WU phase. Depending on the resistance value defects, such transistors may not be fully activated after WUT_{nom}, when the execution of OSs is started. Experimental results have shown that this effect provokes voltage droop phenomena in lines V_{DD}_CC and V_{DD}_PC during the execution of operations.

In particular, we verified that *Df4* causes a voltage droop in line V_{DD}_CC that is always less than 5% of V_DD, which can be considered as negligible. This is because *Df4* does not impact the activation of transistors *P_CC0*, *P_CC1* and *P_CC2* into PS segments connected to the core-cell array, thereby avoiding high voltage droop phenomena on line V_{DD}_CC . However, it has been observed that *Df6* induces high voltage droop in line V_{DD}_PC when OSs are executed after WU phase. Even though *Df6* does not impact the activation of transistors *P_PC0*, such transistors alone cannot maintain line V_{DD}_PC fully charged and stable at V_DD when operations are executed, thereby provoking the observed voltage droop phenomena. In the worst-case situation, voltage droop reaches 400mV, which occurs with PVT = fast, 1.1V, 125C.

The high voltage droop phenomena induced by *Df6* can lead to FB during the execution of such operations, especially due to performance loss issues caused by such degradation of the voltage supplied to the peripheral circuitry. In the next section, our aim is to investigate the execution of OSs after WU phase, in presence of *Df6*, to identify FBs that such defect may cause.

2.1.6.3 Failure Analysis In Presence Of Df6

To better understand the SRAM behavior in presence of *Df6*, we first describe how the SRAM control block generates control signals to perform read and write operations. Next, we present the impacts of *Df6* by means of two experimental scenarios.

Generation of Control Signals

The SRAM control block generates signals to perform operations based on a self-timing mechanism. According to this approach, operations start executing in rising edges of

the clock and the SRAM itself, based on the discharge of a self-timing signal, generates control signals to complete them.

Figure 2.24 depicts the generation of control signals based on such mechanism during a write and a read operation. In Figure 2.24, notation $\overline{BLeq\_SA*}$, $\overline{BLeq\_WD*}$, *Write_EN** and *SAON** designate signals $\overline{BLeq\_SA}$, $\overline{BLeq\_WD}$, *Write_EN* and *SAON*, respectively, of all memory elements, while $\overline{BLeq*}$ represents all control signals driving pre-charge circuits connected to BL pairs, of all memory elements (refer to Section 2.1.6.1 for details about such signals).

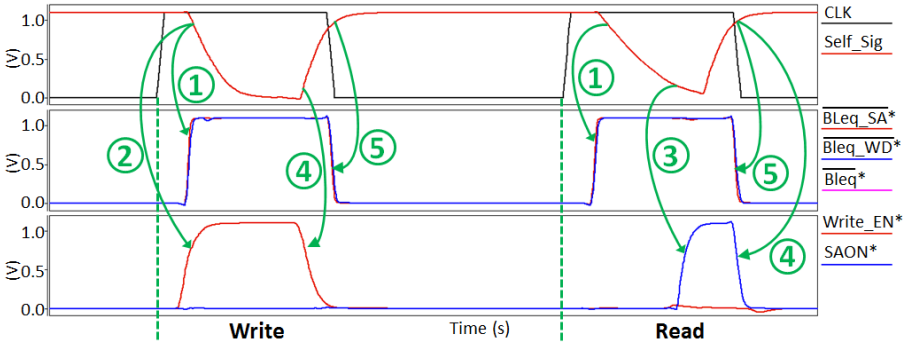


FIGURE 2.24: Generation of control signals based on a self-timing mechanism.

When the SRAM is in stand-by, the self-timing signal (*Self_Sig* in Figure 2.24) is at V_{DD} level. Once a rising edge of the clock occurs, signal *Self_Sig* starts discharging, which turns off all pre-charge circuits. Hence, all control signals driving pre-charge circuits are set to logic '1', as indicated by arrows "1" in Figure 2.24. In the next step (not shown in Figure 2.24), a BL pair is selected to access the I/O circuitry, in each memory element, and a WL is activated to access the addressed core-cells.

During a write operation, write drivers of all memory elements are enabled at the beginning of the fall transition of *Self_Sig* (refer to arrow "2") in order to set the selected BLs to the proper voltage levels. During a read operation, sense amplifiers of all memory elements are enabled when the weak voltage difference between the selected BLs is high enough to perform the read, which corresponds to the end of the fall transition of *Self_Sig* (refer to arrow "3"). At the end of each operation, signal *Self_Sig* is restored to V_{DD}. This turns off write drivers and sense amplifiers (refer to arrows "4") and activates all pre-charge circuits (refer to arrows "5"). The time required for a full cycle of the self-timing signal (discharge to 0V and restore to V_{DD}) reflects approximately the time needed to complete an operation.

We can also observe in Figure 2.24 that the discharge rate of signal *Self_Sig* depends on the operation type (rate is higher in the write operation). Once a rising edge of the clock occurs, a latch inside the SRAM control block holds the primary input signal that indicates the operation type (input *RW* in Figure 2.21) during the whole cycle of signal

Self_Sig. The held data is then used to determine the discharge rate of the self-timing signal.

Finally, it is important to mention that the SRAM is blocked during the discharge of the self-timing signal, when core-cells are being accessed to perform an operation. Hence, if a rising edge of the clock occurs during the discharge of the signal *Self_Sig*, the operation that is expected to start is discarded.

Experimental Scenario 1

In the first experiment, we have executed $OS =$ "$r_1 r_1 w_0 r_0$" with a fixed memory address, which provides access to 64 core-cell (remember that the studied SRAM has 64 memory elements), after WU phase from DS mode, considering WUT_nom as WU time and clock cycle equals to $Tcyc_nom$. Note that all accessed core-cells store logic '1' prior to OS execution. The experiment has been performed with *Df6* set to a high resistance value (25KΩ), such that all transistors *P_PC1* of the PS connected to the peripheral circuitry are turned off when OS is executed. Figure 2.25 shows waveforms obtained through the experiment.

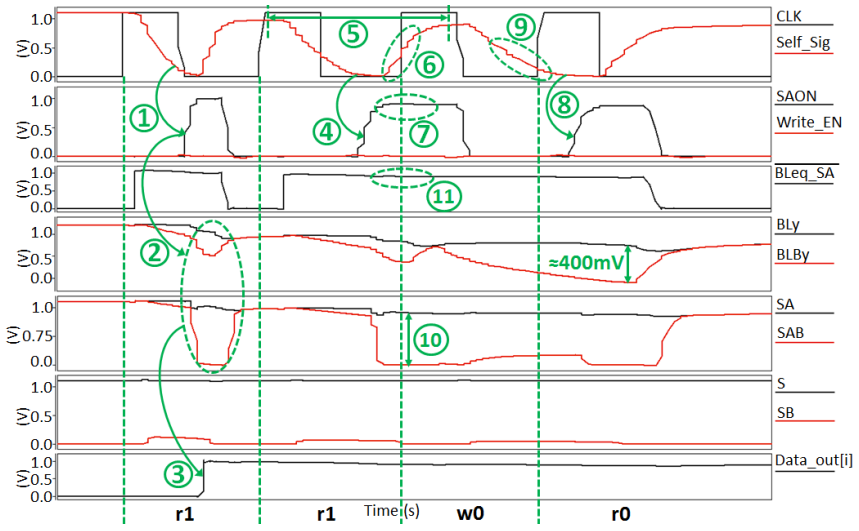


FIGURE 2.25: Execution of operation sequence $r_1 r_1 w_0 r_0$ after wake up phase from deep-sleep mode.

In Figure 2.25, *SAON*, *Write_EN*, $\overline{BLeq\_SA}$ and *SA/SAB* are signals corresponding to a given memory element $i, i \in [0, 63]$, while S and SB correspond to storage nodes of the accessed core-cell $CC_{x,y}$ in memory element i, which is connected to a given selected WL x and to a given selected BL pair $BL_y/BLB_y, x, y \in [0, 511]$. Signal *Data_out[i]* is the output of memory element i, which corresponds to data read from core-cell $CC_{x,y}$.

In this experiment, we first observe that signal *Data_out[i]* is at logic '1' at the end of the OS, while logic '0' was expected. Thus, *Df6* leads to an observable error when the

OS is executed. Secondly, we observe that storage nodes S and SB of $CC_{x,y}$ do not change during OS execution, while a transition $1 \rightarrow 0$ was expected in such core-cell. This is because operation w_0 has not been executed, since write driver is never activated (note that signal *Write_EN* is always at logic '0'). In the remainder of this subsection, the observed phenomena are investigated in further details.

During the first clock cycle, when the first r_1 is performed, we observe that the sense amplifier is correctly activated at the end of the discharge of signal *Self_Sig* (refer to arrow "1" in Figure 2.25). Hence, the weak potential difference between the BLs is amplified through signals SA and SAB (refer to "2") and the output *Data_out[i]* is set to logic '1', as expected (refer to "3"). At the second rising edge of the clock, note that signals driven by the peripheral circuitry (e.g. signal *Self_Sig*) are charged at a voltage level lower than V_{DD} (1.1V), which indicates a voltage droop in line V_{DD}_PC.

During the execution of the second r_1, which occurs in the second clock cycle, the sense amplifier is also correctly activated at the end of the discharge of signal *Self_Sig*, as indicated by arrow "4". However, we observe that the full cycle of *Self_Sig* (discharge to 0V and restore to V_{DD}) lasts for more than a clock cycle (refer to "5"), which is longer compared to the previous r_1. Since the full cycle of the self-timing signal reflects approximately the time required to complete an operation, we can conclude that the voltage droop in supply line V_{DD}_PC causes performance loss. Despite of this issue, the read operation is correctly performed, similarly to the previous one.

Due to the observed performance loss, the third rising edge of the clock (when w_0 is supposed to start) occurs when the self-timing signal is in restore phase, which corresponds to the end of the previous operation (refer to "6"). Moreover, we observe that the sense amplifier is still activated to complete the previous r_1 (refer to "7"). As a consequence, the discharge of *Self_Sig* starts delayed more than half of a clock cycle with respect to the third rising edge of the clock. Although the operation starts executing, we observe that the write driver is not activated (signal *Write_EN* remains set to logic '0'). Instead, we verify the third rising edge of signal $SAON$, which occurs when signal *Self_Sig* is fully discharged, as indicated by arrow "8" in Figure 2.25. Therefore, a read operation is performed instead of the w_0 operation, and then the transition $1 \rightarrow 0$ does not occur in the accessed core-cell $CC_{x,y}$.

This is because the cycle of the self-timing signal corresponding to the previous operation is not completed when the third rising edge of the clock occurs (signal *Self_Sig* is in restore phase). Hence, in such instant, the latch inside the SRAM control block holds the type of the previous operation, which is a read. As a consequence, a read operation is executed in the third clock cycle.

At the fourth rising edge of the clock, we observe the discharge of the self-timing signal corresponding to the previous operation is not completed (refer to "9"). As a consequence, the fourth operation, which is expected to be a r_0, is discarded. At the end of the OS execution, signal *Data_out[i]* remains at logic '1' (the result of the operation executed in the third clock cycle), while logic '0' was expected. Therefore, the observed FB is detected.

Finally, note that BLB_y is discharged over 400mV during the read operation executed in the third clock cycle, while normally it is discharged over 250mV. This is because signals *SA* and *SAB* are not equalized at V_{DD} when such read operation starts (refer to "10"). Instead, *SAB* is at 0V, which pulls down BLB_y to a lower level than the expected one. This occurs because the pre-charge circuit of the sense amplifier is not turned on (i.e. signal $\overline{BLeq\_SA}$ remains at logic '1') during the restore of *Self_Sig* that occurs at the end of the second operation (refer to "11").

Additional experiments have also shown that a read operation may return incorrect data if nodes *SA/SAB* and *WD/WDB* of memory elements are not equalized at V_{DD} at the beginning of the operation. For example, let us consider a core-cell that stores logic '0', connected to a given BL pair BL_y/BLB_y of a memory element, during a read access. When the operations starts, BLB_y is connected directly to *SAB* and *WDB*. If *SAB* or *WDB* is at 0V, this provokes an undesired discharge of BLB_y. At the end of the operation, if BLB_y discharges more than BL_y, the sense amplifier transmits logic '1' to the SRAM output, thus leading to an incorrect read fault.

As described above, the observed faulty behaviors are due to performance loss caused by *Df6*. In the next subsection, we investigate the possibility to mitigate the effect of such defect by modifying either the frequency or the WU time of the memory.

Experimental Scenario 2

The second experimental scenario is based on the execution of the same $OS =$ "$r_1r_1w_0r_0$", after WU phase from DS mode, with a fixed memory address and with *Df6* set to the same resistance value (25KΩ) as in experimental scenario 1. This experimental scenario differs from the first one because we varied either the WU time or the clock cycle used in the electrical simulation instead of using nominal values for both.

Figure 2.26.a shows waveforms generated during the execution of OS when considering WU time and clock cycle equal to $2*WUT_{nom}$ and $Tcyc_{nom}$, respectively, whereas Figure 2.26b presents waveforms obtained with WU time and clock cycle equal to WUT_{nom} and $2 * Tcyc_{nom}$, respectively.

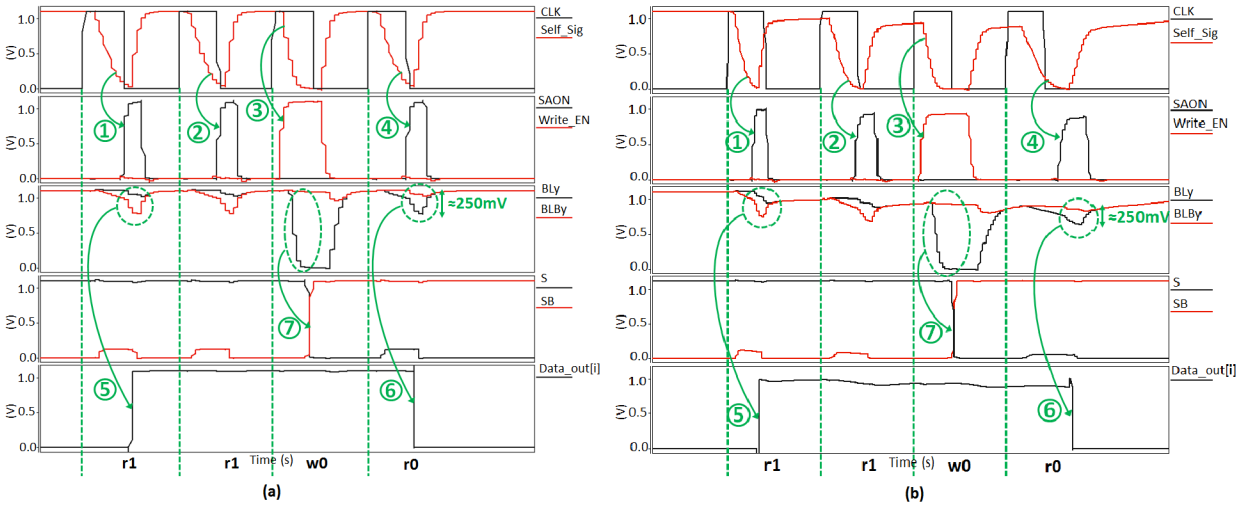


FIGURE 2.26: Execution of operation sequence $r_1 r_1 w_0 r_0$ after wake up phase from deep-sleep mode (a) WU time $2 * WUT_{nom}$ and clock cycle $Tcyc_{nom}$ (b) WU time WUT_{nom} and clock cycle $2 * Tcyc_{nom}$.

In both Figure 2.26.a and Figure 2.26.b, waveforms show that all operations are correctly executed, despite the presence of *Df6*. First, we can observe that, during read operations, the sense amplifier is activated at the end of the fall transition of *Self_Sig* (refer to arrows "1", "2" and "4" in both Figure 2.26.a and Figure 2.26.b), as expected. Moreover, during write operations, we can verify that the write driver is correctly activated at the beginning of the fall transition of *Self_Sig* (refer to arrows "3"). Secondly, we can verify that *Data_out[i]* always contains the result of read operations, i.e. it is set to logic '1' during the first r_1 (refer to arrows "5"), while it is set to logic '0' during the r_0 (refer to arrows "6"). Note that *Data_out[i]* is set according to the weak voltage difference between the BLs generated during the read. Moreover, we can observe that storage nodes S and SB are set to logic '0' and logic '1', respectively, as a result of w_0, thus indicating that such operation has been correctly performed (refer to arrows "7"). Finally, note that BL_y and BLB_y are discharged over 250mV during read operations, as expected. This is because pre-charge circuits connected to all BL pairs, sense amplifiers and write drivers are properly activated at the end of operations.

In particular, no FB occurs in the experiment depicted in Figure 2.26.a because the considered WU time is large enough to guarantee activation of all transistors *P_PC1* of the PS connected to the peripheral circuitry before OS execution. This is confirmed by observing that no voltage droop occurs at nodes driven by V_{DD}_PC (e.g. signal *Self_Sig*) during OS execution. Finally, no FB occurs in the experiment illustrated in Figure 2.26.b because the used clock cycle compensates the performance loss issues caused by *Df6* even if the WU time is the nominal one. This is confirmed by observing that a full cycle of *Self_Sig* never lasts for more than a clock cycle. This experiment

proves that effects caused by *Df6* can be mitigated by reducing memory performance parameters.

Summary

The above discussed experiments have shown that *Df6* can lead to FBs during the execution of operations after WU phase, especially when considering minimum values of WU time (WUT_{nom}) and clock cycle ($Tcyc_{nom}$). We have also performed additional experiments by varying the OS executed after WU phase, PVT conditions and resistance value of *Df6*. In such experiments, we observed other FBs apart from those shown during the above experiments. The complete set of FBs can be enumerated as follows:

- **FB1**: a read operation is discarded, as shown in the experimental scenario 1.
- **FB2**: a write operation is discarded. This FB is similar to FB1, except that the faulty operation is a write.
- **FB3**: a write operation preceded by a read is erroneously executed as a read operation, as shown in the experimental scenario 1.
- **FB4**: a read operation preceded by a write is erroneously executed as a write operation. This FB is similar to FB3, except that the faulty operation is a read.
- **FB5**: a $r_{\overline{x}}$ operation performed right after a r_x returns an incorrect value. Here and below, $x, \overline{x} \in \{0,1\}, x \neq \overline{x}$. This FB is caused by the partial equalization of SA and SAB nodes of memory elements at the beginning of read operations.
- **FB6**: a $r_{\overline{x}}$ operation performed right after a w_x returns an incorrect value. This FB is caused by the partial equalization of WD and WDB nodes of memory elements at the beginning of read operations.

All FBs are caused by performance loss issues induced by degradation of the voltage supplied to the peripheral circuitry. When a FB occurs, the sensitizing operation fails because it is launched when the previous operation is still executing, i.e. in the middle of a cycle of the SRAM self-timing signal. The possible causes of failure can be listed as follows: 1) the SRAM is blocked executing the previous operation (FB1 and FB2), 2) the SRAM control block fails to catch the operation type (FB3 and FB4); 3) equalization of signals *SA* and *SAB* is not completed at the end of a read operation (FB5) and 4) equalization of signals *WD* and *WDB* is not completed at the end of a write operation (FB6).

In this extensive set of experiments, we have also observed that ability of a given OS in sensitizing FBs caused by *Df6* does not depend on the addressed core-cells. For example,

in the experimental scenario 1, the same FB (FB3) is sensitized when each operation of the OS is executed in the same core-cells (i.e. with fixed memory address) and when different core-cells are addressed in each operation. This was expected, since *Df6* does not affect the core-cell array, but the peripheral circuitry.

Moreover, additional experiments shown that a given OS may sensitize a FB only for specific scenarios of PVT conditions and resistance value of *Df6*. For example, no FB occurs when the experiment described in the experimental scenario 1 is performed by changing only the resistance value of *Df6* to 20KΩ. However, if the OS is also changed to "$r_1w_0r_0w_1r_1$", the last read operation sensitizes FB4. This shows that it is very challenging to determine an OS that is capable to sensitize a FB in presence of a given resistance value of *Df6* and under a given PVT condition. This makes the test of such defect also very challenging.

Finally, experimental results presented in experimental scenario 2 have shown that the impacts of *Df6* can be mitigated at the expense of performance in two ways, described as follows:

1. Consider a WU time greater than WUT_{nom} before executing operations after WU command. No FB should occur if such adjusted WU time is large enough to guarantee the correct activation of transistors *P_PC1* when operations are executed after WU phase. This approach has been validated by the experiment described in Figure 2.26.a.
2. Use a clock cycle greater than $Tcyc_{nom}$ when executing operations after WU phase. No FB should occur if such adjusted clock cycle is large enough to compensate the performance loss issues caused by *Df6*, described in the experimental scenario 2. This approach is validated by the experiment described in Figure 2.26.b.

In next section, we exploit the failure analysis presented in this section to present an efficient test solution to detect *Df6*.

2.1.6.4 Test Solution

In this section, we present an efficient March test algorithm (defined in [**?**]) targeting detection of *Df6*, which contain OSs that allow sensitization and detection of the FBs previously identified. Before analyzing the proposed test algorithm, we discuss about special conditions that must be satisfied by a March test in order to detect a subset of the FBs above presented.

Conditions to detect faulty behaviors

The conditions presented in this section particularly refer to detection of FB1, FB2, FB3 and FB4. Such conditions can be summarized as follows:

1. To detect FB1 and FB4, a March test must ensure that two consecutive read operations are executed in core-cells that store opposite logic value.
2. To detect FB2 and FB3, a March test must ensure that write operations are performed in core-cells that store opposite data compared to the one that is supposed to be written.

The first condition is required because, in most commercial SRAMs, the data output (i.e. *Data_out* in the SRAM used as case study) stores the result of the last read operation. Hence, if a r_x operation is not executed, either because it is discarded (FB1) or because it is erroneously executed as a write (FB4), such fault is masked if the last read operation that is correctly performed is a r_x as well. This is because the SRAM output will be set to logic x at the end of the faulty operation (as a result of the previous r_x), which is the expected logic value.

The second condition is required because, if a fault occurs due to a discarded write operation (FB2) or due to a write operation that is erroneously executed as a read (FB3), such fault is masked if the accessed core-cells already store the logic values that is supposed to be written.

Such detection conditions are the base of the March test algorithm presented in the next subsection, called March LZ.

March LZ

The structure of March LZ is the following:

March LZ

$\{$
$\updownarrow (w_1);$ $ME1$
$DSM;$ $ME2$
$WUP;$ $ME3$
$\uparrow (r_1, w_0, r_0);$ $ME4$
$\}$

In March LZ, March element (ME) [**?**] ME1 initializes all core-cells at logic '1', while ME2 switches the SRAM from ACT to DS (represented by notation DSM). In

ME3, the SRAM is turned back to ACT mode, which corresponds to the WU phase (WU command followed by WU time), represented by notation WUP. Finally, ME4 corresponds to operations executed after WU phase to sensitize and detect FBs cause by *Df6*. The time complexity of March LZ is $4N + K$, where N is the total number of SRAM addresses and K is a time constant corresponding to the execution time of ME2 and ME3 (i.e. DSM and WUP).

The analysis presented in the above sections shown that it is not trivial to determine OSs that are effective to sensitize FBs caused by *Df6*. This is because a given OS may be effective only for particular scenarios of PVT conditions and resistance value of the defect. Hence, to maximize the coverage of *Df6*, the March tests we propose have been developed such that all operations executed after WU phase, along with the preceding operations, compose an OS that can potentially sensitize at least one detectable FB caused by *Df6*.

In March LZ, the execution of ME4 generates OSs that can sensitize and detect FB1, FB2, FB3, FB4 and FB5. FB1 can be sensitized and detected by each read operation in ME4. The w_0 operation can sensitize FB2, whereas it is detected by the r_0 operation that follows. FB3 can be sensitized in each execution of the sequence "r_1w_0" and detected by the r_0 operation that follows, while FB4 can be sensitized and detected each time the sequence "w_0r_0" is executed. Finally, FB5 is sensitized and detected by r_0 and the following r_1. We can observe that the condition to detect FB1 and FB4 is fulfilled in March LZ, since two consecutive read operations in ME4 are always operations of opposite logic values. Moreover, March LZ also satisfies the requirement to detect FB2 and FB3, since core-cells are initialized at logic '1' when the w_0 is performed.

It is important to mention that, since *Df6* causes performance loss issues, it is mandatory to execute the algorithm at maximum frequency (i.e. clock cycle set to $Tcyc_{nom}$) to maximize the occurrence of FBs. Moreover, it is also mandatory to consider the minimum WU time (i.e. WUT_{nom}) during test phase to maximize the time interval when operations are executed with transistors P_PC1 deactivated. This also maximizes the occurrence of FBs caused by *Df6*.

Finally, in terms of defect coverage, we verified through electrical simulations that 40KΩ is the minimal resistance value of *Df6* that March LZ can detect in all considered PVT conditions.

2.1.6.5 Conclusions

In this work, we presented an extensive study, based on electrical simulations, to characterize the impacts of resistive-open defects affecting signals driving power gating mechanisms embedded in a commercial low-power SRAMs. For each defect, both a qualitative and a quantitative analysis of its impacts have been presented. We have shown that one particular defect causes performance loss issues, which can provoke faulty behaviors. Based on such failure analysis, an efficient March test has been developed to sensitize and detect the observed faulty behaviors.

This work and its extension are published in [C36, C37, C40, C41, C43, I3, J11] referenced in the Publication list of chapter 4.

2.2 Students

2.2.1 Master students

2.2.1.1 Master 1

- Vincent COUBARD (2008): Réalisation d'un système de fichier pour NAND-FLASH
- Ibrahim AMER (2008): Réalisation d'un outil d'injection de fautes
- Cheikh Bounama NIANG (2009): Mise en oeuvre d'un simulateur de fautes Date
- Ludovic LE-BRAS, Romain SEIGLE (2009): Réalisation d'un dictionnaire de fautes
- Papa Magatte DIAGNE (2009): Réalisation d'un debugger pour microprocesseur
- Sébastien PHERON (2010): Etude de réalisation d'un simulateur de faute Déductif
- Pierre YANG (2010): Implantation d'une interface graphique
- Ranavy HONG (2010): Etude et Réalisation d'un simulateur de fautes concurrent
- Long NGUYEN THANH (2012): Simple robot movement

2.2.1.2 Master 2

- Matteo VEZZANI (2004): Automatic March Test Generator for Static and Linked Faults

- Alessandro SAVINO, Fabio NAZIONALI (2005): Microprocessor software based self test
- Roland MOBEANG (2008): Réalisation d'un outil de gestion des bibliothèques technologiques pour la simulation des circuits numériques
- Ahmed Amine REKIK (2008): Simulateur de mémoires, étude bibliographique
- Laila DAMRI (2009): Implantation d'un Soc TMR sur FPGA
- Liang SHAO (2010): Test of TSV-Based 3D Stacked Integrated Circuits
- Yannick LAFFRAY (2011): Transmission des informations d'un chronotachygraphe et interface utilisateur
- Amine BEN MARIEM (2012): Analyse Automatique de datalog et génération de patterns haute résolution
- Artem MARISOV (2013): Implementation of a TMR scheme at flip-flop level

2.2.2 Ph.D. students

Table 2.11 summarizes the Ph.D. students I co-directed, along with the period of the thesis and the percentage of supervision.

Ph.D. Student	Years	%
Alessandro SAVINO	2006	50
Alexandre ROUSSET	2006-2008	50
Youssef BENABBOUD	2007-2010	50
Fangmei WU	2008-2011	30
Zhenzhou SUN	2011-2014	50
Miroslav VALKA	2010-2014	50
Leonardo ZORDAN	2010-2013	50
Aymen TOUATI	2013-	70

TABLE 2.11: Summary of the Ph.D. Students.

Alessandro SAVINO

Title: Software-based Self-Test of Microprocessor

System-On-Chip technology trends are so advanced that in a single chip one can embed different core, having different functionality, making possible the definition of the Intellectual Properties, such as microprocessor, memories and other peripherals. Microprocessor core testing becomes a very difficult task, due to the high complexity of internal architecture, moreover nature of an embedded core makes impracticable resort

to classic external test mechanisms such Automatic Test Equipments. Rising of Built In Self Test (BIST) overcomes this problem porting the test mechanism and patterns inside the core. BIST solution usually requires to stop the core functionality in order to perform the test. However, critical applications require an on-line test executed in working condition (it means that the component is placed inside the environment where it will work since its lifetime). On-line test nature suggests the typical preferred approach: Software-Based Self-Test (SBST), where test patterns are the microprocessor instructions, provided by the Instruction Set Architecture.

The PhD thesis studied and investigates novel microprocessor test generation methodologies.

Alexandre ROUSSET

Title: Logic Diagnosis for digital circuits

Due to the advances in manufacturing technologies and more aggressive clocking strategies used in modern design, more and more defects lead to failures which can no longer be modeled by classical stuck-at faults. Numerous actual failures exhibit timing or parametric behaviors which are not represented by stuck-at faults. Such failures have to be taken into account during the test process in order to reach acceptable DPM (Defect per Million) figures

The PhD thesis designed and implemented a unified diagnostic framework. The framework was based on an Effect-Cause approach which relies on the two following main operations. The first one is based on Critical Path Tracing algorithm and consists in identifying critical lines in the failing device which can be sources of observed errors. The second one consists in allocating a set of possible fault models to each critical line, so that root causes of failures can be finally determined. The main advantage of this method is that it does not need to explicitly consider each fault model during the identification of critical lines.

I was a member of his thesis defense jury.

Youssef BENABBOUD

Title: Logic Diagnosis for System-on-Chip

System on Chip (SoC) has become a widely accepted architecture for complex and heterogeneous systems that include digital, analog, mixed-signal, radio-frequency, micromechanical, and other types of components on a single piece of silicon. Despite the efficiency of their design and manufacturing processes, SoCs may be affected by defects.

Numerous actual failures exhibit timing or parametric behaviors, which are not represented by stuck-at faults. Such failures have to be taken into account during test and diagnosis of SoCs in order to reach acceptable DPM (Defect per Million) figures. The context of this study is the following. We consider SoCs which are composed of several different cores such as Memory, PLL, ADC, DAC, Logic, DSP, and CPU. To simplify the presentation in this paper, we consider that cores can be classified in three main categories: memory, logic and analog cores respectively.

The PhD thesis proposed a diagnosis framework able to handle a whole SoC. The framework is an extension of the one developed during the PhD thesis of Alexandre ROUSSET, the main modification are: the introduction of a preprocessing step to deal with the complete SoC architecture instead of a simple logic block, the definition of a new algebra for the CPT process during fault localization, and a new fault model allocation procedure. In the case the failure is located in a non-logic core, the logic diagnosis approach will identify the outputs of the core as the location of the failure. The new (system level) diagnosis tool has been evaluated with SoC test cases built from typical ST Microelectronics SoCs. The diagnosis resolution (i.e. absolute number of suspects) has been measured and proves the accuracy of the proposed diagnosis approach.

This work was funded in the context of the research contract STM.

I was a member of his thesis defense jury.

Fangmei WU

Title: Power-Aware Test

Power consumption during structural test can be higher than the function power consumption. Power reduction is thus mandatory to reduce the problems related to the over-test (i.e., yield loss and circuit damage).

The PhD thesis proposed a new power-aware test pattern generation technique for LOS testing. This technique is based on test relaxation and an iterative X-filling process, and reduces power consumption during test at a level that can be compared to the level of power consumption during functional mode. This additional feature (obtaining comparable power consumption during test and during functional mode) is really important as, in addition to solving the yield loss problem, it also avoid test escape (bad chips are declared as good during manufacturing test) that may occur when test power is too much reduced compared to functional power.

This work was funded in the context of the research contract TOETS.

Zhenzhou SUN

Title: Defect Localization Improvement through Transistor Level Diagnosis

The rapid growth in semiconductor field results in an increasing complexity of digital circuits. The ability to identify the root cause of a failing digital circuit is becoming critical for defect localization. Logic diagnosis is the process of isolating the source of observed errors in a defective circuit, so that a physical failure analysis can be performed to determine the root cause of such errors. Effective and precise logic diagnosis is crucial to speed up the failure analysis and eventually to improve the yield. "Effect-Cause" and "Cause-Effect" are the two classical approaches for logic diagnosis. Logic diagnosis provides a list of gates as suspects. However, this approach may not leads to accurate results in the case of the defect is inside a gate.

The PhD thesis proposed a new intra-cell diagnosis method based on "Effect-Cause" approach to improve the defect localization accuracy at transistor level. The proposed approach exploits the CPT (Critical Path Tracing) applied at transistor level. For each suspected cell, we apply the CPT for every given failing test vector. The result is a preliminary list of candidates. Each candidate can be a net or a transistor drain, gate or source. After that, we apply the CPT for each passing test vector in order to narrow down the the list of candidates. The proposed method gives precise localization of the root cause of the observed errors. Moreover, it does not require the explicit use of a fault model.

This work was funded in the context of the research contract STM Grenoble.

I was a member of his thesis defense jury.

Miroslav VALKA

Title: Power aware test and Test of Low Power Devices

Nowadays, electronic products present various issues that become more important with CMOS technology scaling. High operation speed and high frequency are mandatory requests. On the other hand, power consumption is one of the most significant constraints due to large diffusion of portable devices. The need of power efficient electronic devices leads to implement dedicated structures to reduce as much as possible the power consumption. These needs influence not only the design of devices, but also the choice of appropriate test schemes that have to deal with production yield, test quality and test cost. Testing for performance, required to catch timing or delay faults, is therefore mandatory, and it is often implemented through at-speed scan testing for logic circuits. In the meanwhile, specific test solutions have to be developed for the structures dedicated to power savings. This PhD thesis targets both the above aspects: (i) the test for performances in order to guarantee the best tradeoff between test quality and

power consumption during test, (ii) the test of power gating structures allowing power reduction.

This work was funded in the context of the research contract STE.

I was a member of his thesis defense jury.

Leonardo ZORDAN

Title: Test of Low-Power Memories

Nowadays, embedded memories are the densest components within System-On-Chips (SOCs), accounting for more than 90% of the overall SOC area. Among different types of memories, SRAMs are still widely used for realizing complex SOCs, especially because they allow high access performance, high density and fast integration in CMOS designs. On the other hand, high density SRAMs designed with deep-submicrometer technologies have become the main contributor to the overall SOC power consumption. Hence, there is an increasing need to design low-power SRAMs, which embed mechanisms to reduce their power consumption. Moreover, due to their dense structure, SRAMs are more are more prone to defects compared to other circuit blocks, especially in recent technologies. Hence, SRAMs are arising as the main SOC yield detractor, which raises the need to develop efficient test solutions targeting such devices.

In this PhD thesis, failure analysis based on electrical simulations has been exploited to predict faulty behaviors caused by realistic defects affecting circuit blocks that are specific to low-power SRAMs, such as power gating mechanisms and voltage regulation systems. Based on identified faulty behaviors, efficient March tests and low area overhead design for testability schemes have been proposed to detect studied defects. Moreover, the reuse of read and write assist circuits, which are commonly embedded in low-power SRAMs, has been evaluated as an alternative to increase stress in the SRAM during test phase and then improve the defect coverage.

This work was funded in the context of the research contract INTEL.

I was a member of his thesis defense jury.

Aymen TOUATI

Title: Exploring the Impact of Functional Test Programs Re-Used for Power-Aware Testing

High power consumption during at-speed delay fault testing may lead to yield loss and premature aging. On the other hand, reducing too much test power might lead to test escape and reliability problems. Thus, to avoid these issues, test power has to map

the power consumed during functional mode. Existing works target the generation of functional test programs able to maximize the power consumption in functional mode of microprocessor cores. The obtained power consumption will be used as threshold to tune the power consumed during testing.

This PhD thesis investigates the impact of re-using such functional test programs for testing purposes. We propose to apply them by exploiting existing DfT architecture to maximize the delay fault coverage. Then, we combine them with the classical at-speed LOC and LOS delay fault testing schemes to further increase the fault coverage. Preliminary results show that it is possible to achieve a global test solution able to maximize the delay fault coverage while respecting the functional power budget.

This work is partially funded in the context of the research contract LIA LAFISI.

The thesis is supposed to be concluded at the end of 2016.

2.3 Teaching activities

Starting from my Ph.D. studies, I have performed various teaching activities at different levels: engineering students of the Politecnico di Torino as well as the Engineering School of the University of Montpellier 2, master and bachelor students of the University of Montpellier 2 and master students of the University of Science and Technology of Hanoi (USTH - Vietnam).

The topics of my teaching activities (focusing on academic courses only), as well as the description of the course and the targeted students are described in the following paragraphs:

Computer Science:

This class introduces the student to the issues related to computer science and it aims at teaching the use of computer programming using the C and C++ languages.

- Covered topics:
  - C language: Data types, Symbolic constants, Input/output operations (printf and scanf), Control-Flow structures (iterative and conditional), Arrays and multidimensional arrays (of integers, reals and characters), Functions and calls (by reference, by value, pointers), Strings, Command line arguments (argc and argv), Files, Struct, Dynamic memory allocation.

  - C++ language: Classes, Objects, Polymorphisme, Template, STL, operator overloads

- When, where, for who:
  - 2003/04, 2004/05 Politecnico di Torino, Computer Engineering students
  - from 2007/08 until now, Université de Montpellier II, master and bachelor students

- Responsibility: I'm the responsible of the computer science course in the second year of the Master from 2007/08 until now. For all my courses I have prepared both handouts and lab exercices. The latter can be accessed from my home page

Digital System Testing:

The course aims at showing the importance of testing within the design and manufacturing process. It presents the techniques for testing custom Integrated Circuits, microprocessors, memories, and system test.

- Covered topics: Fault modeling, understanding of the tools for testing an embedded system: fault simulator, automatic test pattern generator, tools for automatic scan chain insertion. Concept of Built-In Self-Test (BIST).

- When, where, for who:
  - 2004/05, Politecnico di Torino, Computer Engineering students
  - from 2009/10 until now, Polytech' Montpellier (Université de Montpellier II), Engineering students

- Responsibility: I share the responsibility of the digital system testing course and I have prepared both handouts and lab exercices. The latter can be accessed from my home page

Computer Network:

The course aims at showing the basic principle of computer networking. It presents the main protocols of the TCP/IP stack and the UNIX socket API for programming.

- Covered topics: IEEE ethernet, IPv4, TCP, UDP, NAT .

- When, where, for who:

  - from 2007/08 until now, Université de Montpellier II, master and bachelor students

- Responsibility: I'm the responsible of the computer network course from 2007/08 until now. For all my courses I have prepared both handouts and lab exercices. The latter can be accessed from my home page

Digital System Design:

The course aims at teaching advanced design methodologies for simple digital systems. (Both manual and automatic approaches are presented), highlighting on product life-cycle, focusing on all the aspects of the design phase and finally providing hands-on experience on EDA (Electronic Design Automation) tools

- Covered topics: combinational design, sequential design, VHDL language, simulation and synthesis (manual and automatic).
- When, where, for who:
  - from 2007/08 until now, Université de Montpellier II, master students
  - from 2011/12 until now, University of Science and Technology of Hanoi, master students
- Responsibility: I prepared both handouts and lab exercices. The latter can be accessed from my home page

Advanced MPSoC Architecture:

The course aims at teaching parallel programming paradigm, Multi-Processor SoC Architectures and fault tolerant architectures

- Covered topics: Threads, Process, IPC, Petri Nets, Redundancy, Reliability estimations.
- When, where, for who:
  - from 2007/08 until now, Université de Montpellier II, master students
  - from 2011/12 until now, University of Science and Technology of Hanoi, master students
- Responsibility: I prepared both handouts and lab exercices. The latter can be accessed from my home page

2.4 Research Contracts

NanoTEST

Title: Nano TEST (NanoTEST)

Place and period: 2005-2008, LIRMM

Participants: STMicroelectronics, Infineon, Philips, NXP, Q-Strar Test, Temento, CEA-LETI, TIMA, LIRMM

Funding: Projet CEE EUREKA - MEDEA - n 2A702

Goal of the project: The microelectronics industry is rapidly entering the nanotechnology era with circuit nodes well below 90 nm. Mass production is only economically feasible when the proper test technology is in place to reduce costs, improve quality and cut time to market. It is therefore essential for European chipmakers to develop new in-house tools and proprietary test methodologies in addition to the mainly US-sourced commercial test tools. NanoTEST will ensure development of the tools and standards for system-on-chip and system-in-package technologies. Availability of the right methods and tools earlier than for US and Asian competitors will contribute significantly to commercial success in Europe.

Contribution and results: I was involved in the diagnosis task. My scientific contributions in this project have been published in [C10, C11, C12, C13, J5].

STM

Title: Diagnosis of Integrated Circuits

Place and period: 2007-2009, LIRMM

Participant: LIRMM, STM

Funding: STMicroelectronics Crolles (These CIFRE)

Goal of the project: Analysing the impacts of the defects in the latest technologies and develop meaningful diagnosis framework.

Contribution and results: I was the “co-encadrant” for the Ph.D student involved in the contract (Youssed BENABBOUD). My scientific contributions in this project have been published in [C17, C18, C19, C21, C25].

TOETS Title: Towards One European Test Solution (TOETS)

Place and period: 2008-2011, LIRMM

Participants: NXP, Infineon, Philips, Q-Star Test, Semicon, STMicroelectronics, Temento, iRoC, ATMEL, E2V Semiconductor, JTAG Technologies, Salland Engineering, Advanced Digital Design, Tomorrow Options Microelectronics, Ophtimalia, CEA- LETI, CEA-LIST, University of Twente, LIRMM, TIMA, KULEUVEN, SUPELEC, IMSE-CNM, INESC Porto

Funding: European Project CATRENE CT302

Goal of the project: has the ambition to create a breakthrough in methods and flows used by the test technologies by considering the test in the whole value chain from Design to Application. It faces the giga-scale complexity of SoC with various and numerous implemented functions (RF, analogue, digital, memory), for which test becomes an economical roadblock. Its objectives are the definition of test architectures for minimizing the costs of test operation, to develop alternative test solutions and test flows for test cost reduction and product quality improvement. TOETS is geared toward the secure and safety critical application domain (consumer and medical).

Contribution and results: I was involved in the power-aware test task, in particular I was the "co-encadrant" for the Ph.D student involved in the contract (Fangmei WU) .My scientific contributions in this project have been published in [C23, C24, C27, C34, I2, J6].

INTEL

Title: Test of Low-Power SRAM

Place and period: 2011-2013, LIRMM

Participant: LIRMM, INTEL

Funding: INTEL

Goal of the project: Analysing the impacts of the defects affecting the low-power SRAM produced by INTEL and develop meaningful test solutions.

Contribution and results: I was the "co-encadrant" for the Ph.D student involved in the contract (Leonardo ZORDAN). My scientific contributions in this project have been published in [C28, C36, C37, C40, C41, C43, I3, J11].

STE

Title: Test of Low-Power Circuits

Place and period: 2011-2014, LIRMM

Participant: LIRMM, STE

Funding: STE

Goal of the project: Analysing the impacts of the defects affecting the low-power dedicated circuitry of modern SoCs and develop meaningful test solutions.

Contribution and results: I was the "co-encadrant" for the Ph.D student involved in the contract (Miroslav VALKA). My scientific contributions in this project have been published in [C30, C38, C45, C46, J9, P1, P2].

STM Grenoble

Title: Transistor-Level logic Diagnosis

Place and period: 2011-2014, LIRMM

Participant: LIRMM, STM Grenoble

Funding: STM Grenoble

Goal of the project: Target the defects affecting the transitor-level netlist of combinational library cells, and develop appropriate diagnosis solutions.

Contribution and results: I was the "co-encadrant" for the Ph.D student involved in the contract (Zhenzhou SUN). My scientific contributions in this project have been published in [C35, C42, J10].

LIA LAFISI

Title: French-Italian research LAboratory on hardware-software Integrated Systems (LAFISI)

Place and period: 20013-2016, LIRMM

Participants: LIRMM, Politecnico di Torino

Funding: CNRS and UM2

Goal of the project: The main objective of the LIA "LAFISI" is to promote the French-Italian collaboration in the field of hardware-software integrated systems, by developing novel approaches for built-in self-test and design-for-test, test pattern generation and fault simulation, performance verification, diagnosis, fault tolerance and improvement of system reliability. The wide expertise of the two partners in the fields of automatics, informatics and microelectronics, once combined, will allow to conduct researches on hardware-software integrated systems targeting classical applications such as processors

for computational units or systems-on-chip for mobile communications, or more critical applications such as embedded systems for space, medical, avionic or automotive domains.

Contribution and results: I was actively involved in the creation of the contract, and I'm actually a member of the international laboratory. My scientific contributions in this project have been published in [W4, C44].

Chapter 3

Perspectives

My *cursus* has a clear background on all activities, disciplines and skills related to test and diagnosis of digital systems. At the beginning of my research carrier, these skills were applied to memories as well as microprocessor. After moving to LIRMM, I specialized in diagnosis, power-aware test and test of low-power devices. My perspectives are thus organized following the 3 main themes of my current research activities. In the following I will detail each one ranked in short-, medium- and long-term perspectives.

3.1 Short-term Perspective

IC manufacturers exploit the technology scaling to produce smaller and faster electronic devices. However, manufacturing defects affecting latest technologies lead to dynamic faults (i.e., delay faults). Such faults require at-speed test to be detected. Usually, at-speed test is achieved by exploiting structural test patterns.

At-speed structural test leads to excessive power consumption that can either damage the Circuit Under Test (CUT) or lead to yield loss. Reducing the power during test is a well-known technique but reducing too much the power consumption leads to test escapes phenomena as depicted in Figure 3.1.

Therefore, to cope with the above issue, we have to control this risk by tuning the test power depending on the functional power of the device itself. The knowledge of the actual functional power is thus mandatory. In chapter 2 section 2.1.5, we exploited a generator of functional test programs for microprocessor core. The generator aims at maximizing the power consumption of the target microprocessor, thus the generated programs are good candidates to accurately estimate the functional power limits (i.e., to avoid both over- and under-test).

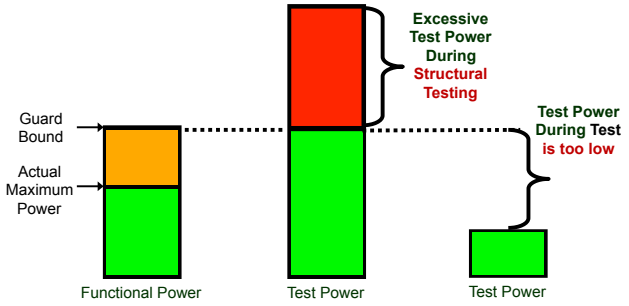


FIGURE 3.1: Over and Under Test phenomena.

Since generated programs maximize the power consumption, they are definitively characterized by a high switching activity. In other words, they could be also good candidates for delay faults. In this context, we would like to investigate more in detail the capability of such generated functional-programs to satisfy other metrics than the power consumption. Two metrics will be considered : structural and functional. The first one is related to the fault coverage, where the fault models are the stuck-at and the transition faults. The second one corresponds to the code coverage, such as statement, branch and toggle coverage.

Basically, the question we would like to answer is: can the functional programs, generated to maximize the power consumption, be re-used for verification and test purposes?

The future work will first consist in the evaluation of the functional test programs generated w.r.t the functional coverage. We will investigate on how much these test programs can meet the design quality needs by exercising each piece of code in the behavioral description of the microprocessor. A first analysis will be based on the most basic metrics, which are the statement, toggle and branch coverage. Then, a deeper analysis will move to a more sophisticated metric, which is the FSM coverage, to evaluate how many states are reached during the application of the functional programs. In a second part we will target the capability of such functional test programs to detect delay faults.

The main goal of this comprehensive evaluation is to verify whether or not the above metrics are in some way correlated in order to provide a meaningful test that can be reused many times during the overall product flow (i.e., during verification and for power aware test).

3.2 Medium-term Perspective

Guaranteeing the reliability, robustness, and manufacturability of complex and multi-technology electronic systems for 2020 and beyond is becoming a major challenge. There is experimental evidence that defects occurring in such emerging systems may escape existing manufacturing testing and diagnosis approaches.

In the case of remote, mission-critical and safety-critical applications, defect escapes may result in catastrophic consequences, placing in danger human lives, causing environmental accidents, jeopardizing the trustworthiness and integrity of data communication, causing non-repairable damages, etc. Therefore, efficient diagnosis schemes to locate and assess failures at different system levels and point to reliability hazards are of vital importance. Diagnosis offers insight about the failing part of the system that needs to be repaired, about the environmental conditions that can jeopardize the system's health, and about corrective actions that should be applied to prevent failure re-occurrence and, thereby, expand the safety features.

Efficient diagnosis schemes are also essential at the design stage to diagnose the sources of failures in the first prototypes. In this case, diagnosis tools can be combined with post-silicon validation tools to improve the debugging procedure. In this way, diagnosis can help to reduce design iterations and to meet the time-to-market goal.

Moreover, in high-volume production, diagnosing the sources of failures assists the designers in collecting valuable information regarding the underlying failure mechanisms, in order to enhance yield for future product generations through improvement of the manufacturing environment and development of design techniques that minimize the failure rate.

The outputs of fault diagnosis are the defect locations and the "strength" of the defects. The first step in the diagnosis flow is to perform Electrical Diagnosis (ED) which consists of crafting test stimuli and extracting test signatures that pinpoint with the highest resolution possible the area on the die where defects have occurred. The outcome of the first step guides the second step which consists of performing Failure Analysis (FA). The aim of FA is to physically examine the circuit to highlight anomalies in the operation and reduce further the area that should be analyzed with more scrutiny. As an example, a thermal camera can be used to locate hot spots inside the system. In the third and final step, based on the outcome of the FA, the circuit is submitted to Physical Failure Analysis (PFA) where selective de-layering and cross-sectioning of the die is performed using different types of imaging tools, such as Focused Ion Beam (FIB) etching, as shown in Figure 3.2. The aim of PFA is to confirm with incontestable certainty the defect locations and the "strength" of the defects.

It is clear that the level of success of ED determines to a very large extent the success of the PFA, as well as the time to diagnose the failures. The ED needs to be able to isolate with as much rigor as possible the area wherein the fault has occurred.

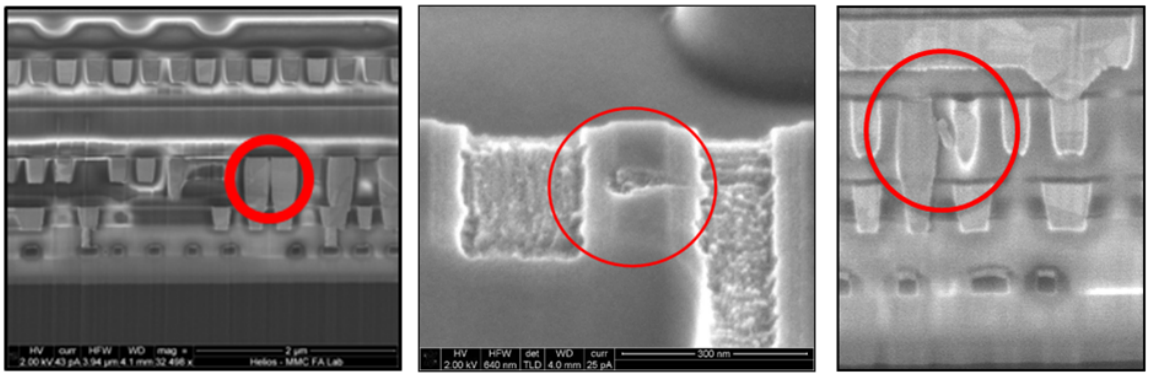


FIGURE 3.2: Cross sectioning and delayering of dies showing the defect location as identified by the PFA.

Furthermore, a modern system typically consists of independent and heterogeneous blocks and each block may comprise memory, digital, Analog and Mixed-Signal (AMS) circuits, as shown in Figure 3.3. Diagnosis solutions are circuit-dependent, that is, different and largely distinct diagnosis solutions apply according to the circuit type.

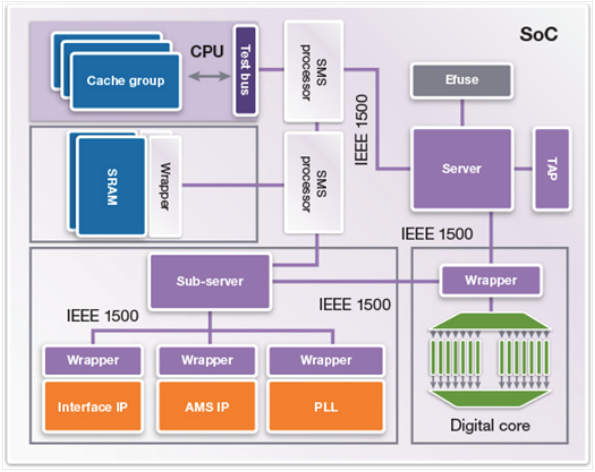


FIGURE 3.3: Heterogeneous System-on-Chip (courtesy of Synopsis).

The perspectives in this context mainly consist in the development of ED methodologies and tools with the aim to identify defects down to the transistor level, in order to speed-up and guide appropriately the PFA and improve its success rate. Among the different cores that compose a complex SoC (as shown in Figure 3.3) we will focus on the digital IP blocks.

Concerning digital IP blocks, existing ED commercial tools and methodologies present several limitations that can be summarized as follows:

- Too many defect candidates are reported, thus slowing down FA and reducing the chances that PFA successfully points to the actual defect that has occurred.
- There is no ED approach for the so-called "purely-functional" circuits, e.g. circuits that do not embed design-for-test circuitry which enables structural test.
- Each candidate provided by existing ED approaches is an interconnection between logic gates (e.g. inter-cell defect) and, therefore, existing ED approaches do not account for the case where the real defect is located inside one logic gate (e.g. intra-cell defect).

To address the challenge of diagnosing intra-cell defects, three different approaches will be investigated. The first approach will be based on a combination of cause-effect and effect-cause analyses to obtain a list of candidates. Then, a diagnostic test pattern generator will be developed for distinguishing between equivalent intra-cell defects. The second approach aims at proposing a cell design modification, in order to facilitate the ED (e.g. design-for-ED). For example, extra inputs can be added to the cell in order to force the cell to behave in a specific way depending on the intra-cell defect that is present. The third approach will exploit knowledge regarding the layout of the cell (e.g. layout-aware ED). These three approaches may turn out to be complementary and, thereby, a mixture of them could be used for developing an optimal ED flow for intra-cell defects that culminates in no or little defect ambiguity. As a by-product, an ED flow targeting specifically intra-cell defects can be used to prune the list of candidate inter-cell defects and to facilitate defect ranking. Finally, to address the challenge of purely-functional digital circuits, our approach will be to resort on functional test and build on top a dedicated functional diagnosis flow. This novel flow will be based on generating a fault dictionary which contains a set of functional tests able to distinguish between defects. This fault dictionary approach will result in a list of candidate inter-cell defects which, thereafter, can be pruned by applying the ED flow for intra-cell defects.

Another important aspect is related to the system-level. Thus, another research perspective is exploiting the information extracted during the IP block diagnosis to develop a system-level ED. The main idea is to design and embed in to the SoC dedicated circuitry to guide the test engineer during the diagnosis. The goal is to facilitate the localization of the faulty block(s). The required inputs by the system-level ED are the knowledge of the System under Diagnosis (SuD) and the test infrastructure already present on-chip to support semiconductor testing. Examples of infrastructure to be reused for the purpose of ED are scan-paths, Built-In Self-Test (BIST) and Design-for-Test (DfT) facilities, pattern compression/decompression schemes, on-chip sensors, signal buses, etc. The reusability requires some standardization, which is already observed, for instance,

in the shift towards IJTAG or IEEE 1500. When necessary, additional test instruments will be embedded to enhance diagnostic information paying attention that such instruments must be non-intrusive, transparent, and incur low-overhead. The output of the system-level ED is the suspect IP block or the interconnection between IP blocks that is responsible for system failure. Thereafter, the appropriate IP block-level ED procedure will be executed. Each ED procedure must be able to identify the root cause of observed failures at transistor-level or rate defects according to their probability of occurrence.

3.3 Long-term Perspectives

The large diffusion of portable devices (i.e., smartphone, tablet, ...) and the continuous scale down of technology require the production of Integrated Circuits (ICs) having an efficient management of energy consumption. The latter is of paramount importance for battery handheld devices. The autonomy and lifespan of such devices directly depends on procedures aimed at saving energy. Achieving high-energy efficiency for these components, while maintaining a high degree of performances, has been a major issue in low power integrated circuits from a hardware design standpoint. Energy savings can also be obtained using a software-based control. Most typically, energy hungry components are switched off during idle times.

The overall IC power consumption has two contributors: (i) static and (ii) dynamic power. The static power only depends on the leakage currents. In order to reduce the static power consumption the so-called power-gating techniques have been proposed so far in the literature. The basic idea behind power-gating relies on splitting the IC into different blocks called power islands. Each power island can be switched on/off independently of the others. Using this technique, if one or more power islands are not used by the running application, they are switched off, in order to reduce the static power consumption. Power-gating is usually implemented by means of power switches.

In this context, we will target the dynamic power that is strictly related to the running application. In the literature several techniques have been already presented. The so-called Adaptive Voltage Scaling (AVS) technique aims at modifying the supply voltage level of an IC in order to reduce as much as possible the power consumption for a given running application. Actually, it exists an “off-line” AVS technique where the required supplied voltage level is pre-determined during characterization/production phase. In this case, the applied voltage level is usually determined by using the worst-case scenario that could be far from the optimum. Moreover, it is limited by strictly chosen set of applications. On the other hand, the “on-line” AVS technique aims at automatic tuning the supply voltage level. In the literature can be found several solutions based on

"power" sensors, where depending on the measured value, the Power Management Unit (PMU) can increase or decrease the supplied voltage level. Main limitations of existing techniques represent the own power sensor because (i) it does not allow catch the local voltage variations caused by currently running application(s) and (ii) it measures only an averaged value of power. Moreover, the measured value has to be compared to the threshold one that is again obtained in "off-line" mode based on applied set of chosen applications. Another existing solution is based on the use "performance" sensors in order to understand if the performance of the current application is good enough or not. Existing performance sensors represent Ring oscillators and Canary circuits. However, they might fail to capture within-die variations that are local to real circuits such as random manufacturing variations and circuit aging.

The main goal relies on the development of meaningful architecture and design guidelines for tuning the supply voltage level of an IC. We will study further energy savings that can be obtained by considering different performance modes and controlling the required power supply. Each performance mode of a Device Under Control (DUC) has a different power consumption associated with it. The implementation of such an approach is not straightforward. A major challenge is to guarantee the performance level during each DUC mode by controlling the power supply during all the operational life of the device. We will study a closed loop control scheme aiming to adjust the power consumption for integrated DUC according to the required performance mode. Since the DUC performances are not directly available for measurement, the idea is to estimate them using stored regression functions. The estimation inputs are signals coming from embedded jitter based power noise sensors that have been developed in the previous works.

The main advantage w.r.t. the state-of-the-art will be (i) the absence of ring oscillators and/or canary circuits, (ii) the absence of power sensor(s) that will be replaced by power supply variations detector (PSVD) and finally (iii) removing all off-line characterization procedures. Our goal is to be sensitive as much as possible to the running application, to the in die- variations, circuit aging and any random manufacturing variations that is not the case for the existing solutions. To meet our goal, we propose to develop architecture composed of (i) performance and PSVD sensors and a (ii) control unit (CU) as shown the Figure 3.4.

The first sensor represents a PSVD able to carefully determine the local voltage supply variations on measured IP. For this purpose, it uses the jitter measurement of distributed clock signal. The clock signal is propagated across clock tree composed of series of buffers that are sensitive to the voltage supply variations. In order to determine the local voltage variations on measured IP, the set of paths from distributed network of clock tree have

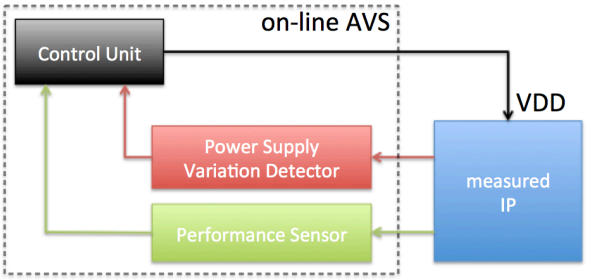


FIGURE 3.4: Proposed Principle of Power Aware Auto-Adaptive Mechanism.

to be chosen. Dependent on the "stress" level of measured IP (i.e., caused by applying various applications), the voltage supply variations on power grid in the area closed to the choosing paths are changing proportionally with jitter of propagated clock signal through given path. Thanks this dependency, by simple measuring of jitter on chosen clock path, the voltage supply variations can be determined.

The second one is the performance sensor. Actually this sensor will be the responsible of the measure of the performance quality achieved by the IC under a given voltage supply value. As the target is to avoid any special circuitry (i.e., ring oscillator/ canary circuits), we propose to modify the architecture of the IC to embed the sensor, so that it will be sensitive to the IC variations (i.e., process variations, aging effects and so on). Basically, when the supplied voltage level is lowered by the CU, we have to understand if the IC can still execute correctly the application or it requires a higher level of supplied voltage. However, one consequence could be that the application may fail (i.e., it produces a wrong output) due to the too low supplied voltage level. To tackle this problem we propose to investigate the impact of re-using error detection circuitry. In this case, failures due to a too low voltage level will be detected and thus the CU can react by increasing the voltage level. To avoid the re-execution of the whole application, we have to implement roll back mechanism able to restore a safe state of the current application.

Finally, the CU closes the loop and it provides the final voltage supply to the measured IP. It takes as the input the data from both sensors (i.e., performance sensor and PSVD) in form of signatures and it calculates the error (i) for performance and after for (ii) voltage supplies variations. If performance is too much degraded and the measured IP is not working properly, the roll back mechanism on elevated voltage supply is activated and context of currently running application is restored. However, if performance is degraded but measured IP is still working properly, the PSVD will adapt the voltage supply level in order to provide the requested performance.

To conclude, this kind of objectives require several skills to success: analog design, digital design, fault tolerance, correction and recovery mechanisms.

Chapter 4

Publications

4.1 Books

B1 A. Bosio, L. Dilillo, P. Girard, A. Virazel, S. Pravossoudovitch, "Advanced Test Methods for SRAMs Effective Solutions for Dynamic Fault Detection in Nanoscale Technologies", Springer, 2009.

4.2 Patents

P2 M. Valka, A. Bosio, P. Debaud, P. Girard, S. Guilhot, "Adaptive voltage scaling mechanism based on voltage shoot measurement", In WO Patent App. PCT/EP2013/066,669, 2014.

P1 M. Valka, A. Bosio, M. Broutin, P. Debaud, P. Girard, S. Guilhot, "Efficient Power Supply Noise Measurement Based on Timing Uncertainty", In WO Patent App. PCT/EP2013/066,668, 2014.

4.3 Journals

J11 L. H. Bonet Zordan, A. Bosio, L. Dilillo, P. Girard, A. Virazel, N. Badereddine, "On the Test and Mitigation of Malfunctions in Low-Power SRAMs", In Journal of Electronic Testing, Springer US, vol. 30, no. 5, pp. 611-627, 2014.

J10 Z. Sun, A. Bosio, L. Dilillo, P. Girard, S. Pravossoudovich, A. Virazel, E. Auvray, "Intra-Cell Defects Diagnosis", In Journal of Electronic Testing, Springer US, vol. 30, no. 5, pp. 541-555, 2014.

J9 P. Bernardi, M. De Carvalho, E. Sanchez, M. Sonza Reorda, A. Bosio, L. Dilillo, M. Valka, P. Girard, "Fast Power Evaluation for Effective Generation of Test Programs Maximizing Peak Power Consumption", In ASP article of Low Power Electronics, vol. 9, no. 2, pp. 253-263, 2013.

J8 A. Todri, A. Bosio, P. Girard, A. Virazel, "Uncorrelated Power Supply Noise and Ground Bounce Consideration for Test Pattern Generation", In IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 21, no. 5, pp. 958-970, 2013.

J7 A. Savino, S. Di Carlo, G. Politano, A. Benso, A. Bosio, G. Di Natale, "Statistical Reliability Estimation of Microprocessor-Based Systems", In IEEE Transactions on Computers, vol. 61, no. 11, pp. 1521-1534, 2012.

J6 F. Wu, L. Dilillo, A. Bosio, P. Girard, S. Pravossoudovitch, A. Virazel, M. Tehranipoor, X. Wen, N. Ahmed, "A Comprehensive Analysis of Transition Fault Coverage and Test Power Dissipation for LOS and LOC Schemes", In ASP article of Low Power Electronics, vol. 6, no. 2, pp. 359-374, 2010.

J5 A. Bosio, P. Girard, S. Pravossoudovitch, A. Virazel, "A Comprehensive Framework for Logic Diagnosis of Arbitrary Defects", In IEEE Transactions on Computers, vol. 59, no. 3, pp. 289-300, 2010.

J4 A. Benso, A. Bosio, S. Di Carlo, P. Prinetto, "Are IEEE-1500-Compliant Cores Really Compliant to the Standard?", In IEEE Design Test of Computers, vol. 26, no. 3, pp. 16-24, 2009.

J3 A. Benso, A. Bosio, S. Di Carlo, G. Di Natale, P. Prinetto, "March Test Generation Revealed", In IEEE Transactions on Computers, vol. 57, no. 12, pp. 1704-1713, 2008.

J2 A. Bosio, G. Di Natale, "March Test BDN, a new March Test for Dynamic Faults", In article of Control Engineering and Applied Informatics, vol. 10, no. 2, pp. 3-9, 2008.

J1 A. Bosio, S. Di Carlo, G. Di Natale, P. Prinetto, "March AB, a state-of-the-art march test for realistic static linked faults and dynamic faults in SRAMs", In IET Computers Digital Techniques, vol. 1, no. 3, pp. 237-245, 2007.

4.4 Invited Papers

I4 A. Bosio, L. Dilillo, P. Girard, A. Todri, A. Virazel, "Why and How Controlling Power Consumption during Test: A Survey", In IEEE Asian Test Symposium (ATS), pp. 221-226, 2012.

I3 L. B. Zordan, A. Bosio, L. Dilillo, P. Girard, S. Pravossoudovitch, A. Todri, A. Virazel, N. Badereddine, "Failure Analysis and Test Solutions for Low-Power SRAMs", In IEEE Asian Test Symposium (ATS), pp. 459-460, 2011.

I2 A. Bosio, L. Dilillo, P. Girard, A. Todri, A. Virazel, K. Miyase, X. Wen, "Power-Aware Test Pattern Generation for At-Speed LOS Testing", In IEEE Asian Test Symposium (ATS), pp. 506-510, 2011.

I1 A. Benso, A. Bosio, P. Prinetto, A. Savino, "A Black-Box-Oriented Test Methodology", In IEEE East-West Design & Test Workshop (EWDTW), pp. 11-15, 2006.

4.5 Conferences

C47 Z. Sun, A. Bosio, L. Dilillo, P. Girard, A. Virazel, E. Auvray, "On the Generation of Diagnostic Test Set for Intra-cell Defects", In 2014 IEEE 23rd Asian Test Symposium (ATS), pp. 312-317, 2014.

C46 M. Valka, A. Bosio, L. Dilillo, P. Girard, A. Todri, A. Virazel, P. Debaud, S. Guilhot, "iBoX: Jitter based Power Supply Noise sensor", In 2014 19th IEEE European Test Symposium (ETS), pp. 1-2, 2014.

C45 M. Valka, A. Bosio, L. Dilillo, P. Girard, A. Todri, A. Virazel, P. Debaud, S. Guilhot, "Test and diagnosis of power switches", In IEEE 17th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS), pp. 213-218, 2014.

C44 S. Bernabovi, P. Bernardi, A. Bosio, L. Dilillo, P. Girard, A. Todri, A. Virazel, "An intra-cell defect grading tool", In IEEE 17th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS), pp. 298-301, 2014.

C43 L.B. Zordan, A. Bosio, L. Dilillo, P. Girard, A. Todri, A. Virazel, N. Badereddine, "A built-in scheme for testing and repairing voltage regulators of low-power srams", In IEEE VLSI Test Symposium (VTS), pp. 1-6, 2013.

C42 S. Zhenzhou, A. Bosio, L. Dilillo, P. Girard, A. Todri, A. Virazel, E. Auvray, "Effect-cause intra-cell diagnosis at transistor level", In International Symposium on Quality Electronic Design (ISQED), pp. 460-467, 2013.

C41 L.B. Zordan, A. Bosio, L. Dilillo, P. Girard, A. Todri, A. Virazel, N. Badereddine, "Test solution for data retention faults in low-power SRAMs", In Design, Automation Test in Europe Inproceedings Exhibition (DATE), pp. 442-447, 2013.

C40 L.B. Zordan, A. Bosio, L. Dilillo, P. Girard, A. Todri, A. Virazel, N. Badereddine, "On the reuse of read and write assist circuits to improve test efficiency in low-power SRAMs", In IEEE International Test Inproceedings (ITC), pp. 1-10, 2013.

C39 P. Bernardi, M. De Carvalho, E. Sanchez, M.S. Reorda, A. Bosio, L. Dilillo, P. Girard, M. Valka, "Peak Power Estimation: A Case Study on CPU Cores", In IEEE Asian Test Symposium (ATS), pp. 167-172, 2012.

C38 M. Valka, A. Bosio, L. Dilillo, P. Girard, A. Todri, A. Virazel, P. Debaud, S. Guilhot, "Power Supply Noise Sensor Based on Timing Uncertainty Measurements", In IEEE Asian Test Symposium (ATS), pp. 161-166, 2012.

C37 L.B. Zordan, A. Bosio, L. Dilillo, P. Girard, A. Todri, A. Virazel, N. Badereddine, "Low-power SRAMs power mode control logic: Failure analysis and test solutions", In IEEE International Test Inproceedings (ITC), pp. 1-10, 2012.

C36 L.B. Zordan, A. Bosio, L. Dilillo, P. Girard, A. Todri, A. Virazel, N. Badereddine, "Defect analysis in power mode control logic of low-power SRAMs", In IEEE European Test Symposium (ETS), pp. 1-1, 2012.

C35 S. Zhenzhou, A. Bosio, L. Dilillo, P. Girard, A. Todri, A. Virazel, E. Auvray, "Fault Localization Improvement through an Intra-Cell Diagnosis Approach", In International Symposium for Testing and Failure Analysis, pp. 509-519, 2012.

C34 K. Miyase, Y. Uchinodan, K. Enokimoto, Y. Yamato, X. Wen, S. Kajihara, F. Wu, L. Dilillo, A. Bosio, P. Girard, A. Virazel, "Effective Launch-to-Capture Power Reduction for LOS Scheme with Adjacent-Probability-Based X-Filling", In IEEE Asian Test Symposium (ATS), pp. 90-95, 2011.

C33 L. Dilillo, A. Bosio, M. Valka, P. Girard, S. Pravossoudovitch, A. Virazel, "Error Resilient Infrastructure for Data Transfer in a Distributed Neutron Detector", In IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), pp. 294-301, 2011.

C32 P. Bernardi, M.S. Reorda, A. Bosio, P. Girard, S. Pravossoudovitch, "On the Modeling of Gate Delay Faults by Means of Transition Delay Faults", In IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), pp. 226-232, 2011.

C31 A. Todri, A. Bosio, L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, "Power supply noise and ground bounce aware pattern generation for delay testing", In IEEE International New Circuits and Systems Inproceedings (NEWCAS), pp. 73-76, 2011.

C30 M. Valka, A. Bosio, L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, E. Sanchez, M. De Carvalho, M.S. Reorda, "A Functional Power Evaluation Flow for Defining Test Power Limits during At-Speed Delay Testing", In IEEE European Test Symposium (ETS), pp. 153-158, 2011.

C29 A. Todri, A. Bosio, L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, "A study of path delay variations in the presence of uncorrelated power and ground supply noise", In IEEE International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS), pp. 189-194, 2011.

C28 L.B. Zordan, A. Bosio, L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, N. Badereddine, "Optimized march test flow for detecting memory faults in SRAM devices under bit line coupling", In Design and Diagnostics of Electronic Circuits Systems (DDECS), 2011 IEEE 14th International Symposium on, pp. 353-358, 2011.

C27 F. Wu, L. Dilillo, A. Bosio, P. Girard, S. Pravossoudovitch, A. Virazel, M. Tehranipoor, K. Miyase, X. Wen, N. Ahmed, "Power reduction through X-filling of transition fault test vectors for LOS testing", In International Inproceedings on Design Technology of Integrated Systems in Nanoscale Era (DTIS), pp. 1-6, 2011.

C26 P. Rech, A. Bosio, P. Girard, S. Pravossoudovitch, A. Virazel, L. Dilillo, "A Memory Fault Simulator for Radiation-Induced Effects in SRAMs", In IEEE Asian Test Symposium (ATS), pp. 100-105, 2010.

C25 Y. Benabboud, A. Bosio, L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, O. Riewer, "A Comprehensive System-on-Chip Logic Diagnosis", In IEEE Asian Test Symposium (ATS), pp. 237-242, 2010.

C24 F. Wu, L. Dilillo, A. Bosio, P. Girard, S. Pravossoudovitch, A. Virazel, M. Tehranipoor, K. Miyase, X. Wen, N. Ahmed, "Is test power reduction through X-filling good enough?", In IEEE International Test Inproceedings (ITC), pp. 1-1, 2010.

C23 F. Wu, L. Dilillo, A. Bosio, P. Girard, S. Pravossoudovitch, A. Virazel, J. Ma, W. Zhao, M. Tehranipoor, X. Wen, "Analysis of power consumption and transition fault coverage for LOS and LOC testing schemes", In IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS), pp. 376-381, 2010.

C22 A. Bosio, P. Girard, S. Pravossoudovitch, P. Bernardi, M.S. Reorda, "An Exact and Efficient Critical Path Tracing Algorithm", In IEEE International Symposium on Electronic Design, Test and Application (DELTA), pp. 164-169, 2010.

C21 Y. Benabboud, A. Bosio, L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, O. Riewer, "Delay Fault Diagnosis in Sequential Circuits", In IEEE Asian Test Symposium (ATS), pp. 355-360, 2009.

C20 A. Bosio, P. Girard, S. Pravossoudovich, P. Bernardi, M.S. Reorda, "An efficient fault simulation technique for transition faults in non-scan sequential circuits", In International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS), pp. 50-55, 2009.

C19 Y. Benabboud, A. Bosio, L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, L. Bouzaida, I. Izaute, "Comprehensive bridging fault diagnosis based on the SLAT paradigm", In International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS), pp. 264-269, 2009.

C18 Y. Benabboud, A. Bosio, L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, L. Bouzaida, I. Izaute, "A fault-simulation-based approach for logic diagnosis", In International Inproceedings on Design Technology of Integrated Systems in Nanoscal Era (DTIS), pp. 216-222, 2009.

C17 Y. Benabboud, A. Bosio, L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, L. Bouzaida, I. Izaute, "A case study on logic diagnosis for System-on-Chip", In International Symposium on Quality Electronic Design (ISQED), pp. 253-259, 2009.

C16 A. Bosio, G. Di Natale, "LIFTING: A Flexible Open-Source Fault Simulator", In IEEE Asian Test Symposium (ATS), pp. 35-40, 2008.

C15 P. Oehler, A. Bosio, G. Di Natale, S. Hellebrand, "A Modular Memory BIST for Optimized Memory Repair", In IEEE International On-Line Testing Symposium (IOLTS), pp. 171-172, 2008.

C14 A. Ney, A. Bosio, L. Dilillo, G. Girard, S. Pravossoudovitch, A. Virazel, "A signature-based approach for diagnosis of dynamic faults in SRAMs", In International Inproceedings on Design and Technology of Integrated Systems in Nanoscale Era (DTIS), pp. 1-6, 2008.

C13 A. Rousset, A. Bosio, P. Girard, C. Landrault, S. Pravossoudovitch, A. Virazel, "Improving Diagnosis Resolution without Physical Information", In IEEE International Symposium on Electronic Design, Test and Applications (DELTA), pp. 210-215, 2008.

C12 A. Rousset, A. Bosio, P. Girard, C. Landrault, S. Pravossoudovitch, A. Virazel, "Fast Bridging Fault Diagnosis using Logic Information", In IEEE Asian Test Symposium (ATS), pp. 33-38, 2007.

C11 A. Rousset, A. Bosio, P. Girard, C. Landrault, S. Pravossoudovitch, A. Virazel, "DERRIC: A Tool for Unified Logic Diagnosis", In IEEE European Test Symposium (ETS), pp. 13-20, 2007.

C10 A. Rousset, A. Bosio, P. Girard, C. Landrault, S. Pravossoudovitch, A. Virazel, "A Mixed Approach for Unified Logic Diagnosis", In IEEE Design and Diagnostics of Electronic Circuits and Systems (DDECS), pp. 1-4, 2007.

C9 A. Benso, A. Bosio, S. Di Carlo, R. Mariani, "A Functional Verification based Fault Injection Environment", In IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT), pp. 114-122, 2007.

C8 A. Benso, A. Bosio, S. Di Carlo, G. Di Natale, P. Prinetto, "Memory Fault Simulator for Static-Linked Faults", In IEEE Asian Test Symposium (ATS), pp. 31-36, 2006.

C7 A. Benso, A. Bosio, S. Di Carlo, G. Di Natale, P. Prinetto, "ATPG for Dynamic Burn-In Test in Full-Scan Circuits", In IEEE Asian Test Symposium (ATS), pp. 75-82, 2006.

C6 A. Benso, A. Bosio, S. Di Carlo, G. Di Natale, P. Prinetto, "A 22n March Test for Realistic Static Linked Faults in SRAMs", In IEEE European Test Symposium (ETS), pp. 49-54, 2006.

C5 A. Benso, A. Bosio, S. Di Carlo, G. Di Natale, P. Prinetto, "A Unique March Test Algorithm for the Wide Spread of Realistic Memory Faults in SRAMs", In IEEE Design and Diagnostics of Electronic Circuits and systems (DDECS), pp. 155-156, 2006.

C4 A. Benso, A. Bosio, S. Di Carlo, G. Di Natale, P. Prinetto, "Automatic March Tests Generations for Static Linked Faults in SRAMs", In Design, Automation and Test in Europe (DATE), vol. 1, pp. 1-6, 2006.

C3 A. Benso, A. Bosio, P. Prinetto, A. Savino, "An on-line software-based self-test framework for microprocessor cores", In International Inproceedings on Design and Test of Integrated Systems in Nanoscale Technology (DTIS), pp. 394-399, 2006.

C2 A. Benso, A. Bosio, S. Di Carlo, G. Di Natale, P. Prinetto, "March AB, March AB1: new March tests for unlinked dynamic memory faults", In IEEE International Test Inproceedings (ITC), pp. 8 pp.-841, 2005.

C1 A. Benso, A. Bosio, S. Di Carlo, G. Di Natale, P. Prinetto, "Automatic March tests generation for static and dynamic faults in SRAMs", In IEEE European Test Symposium (ETS), pp. 122-127, 2005.

4.6 Workshop

W4 A. Touati, A. Bosio, L. Dilillo, P. Girard, A. Todri, A. Virazel, P. Bernardi, "A Comprehensive Evaluation of Functional Programs for Power-Aware Test", In 2014 IEEE 23rd North Atlantic Test Workshop (NATW), pp. 69-72, 2014.

W3 A. Bosio, P. Girard, S. Pravossoudovich, P. Bernardi, "SoC Symbolic Simulation: a case study on delay fault testing", In IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS), pp. 1-6, 2008.

W2 A. Benso, S. Di Carlo, P. Prinetto, A. Bosio, "Automating the IEEE std. 1500 compliance verification for embedded cores", In IEEE International High Level Design Validation and Test Workshop (HLVDT), pp. 171-178, 2007.

W1 A. Benso, A. Bosio, S. Di Carlo, G. Di Natale, P. Prinetto, "Automatic March tests generation for multi-port SRAMs", In IEEE International Workshop on Electronic Design, Test and Applications (DELTA), 2006.

Chapter 5

Selection of 5 best papers

J8 A. Todri, A. Bosio, P. Girard, A. Virazel, "Uncorrelated Power Supply Noise and Ground Bounce Consideration for Test Pattern Generation", In IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 21, no. 5, pp. 958-970, 2013.

J7 A. Savino, S. Di Carlo, G. Politano, A. Benso, A. Bosio, G. Di Natale, "Statistical Reliability Estimation of Microprocessor-Based Systems", In IEEE Transactions on Computers, vol. 61, no. 11, pp. 1521-1534, 2012.

J5 A. Bosio, P. Girard, S. Pravossoudovitch, A. Virazel, "A Comprehensive Framework for Logic Diagnosis of Arbitrary Defects", In IEEE Transactions on Computers, vol. 59, no. 3, pp. 289-300, 2010.

J4 A. Benso, A. Bosio, S. Di Carlo, P. Prinetto, "Are IEEE-1500-Compliant Cores Really Compliant to the Standard?", In IEEE Design Test of Computers, vol. 26, no. 3, pp. 16-24, 2009.

J3 A. Benso, A. Bosio, S. Di Carlo, G. Di Natale, P. Prinetto, "March Test Generation Revealed", In IEEE Transactions on Computers, vol. 57, no. 12, pp. 1704-1713, 2008.

Uncorrelated Power Supply Noise and Ground Bounce Consideration for Test Pattern Generation

Aida Todri, *Member, IEEE*, Alberto Bosio, *Member, IEEE*, Luigi Dilillo, *Member, IEEE*, Patrick Girard, *Senior Member, IEEE*, and Arnaud Virazel, *Member, IEEE*

***Abstract*— Power supply noise and ground bounce can cause considerable path delay variations. Capturing the worst case power supply noise at a gate level is not a sufficient indicator for measuring the worst case path delay. Furthermore, path delay variations depend on multiple parameters such as input stimuli, cell placement, switching frequency, and available decoupling capacitors. All these variables obscure the rapport between supply noise and path delay and make the selection of stimuli for worst case path delay a difficult task during test pattern generation. In this paper, we utilize power supply noise and ground bounce distribution along with physical design data to generate test patterns for capturing worst case path delay. We propose accurate close-form mathematical models for capturing the effect of power supply noise and ground bounce on path delay. These models are based on modified nodal analysis formulation of power and ground networks, where current waveforms are obtained from levelized simulation and cell library characterization. The proposed test pattern generation flow is a simulated-annealing-based iterative process, which utilizes mathematical models for capturing the impact of supply noise on path delay for a given input pattern. We perform experiments on ITC'99 benchmarks and show that path delay variation can be considerable if test patterns are not properly selected.**

***Index Terms*— Automatic test pattern generation (ATPG), deep submicrometer, delay test, ground bounce, pattern selection, power supply noise, timing analysis.**

I. Introduction

THE ONGOING miniaturization of circuits at the nanometer regime has introduced significant changes on the device's parasitics and behavior. Circuit densities increase with each nanotechnology generation because of smaller devices and larger dies, and, consequently, current density and total current consumption increase accordingly. Simultaneously, circuits with high switching frequencies impose faster current transients on power and ground distribution networks. Transient currents increase exponentially with each technology node and cause significant deviations on the voltage distribution. Such deviations of the voltage levels from their nominal values are referred to as "power supply noise and ground bounce." Both these conditions are undesirable, as they significantly impact signal propagation. Analysis shows that power supply noise and ground bounce can considerably affect circuit's performance [1]. Furthermore, simulations show that delay can have a speed-up/slow-down effect depending on the noise conditions on the neighboring gates and/or the crosstalk between gates as shown, respectively, by [2] and [3]. We consider the *uncorrelated* behavior of power supply noise and ground bounce (independent noise peaks and frequencies) in order to represent them as realistically as they would occur in an actual design. Gates can be placed in different locations on chip and they do not experience the same power or ground noise due to temporal and spatial switching. Also, power and ground parasitics for each cell can vary because of their proximity to the nearest power and ground pins. Moreover, as all gates share the same power and ground network, there is also noise transfer that occurs from one region to its neighboring regions, which can cause further delay variations. Another important factor that leads to uncorrelated noise is the amount of decoupling capacitance available at a given region. In general, decoupling capacitors are not evenly distributed, resulting in different amounts of generated noise. Owing to the aforementioned reasons, we treat power supply noise and ground bounce as uncorrelated.

Traditionally, the impact of power supply noise on delay was considered at the cell library development step where each cell was characterized for the worst case voltage drop. Such approach assumes that all cells experience the worst case voltage drop, which is unrealistic. Several other approaches have been proposed in the literature which can be grouped into two main areas: 1) power supply noise aware timing analysis methods and 2) power supply noise aware test pattern generation. In the first group, there has been a substantial amount of work on how to estimate power supply noise-induced worst case delay, notably [4]–[9]. In [4], the authors propose a method to compute the upper bound of circuit delay under voltage variations. A vectorless approach is presented in [5] to estimate the maximum delay under power supply noise, and a delay maximization problem is formulated as an optimization problem. Similarly, the authors in [6]–[9] provide a worst case delay analysis taking into account power supply variations. In the second group of works, such as [10]–[14], the authors propose different techniques for test pattern generation while considering the impact of power supply noise. These works target critical path delay maximization under power supply noise while maximizing switching activity using approaches based either on the Monte Carlo

Manuscript received September 6, 2011; revised January 19, 2012; accepted April 12, 2012. Date of publication June 6, 2012; date of current version April 22, 2013.

The authors are with the Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier (LIRMM UMR 5506), Montpellier 34095, France (e-mail: todri@lirmm.fr; bosio@lirmm.fr; dilillo@lirmm.fr; girard@lirmm.fr; virazel@lirmm.fr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2012.2197427

method or on genetic algorithms. These existing delay-testing and timing-analysis techniques capture worst case timing scenarios which might not reflect the worst case circuit delay. This is due to the following: 1) the model is based on simplified logic-level delay fault models, where physical design information such as the $\{R, L, C\}$ parasitics of the circuits, package, power/ground network, and available decoupling capacitor information are ignored; 2) the combined and uncorrelated impact of power supply noise and ground bounce is not considered which can lead to either delay speedup or slowdown; and 3) impact of resonance frequency on path delay is ignored. Power supply noise and ground bounce in the range of resonance frequencies have been shown as the dominant noise component for high-performance microprocessors [15]. For the reasons mentioned above, we believe that test pattern generation in presence of supply noise deserves reexamination and an effort to understand and capture the interdependencies among path delay variations and noise conditions.

In this paper, we propose a pattern generation technique that takes into account combined effect of power supply noise and ground bounce on path delay as a function of applied inputs. Noise impact on delay is highly dependent on the applied input patterns. We provide mathematical models to represent the circuit based on physical extracted data after it has been placed and routed with power/ground grids. We propose close-form mathematical models to capture the impact of input patterns on path delay in the presence of power supply noise and ground bounce. We use a simulated annealing (SA)-based approach to find patterns that maximize critical path delay. Our method generates patterns to cause such power supply noise and ground bounce distribution that leads to maximum path delay. The contributions of this paper are summarized as follows.

1) We propose accurate and close-form mathematical models to derive the impact of input test patterns on path delay in the presence of noise.
2) We propose a path delay calculation method that takes into account the amount of noise on neighboring cells and switching frequency.
3) We propose a test pattern generation flow that takes into account circuit physical design data (i.e., parasitics, pad/pin location, and cell placements) and speed-up/slow-down effects of noise on path delay.
4) The proposed technique is versatile and can be utilized for delay testing and/or timing analysis techniques.

Furthermore, in contrast to previous works which initially aimed to find patterns for maximum supply noise and then compute delay, our method targets directly to find the worst case delay which might not necessarily occur under worst case power supply noise due to path delay speed-up/slow-down phenomena from the noise conditions on neighboring gates.

The rest of this paper is organized as follows. A motivational example is presented in Section II. The delay model considering power supply noise and ground bounce is presented in Section III. In Section IV, we present our test pattern generation flow in the presence of power supply noise and ground bounce. Experimental results are presented in Section V. We conclude this paper in Section VI.

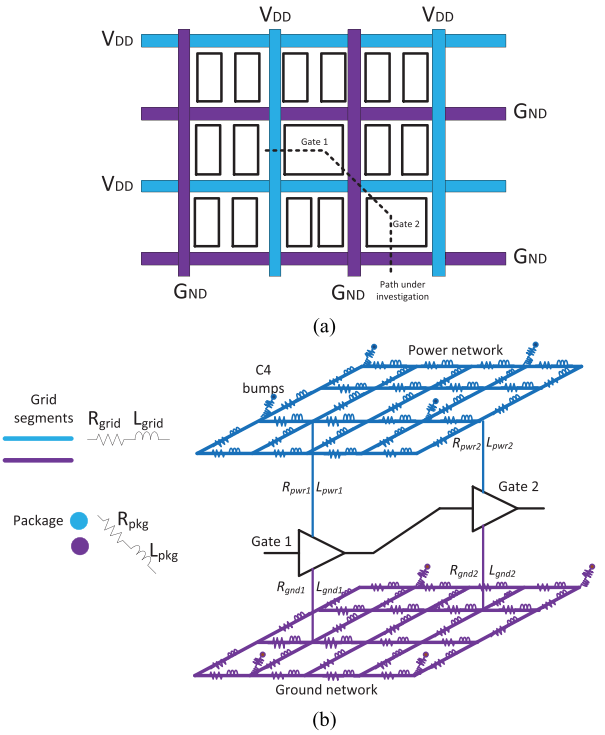


Fig. 1. (a) Illustration of gate placement on chip and (b) representative model for the two-stage buffer circuit used for path delay analysis in the presence of power supply noise and ground bounce.

II. Motivational Example

Power supply noise and ground bounce can cause path delay variations. To highlight the impact of power supply noise and ground bounce on path delay, we provide the analysis of a sample circuit as shown in Fig. 1. A similar analysis was performed in [3] and [5], but we extend such analysis on mesh networks along with decoupling capacitors for capturing the impact of supply noise and resonance frequency on the path delay.

In this paper, we consider on-die power and ground networks along with controlled-collapse chip-connection (C4) package bumps, on-chip decoupling capacitors, and switching circuits. Printed circuit board parasitics are not considered and are beyond the scope of this paper. The sample circuit is a two-stage buffer chain implemented in 90 nm with $V_{DD} = 1$ V. As shown in Fig. 1(a), the buffer gates share the same global power and ground networks, however, they can be placed in different locations and proximities from the power and ground pins. Fig. 1(b) shows the circuit model which we utilize for our analysis. Power and ground networks are represented with their extracted parasitics of resistance R, capacitance C, and self-inductance L. We ignore mutual inductances. The extracted values are based on the dimensions of power/ground tracks as used in [16]. We include package parasitics represented by extracted R and L values as described in [17].

TABLE I
IMPACT OF VOLTAGE DROP ONLY ON DELAY

	Gate 1	Gate 2	Delay Ratio	% Impact
No ground bounce	-0.1V (voltage overshoot)	-0.1V	0.88	23% speed-up
		0V	1.014	1.4% slow-down
		0.1V	1.2	20% speed-up
	0V (no voltage droop)	-0.1V	0.86	14% speed-up
		0V	1	Nominal delay
		0.1V	1.18	18% slow-down
	0.1V (voltage droop)	-0.1V	0.83	17% speed-up
		0V	0.95	5% speed-up
		0.1V	1.16	16% slow-down

TABLE II
IMPACT OF GROUND BOUNCE ONLY ON DELAY

	Gate 1	Gate 2	Delay Ratio	% Impact
No power supply noise	No ground bounce	No ground bounce	1	Nominal delay
	No ground bounce	0.1V ground bounce	0.98	0.2% speed-up
	0.1V ground bounce	No ground bounce	1.074	7.4% slow-down
	0.1V ground bounce	0.1V ground bounce	1.042	4.2% slow-down

We perform HSPICE transient analysis on the sample circuit and measure delay variations as a function of power supply noise and ground bounce. In the following subsections, we report on path delay variations by performing: 1) stand-alone power supply noise analysis; 2) stand-alone ground bounce analysis; and 3) combined power and ground noise analysis with respect to resonance frequency.

A. PSN Impact on Delay Variations

In this experiment, to capture the impact of power supply noise only, the ground network is considered ideal. Delay is plotted as a function of measured maximum power supply noise (represented as voltage drop) on each gate, as shown in Fig. 2. Delay variations are plotted as delay ratios with respect to nominal delay with no noise on the circuit. Negative (positive) values on the x- and y-axis present the measured voltage overshoot (undershoot) from V_{DD}. Table I presents the percentages of delay variations. Two observations from this experiment can be made. 1) Depending on the noise conditions on each gate, path delay can increase/decrease. 2) The worst case voltage droop on both gates does not lead to worst path delay.

B. Ground Bounce Impact on Delay Variations

The same circuit is used to analyze the impact of ground bounce on delay. Fig. 3 shows the path delay map as a function of measured ground bounce on each gate. Table II shows the percentages of path delay variations. The main observation from this analysis is that the worst path delay does not occur when both gates experience worst case ground bounce. As shown in Fig. 3, the worst case path delay is on lower right-hand corner of the map, when gate 1 has the largest ground bounce and gate 2 has no ground bounce.

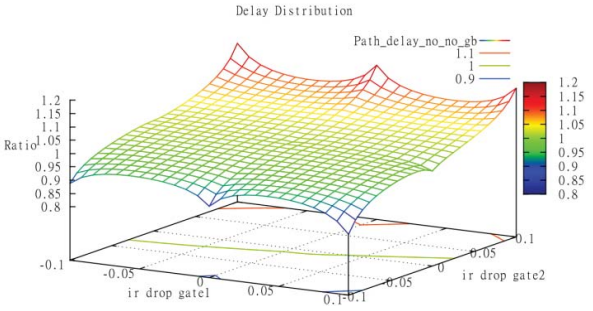


Fig. 2. Path delay variation in presence of power supply noise.

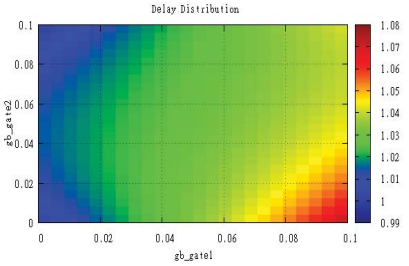


Fig. 3. Path delay map as a function of ground bounce only.

C. Uncorrelated PSN and Ground Bounce Impact on Path Delay

In this experiment, we perform path delay analysis with both power supply noise and ground bounce. We use the same two-buffer circuit. Figs. 4 and 5 show different path delay variations with respect to power supply noise and ground bounce. The delays are represented as ratios with respect to nominal delay with no noise.

Path delay variations are plotted for four cases: 1) both gates have no ground bounce [Fig. 4(a)]; 2) only gate 2 experiences ground bounce [Fig. 4(b)]; 3) only gate 1 experiences ground bounce [Fig. 4(c)]; and 4) both gates experience ground bounce [Fig. 4(d)]. We observe that considering uncorrelated power and ground noise introduces further delay variations. For example, there is a decrease on path delay when gate 2 suffers from ground bounce versus the case when both gates have no ground bounce as shown in Fig. 5(a). In Fig. 5(a), there are two delay distribution layers where one layer shows delay distribution with no ground bounce on both gates (red layer) and the other layer shows delay distribution with ground bounce on gate 2 only (green layer). Their overlap shows the delay speed-up effect that occurs when gate 2 has ground bounce. In the case when only gate 1 has ground bounce, there is a slow-down effect as shown in Fig. 5(b).

We repeat the above experiments with varying input signal switching frequency between 150 MHz to 1 GHz in order to capture path delay variations with resonance frequency. Resonance frequency on chip is created due to large package inductance L and on-chip capacitance C, which together create a series LC tank. The LC tank creates an oscillator where energy is being transferred between the inductance and capacitor leading to excessive voltage harmonics on power and ground networks. Moreover, as power and ground networks

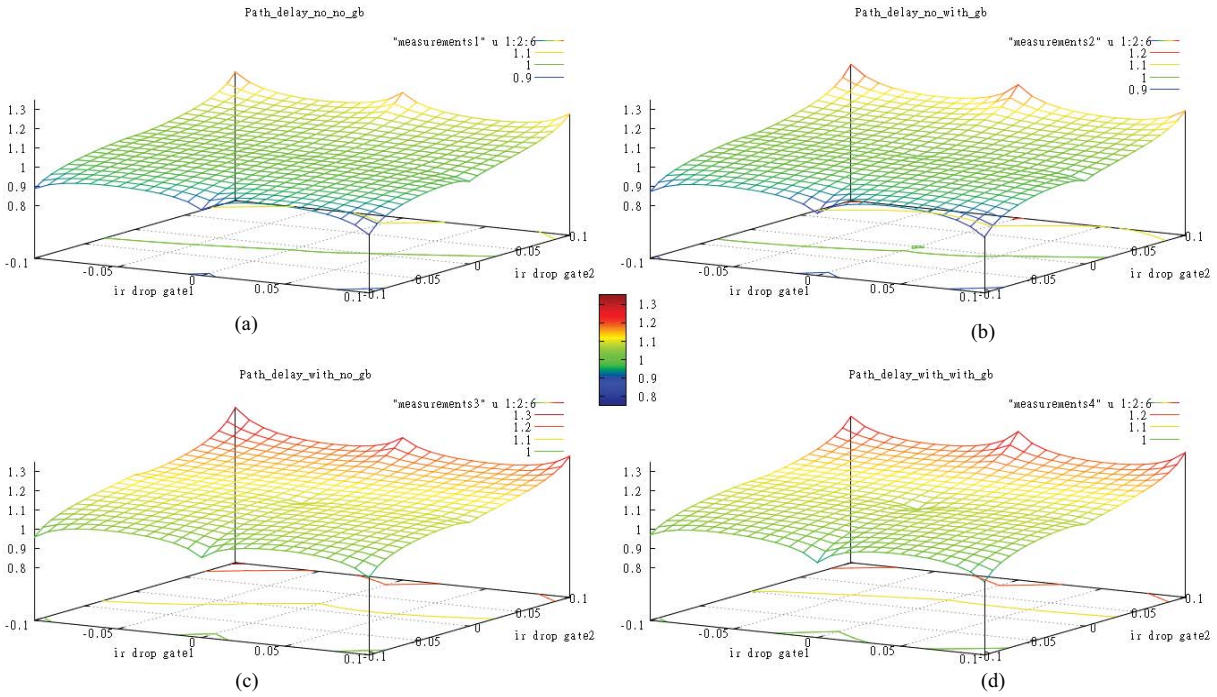


Fig. 4. Path delay variations as a function of power supply noise on both gates. (a) No ground bounce on any of the gates. (b) Ground bounce only on gate 1. (c) Ground bounce only on gate 2. (d) Ground bounce on both gates.

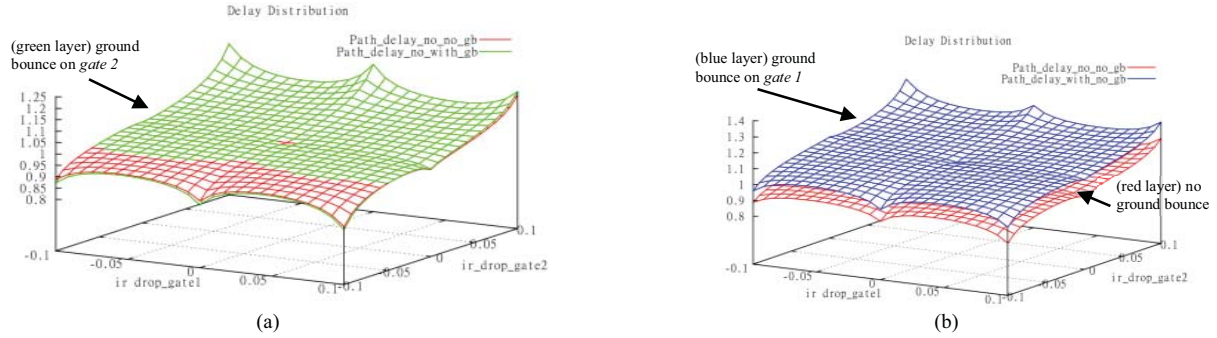


Fig. 5. Path variations with power and ground supply noise showing speedup and slowdown. (a) Ground bounce injected on gate 2. (b) Slowdown when ground bounce injected on gate 1.

cover a significant on-chip area, they provide a large amount of parasitic resistance (R), inductance ($j\omega L$) and capacitance ($1/j\omega C$), which are sensitive to frequency (ω) variations and can considerably change network impedance $Z = R + j\omega L + 1/j\omega C$. Consequently, power and ground network impedance increases with resonance frequency which further increases the supply noise. In [16], the authors have studied the impact of package inductance at different frequencies to estimate the amount of supply noise generated. They have concluded that there are high- and mid/low-frequency supply noises generated. High-frequency noise is a localized phenomenon due to the effect of the neighboring decoupling capacitors. The mid- to low-frequency resonance have a larger and an additive impact on every neighboring gate, further overwhelming each gate's localized high-frequency effects. In our experiment, we measure path delay variations with varying switching frequency.

Power supply noise is derived by integrating the supply voltage over switching period such as $noise = \int_{ts}^{te}(V_{\mathrm{DD}} - V_t)$ where ts and te are the starting and ending switching times. The measured noise represents the area of voltage drop under nominal voltage level. Ground bounce is similarly measured. Fig. 6 depicts the area for representing power supply noise.

In Fig. 7, we show the measured supply noise on each buffer gate as a function of the switching frequency of the input signal. We observe two resonance peaks from each buffer gate. The first gate has a peak on the supply noise around 250 MHz, while the second gate has a peak on supply noise around 500 MHz. This is due to the coupling of the package inductance with capacitance of each gate thereby creating two mid-frequency resonance effects. Fig. 7 also shows the path delay variation as a function of the switching frequency. Resonance frequency further complicates the relationship between supply noise and delay and makes the selection of stimuli for

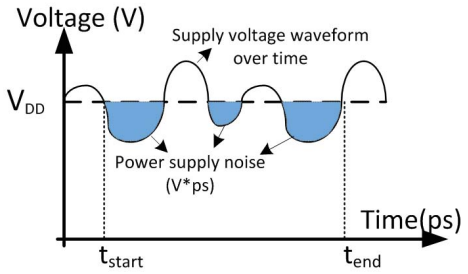


Fig. 6. Illustration of power supply noise measurement.

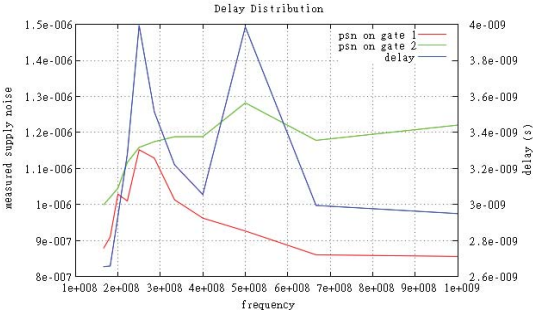


Fig. 7. Supply noise and path delay variations with switching frequency.

the worst case path delay a difficult task during test pattern generation.

From these experiments, we observe the effect of uncorrelated power supply noise and ground bounce on path delay as follows: 1) performance degradation due to reduced voltage level between power and ground; 2) delay increase/decrease due to noise conditions on a gate and its neighboring gates; and 3) augmented supply noise and increased path delays due to the resonance frequency.

Thus, path delay variations are dependent on multiple variables such as input stimuli, physical placement, package parasitics, resonance frequency, and available decoupling capacitors. Hence, capturing the worst case delay by considering all these variables is a complicated task. In the following section, we describe our models and pattern generation method in the presence of power supply noise and ground bounce.

III. Delay Model for Power Supply Noise and Ground Bounce

In this section, we present our approach for modeling the effect of power supply noise and ground bounce on path delay. Our approach consists of two main parts: 1) current derivation process and 2) path delay circuit analysis. Pattern generation flow iterates between these two processes to identify the input stimuli that generate the worst case path delay in the presence of power supply noise and ground bounce.

A. Current Derivation Process

In this step, we derive the amount of current drawn by switching gates on the circuit. Power supply noise and ground bounce are dependent on the instantaneous currents flowing through power and ground networks and their parasitic impedance values. Accurate current waveforms must be obtained in order to accurately derive the amount of noise on the circuit. The process of deriving the current consumed by each gate is organized in three steps: 1) library characterization; 2) circuit levelization; and 3) current derivation.

1) Library Characterization: Here, we derive the current waveform for each cell in the library as a function of its primary input conditions. SPICE netlist of each cell is simulated and current waveforms with respect input patterns are obtained. We store the current characteristics, i.e., peak current I_p, leakage current I_l, transition time t_r, and peak time t_p, for each input condition in a lookup table (LUT). Such characterization allows us to transform each cell into a current source (triangular waveform) model appropriate to its input conditions.

These waveforms are computed only once and are used during the test pattern generation step for identifying the current consumption based on a given input pattern. We note that current waveform characteristics $\{I_p, I_l, t_r, t_p\}$ are obtained for ideal power and ground conditions. These current models are later inserted on the actual power and ground networks for more accurate power and ground network analysis.

2) Circuit Levelization: The objective of this step is to obtain input transitions for each gate on the netlist. We utilize a levelized simulation algorithm in order to propagate the transitions from primary inputs to primary outputs [18], [19]. The algorithm begins with primary inputs that are assigned a level number zero. A level number can be assigned to a gate only if all gate inputs have been assigned level numbers. Similarly, a net can be assigned a level number only if all driving gates have been assigned level numbers. The level assignment process is iterative until all the nets and gates on the netlist have been levelized and primary outputs have been reached. Once the netlist is levelized, we perform levelized simulation where primary input transitions are propagated in an orderly fashion throughout the gates on the netlist.

We note that there exist other methods and commercial tools that perform waveform simulation for a given input pattern [20], [21]. We employ the levelized simulation algorithm which is incorporated in our pattern generation flow.

3) Current Derivation: After the netlist is levelized and input transitions are propagated through each gate, we derive each gate's appropriate current waveform. The idea is to utilize LUTs obtained from library characterization step in order to represent each gate as a current source model.

As we propagate transitions throughout the netlist, there are two main tasks being performed: 1) current modeling based on LUT match-up with input transitions and 2) delay accumulation as transitions are propagated in the levelized netlist. The first task serves to identify the current source $\{I_p, I_l, t_r\}$, while second task serves to identify peak transition time $\{t_p\}$. By keeping track of $\{t_p\}$ for each cell, we ensure that in a given clock cycle all cells are not switching at the same time but rather shifted in time by the accumulated delay for each level of the netlist. The delay of each level of the

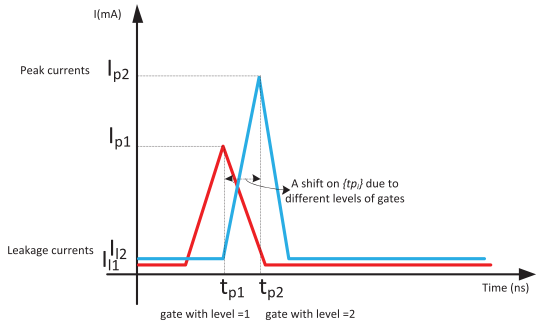


Fig. 8. Illustration of the current source model for different gates on the levelized netlist.

netlist is based on

$$\text{delay}_{\text{level}} = \sum_{i=1}^{nr \text{ of levels}} \text{delay}_{\text{gate}_i}. \quad (1)$$

Fig. 8 illustrates such a concept where the current waveform of the gate with level 1 has peak time $\{tp_1\}$ and the gate with level 2 has peak time $\{tp_2\}$.

In this step, as we still have not derived the actual gate delay, we utilize the gate delay measured with an ideal power and ground network. We note that the levelized delay is simply used for representing realistically the switching times of current sources as they would occur during circuit's operation. Gate delays in the presence of supply noise are derived in the next section.

In summary, our objective is to obtain fast and accurate current source models according to input transitions of the circuit. The derived current source models are a function of input patterns as

$$I = f(\text{patterns}) \quad (2)$$

where *patterns* represent the input conditions and I is the current vector of size $n \times 1$ where n is the total number of nodes on the circuit for both the power and ground networks. We note that not every node has a current source attached to it. In the following subsections, we introduce circuit modeling concept which takes into consideration cell placement.

B. Path Delay Circuit Analysis

The key objective of this paper is to utilize circuit physical design information that is extracted after placement and routing. We devise physical design data in mathematical models for performing accurate power supply noise and ground bounce analysis with respect to applied input patterns. Computed power supply noise and ground bounce are then used to derive each gate's delay while considering noise conditions on its neighboring gates and switching frequency. We develop a flow where path delay is derived with respect to power/ground parasitics and input stimuli are represented as switching current sources and switching frequency.

Path delay circuit analysis is performed in three steps: 1) circuit modeling; 2) power supply noise and ground bounce derivation; and 3) path delay calculation.

1) Circuit Modeling: In this paper, we utilize the circuit netlist that is extracted after the design has been placed and routed and power/ground networks are inserted. The extracted netlist provides R, L, and C parasitic information of the circuit, package, power/ground networks, and pin/cell placements. Power and ground networks are modeled by using the extracted resistance and capacitance parasitics $\{R_{\text{pwr}}, R_{\text{gnd}}, C_{pg}\}$ while package is modeled by its inductance and resistance parasitics $\{R_{\text{pkg}}, L_{\text{pkg}}\}$. Note that we only consider self-inductance and ignore mutual inductance on power/ground networks. While mutual inductance can alter power grid impedance, it also results in excessively large analysis runtimes. As the goal of this paper is to identify quickly and accurately the impact of input patterns on path delay in presence of power supply noise and ground bounce, we ignore mutual inductances.

Current sources inserted between power and ground networks are current models obtained from current derivation process in the previous section. Their locations are derived from cell placement data of the extracted netlist. The initial circuit netlist in verilog and commercial CAD tool (Cadence SoC Encounter[1]) is used for place and route and generate the extracted netlist.

Fig. 9 shows the physical layout design for a sample circuit from ITC'99 benchmarks and a simplified two-cell circuit to represent modeling. We note that, for the circuit sample in Fig. 9, ground and power network is represented as a mesh topology, however, tree topologies can also be extracted depending on the design style.

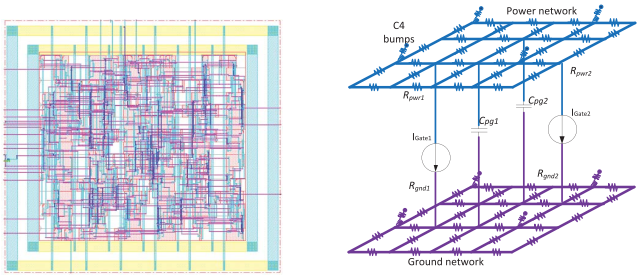
The goal of the circuit modeling step is to represent physical design information of the circuit in a mathematical model which we can accurately analyze. We utilize the modified nodal analysis (MNA) [22] approach to represent the extracted circuit into a mathematical model using Kirchhoff's law node equations as in (3) and (4)

$$(G_{nxn} + sC_{nxn})V_{nx1} = I_{nx1} \quad (3)$$

$$\left(\begin{bmatrix} G^p_{11} & \dots & G^p_{1m} & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ G^p_{m1} & \dots & G^p_{mm} & 0 & \dots & 0 \\ 0 & \dots & 0 & G^g_{11} & \dots & G^g_{1m} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & G^g_{m1} & \dots & G^g_{mm} \end{bmatrix} + s \begin{bmatrix} C_{11} & \dots & 0 & -C_{11} & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & C_{mm} & 0 & \dots & -C_{mm} \\ -C_{11} & \dots & 0 & C_{11} & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & -C_{mm} & 0 & \dots & C_{mm} \end{bmatrix} \right) \begin{bmatrix} V^p_1 \\ \vdots \\ V^p_m \\ V^g_1 \\ \vdots \\ V^g_m \end{bmatrix} = \begin{bmatrix} I_1 \\ \vdots \\ I_m \\ -I_1 \\ \vdots \\ -I_m \end{bmatrix} \quad (4)$$

where G^p_{ij} is impedance between nodes i and j in the power network and G^g_{ij} is the impedance between nodes i and j in the ground network. Capacitors between power and ground nodes are represented by C_{ij}. V_j is voltage at node i where the top half of the vector represents power network nodes, V^p_i and bottom half represents ground network nodes, V^g_i where

[1]Available online at http://www.cadence.com/products/di/soc_encounter/pages/default.aspx.



(a) (b)

Fig. 9. (a) Physical design of the circuit. (b) Circuit model representation.

$V = [V_i^p, V_i^g]$. Current sources represent the current consumed by the cells connected between power and ground networks, and vector Ii is the current source vector where the top half presents the current sources connected to power network and the bottom half presents the same current source connected to ground grid with opposite current flow. The location of current sources is obtained from cell placement. m is the number of power (ground) nodes where $n = 2 \cdot m$. Some of the nodes on power and ground networks do not have a current source connected to them and these nodes are represented by a zero current source on the I vector. Furthermore, some of the nodes on power and ground networks serve as V_{DD} ($\mathrm{G_{ND}}$) pins, which can be presented by modifying (4) to identify them as voltage sources. We note that mathematical formulation is represented in the frequency domain (s).

2) Power Supply Noise and Ground Bounce Derivation: The mathematical formulation in (4) provides all physical design information of the circuit, where the G and C matrices are obtained from the extracted netlist. Values of the I vector vary according to the applied input stimuli and derived as described in the previous section. Thus, for a given input pattern where G, C, and I are known, the only unknown remains node voltage vector V. Equation (3) is a set of linear equations with n unknowns which can be accurately solved using matrix manipulations. We utilize MATLAB [12] to perform matrix computations where node voltages are expressed as

$$V = (G + sC)^{-1} I. \tag{5}$$

Node voltages vector V is further used to obtain power supply noise, ground bounce, and supply noise as shown in (6)–(8)

$$\mathrm{PSN}_i = \int_{ts}^{te} (V_{\mathrm{DD}} - V_i^p) dt \tag{6}$$

$$GB_i = \int_{ts}^{te} (V_i^g - V_{\mathrm{gnd}}) dt \tag{7}$$

$$SN_i = \mathrm{PSN}_i + GB_i \tag{8}$$

where PSNi, GBi, and SNi are the power supply noise, ground bounce, and supply noise for cell i, and t_s, t_e are the starting and ending switching times.

Taking the inverse matrix can be a computationally expensive task, and in this flow we compute inverse matrix only once and utilize it with different I vectors (input patterns) to derive the voltage distribution on power and ground networks.

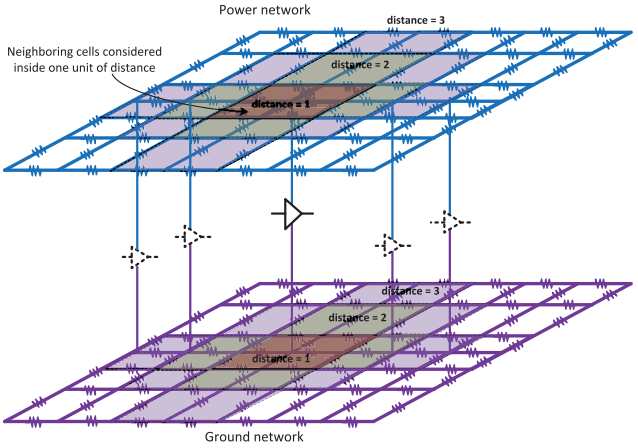


Fig. 10. Illustration of neighboring cells considered for delay analysis.

Similarly, computation of power supply noise and ground bounce is performed every time with a new I vector (input pattern). The inverse matrix can be efficiently obtained using various techniques, i.e., model order reduction, exploiting matrix sparsity, multigrid method, etc. It is not the focus of this paper to elaborate on these methods, but there exist efficient solvers for the inverse matrix problem.

3) Delay Characterization: In this step, we aim to capture delay variations as a function of power supply noise and ground bounce on a gate and its neighboring gates. As shown in the motivational example, noise conditions on a cell and its neighboring cells can cause either path delay increase or decrease. Such a phenomenon is further exacerbated in the presence of resonance frequency.

Delay characterization is performed by: 1) deriving the gate delay in presence of power and ground noise, noise impact from neighboring gates, and switching frequency and 2) deriving path delay based on gate delays. We start by characterizing the relationship between gate delay and noise with respect to delay coefficient β_i. For each cell on the library, we perform an HSPICE simulation with different power and ground voltage levels $\{V_i^p, V_i^g\}$ and switching frequency, $\{w\}$ in order to compute its delay variations. These simulations are performed on corner cases, i.e., no, mid- and high-level noise and low, medium, and high switching frequencies. The results obtained are utilized on a regression analysis in order to obtain the *coefficients* β_i that lead to estimate gate delay as in (9)

$$\tau_{\mathrm{cell}_i} = \beta_i^p V_i^p + \beta_i^g V_i^g + \sum_{j \epsilon \mathrm{neigh}} (\beta_j^p V_j^p + \beta_j^g V_j^g) + \beta_i^w w \tag{9}$$

where β_i^p is delay coefficient from the power node voltage of cell i, β_i^g is delay coefficient from ground node voltage of cell i, β_j^p is delay coefficient from power node voltage of neighboring cell j, β_j^g is the delay coefficient from ground node voltage of neighboring cell j, β_i^w is delay coefficient of cell i for frequency w, and $\tau_{\mathrm{cell_i}}$ is the delay of cell i.

Neighboring cells are chosen based on the Manhattan distance between them and the cell under investigation. We perform simulation and quantify the impact on path delay from cells located in different Manhattan distances as shown in Fig. 10. Table III provides the experimental results. In this

TABLE III
PROXIMITY OF NEIGHBORING CELLS IMPACT ON CELL DELAY

Neighboring cells included	Delay (s)	% Difference wrt. to actual delay
No neighboring cells	3.22E-10	18.27%
1st distance	3.58E-10	9.14%
1st + 2nd distances	3.65E-10	7.36%
1st+2nd+ 3rd distances (actual delay)	3.94E-10	0%

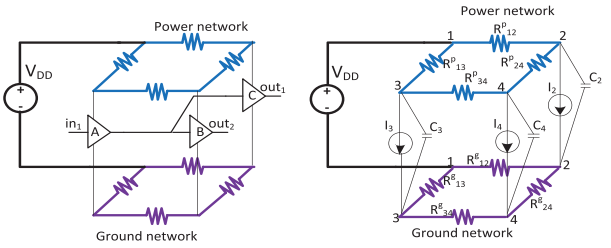


Fig. 11. Sample circuit for illustration of delay analysis with power supply noise and ground bounce.

experiment, neighboring cells have a slow-down effect on path delay. There is a 22% percentage difference on path delay when no neighboring cells are included versus when all neighboring cells are included. For this experiment, the actual path delay is measurement when first, second, and third unit distance neighboring cells are considered. In the case when only the first distance neighboring cells are considered, we obtain less than 10% path delay difference. In our analyzes, we consider neighboring cells located within one Manhattan distance from the cell under investigation, as shown in Fig. 10.

The *delay coefficients* are derived by solving the linear least square regression. Such mathematical formulation allows us to capture delay speedup/slowdown due to the noise conditions on the current cell and its neighboring cells as a function of the switching frequency. Once regression analysis is performed, the path delay is computed as

$$\tau_{\text{path}} = \sum_{\text{cell}_i \epsilon_{\text{path}}} \tau_{\text{cell}_i}. \tag{10}$$

Here, we provide a sample circuit in which we perform all the aforementioned steps in order to exemplify our flow. Fig. 11 shows a sample model of a three-gate circuit.

The sample circuit has three inverter gates. The current characterization LUT for the inverter is shown in Fig. 12(a). The sample circuit has one input and two outputs. We investigate the rising condition on the input by applying input pattern $<V_1, V_2>=<0,1>$. Levelized circuit netlist, propagated transitions, and the appropriate current source model for each gate are shown in Fig. 12(b). There are four nodes on the power and ground network, respectively, with a total of eight nodes. Equation (11) shows the matrix formulations where G_{8x8}, C_{8x8}, and I_{8x1} are expressed in the Laplace $s-$domain and are known variables. Power and ground node voltages in V_{8x1} are unknown and can be solved accurately using any linear algebra package solvers. Once the node voltages are obtained, they are

LUT - INV

Input transitions	Current source model
Low	{0, 0, 0}
High	{0, 0, 0}
Low-to-High	{120uA, 100nA, 0.5n}
High-to-Low	{55uA, 100nA, 0.5n}
Delay coefficients	$\beta^p_{inv}=8.14e-10$ $\beta^g_{inv}=9.05e-10$ $\beta^p_{neighbor}=-4.39e-10$ $\beta^g_{neighbor}=-7.16e-10$ $\beta^w_{inv}=0.0412$

(a)

Circuit netlist

For a given pattern <0,1> (low-to-high)

	Gate A	Gate B	Gate C
level	0	1	1
propagated transitions	*low-to-high*	*high-to-low*	*high-to-low*
tp	1.5n	1.8ns	1.8ns
current source model	{120uA, 100nA, 0.5n}	{55uA, 100nA, 0.5n}	{55uA, 100nA, 0.5n}

(b)

Fig. 12. (a) LUT for INV gate derived from library characterization step. (b) Circuit netlist levelization and current source modeling for each gate.

used to the derive path delay, i.e., the path from cell A to B as shown in (12) by using (9) and (10). We note that $\{\beta\}$ coefficients are already precomputed as described in the previous subsection. The path delay computed from the mathematical equations is 3.9×10^{-10} s versus 4.03×10^{-10} s obtained from HSPICE (3.2% difference). Thus, throughout our flow, we need to compute the inverse of $(G+sC)^{-1}$ only once while vector I will change with respect to the input pattern

$$(G + sC)V = I$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{R^p_{12}}+\frac{1}{R^p_{24}} & 0 & -\frac{1}{R^p_{24}} \\ 0 & 0 & \frac{1}{R^p_{13}}+\frac{1}{R^p_{34}} & -\frac{1}{R^p_{34}} \\ 0 & -\frac{1}{R^p_{24}} & -\frac{1}{R^p_{34}} & \frac{1}{R^p_{24}}+\frac{1}{R^p_{34}} \end{bmatrix} \begin{bmatrix} 0 \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{R^g_{12}}+\frac{1}{R^g_{24}} & 0 & -\frac{1}{R^g_{24}} \\ 0 & 0 & \frac{1}{R^g_{13}}+\frac{1}{R^g_{34}} & -\frac{1}{R^g_{34}} \\ 0 & -\frac{1}{R^g_{24}} & -\frac{1}{R^g_{34}} & \frac{1}{R^g_{24}}+\frac{1}{R^g_{34}} \end{bmatrix}$$

$$s \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & C_2 & 0 & 0 \\ 0 & 0 & C_3 & 0 \\ 0 & 0 & 0 & C_4 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & -C_2 & 0 & 0 \\ 0 & 0 & -C_3 & 0 \\ 0 & 0 & 0 & -C_4 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & -C_2 & 0 & 0 \\ 0 & 0 & -C_3 & 0 \\ 0 & 0 & 0 & -C_4 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & C_2 & 0 & 0 \\ 0 & 0 & C_3 & 0 \\ 0 & 0 & 0 & C_4 \end{bmatrix} \begin{bmatrix} V^p_1 \\ V^p_2 \\ V^p_3 \\ V^p_4 \\ V^g_1 \\ V^g_2 \\ V^g_3 \\ V^g_4 \end{bmatrix} = \begin{bmatrix} V_{DD}(s) \\ I_2(s) + \frac{V_{DD}(s)}{R^p_{12}} \\ I_3(s) + \frac{V_{DD}(s)}{R^p_{13}} \\ I_4(s) \\ 0 \\ -I_2(s) \\ -I_3(s) \\ -I_4(s) \end{bmatrix} \tag{11}$$

$$\begin{aligned} \tau^A_{\text{cell}} &= \beta^p_A V^p_3 + \beta^g_A V^g_3 + \left(\beta^p_B V^p_4 + \beta^g_B V^g_4\right) + \beta^w_A w \\ \tau^B_{\text{cell}} &= \beta^p_B V^p_4 + \beta^g_B V^g_4 + \left(\beta^p_A V^p_3 + \beta^g_A V^g_3 + \beta^p_C V^p_2 + \beta^g_C V^g_2\right) \\ &\quad + \beta^w_B w \\ \tau^{AB}_{\text{path}} &= \tau^A_{\text{cell}} + \tau^B_{\text{cell}}. \end{aligned} \tag{12}$$

In the next section, we use these mathematical models in our test pattern generation flow in order to accurately determine the impact of the input patterns on the path delay in the presence of power supply noise and ground bounce.

IV. TEST PATTERN GENERATION FLOW CONSIDERING POWER SUPPLY NOISE AND GROUND BOUNCE

To identify path delay faults, a vector pair needs to be applied to the circuit. One solution to finding the maximum

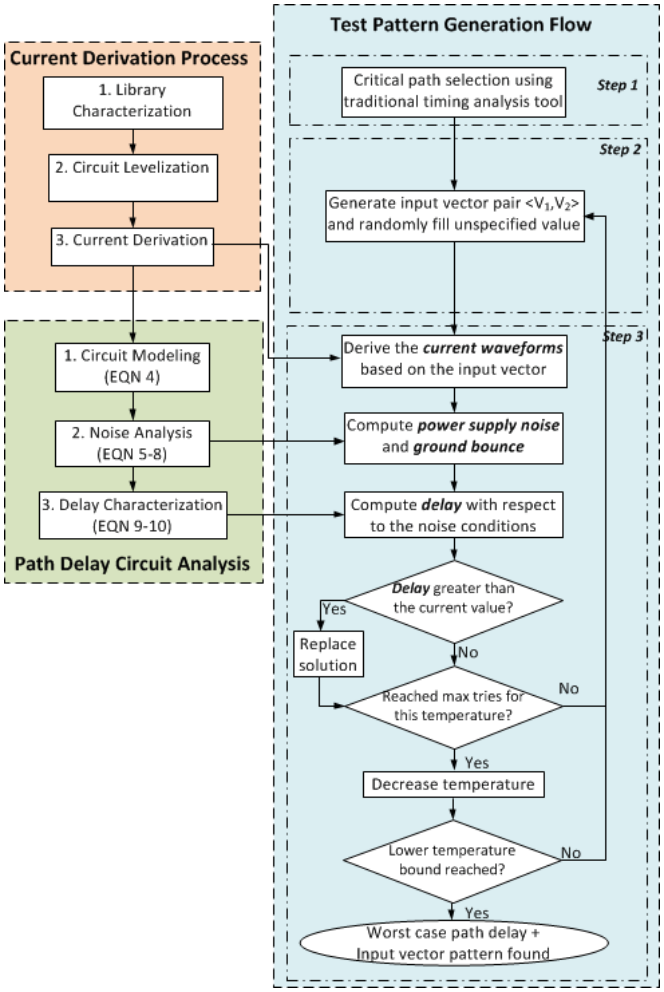


Fig. 13. Test pattern generation flow based on SA iterative process for capturing the worst case path delay in the presence of power supply noise and ground bounce with respect to switching frequency.

path delay in the presence of noise is to simulate all possible two-vector patterns for a given circuit. However, this is simply infeasible, as it would require a significantly large number of simulations. We propose a pattern generation flow that makes use of the closed-form equations described in the previous section to estimate delay based on the input patterns in the presence of supply noise.

Our test pattern generation flow for path delay faults considering power supply noise and ground bounce consists of three steps: 1) path selection; 2) vector pair generation; and 3) SA-based pattern generation.

A. Path Selection

We employ the commercial static timing analysis tool Synopsys Primetime [23] to identify the critical paths in the design. Only a small subset of paths is selected by the tool, listing the longest paths based on the timing report. From this small subset, we select only top 10% of critical paths to apply our pattern generation flow. We note that our pattern generation method is independent of the path selection process and it can be applied to any selected path.

TABLE IV
VARIOUS INPUT PATTERNS AND THEIR DELAY AND NOISE MEASUREMENTS FOR THE *b*01 CIRCUIT

<V1,V2>=(XX011,XX010)
V1=XX011 V2=XX010

Vector Filling <V1,V2>	Delay (s)	Supply Noise (V*s) (SN=PSN+GB)	% Diff wrt min. Delay
(00011,00010)	6.824E-10	2.280E-09	15%
(00011,01010)	6.871E-10	2.877E-09	16%
(01011,00010)	6.708E-10	3.082E-09	13%
(01011,01010)	6.764E-10	2.654E-09	14%
(00011,10010)	6.789E-10	3.081E-09	15%
(00011,11010)	6.895E-10	3.434E-09	16%
(01011,10010)	6.637E-10	2.186E-09	12%
(01011,11010)*	7.040E-10	3.513E-09	19%
(10011,00010)	6.843E-10	3.190E-09	15%
(10011,01010)	6.800E-10	2.329E-09	15%
(11011,00010)	6.659E-10	3.946E-09	12%
(11011,01010)	5.929E-10	3.481E-09	0%
(10011,10010)	6.764E-10	2.655E-09	14%
(10011,11010)	6.889E-10	3.985E-09	16%
(11011,10010)	6.631E-10	3.702E-09	12%
(11011,11010)	6.670E-10	2.955E-09	12%

** Selected by our SA-based pattern generation*

TABLE V
DELAY VARIATIONS FOR CRITICAL PATHS OF THE *b*01 CIRCUIT

b01	Delay (s)	PSN (V*s)	GB (V*s)	Supply Noise (PSN+GB)
Path 1 (most critical)	1.98E-10	2.04E-11	2.24E-11	4.28E-11
Path 2	2.12E-10	2.87E-11	4.64E-11	7.51E-11
Path 3	1.87E-10	1.09E-11	1.69E-11	2.78E-11
Path 4 (least critical)	1.76E-10	2.00E-11	3.51E-11	5.51E-11

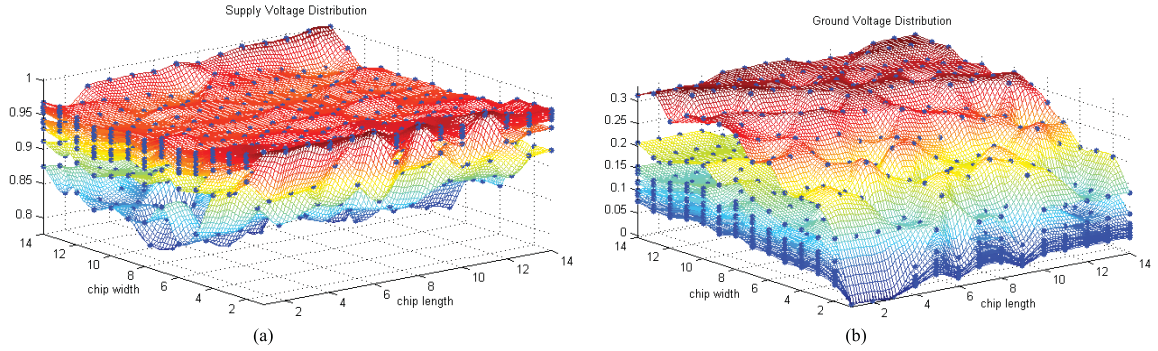
B. Vector Pair Generation

Test vectors are generated such that the target path is sensitized under given propagation condition (robust, nonrobust, etc.). As shown in [24], defects on robustly testable paths are guaranteed to be detected regardless of the delays outside the targeted paths, while defects on nonrobustly testable paths can be detected if transitions on certain signals not belonging to the target path are not late.

In this paper, we use the commercial Tetramax ATPG tool [25] to generate partially specified input vector pairs for the selected critical paths. In this step, we attempt to leave as many unspecified (X value) primary input values as possible so that we can apply X filling by considering their impact on path delay in the presence of supply noise.

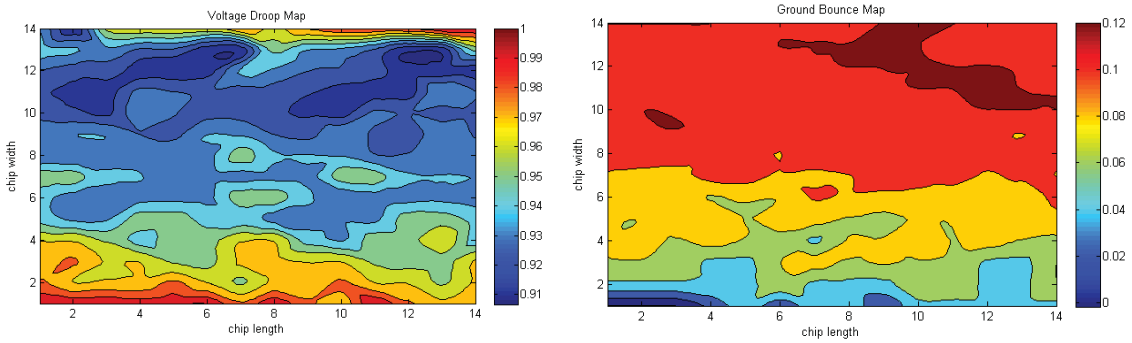
C. Test Pattern Generation

Different assignments of unspecified primary input values can result in different path delays and supply noise. This is because path delay is dependent on the number of inputs, which are switching and the internal switching activity on the circuit. For the selected critical path, the objective is to generate an *input vector pair* such that the impact of power supply noise and ground bounce on path delay is maximized. We develop a SA-based iterative process in order to evaluate the effect of supply noise on delay for each generated input



(a) (b)

Fig. 14. (a) Voltage distribution on power network for different input patterns for the *b*01 circuit. (b) Voltage distribution on ground network for various input patterns for the *b*01 circuit.



(a) (b)

Fig. 15. (a) Voltage drop and (b) ground bounce maps for the *b*01 circuit for the selected input vector from test pattern generation flow.

vector pair. SA is a well-known optimization technique widely used for various applications.

The iterative flow is based on: 1) generating a test vector pattern by filling unspecified primary input values; 2) computing current waveforms by using library characterization data as in Section III-A; 3) computing power supply noise and ground bounce using closed-form equations as in Section III-B; and 4) computing path delay in presence of noise as in Section III-B.

In the first two steps, unspecified input values are randomly filled by either a 0 or 1. Thus, the number of generated patterns is dependent on the number of unspecified input values. In the third and fourth steps, the generated input test pattern is evaluated for supply noise and path delay. Computed path delay serves as the evaluation function for the generated input test pattern. The temperature parameters for SA are set to $T_{i+1} = T_i \cdot CR^{k-1}$, where the cooling rate is $CR = 0.92$ and k is the cooling step in the iteration loop. For each temperature step, equilibrium is reached if there is no more change in path delay for a perturbed input vector configuration. SA iterates among different input patterns in order to identify the pattern that generated the maximum delay in the presence of power supply noise and ground bounce with respect to switching frequency. Fig. 13 illustrates the proposed SA-based approach for pattern generation flow.

V. Experimental Results

Our experiments are conducted on the combinational part of ITC'99 [26] benchmarks which are described in register transfer level and synthesized using STMicroelectronics 90-nm cell library with $V_{DD} = 1$ V. The gate level netlists are generated and imported to the SoC Encounter where physical layout information is obtained after power/ground network design, floorplanning, placement, and routing. Timing information and critical path lists obtained from Synopsys PrimeTime along with the circuit netlist are fed to TetraMax to generate input test patterns with unspecified input values (X values). In this paper, we use 10% of worst critical path reported from PrimeTime. The extracted netlist with $\{R, L, C\}$ parasitics of the power /ground network, interconnects, and cell placement is then provided to MATLAB[2] where the SA-based test pattern generation is implemented. All the mathematical models and equations described in the previous sections are implemented in MATLAB. The voltage drop constraints were set to 10% of nominal voltage values. The experiments were run on a Linux machine with a speed of 2.5 GHz, memory 4 GB of RAM, and capacity of 250 GB. We studied the effect of power supply noise and ground bounce on path delay and multiple critical path behavior and applied our pattern generation flow on several circuits.

A. Impact of Power Supply Noise and Ground Bounce on Path Delay

We experiment with the *b*01 benchmark of ITC'99 to demonstrate the impact of power supply noise and ground bounce on path delay.

[2]Available online at http://www.mathworks.fr/.

TABLE VI
CIRCUIT DESCRIPTION

circuits	PI	PO	Gates	Critical Paths
b01	5	5	26	5
b02	5	4	14	4
b03	34	30	80	30
b04h	77	67	273	66
b04f	77	67	273	66
b06	6	9	38	9
b07	50	50	205	49
b08	30	22	78	21
b09	29	29	78	28
b10	28	18	96	17
b13	63	54	154	48
b18	3358	3342	57470	100
b19	1024	1605	77734	100
b20	126	75	4724	100
b21	522	512	5595	100
b22	186	110	6993	100

TABLE VII
RESULTS OF SA-BASED TEST PATTERN GENERATION FLOW FOR SEVERAL ITC'99 CIRCUITS

circuits	SN=PSN+GB (V*s)	Delay (s)	Runtime (s)
b01	4.287E-11	1.984E-10	93
b02	4.902E-11	2.306E-10	137
b03	1.525E-09	8.386E-11	250
b04h	1.328E-09	6.978E-12	270
b04f	7.708E-10	7.786E-12	270
b06	1.069E-11	3.992E-10	262
b07	4.902E-11	2.306E-10	314
b08	4.902E-11	2.148E-10	362
b09	7.956E-10	5.440E-10	289
b10	4.337E-11	4.850E-09	393
b13	9.852E-11	5.540E-09	1048
b18	2.264E-09	7.360E-09	7230
b19	1.186E-09	7.188E-09	5460
b20	2.498E-09	8.126E-09	4258
b21	3.855E-09	8.380E-09	5630
b22	3.995E-09	9.90E-09	2456

The number of primary inputs for the $b01$ benchmark is 5, which allows us to perform a thorough analysis of various input vectors. Input vectors are $<V_1 = XX011, V_2 = XX010>$ and there are 16 possible vectors that can be generated by specifying either a 0 or 1 on X values. To highlight the importance of considering both power supply noise and ground bounce for path delay testing, we evaluate the path delay and supply noise generated from each possible input vector $<V_1, V_2>$. Table IV shows our results. We have highlighted the minimum and maximum path delay and supply noise (power supply noise and ground bounce) for all vectors. There are two main observations from these experiments. First, we observe that the maximum supply noise on the circuit does not lead to the maximum path delay. This is due to path delay speed-up/slow-down phenomena triggered from noise conditions on the cell and its neighboring cells. Second, there is a maximum up to 19% of the measured path delay difference between the input vector selected from our SA-based approach and the pattern with minimum delay [$< V_1, V_2 >=$ (11011, 01010)]. These experimental results clearly indicate the need for a new delay testing technique that takes into account the impact that power supply noise and ground bounce can cause on path delay.

TABLE VIII
COMPARISON RESULTS OF SA-BASED TEST PATTERN GENERATION FLOW WITH 0 FILLING, 1 FILLING, AND RANDOM FILLING METHODS FOR SEVERAL ITC'99 CIRCUITS

circuits	Measured Delay (s)				% difference with our method		
	F0	F1	FR	Our Method	F0	F1	FR
b01	1.64E-10	1.75E-10	1.74E-10	1.98E-10	17.4%	11.8%	12.4%
b02	1.78E-10	1.74E-10	1.83E-10	2.31E-10	22.8%	24.5%	20.5%
b03	8.26E-11	8.25E-11	8.20E-11	8.39E-11	1.4%	1.6%	2.2%
b04h	6.92E-12	6.90E-12	6.52E-12	6.98E-12	0.8%	1.1%	6.6%
b04f	7.46E-12	7.40E-12	7.26E-12	7.79E-12	4.1%	4.9%	6.7%
b06	3.20E-10	2.82E-10	2.90E-10	3.99E-10	19.7%	29.3%	27.3%
b07	1.90E-10	1.60E-10	1.58E-10	2.31E-10	17.4%	30.5%	31.4%
b08	2.90E-11	3.28E-11	1.11E-11	2.15E-10	86.5%	84.7%	94.8%
b09	2.68E-10	5.43E-10	3.64E-10	5.44E-10	50.8%	0.2%	33.1%
b10	4.82E-09	4.75E-09	4.42E-09	4.85E-09	0.7%	2.1%	8.9%
b13	4.50E-09	4.64E-09	5.00E-09	5.54E-09	18.7%	16.2%	9.7%
b18	6.88E-09	6.72E-09	7.25E-09	7.36E-09	6.5%	8.7%	1.5%
b19	6.58E-09	6.32E-09	7.05E-09	7.18E-09	8.5%	12.1%	1.9%
b20	7.42E-09	7.54E-09	8.10E-09	8.13E-09	8.7%	7.2%	0.3%
b21	7.02E-09	6.54E-09	7.47E-09	8.38E-09	16.2%	22%	10.9%
b22	5.75E-09	5.72E-09	6.70E-09	9.90E-09	41.9%	42.2%	32.3%
avg					**20.1%**	**18.7%**	**18.8%**

Fig. 14(a) and (b) show the voltage distribution on power and ground networks generated throughout the pattern generation flow. As shown in Fig. 14(a), there are many layers of voltage distribution due to different input patterns applied on the circuit. Our objective in this paper is to not select the pattern with minimum or maximum voltage drop or ground bounce, but rather to select a pattern that causes maximum delay in the presence of supply noise. Similarly, Fig. 14(b) shows the various voltage distributions on the ground network for different input patterns as listed in Table IV. Fig. 15 shows the voltage drop and ground bounce map for the the selected input pattern that causes maximum path delay.

B. Impact of Multiple Critical Paths

Here, we investigate critical paths for the benchmark $b01$ provided from the static timing analysis tool and compute their path delay variations in the presence of power supply noise and ground bounce. The results are listed in Table V. Paths are listed based on their criticality where path1 is the most

critical and path4 is the least critical. From the results shown in Table V, we observe that path delay of path2 is larger than the delay of path1 when both power supply noise and ground bounce are considered.

Such an observation indicates that critical paths selected by the timing analysis tools might not necessarily be the actual critical paths of the circuit, as the impacts of power supply noise, ground bounce, and resonance frequency are ignored. Therefore, for accurate results, the test pattern generation methodology should be combined with the critical path selection technique in order to take into account the impact of power supply noise and ground bounce.

C. Test Pattern Generation Flow

We apply our test pattern generation flow to the combination part of circuits on the ITC'99 benchmark. Table VI shows the list of circuits and their characteristics in terms of the number of inputs, outputs, and critical paths (reported by the static timing analysis tool). We apply our SA-based test pattern flow on the circuits and list their supply noise (power supply noise and ground bounce) and path delay. The results are listed in Table VII.

The number of the generated patterns greatly depends on the number of unspecified input values which also impact the runtime and the quality of the solution obtained by test pattern generation flow. The quality of the solutions depends on the number of X values, as a smaller number of X values on the input pattern imposes less flexibility to our pattern generation flow for finding a pattern that generates maximum path delay. Additionally, the runtime grows proportionally with the number of X values, as it increases the number of patterns and mathematical computations to be evaluated. Furthermore, the choice of the simulator (MATLAB) to perform the analysis of the linear system (power/ground network analysis) can also contribute to the long runtime.

We implement three other methods for comparison. The first method performs 0 filling on the unspecified input values and labeled as $F0$. The second method performs 1 filling on the unspecified input values and is labeled $F1$. The third method performs random filling on the unspecified input values and is labeled *FR*. Table VIII shows the results. We obtain that 0, 1, and random fillings underestimate the impact of supply noise on path delay. These experiments clearly indicate the need for a power supply noise and ground bounce aware test pattern generation tool. As future work, we aim to integrate signal integrity issues (i.e., crosstalk) and switching activity distribution (i.e., accurate hot spot and voltage droop distribution) and combine them with the path selection step for more accurate path delay computation in the presence of power supply noise and ground bounce.

VI. Conclusion

Current path delay testing techniques do not consider the combined impact of power supply noise and ground bounce on path delay. In this paper, we proposed close-form mathematical models for capturing the impact of supply noise on path delay variation for generating suitable input test patterns. We proposed an SA-based pattern generation technique which, for its fitness function, uses the mathematical models for deriving accurately the impact of supply noise on path delay. Experimental results showed considerable differences in path delays when both power supply noise and ground bounce effects were considered.

References

[1] M. Tehranipoor and K. Butler, "Power supply noise: A survey on effects and research," *IEEE Design Test Comput.*, vol. 27, no. 2, pp. 51–67, Mar. 2010.

[2] A. Todri, A. Bosio, L. Dilillo, P. Girard, S. Pravossoudouvitch, and A. Virazel, "A study of path delay variations in the presence of uncorrelated power and ground supply noise," in *Proc. IEEE Symp. Design Diag. Electron. Circuits Syst.*, Apr. 2011, pp. 164–170.

[3] L. H. Chen, M. Marek-Sadowska, and F. Brewer, "Coping with buffer delay change due to power and ground noise," in *Proc. 39th Design Autom. Conf.*, 2002, pp. 860–865.

[4] G. Bai, S. Bobba, and I. Hajj, "Static timing analysis including power supply noise effect on propagation delay in VLSI circuits," in *Proc. Design Autom. Conf.*, 2001, pp. 295–300.

[5] S. Pant, D. Blaauw, V. Zolotov, S. Sundareswaran, and R. Panda, "Vectorless analysis of supply noise induced delay variation," in *Proc. Int. Conf. Comput.-Aided Design*, 2003, pp. 184–192.

[6] D. Kouroussis, R. Ahmadi, and F. N. Najmm, "Worst-case circuit delay taking into account power supply variations," in *Proc. 41st Design Autom. Conf.*, 2004, pp. 652–657.

[7] S. Pant and D. Blaauw, "Static timing analysis considering power supply variations," in *Proc. IEEE Int. Conf. Comput.-Aided Design*, Nov. 2005, pp. 365–371.

[8] T. Ahmadi and F. Najm, "Timing analysis in presence of power supply noise and ground voltage variations," in *Proc. EEE Int. Conf. Comput.-Aided Design*, 2003, pp. 1–8.

[9] T. Okumura, F. Minami, K. Shimazaki, K. Kuwada, and M. Hashimoto, "Gate delay estimation in STA under dynamic power supply noise," in *Proc. 15th Asia South Pacific Design Autom. Conf.*, 2010, pp. 775–780.

[10] J.-J. Liou, A. Krstic, Y.-M. Jiang, and K.-T. Cheng, "Path selection and pattern generation for dynamic timing analysis considering power supply noise effects," in *Proc. IEEE Int. Conf. Comput. Aided Design*, Nov. 2000, pp. 493–496.

[11] J. Ma, J. Lee, and M. Tehranipoor, "Layout-aware pattern generation for maximizing supply noise effects on critical paths," in *Proc. VLSI Test Symp.*, 2009, pp. 221–226.

[12] N. Ahmed, M. Tehranipoor, and V. Jayaram, "Supply voltage noise aware ATPG for transition delay faults," in *Proc. 25th IEEE VLSI Test Symp.*, May 2007, pp. 179–186.

[13] M. Nourani and A. Radhakrishnan, "Power-supply noise in SoCs: ATPG, estimation, and control," in *Proc. IEEE Int. Test Conf.*, Nov. 2005, pp. 505–516.

[14] A. Krstic, Y. Jiang, and K. T. Cheng, "Pattern generation for delay testing and dynamic timing analysis considering power-supply noise effects," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 3, pp. 416–425, Mar. 2001.

[15] T. Rhal-Arabi, G. Taylow, M. Ma, and C. Webb, "Design and validation of Pentium III and Pentium 4 processors power delivery," in *Symp. VLSI Circuit Dig. Technol. Papers*, 2002, pp. 220–223.

[16] S. Pant and E. Chiprout, "Power grid physics and implications for CAD," in *Proc. Design Autom. Conf.*, 2006, pp. 199–204.

[17] H. Chen and J. S. Neely, "Interconnect and circuit modeling techniques for full-chip power supply noise analysis," *IEEE Trans. Compon. Packag. Manuf. Technol.*, vol. 21, no. 3, pp. 209–215, Aug. 1998.

[18] R. J. Smith, "Fundamentals of parallel logic simulation," in *Proc. Design Autom. Conf.*, 1986, pp. 2-12.

[19] S. Even, *Graph Algorithms*. Rockville, MD: Computer Science, 1979.

[20] *Modelsim* (2009) [Online]. Available: http//www.mentorgraphics.com

[21] *ISE Simulator* (2009) [Online]. Available: http//www.xilinx.com

[22] R. L. Boylestad, *Introduction to Circuit Analysis*. Englewood Cliffs, NJ: Prentice-Hall, 2000.

[23] *Golden Timing Signoff Solution and Environment*. (2012) Synopsys, Inc., Mountain View, CA [Online]. Available: http://www.synopsys.com/Tools/Implementation/SignOff/Pages/PrimeTime.aspx

[24] L.-C. Chen, S. Gupta, and M. A. Brewer, "High quality robust tests for path delay faults," in *Proc. IEEE VLSI Test Symp.*, Apr.–May 1997, pp. 88–93.
[25] *Automatic Test Pattern Generation*. Synopsys, Inc., Mountain View, CA (2007) [Online]. Available: http://www.synopsys.com/Tools/Implementation/RTLSynthesis/Pages/TetraMAXATPG.aspx
[26] *ITC'99 Benchmarks* (1999) [Online]. Available: http://www.cerc.utexas.edu/itc99-benchmarks/bendoc1.html

Aida Todri (M'03) received the B.S. degree in electrical engineering from Bradley University, Peoria, IL, the M.S. degree in electrical engineering from Long Beach State University, Long Beach, CA, and the Ph.D. degree in electrical and computer engineering from the University of California, Santa Barbara, in 2001, 2003, and 2009, respectively.

She is currently a Researcher with the French National Center of Scientific Research (CNRS), Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier (LIRMM). Previously, she was an Research and Development Engineer with the Fermi National Accelerator Laboratory, IL. She has also held positions at Mentor Graphics, Cadence Design Systems, STMicroelectronics, and IBM TJ Watson Research Center. Her current research interests include nanometer-scale issues in high-performance VLSI design with emphasis on power, thermal, signal integrity, and reliability issues, as well as circuits and systems for emerging technologies.

Dr. Todri was a recipient of the John Bardeen Fellowship in Engineering in 2009 at Fermilab.

Alberto Bosio (M'06) received the Ph.D. degree in computer engineering from Politecnico di Torino, Torino, Italy, in 2006.

He is currently an Associate Professor with the Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier (LIRMM), University of Montpellier, Montpellier, France. His current research interests include computer-aided designs, logic diagnosis, functional verification, and dependability.

Patrick Girard (M'92–SM'09) received the M.S. degree in electrical engineering and the Ph.D. degree in microelectronics from the University of Montpellier, Montpellier, France, in 1988 and 1992, respectively.

He is currently a Research Director with the French National Center for Scientific Research (CNRS), and a Chair with the Microelectronics Department, Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier, Montpellier. His current research interests include all aspects of digital testing and memory testing, reliability and fault tolerance, and test of 3-D integrated circuits.

Dr. Girard holds the Technical Activities Chair of the Test Technology Technical Council (TTTC) of the IEEE Computer Society. He has served as a Vice-Chair of the European TTTC of the IEEE Computer Society, and also on numerous conference committees. He is the founder and Editor-in-Chief of the *ASP Journal of Low Power Electronics* and an Associate Editor of the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS and the *Journal of Electronic Testing-Theory and Applications* (Springer). He is a Golden Core Member of the IEEE Computer Society.

Luigi Dilillo (M'03) received the Ph.D. degree in microelectronics from the University of Montpellier, Montpellier, France, in 2005.

He is currently a CNRS Researcher with the Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier (LIRMM), University of Montpellier 2, Montpellier. He has published articles in diverse disciplines, including memory testing, power-aware testing, and radiation effects on electronic devices.

Arnaud Virazel (M'98) received the M.S. degree in electrical engineering and the Ph.D. degree in microelectronics from the University of Montpellier, Montpellier, France, in 1997 and 2001, respectively.

He is currently an Assistant Professor with the University of Montpellier 2, Montpellier, and works in the Microelectronics Department, Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier (LIRMM), Montpellier. His current research interests include the various aspects of digital testing, DfT, BIST, diagnosis, delay testing, power-aware testing, and memory testing.

Dr.Virazel served on technical program committees of EUC in 2005. He has been associated with the ISQED since 2008, DTIS since 2008, DATE since 2009, and ETS since 2010. He also serves as a reviewer for many conferences and journals on testing.

Statistical Reliability Estimation of Microprocessor-Based Systems

Alessandro Savino, Stefano Di Carlo, *Senior Member*, *IEEE*, Gianfranco Politano, *Member*, *IEEE*, Alfredo Benso, *Senior Member*, *IEEE*, Alberto Bosio, *Member*, *IEEE*, and Giorgio Di Natale, *Member*, *IEEE*

Abstract—What is the probability that the execution state of a given microprocessor running a given application is correct, in a certain working environment with a given soft-error rate? Trying to answer this question using fault injection can be very expensive and time consuming. This paper proposes the baseline for a new methodology, based on microprocessor error probability profiling, that aims at estimating fault injection results without the need of a typical fault injection setup. The proposed methodology is based on two main ideas: a one-time fault-injection analysis of the microprocessor architecture to characterize the probability of successful execution of each of its instructions in presence of a soft-error, and a static and very fast analysis of the control and data flow of the target software application to compute its probability of success. The presented work goes beyond the dependability evaluation problem; it also has the potential to become the backbone for new tools able to help engineers to choose the best hardware and software architecture to structurally maximize the probability of a correct execution of the target software.

Index Terms—Microprocessor reliability, safety-critical systems, statistical analysis

1 Introduction

As microprocessor technology scales down to the very deep submicron range, high-production variability, voltage scaling, and high-operating frequency increase the hardware susceptibility to (soft) errors [1], [2], [3], [4], [5], [6], [7], [8]. This has a negative impact on the reliability of a wide range of computer-based applications which are critical to our health, safety, and financial security. Since 1996, several studies reported cases of large computer system failures caused by cosmic-ray-induced soft-errors [9], [10].

Several techniques have been proposed to protect digital circuits against soft-errors, e.g., radiation-hardened technologies [11], [12], error detection/correction codes [13], and redundant architectures [14], [15]. Software Implemented Hardware Fault Tolerance (SIHFT) also gained attention in the last decade [16], [17]. These techniques have a negative impact on systems' performance, power consumption, area, and design complexity. Their application must therefore be carefully evaluated depending on the soft-error rate (SER) of the target system.

Unfortunately, tools and techniques to estimate the susceptibility of a computer system to soft errors, taking into account both the hardware and the software domain, are not readily available or fully understood. The execution of a program may mask a large amount of soft errors. In fact, at the system level soft errors do not matter as long as the final outcome of the program is correct. To efficiently tradeoff between fault tolerance cost and system reliability one has to ask: what is the probability of a program P to have a correct execution state given a certain hardware (raw) soft-error rate? Fault injection is a viable solution to answer this question [18], [19], [20]. However, it can be very expensive and time consuming.

This paper proposes the baseline for a new methodology to estimate computer-based systems reliability against soft-errors. The target microprocessor is first characterized to profile the probability of successful execution of each instruction of its Instruction Set Architecture (ISA). A static and very fast analysis of the control and data flow of the executed software is then performed to compute its probability of successful execution in case of soft errors. The presented method has the potential to help engineers to choose the best hardware and software architecture to minimize the impact of soft errors on the system's reliability.

This paper is organized as follows: Section 2 shortly overviews the related literature, while Sections 3 and 4 present the proposed model whose experimental validation is given in Section 5. To conclude, Section 6 introduces future improvements and Section 7 summarizes the main contributions of the paper.

2 Related Works

Previous works on the estimation of the Soft-Error Rate of an IC can be classified into three categories, namely *circuit-level*, *gate-level*, and *architectural-level*.

Circuit-level SER estimation tries to estimate the probability of an error (glitch) at the output of a logic gate hit by a particle. This is mandatory to define technological mitigation techniques to soft errors [21], [22].

- *A. Savino, S. Di Carlo, G. Politano, and A. Benso are with the Department of Control and Computer Engineering, Politecnico di Torino, Corso Duca degli Abruzzi 24, Torino I-10129, Italy. E-mail: {alessandro.savino, stefano.dicarlo, gianfranco.politano, alfredo.benso}@polito.it.*
- *A. Bosio and G. Di Natale are with the Laboratoire d'Informatique, de Robotique et de Microelectronique de Montpellier, University of Montpellier II/CNRS, 161, rue Ada, Montpellier 34392 Cedex 5, France. E-mail: {alberto.bosio, giorgio.dinatale}@lirmm.fr.*

Manuscript received 10 Dec. 2010; revised 26 Aug. 2011; accepted 3 Sept. 2011; published online 30 Sept. 2011.
Recommended for acceptance by B. Parhami.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2010-12-0680.
Digital Object Identifier no. 10.1109/TC.2011.188.

Gate-level SER estimation moves the focus to the nodes of a netlist [23]. Estimating the error susceptibility of a node requires computing the probability of sensitizing the node with an input vector able to propagate the erroneous value to one of the outputs of the circuit [24]. This however requires the simulation of several random vectors whose number significantly increases with the size of the circuit [22], [25], [23], [21], [26], [27].

Only recently the research on SER estimation has moved from circuit and gate-level to the architectural level [28], [29], [30], [31], [32]. The *Architectural Vulnerability Factor* (AVF) expresses the probability of a system error caused by a raw error in a particular hardware structure [29]. In fact, several raw errors occurring at the device/circuit level are masked at the architectural level (e.g., 85 percent, Wang et al. [31]) due to low-resource utilization and introduction of computational blocks that affect performance but not correctness (e.g., branch prediction unit). Several publications propose methods to estimate the AVF of a functional block [28], [29], [30], [31], [32], [33], [34]. An interesting solution that includes the software layer is provided by Sridharan and Keli [35]. They propose to compute a Program Vulnerability Factor (PVF) for a set of benchmarks that can be then used to save computation while calculating the AVF of several microprocessors. Differently to what we propose in this paper, the final software workload is not explicitly considered. Only a few publications try to introduce this concept [36], [37]. However, apart from using fault injection, to the best of the authors' knowledge, a very efficient algorithm to estimate the error probability of a computer system taking into account its hardware, architecture, and running software, is still missing, thus motivating the research proposed in this paper.

3 Soft-Error and System Models

This section shortly introduces the soft-error model considered in this work, together with some basic concepts required to perform the proposed reliability analysis.

3.1 Soft-Errors Model

Neutron radiations from cosmic rays, alpha particles from packaging materials, and environmental/design variations are common causes of perturbations of digital circuit's nodes that manifest as current pulses of very short duration. If this happens in the hold state of a memory cell or in a flip-flop, the content of the storage element is flipped, causing a soft error. This model is referred to as Single-Event Upset (SEU) and represents the target error model of this work. Perturbations can also cause a glitch in a combinational node of a circuit causing a Single-Event Transient (SET). If a SET is latched into a sequential logic unit, it then manifests as a SEU. SETs have been considered for long time negligible due to different natural masking effects [38], but are becoming a significant source of errors as technology nodes scale down below 100 nm [38], [39].

3.2 Soft-Error Rate and System Modeling

The raw soft-error rate of an electronic component, also denoted with $\lambda_{comp}(t)$, is the rate at which the device encounters or is predicted to encounter soft errors. Vendors express the SER either as number of failures-in-time (FIT) or as mean-time-between-failures (MTBF). SER can be used together with a probability distribution to define a *reliability function* $\mathcal{R}(t)$ and a *failure function* $\mathcal{F}(t)$ providing, respectively, the probability of no error (success) and failure of the component before time t [40]. Given a constant raw error rate λ_{comp} the exponential distribution is a good approximation to model the reliability of an electronic device

$$\mathcal{R}(t) = e^{-\lambda_{comp} t}, \tag{1}$$

$$\mathcal{F}(t) = 1 - \mathcal{R}(t) = 1 - e^{-\lambda_{comp} t}. \tag{2}$$

Other distributions such as the Weibull distribution or log-normal distribution can be used when raw error rates are not constant. In practice, λ_{comp} is small enough to reasonably allow considering these two probabilities as constants over a period of time as short as the execution of a program P. In this paper, T_M (*mission time*) denotes the time during the life of the component at which its reliability is evaluated, and $\mathcal{R}(T_M)$ and $\mathcal{F}(T_M)$ denote its raw probability of success and failure at that time.

Among the different devices that constitute a computer systems, the microprocessor is by far the most critical and complex component. A microprocessor can be split into two set of resources called storage elements and operators. The set $S = \{s_i \mid i \in [1, \#S]\}$ of storage elements includes registers and memory elements where data processed by instructions are stored. The set $OP = \{op_i \mid i \in [1, \#OP]\}$ of operators contains all remaining microprocessor blocks (e.g., control state machines, arithmetical units, branch prediction units, etc.) that are used during the execution of an instruction to process data contained into storage elements. Assuming an equal spatial distribution of failures in a component, the raw error rate of a resource λ_{res} is a portion of λ_{comp} proportional to the fraction of its silicon area A_{res} over the total area of the component A_{comp}: $\lambda_{res} = \lambda_{comp} \frac{A_{res}}{A_{comp}}$

According to the models proposed in this section, by denoting with C_{res} the not-deterministic event "the resource res is correct at time $0 \leq t \leq T_M$," and with NC_{res} its complementary event, then the raw probabilities of success and failure of a resource at T_M are

$$P(C_{res}) = \mathcal{R}(T_M)|_{\lambda=\lambda_{res}} = e^{-\lambda_{comp} \cdot \frac{A_{res}}{A_{comp}} T_M}, \tag{3}$$

$$P(NC_{res}) = \mathcal{F}(T_M)|_{\lambda=\lambda_{res}} = 1 - e^{-\lambda_{comp} \frac{A_{res}}{A_{comp}} T_M}. \tag{4}$$

These two probabilities represent the basis of our reliability estimation model. As a first approximation, the set of events $\{C_{res} \mid res \in (OP \cup S)\}$ can be considered independent. Dependencies are in fact introduced by the execution of the program, and they will be taken into account in the model by the introduction of specific heuristics.

4 Software Reliability Estimation

In the context of this work, estimating the failure probability of a computer system running a program P means estimating the probability of observing an error in the outcome of the program (assumed bugs-free) running on a hardware system affected by soft errors only. Soft errors in the hardware may

be masked either because they affect idle resources, or because the program's execution somehow overwrites the error. Based on this assumption, this section introduces an analytical model to estimate the probability of success of a program in presence of soft errors in the hardware.

4.1 Analytical Model

Programs are analyzed using the concept of *program trace*s.

Definition 1. *A program trace is an ordered sequence of k instructions (k-tuple) executed while running a program P: $T = \langle I_1, I_2, \ldots, I_k \rangle$ with $I_i \in ISA, \forall 1 \leq i \leq k$ (ISA identifies the Instruction Set Architecture of the target microprocessor).*

Program traces are a general concept used with many variants in software engineering whenever the sequence of instructions executed by a program must be recorded or statically computed to perform further analysis [41].

The probability of success of a trace T can be predicted by computing the probability of a correct program's outcome during the execution of each instruction of the trace. The outcome of the program is usually a portion of the entire state of the system. This is modeled introducing the concept of *active state*.

Definition 2. *The active state of a program P during the execution of an instruction $I \in T$, denoted with $A_I^T \subseteq S$, is the set of storage elements representing the outcome of the program when executing the instruction I.*

Programs whose output is evaluated only once at the end of the execution have a not-empty active state only for the last instruction of the trace; programs whose output is continuously evaluated have a not-empty active state for each instruction of the trace. The definition of the active state is application dependent and usually defined by the programmer. In the worst case scenario, the complete state of the system can be considered as active.

The execution of each instruction of a trace may propagate or mask errors among resources, thus modifying their raw probability of success defined in (3). This propagation must be evaluated by modeling the way executed instructions react to soft errors in the hardware. An instruction $I \in T$ can be modeled as a triplet $I = \langle OUT_I, \mathcal{OP}_I, \mathcal{IN}_I \rangle$ where:

- $OUT_I = \{out_1, \ldots, out_z \mid out_i \in S\}$ is the set of storage elements updated by the instruction (outputs),
- $\mathcal{OP}_I : OUT_I \longmapsto O \subseteq OP$ is a function that defines, for each output, the set O of operators required for its computation, and
- $\mathcal{IN}_I : OUT_I \longmapsto J \subseteq S$ is a function that defines, for each output, the set J of storage elements (operands) required for its computation.

All instructions include the program counter into OUT_I since the execution of an instruction always updates this register. Errors in the control flow of the program can be considered including the program counter into the active state.

Let us denote with ex a stochastic variable indicating the execution of the instruction $I_{ex} \in T$ and with $f \in [1, k]$ (k denotes the number of instructions of the trace) a stochastic variable indicating that a soft error occurs in the hardware during the execution of the instruction $I_f \in T$. The probability of success of each storage element $s \in S$ given that $ex = f$, i.e., a soft error manifests in the hardware while the instruction is executed, can be computed as follows:

1. the initial probability of success of each storage element $s \in S$ ($P'(C_s)$), is the probability of an error-free resource (event C_s) or a resource with an error (event NC_s) that is masked by the hardware

$$\begin{aligned} P'(C_s) &= P(C_s \cup (M_s \cap NC_s)) \\ &= P(C_s) + P(M_s)P(NC_s). \end{aligned} \tag{5}$$

   M_{res} is the event: "an error in res is masked" and $P(M_{res})$ is the error masking probability of the resource;
2. the final probability of success of each output $s \in OUT_{I_f}$ of the instruction ($P(C_s \mid ex = f)$), is computed considering that an output is correct if: a) all operators required for its computation are error free or able to mask the error (this event is denoted with $C_{\mathcal{OP}_{I_f}(s)}$ and its probability defined in (8)) and b) all operands required for the computation are error free (events C_{in}, $\forall in \in \mathcal{IN}_{I_f}(s)$) or able to mask the error. This is formally expressed in the following equation:

$$\begin{aligned} P(C_s \mid ex = f) &= P\left(C_{\mathcal{OP}_{I_f}(s)} \bigcap \left\{ \left[\bigcap_{\forall in \in \mathcal{IN}_{I_f}(s)} C_{in} \right] \right.\right. \\ &\quad \left.\left. \bigcup \left[DM_{I_f} \bigcap \left(1 - \bigcap_{\forall in \in \mathcal{IN}_{I_f}(s)} C_{in} \right) \right] \right\} \right) \\ &= P(C_{\mathcal{OP}_{I_f}(s)}) \cdot \left\{ \prod_{\forall in \in \mathcal{IN}_{I_f}(s)} P'(C_{in}) \right. \\ &\quad \left. + P(DM_{I_f}) \cdot \left(1 - \prod_{\forall in \in \mathcal{IN}_{I_f}(s)} P'(C_{in}) \right) \right\}, \end{aligned} \tag{6}$$

   where DM_{I_f} represents the event: "the execution of I_f masks an error in one of its operands" and $P(DM_{I_f})$ is the probability that an instruction masks an error in its operands. This probability can be computed either with fault injection experiments or, as explained later in this paper, by an analytical analysis of each instruction; and
3. the final probability of success of all storage elements not in the output set of the instruction ($\forall s \in S - OUT_{If}$) is not affected by the execution and is computed as follows:

$$P(C_s \mid ex = f) = P'(C_s). \tag{7}$$

$P(C_{\mathcal{OP}_{I_f}(s)})$ used in (6) denotes the probability of success of all operators used to compute the resource s. Similarly to (5), it can be computed as the probability of respecting, for all considered operators, the following conditions: 1) the

operator is error free or, 2) the operator manifests an error but the error is masked. This can be formalized as follows:

$$P(C_{\mathcal{OP}_{I_f}(s)}) = P\left(\bigcap_{\forall op \in \mathcal{OP}_{I_f}(s)} [C_{op} \cup (M_{op} \cap NC_{op})]\right) = \prod_{\forall op \in \mathcal{OP}_{I_f}(s)} [P(C_{op}) + (P(M_{op}) \cdot P(NC_{op}))]. \quad (8)$$

With this model, the contribution of a fault tolerant operator to (8) is: $P(C_{op}) + (1 \cdot P(NC_{op})) = \mathcal{R}_{op}(T_M) + (1 - \mathcal{R}_{op}(T_M)) = 1$. This correctly models that the fault tolerance mechanism resets the contribution of this operator to the error probability of other resources.

Similarly to the case $ex = f$, the probability of success of each resource at the end of the execution of an instruction $I_j \in T$ following I_f (i.e., $ex = j > f$) is computed taking into account that an error in one of the operands can be propagated to one of the outputs:

1. the probability of success of each storage element not in OUT_{I_j} ($\forall s \in S - OUT_{I_j}$) is constant

$$P(C_s \mid ex = j \ \wedge \ j > f) = P(C_s \mid ex = j - 1). \quad (9)$$

2. the probability of success of each storage element $s \in OUT_{I_j}$ is the probability that all operands of I_j are correct or that at least one operand of I_j is not correct but the error is masked by the execution of the instruction

$$P(C_s \mid ex = j \ \wedge \ j > f) = P\left(\left[\bigcap_{\forall in \in \mathcal{IN}_{I_j}(s)} C_{in}\right] \cup \left[DM_{I_j} \bigcap \left(1 - \bigcap_{\forall in \in \mathcal{IN}_{I_j}(s)} C_{in}\right)\right]\right) = \prod_{\forall in \in \mathcal{IN}_{I_j}(s)} P(C_s \mid ex = j - 1) + P(DM_{I_j}) \cdot \left(1 - \prod_{\forall in \in \mathcal{IN}_{I_j}(s)} P(C_s \mid ex = j - 1)\right). \quad (10)$$

When evaluating the execution of a trace T, soft errors may manifest during any of the k instructions of the trace (i.e., $1 \leq f \leq k$). According to our model, the correctness of a resource $s \in S$ during the execution of an instruction I_j ($ex = j$) depends on the instant the soft error manifests in the hardware (I_f). A set of j error conditions must therefore be analyzed: 1) the error manifests during the first instruction of the trace ($f = 1$), 2) the error manifests during the second instruction of the trace ($f = 2$) and so on until the case ($f = j$). The probability of success of the resource for each error condition can be computed according to (6), (7), (9), and (10). Since all error conditions have the same probability and represent disjoint events, the probability of success of a resource after the execution of a generic instruction I_j can be computed as follows:

$$\begin{aligned} &P(C_s \mid ex = j, \ \forall 1 \leq f \leq j) \\ &= P\left(\left[\bigcup_{x=1}^{j-1} ((f = x) \cap (C_s \mid ex = j \ \wedge \ j > f))\right] \cup [(f = j) \cap (C_s \mid ex = f)]\right) \\ &= \sum_{x=1}^{j-1} \left(\frac{1}{j}\right) \cdot P(C_s \mid ex = j \ \wedge \ j > f) + \left(\frac{1}{j}\right) \cdot P(C_s \mid ex = f). \end{aligned} \quad (11)$$

The jth instruction I_j of a program trace T is considered correctly executed if all storage elements of its active state $A^T_{I_j}$ are error free. Denoting C_{I_j} the nondeterministic event "the instruction I_j is correctly executed," the probability of success of the instruction given that $A^T_{I_j} \neq \oslash$ is defined as

$$P(C_{I_j}) = P\left(\bigcap_{\forall s \in A^T_{I_j}} (C_s \mid ex = j, \forall 1 \leq f \leq j)\right). \quad (12)$$

Computing (12) is not trivial, since the execution of an instruction introduces dependencies among storage elements. Algorithm 1 proposes an heuristic to evaluate these dependencies. It produces a subset of the active state containing independent resources that can be used to compute (12). In Algorithm 1, D is a integer matrix with each row corresponding to an instruction of T and each column to one of the storage elements. The condition D[j][i] $\neq 0$ indicates that, during the execution of the jth instruction, the resource i must be discarded when computing (12) since its contribution has already been taken into account in a different set of resources. On the other hand, D[j][i] $= 0$ denotes that the resource must be considered since its contribution was not considered before. When the program starts (j $= 1$) all resources are independent (Algorithm 1, row 2). For a generic instruction I_j, the status of each resource in D is initially set to those of the previous instruction (Algorithm 1, row 5), and then the outputs of the instruction are considered. The overall idea is that each output already includes the contribution of the corresponding operands that can therefore be excluded from the set of resources to consider (Algorithm 1, row 14-16). If more than one output is computed based on the same set of operands, only one of these outputs must be considered (Algorithm 1, row 18-25). Whenever a storage element is written, the operands used during the last instruction targeting the same resource must be considered again (Algorithm 1, row 14-16). The array LW stores, for each storage element, the index of the last instruction of the trace where the resource was written. Algorithm 1 is an approximated approach to take into account dependencies among resources; however, the experimental results of Section 5 will show that it is able to provide estimations with a reasonable level of confidence.

Algorithm 1. Algorithm to compute the subset of independent resources for an instruction

Require: j: index of the evaluated instruction

1: **if** j $= 1$ **then**

2: D[j] $= (0, \ldots, 0)$

```
    LW = (0, ..., 0)
else
    D[j] = D[j − 1]
end if
for i = 1 to count(OUT_{I_j}) do
    s = OUT_{I_j}[i]
    if LW[s]<>0 then
        for all r in IN_{I_LW[s]}(s) do
            D[j][r] = D[j][r] − 1
        end for
    end if
    for all r in IN_{I_j}(s) do
        D[j][r] = D[j][r] + 1
    end for
    LW[s] = j
    D[j][s] = 0
    for k = 1 to i − 1 do
        x = OUT_{I_j}[k]
        if IN_{I_j}(x) = IN_{I_j}(s) then
            D[j][s] = 1
            break
        end if
    end for
end for
```

Based on Algorithm 1, the probability expressed in (12) can be estimated as follows:

$$P(C_{I_j}) \cong \prod_{\forall s \in A^T_{I_j} | \mathrm{D[j][s]}=0} P(C_s \mid ex = j, \forall 1 \leq f \leq j). \quad (13)$$

Given that (13) provides the probability of success of each instruction of a trace, the probability of success of the full trace T ($P(C_T)$) can be approximate as the average probability of success of those instructions characterized by a not empty active state

$$P(C_T) \cong \frac{1}{count(I_i \mid A^T_{I_i} \neq \oslash)} \sum_{\forall I_i | A^T_{I_i} \neq \oslash} P(C_{I_i}). \quad (14)$$

Several traces can be generated by the execution of a program, depending on the specific workload. Let us denote with $\mathcal{T}_P$ the complete set of possible traces of a program P, with each trace $T \in \mathcal{T}_P$ an independent event characterized by an execution probability $P(T)$ and $\sum_{\forall T \in \mathcal{T}_P} P(T) = 1$. The probability of success of the program P ($P(C_P)$), can be computed as a weighted average of the probability of success of each trace

$$P(C_P) = \sum_{\forall T \in \mathcal{T}_P} P(T) \cdot P(C_T). \quad (15)$$

Generating the full set of traces of a real application is obviously often not feasible in a reasonable computational time. A subset of all possible traces (TS), must therefore be sampled in order to statistically represent a significant group of execution alternatives. The more traces are sampled, the better (15) will estimate the reliability of the system as the probability of success of the program in presence of soft errors in the hardware. In order to take into account the contribution of all traces not included in TS, (15) can be rewritten as follows:

$$P(C_P) \cong \sum_{\forall T \in TS} (P(T) \cdot P(C_T)) + \left(1 - \sum_{\forall T \in TS} P(T)\right) \cdot \mathcal{R}(T_M)|_{\lambda=\lambda_{comp}}. \quad (16)$$

The first portion of (16) computes (15) on TS. The second portion of the formula considers that in all situations not included in TS the probability of success of the program can be approximated to the worst case represented by the raw reliability of the component defined in (1).

4.2 Program Traces Generation

Two approaches can be followed to obtain a relevant set of traces for the proposed reliability estimation model.

Whenever a strong, statistically relevant set of inputs for the target software is available, it can be exploited to derive a corresponding set of traces. Several runs of the program are executed, each with a different input, and runtime information about executed instructions and accessed data are recorded to compose each trace. The probability assigned to each trace ($P(T)$ in (16)) can be uniformly distributed or calculated based on the knowledge of the probability of occurrence of the corresponding inputs. However, in several situations in which very early design exploration is performed, a statistically relevant set of inputs might not be available, or it might be difficult to estimate how much it covers the set of possible executions. For these situations, this paper presents an algorithm that generates a set of traces by performing a static analysis of the program's binary code. The goal of this algorithm is to cover as many parts as possible of the control-flow graph of the application, providing also a metric to measure how many of the possible paths have been covered.

The control-flow graph of a program P, is a labeled directed graph $CFG_P = (Instr, A, L)$ where:

- $Instr = \{I_i \mid I_i \in ISA\}$ is the set of nodes of the graph, with each node representing a single instruction of the program,
- $A = \{(I_i, I_j) \mid I_i, I_j \in ISA\}$ is the set of arcs modeling allowed sequences of instructions, and
- $L : A \rightarrow labels$ is a function that maps each arc to a label.

Each CFG has two special nodes denoted with I_{start} and I_{end} representing the entry point and the exit point of the program. Multiple exit points are connected to a single node. In our model, the label of an arc (I_i, I_j) is the probability $p_{i,j}$ of crossing the arc during the execution of the program. $p_{i,j}$ can be assigned applying the following policies:

1. If I_j is the only direct successor of I_i, and therefore it is not a branch instruction, the probability of crossing the arc (I_i, I_j) is equal to 1;
2. If I_i has m direct successors on the graph, i.e., there are m arcs directed from I_i, and no runtime information about the probabilities of crossing each arc is available, then each arc is assigned a probability equal to $\frac{1}{m}$; and
3. If I_i has m direct successors, and from the knowledge of the program or from runtime information it is possible to conclude that some of the arcs are less probable than others (e.g., arcs that terminate the

program in case of errors), custom probabilities can be assigned given that the sum of the probabilities of the arcs directed from the node must be equal to 1. Variable probabilities can be also assigned, modeling for instance loops that start with a high probability that decreases when the number of iterations increases.

The control-flow graph of a program can be automatically generated by statically analyzing its binary code with tools such as Diablo [42].

A modified depth-first search algorithm on the CFG of the program named Traces Generation Algorithm (TGA) is used to statically compute a set of execution traces (Algorithm 2). The main problem of this approach is that, in the case of loops, the number of traces that can be generated is theoretically infinite. A set of terminating conditions is therefore introduced to stop the generation either when the computed traces provide the desired coverage of the CFG, or when a maximum number of traces has been generated.

Algorithm 2. Traces Generation Algorithm
Require: TS $\leftarrow \oslash$,
TARGET_CEP $\leftarrow$ [0, 1],
MAX_T,
CEP $\leftarrow 0$,
STOP_IF_ALL_ARCS_COVERED $\leftarrow$ {true,false}

```
1: TGA (node = I_start, T = ∅, prob = 1)
2: T ← T ∪ node
3: if node = v_end then
4:     TS ← TS ∪ T
5:     CEP ← CEP + prob
6:     mark all arcs traversed by T as visited
7:     if all_arc_visited AND
       STOP_IF_ALL_ARCS_COVERED=true then
8:         exit
9:     end if
10:    if |TS| = MAX_T then
11:        exit
12:    end if
13:    if CEP >= TARGET_CEP then
14:        exit
15:    end if
16:    return
17: else
18:    for all I in directed_successors(node) do
19:        newprob ← newprob *p_{node,v}
20:        TGA (v,T,newprob)
21:    end for
22: end if
```

TGA is a recursive algorithm that requires the following set of global variables:

- TS: the set of generated traces. It is an empty set when the algorithm starts;
- TARGET_CEP: according to (15), each trace is associated with an execution probability $P(T)$. If all possible traces of a program can be generated, their cumulative execution probability (CEP) is equal to 1, thus guaranteeing the full coverage of all execution paths. When instead, the number of possible traces is theoretically infinite, TARGET_CEP is the minimum cumulative execution probability that has to be reached before stopping the trace generation. This value is also used as a metric of the completeness of the generated set of traces;
- MAX_T: is an upper bound on the number of generated traces. It forces the algorithm to stop even if TARGET_CEP has not been reached; and
- STOP_IF_ALL_ARCS_COVERED: if set to true, this flag allows to stop the generation when all arcs of the CFG have been traversed at least once. This represents the minimum set of traces that must be considered to analyze a program. It can be used for early and very fast evaluations.

TGA begins the generation considering the starting node I_s and an empty trace T with execution probability equal to 1 (Algorithm 2, row 1). It adds the current node to the trace (Algorithm 2, row 2) and then checks if the current node corresponds to I_{end} to detect whether the end of a trace has been reached (Algorithm 2, row 3).

In case the current trace is not complete (Algorithm 2, rows 18-21), the algorithm selects iteratively each direct successor of the current node and, for each corresponding arc, it generates a new trace by recursively calling itself (Algorithm 2, row 20). The probability of the new trace is the product of the current probability by the probability of execution of the arc (Algorithm 2, row 19).

In case the current trace is complete (Algorithm 2, rows 4-16), it is added to the set TS (Algorithm 2, row 4). CEP (Algorithm 2, row 5) and the set of arcs traversed at least once (Algorithm 2, row 6) is updated. The different terminating conditions are then evaluated. Rows 7-9 terminate the generation if all arcs have been traversed at least once and STOP_IF_ALL_ARCS_COVERED is set to true. Rows 10-12 stop the generation if MAX_T traces have been generated and, finally, rows 13-15 stop the generation if target TARGET_CEP has been reached. If none of these conditions are true, the generation continues exploring additional paths on the graph.

Listing 1 shows a simple example of a program, coded for the Intel 8088 microprocessor, counting the number of elements of an array. Items are stored in memory at address 0100 h and range boundaries are passed through the stack. The program loops until all items are evaluated (CX is used to count the number of items in the array passed in the stack). In order to simplify the example, the program omits any context saving operation.

```
1  pop cx            ;counter
2  pop ax            ;get upper & lower
3  pop bx            ;limits in ax & bx
4  mov si, 0         ;number counter
5  mov di, 0100h     ;get initial location
6  lp: mov dx, word ptr [di] ;get the content
7  cmp dx, ax        ;check the upper
8  jle lw            ;if number is lower
9  jmp nxt           ;if number is larger
10 lw: cmp dx, bx    ;check lower limit
11 jge lim           ;if number is larger
12 jmp nxt           ;if number is lower
13 lim: inc si       ;increment counter
14 nxt: add dx, 2    ;get next location
15 loop lp           ;repeat until items
```

Listing 1. Intel 8088 example program.

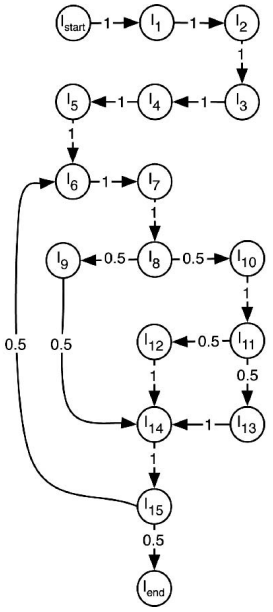


Fig. 1. Control-flow graph statically computed from the binary code of Listing 1.

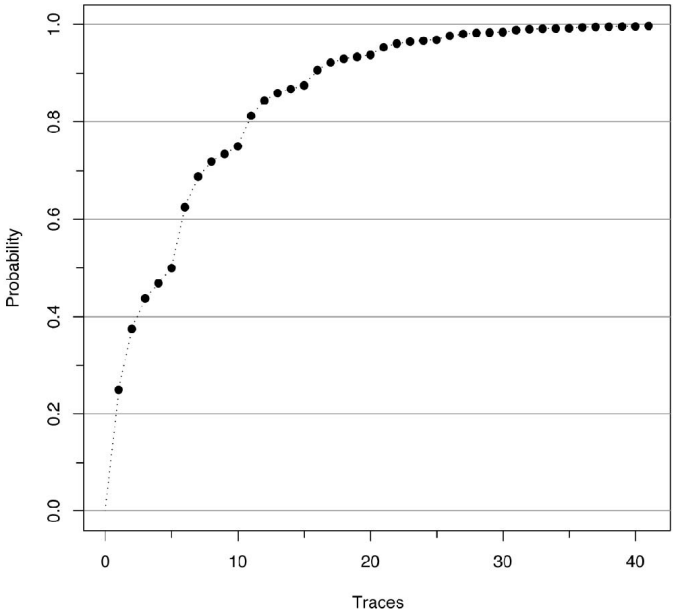


Fig. 2. Plot of the cumulative trace execution probability versus the number of generated traces.

The CFG of the program is summarized in Fig. 1. No runtime information about the probability of traversing each arc is available. In case of branches, all arcs directed from the node have been assigned with the same execution probability (policies 1 and 2). The CFG shows different paths and a loop.

By executing Algorithm 2 with STOP_IF_ALL_ARCS_COVERED set to true, the following set of traces is generated:

- $$T1 = <I_{start}, I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8, I_{9,}I_{14}, I_{15}, I_{end}> \quad (P_{T1} = 0.25)$$
- $$T2 = <I_{start}, I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8, I_{10}, I_{11}, I_{12}, I_{14}, I_{15}, I_{end}> \; (P_{T2} = 0.125)$$
- $$T3 = <I_{start}, I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8, I_{10}, I_{11}, I_{13}, I_{14}, I_{15}, I_{end}> \; (P_{T3} = 0.125)$$
- $$T4 = <I_{start}, I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8, I_{9,}I_{14}, I_{15}, I_6, I_7, I_{8,}I_9, I_{14}, I_{15}, I_{end}> \; (P_{T4} = 0.0625).$$

This set allows to reach a CEP equal to 0.5625. Fig. 2 plots how CEP increases by increasing the number of generated traces. By traversing the loop multiple times, i.e., arc (I_{15}, I_6), additional execution alternatives can be evaluated reaching, with about 40 traces, a CEP almost equal to 1.

5 Experimental Model Validation

This section presents the experimental setup used to validate the model proposed in the previous sections. It covers three main aspects: the microprocessor characterization, the statistical reliability estimation, and the validation and discussion of the results.

5.1 Microprocessor Characterization

The microprocessor characterization is a key operation that must be performed only once, independently from the program that will be executed in the system. Two microprocessor cores have been characterized in this paper: the Intel 8088 and the OpenRISC1200.

The Intel 8088 (hereinafter referred to as 8088) has the same architecture of the more famous Intel 8086 with the only difference being that the external data bus width is reduced from 16 to 8 bit. It is a CISC microprocessor with a very simple two-stage pipeline. It is equipped with 16-bit registers grouped as follows: four general purpose registers (AX, BX, CX, DX) also accessible as eight 8-bit registers; four memory indexing registers (stack-pointer SP, base-pointer BP, source-index SI, destination-index DI); four segment registers (code segment CS, data segment DS, stack segment SS, extra segment ES) and two registers for controlling the execution flow (program counter PC, status flags SF). The 8088 ISA contains 111 instructions without floating-point support. The microprocessor model is provided by the HT-LAB toolkit [43]. This toolkit, distributed under the GNU license, includes the VHDL code of a complete 8088-based system: the processor, the ROM and the RAM, some peripheral devices, and a set of facilities to convert assembly code in a format that can be directly included and executed in the VHDL code.

The OpenRISC1200 (hereinafter referred to as OR1200) is a 32-bit scalar RISC microprocessor with Harvard architecture and five stage integer pipeline. It has 32 general purpose 32-bit registers, caches, virtual memory support, and basic DSP functions. It supports the ORBIS32 instruction

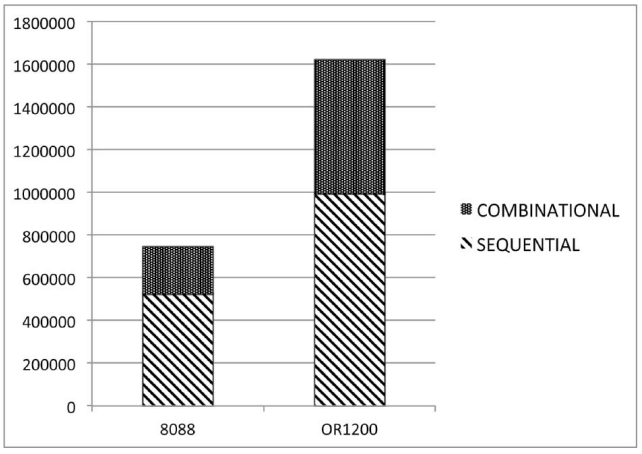


Fig. 3. Microprocessors area summary provided in equivalent gates.

set for a total of 215 instructions. The instruction set includes 32-bit integer instructions, basic DSP instructions, 32-bit Load and Store instructions, program control flow instructions, and some special instructions. The VHDL model of the OR1200 is freely available on the OpenCores website (http://www.opencores.org).

Both processors have been synthesized using Synopsis Design Compiler with the AMS 350 nm technology library. The choice of the target library could lead to small fluctuations in the reliability results, but this issue is beyond the scope of this paper. Fig. 3 provides a summary of the area occupation of the two cores that gives an idea of the complexity of the two microprocessors. The 8088 accounts for a total of 652 flip-flops while the OR1200 accounts for a total of 1,891 flip-flops, all of them considered as potential target SEU locations.

The microprocessor characterization process estimates the masking probabilities $P(C_{\mathcal{OP}_I})$ and $P(DM_I)$ required to compute (6) and (10).

The most efficient way to compute $P(C_{\mathcal{OP}_I})$ is to setup a fault injection campaign. For each instruction <INSTR> of the ISA, and for each possible combination of operands, fault injection has been performed with the microprocessor executing a simple program composed of the target instruction preceded and followed by a set of NOP instructions. This solution guarantees that <INSTR> traverses all stages of the pipeline (two for the 8088 and five for the OR1200) without other instructions interfering with its operations. The same instruction is simulated several times with different operands in order to explore different execution conditions. An average of 10,000 SEUs for the 8088 and 30,000 SEUs for the OR1200 has been injected for each variant of each instruction. At this stage, fault injection only targets operators and it does not include flip-flops associated with operands and output registers that will be considered instead when estimating $P(DM_I)$. Obviously, the size of the fault list, and therefore the length of the fault injection experiment, heavily depends on the number of registers and instructions of the microprocessor.

The effort required to perform this part of the characterization can be extremely variable depending on several factors. The most important ones are the available computational resources, which impact the time required to perform the experiments and elaborate the results, the level of detail of the microprocessor model, which directly affects both the confidence in the generated fault list and the reliability of its injection, and the chosen fault injection mechanism (hardware, software, or simulation based), which determines the cost and precision of the injection results. Nevertheless, it is worth reminding that having to consider only the microprocessor without any workload but the individual instructions of its ISA, the fault list generation is faster, easier, and more complete because it can be exhaustively generated with a simple software program.

Differently from $P(C_{\mathcal{OP}_I})$, $P(DM_I)$ can be analytically computed by analyzing the behavior of each instruction without the need of performing VHDL simulations. Instead, a set of C programs exhaustively performs this analysis simulating the behavior of each instruction in presence of faults in its operands. Let's take as an example two instructions, ADD and CMP (compare) for the 8088:

- ADD computes the sum of two 16-bit operands and stores it into a new 16-bit word. Regardless of their value, any error in one of the operands will generate an error in the output result. $P(DM_{ADD})$ is therefore equal to 0.
- CMP compares two 16-bit operands. The result in this case heavily depends on the value of the compared data. By analyzing all possible combinations and errors, a $P(DM_{CMP}) = 0.95$ is obtained. This means that 95 percent of the errors in the operands will be masked by the instruction itself.

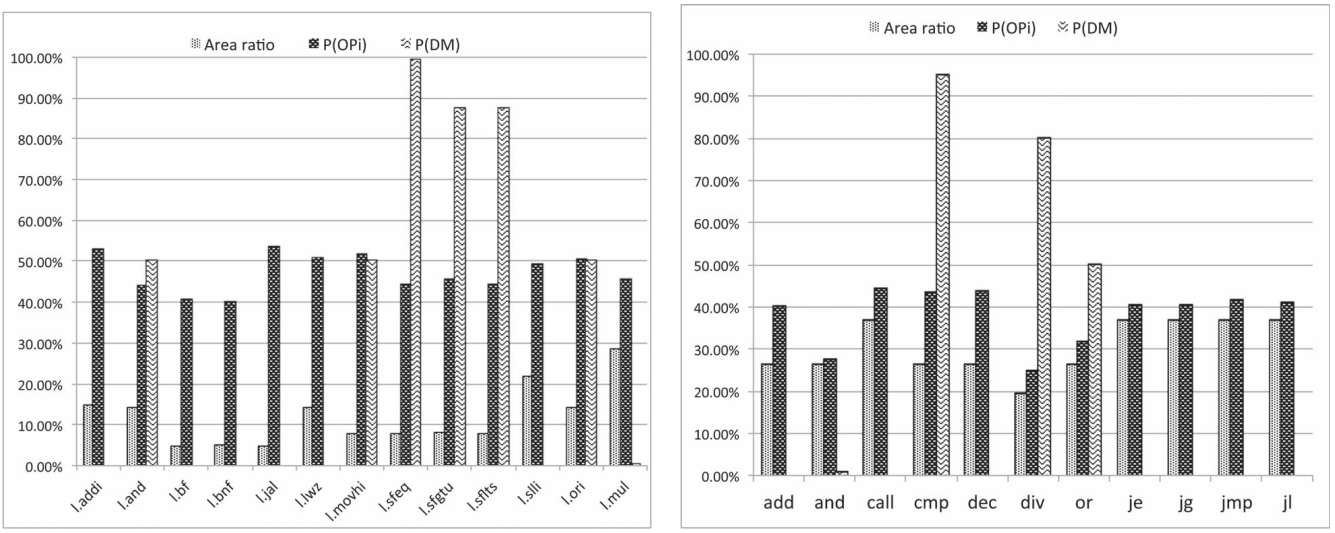
Fig. 4 reports an example of the characterization of a subset of instructions for the two considered processors. For each instruction, the figure reports: 1) the operators area ratio (i.e., the number of flip-flops of the used operators over the total number of flip-flops) required to compute (3) and (4) for the operators, 2) the overall operators masking probability $P(C_{\mathcal{OP}_I})$, and 3) the data masking probability $P(DM_I)$. The figure highlights that the 8088 has a lower capability of masking errors in the hardware compared to the OR1200. As shown in the following sections, this will negatively reflect on the reliability at the system level.

5.2 Experiments and Validation

Experiments have been conducted on three application programs, two of them (QSORT and AES) obtained from the MiBench Ver. 1.0 benchmarks [44]:

1. *HUFFMAN*: performs Huffman encoding applied to a list of 16 symbols. The result is the Huffman code associated with each symbol.
2. *QSORT*: sorts a given array of integer numbers stored in the main memory using the quick sort algorithm.
3. *AES*: performs AES encryption of a 138-Bytes message.

The reliability of the two microprocessors while running these three benchmarks has been assessed both by applying the proposed estimation model, and by executing a very extensive fault injection campaign aimed at confirming the estimated results. In order to reduce the complexity of the fault injection experiments, the three benchmarks do not contain I/O instructions and all input data are predefined and stored in the RAM along with the program's binary code.



(a) OR1200 (b) 8088

Fig. 4. Characterization of a subset of instructions of the (a) OR1200 and (b) 8088 microprocessors.

All experiments have been performed on a workstation equipped with a dual Intel Xeon@3.16 GHz quad core processor and 32 GB of RAM.

The proposed estimation model has been coded in a C program. The tool includes a library of parsers for the assembly language of the two considered microprocessors, and it implements a multithread architecture to fully exploit the parallelism offered by the available workstation. Experiments only focused on faults in the microprocessor. For this reason, each instruction that writes data outside the microprocessor (e.g., any instruction storing data in the memory) has a not empty active state. When generating program traces using Algorithm 2, the knowledge of the source code is used to assign custom variable probabilities to the different branch instructions. Several golden runs of the program with random data have been performed to obtain an estimation of the most probable branches of the program. This makes it possible to reduce the number of traces required to reach the desired CEP level.

The fault injection campaign has been performed resorting to a custom fault simulation environment developed at LIRMM [45]. To fairly compare performances, the fault injector proposed in [45] has been extended to allow multiprocess simulations. Fault injection experiments have been setup as follows:

1. The overall system, including the microprocessor, RAM, ROM, etc., is simulated and all activities over the primary inputs and outputs of the processors (control signals, data, and address buses) are logged in an external file. A table mapping each instruction of the target program to its execution time expressed in terms of clock cycle is also generated.
2. Fault simulation of SEUs in the microprocessor's flip-flops is performed while applying the inputs stored in step 1. An exhaustive fault injection campaign is performed. All possible clock cycles as well as all possible flip-flops have been considered as target fault locations. This makes it possible to reach 100 percent of confidence in the simulated results.
3. A report is generated starting from the results of the fault simulations. This report summarizes, for each instruction, how many faults are detected and how many are masked.

Fig. 5 compares the performance of the proposed method compared to fault injection in terms of CPU time. Results are provided in hours of CPU time using a logarithmic scale. It is evident how the proposed model outperforms fault injection, reducing the computation time by several orders of magnitude. When considering the fault injection campaign, the 8088 is the most critical core in terms of computation time. This is due to the fact that instructions of the 8088 ISA usually require multiple clock cycles to be executed, strongly increasing the simulation time of the synthesized core. For the proposed method the situation is instead inverted. The computation time is mainly affected by the number of instructions composing the program. The

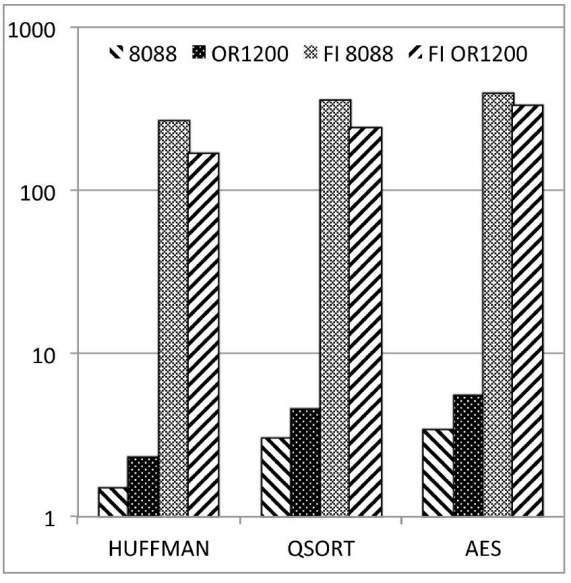


Fig. 5. Comparison of the computation time between the proposed model and the fault injection analysis. Time is expressed in hours of CPU and reported using a logarithmic scale.

TABLE 1
Summary of the Experimental Setup

Processor	Benchmark	Instructions	CC	Traces	CEP
8088	HUFFMAN	285	13,273	~1,000	0,95
	QSORT	112	26,891	~5,000	0,95
	AES	3,163	32,652	~2,000	0,95
OR1200	HUFFMAN	453	1,405	~1,000	0,95
	QSORT	180	2,301	~5,000	0,95
	AES	5,758	7,606	~2,000	0,95

OR1200, which like all RISC processors only implements simple instructions, requires more instructions to code a program (see Table 1), therefore making the reliability analysis more time consuming.

Fig. 6 compares the storage requirement for the two methods. Storage requirement is reduced compared with fault injection, especially considering the 8088 whose fault injection requires saving long simulations in terms of clock cycles. Overall, the amount of data to store is quite small and does not represent a critical issue for the analysis.

To conclude the description of the experimental setup, Table 1 summarizes the information about the complexity of the different benchmarks in terms of number of instructions and average number of clock cycles for an execution (CC). It also reports the number of traces that have been simulated along with the reached CEP. For all experiments the trace generation algorithm has been executed with a $\mathrm{TARGET\_CEP} = 0.95$.

5.3 Results

Fig. 7 proposes six plots that summarize the results of the reliability analysis performed on the six case studies. Each plot reports the following three curves:

- *Raw rel. fun.*: it is the raw reliability function of the microprocessor computed according to (1) for a mission time T_M of six years and an error rate $\lambda = 0.019 \cdot 10^{-6}$ for both the 8088 and the OR1200 (the specific value of λ characterizes the technology and does not influence the accuracy of the prediction).
- *FI based rel. fun.*: it is the reliability function estimated considering the results of the fault injection as: $\mathcal{R}(T_M) = e^{-\lambda T_M} + (1 - e^{-\lambda_{8088} T_M}) \cdot P_{mask}$. This function takes into account the probability of having a fault free device at time T_M, and the probability of having a faulty device whose error is masked with a given probability. The masking probability for the given benchmark, reported in Table 2, has been computed according to the fault injection results as the number of masked faults over the total number of injected faults.
- *Estimated rel. fun.*: it is the reliability function estimated with the proposed model considering different values of T_M.

Fig. 7 clearly shows that the estimated reliability function is in general able to approximate the fault injection based reliability function, thus confirming the capability of the proposed method to efficiently estimate the reliability of the target microprocessor considering the running program. This can be better appreciated looking at Fig. 8 that reports

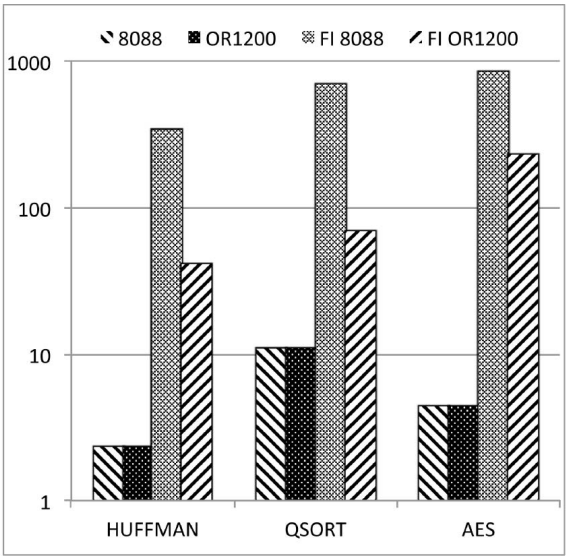


Fig. 6. Comparison of the storage requirement between the proposed model and the fault injection analysis. Results are expressed in MBytes using a logarithmic scale.

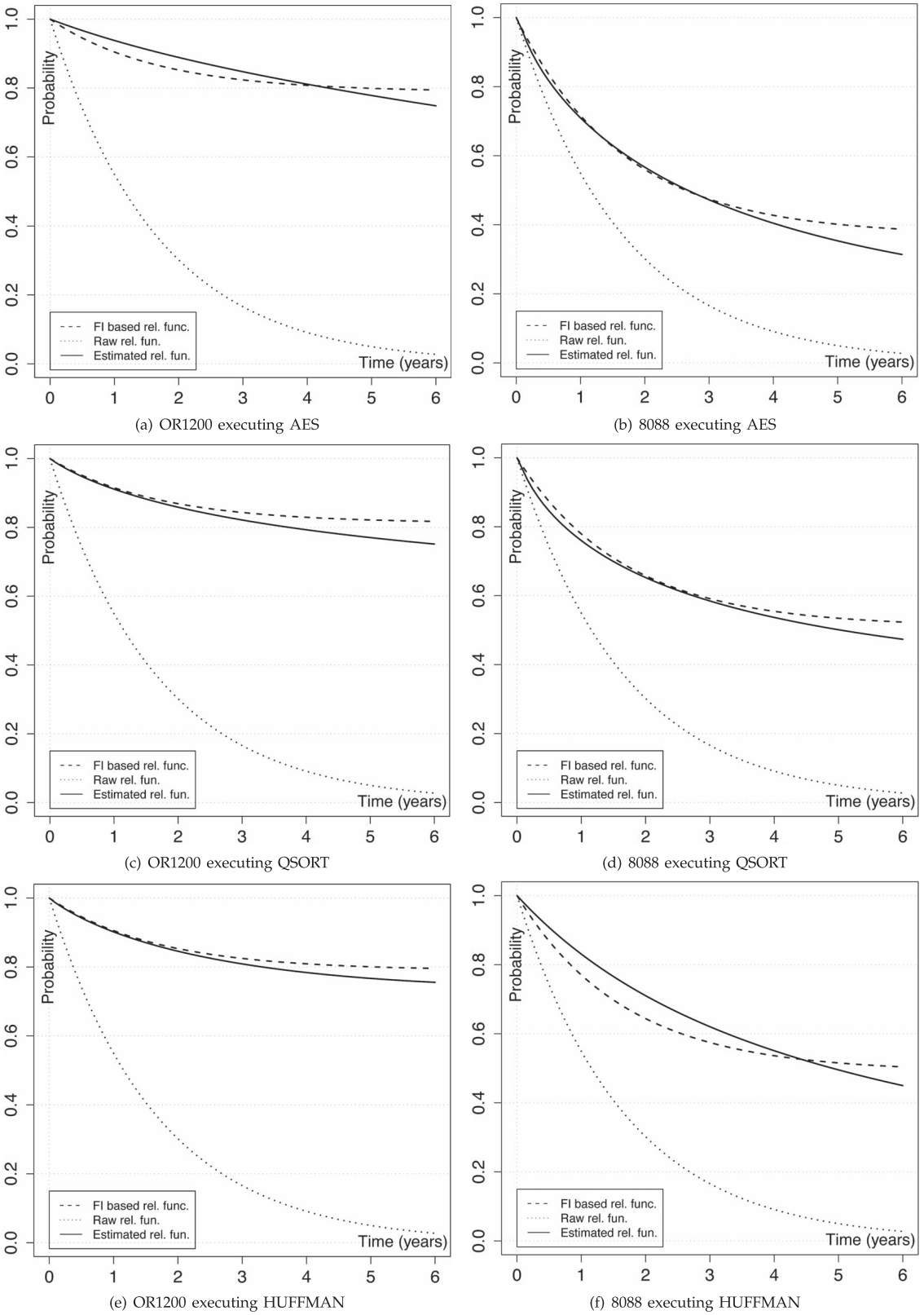
the error between the estimated reliability function and the fault injection curve. With a reasonable number of traces this error is always lower than 7 percent guaranteeing a good confidence in the prediction.

Looking at the experimental results, one can notice that the estimation error increases with the increment of the mission time. A portion of this error can be accounted to the approximated characterization of the microprocessor, and to the impossibility of exploring the complete set of possible traces and reaching $\mathrm{CEP} = 1$. However, by analyzing the way our model works, the majority of the error is probably introduced by the heuristic used to take into account resources dependencies. This introduces a certain error in the estimation that becomes evident when the mission time T_M, and consequently the fault probability of the single resources, increases.

To conclude, Fig. 9 shows how the proposed method can be used to perform very fast early design exploration. It reports the estimated reliability function of the 8088 running HUFFMAN with four different fault tolerant configurations of the microprocessor:

1. all ALU's internal flip-flops are fault-tolerant (ALU-FT),
2. all microprocessor's user registers are fault-tolerant (REG-FT),
3. both the ALU and the registers are fault-tolerant (ALU+REG-FT), and
4. all resources of the microprocessor are fault-tolerant (ALL-FT).

The four configurations can be easily analyzed by changing the masking probabilities of the different resources. Even if working with a simple microprocessor, Fig. 9 clearly demonstrates the potential of the proposed tool. In this specific case study, introducing a fault tolerant ALU has a minimal impact on the overall reliability of the system (8088 versus ALU-FT), while protecting the registers provides a major improvement. Although this is somehow expected, it is interesting to note that protecting the whole processor provides a minimal improvement in the overall reliability



(a) OR1200 executing AES (b) 8088 executing AES

(c) OR1200 executing QSORT (d) 8088 executing QSORT

(e) OR1200 executing HUFFMAN (f) 8088 executing HUFFMAN

Fig. 7. Result of the reliability analysis for the two microprocessors running the three considered benchmarks. Each plot shows the raw reliability function of the microprocessor (raw rel. fun.), the reliability function estimated through fault injection (FI based rel. fun.) and the reliability function estimated with the proposed model (Estimated rel. fun.).

with respect to protecting only the registers. It is worth remembering here that, according to (16), even if all resources of the processor are fault tolerant the estimated probability decreases with T_M every time the CEP of the generated traces is not equal to one, confirming the estimation of Fig. 9.

6 Future Improvements

The reliability analysis proposed in Sections 4.1 and 4.2 is clearly limited by the complexity of the analyzed software, and in particular by the complexity of the corresponding CFG. This section discusses how this complexity could be managed by exploiting the intrinsic hierarchy of a program.

TABLE 2
Summary of Fault Injection Experiments in Terms of Injected and Masked SEUs

Benchmark	Benchmark	Injected	Masked	Mask Prob
8088	HUFFMAN	8,627,450	4,296,779	~0.49
	QSORT	17,479,150	8,835,476	~0.51
	AES	21,289,114	7,876,973	~0.37
OR1200	HUFFMAN	2,656,855	2,098,916	~0.79
	QSORT	3,560,753	2,892,176	~0.81
	AES	14,382,946	11,362,528	~0.79

Considering the simple CFG including a call to a function F reported in Fig. 10a, the CFG can be clearly partitioned into two portions: 1) the main program, and 2) the function F (gray part of Fig. 10a).

The full portion of the graph modeling the function F can be collapsed into a single node (Node F of Fig. 10b), defining a new high-level instruction. The new instruction will be characterized by a set of input operands including all function parameters, a set of output values corresponding to the return values of the function, and finally, a single operator corresponding to the actual computation performed by the function with a probability of success computed using (15) by considering the CFG of the function in isolation. The state of this instruction will contain both local and global variables of the program, but in general, the active state of the function will include the function return values only. This collapsing technique has the potential to reduce the complexity of the CFG by analyzing portions of it in isolation. This will allow the reduction of the amount and complexity of the traces to analyze, thus allowing the management of very complex applications.

7 Conclusion

This paper proposed a new reliability evaluation methodology targeting microprocessors running a software application. Compared to fault injection, the proposed approach makes it possible to save a considerable amount of time: fault injection is used only once for a one time, reusable, characterization of the microprocessor in terms of probability of success of each of its instructions in the presence of a soft error in the hardware. The overall reliability of the microporocessor running a given workload is then computed with a purely probabilistic approach. The same characterization can then be reused every time the same CPU is used to build a new system or a new application software needs to be evaluated. The proposed method makes it possible to perform early exploration of design alternatives giving the possibility of comparing the system reliability using different processor architectures, even before the actual system's design is available. In the long run, the diffusion of this approach could lead to the availability of libraries of microprocessor characterizations (freely available or proprietary) that would allow users to evaluate the reliability of microprocessor-based systems without the need of neither a single fault-injection

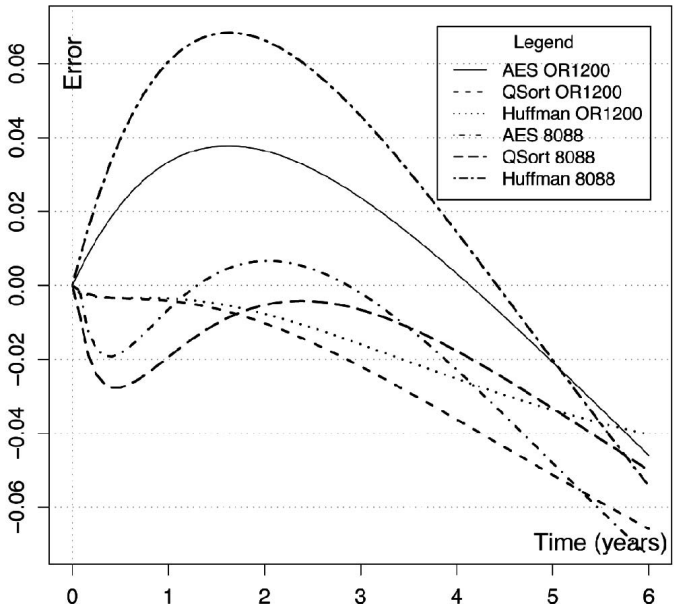


Fig. 8. Plot of the error between the fault injection based reliability function and the estimated reliability function for the considered benchmarks and microprocessors.

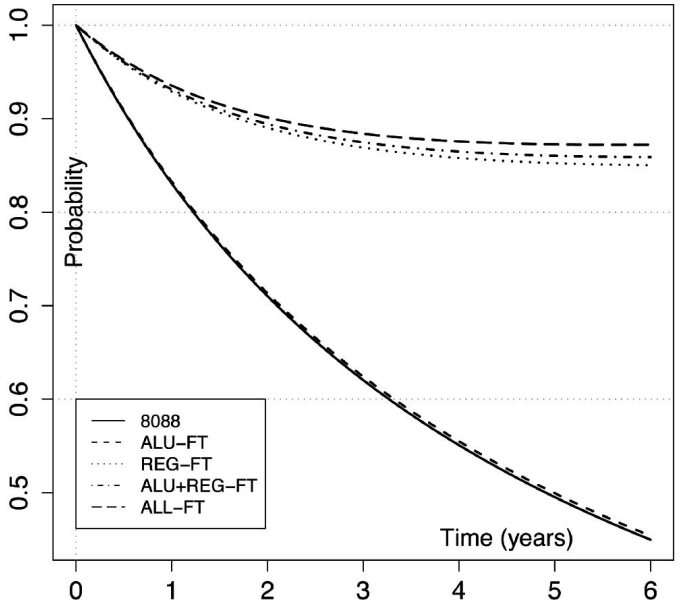


Fig. 9. Early design exploration for the implementation of fault-tolerant resources. The graphs shows the estimated reliability function of the 8088 running HUFFMAN with four different fault tolerant configurations of the microprocessor.

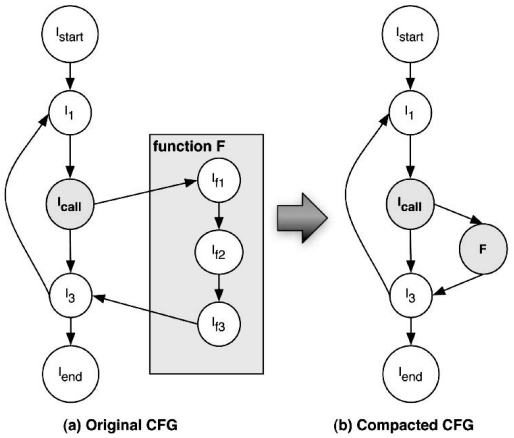


Fig. 10. Control flow graph reduction.

campaign, nor a deep knowledge of the microprocessor architecture (usually proprietary).

There is still a room for several improvements. Simulation and computational constraints do not allow to manage more than one program at a time, or to consider the introduction of operating system code. In order to manage very complex applications, further optimizations such as the one proposed in Section 6 must be implemented to brake down the complexity into more manageable subproblems. Since the execution time is not a part of the model, in its current form the proposed approach does not allow the targeting of real-time constraints. In order to obtain an even more precise reliability estimation, the proposed heuristic for the computation of the dependency among resources, which represents one of the most critical elements of the model, can be further refined.

Experimental results performed on the Intel 8088 and the OpenRISC1200 microprocessors are very promising. All the presented experiments show very small differences in the reliability estimation between this approach and a traditional fault injection experiment, but with a huge saving in computation time. The complexity of the microprocessors used for the experiments is not very high, but they nevertheless include several of the most critical functionalities of state-of-the-art devices (e.g., pipelines, floating-point units, etc.). The results suggest that there is no reason to believe that the proposed methodology would not be applicable to more complex microprocessors, provided that the resources to characterize them are available. It should also be considered that if the complexity of a modern microprocessor does not allow its characterization as proposed in this paper, it would neither allow a reliable fault injection campaign.

REFERENCES

[1] S. Kumar and A. Aggarwal, "Self-Checking Instructions: Reducing Instruction Redundancy for Concurrent Error Detection," *Proc. 15th Int'l Conf. Parallel Architectures and Compilation Techniques,* pp. 64-73, 2006.

[2] A. Shye, J. Blomstedt, T. Moseley, V. Reddi, and D. Connors, "Plr: A Software Approach to Transient Fault Tolerance for Multicore Architectures," *IEEE Trans. Dependable and Secure Computing,* vol. 6, no. 2, pp. 135-148, Apr.-June 2009.

[3] R. Baumann, "Technology Scaling Trends and Accelerated Testing for Soft Errors In Commercial Silicon Devices," *Proc. IEEE Int'l On-Line Testing Symp.,* p. 4, 2003.

[4] B.R., "Soft Errors in Advanced Computer Systems," *IEEE Design and Test of Computers,* vol. 22, no. 3, pp. 258-266, May/June 2005.

[5] S. Borkar, "Tackling Variability and Reliability Challenges," *IEEE Design and Test of Computers,* vol. 23, no. 6, p. 520, June 2006.

[6] S. Borkar, "Thousand Core Chips: A Technology Perspective," *Proc. 44th Ann. Design Automation Conf.,* pp. 746-749, 2007.

[7] P. Dodd, "Physics-Based Simulation of Single-Event Effects," *IEEE Trans. Device and Materials Reliability,* vol. 5, no. 3, pp. 343-357, Sept. 2005.

[8] S. Mitra, M. Zhang, T. Mak, N. Seifert, V. Zia, and K.S. Kim, "Logic Soft Errors: A Major Barrier to Robust Platform Design," *Proc. IEEE Int'l Test Conf.,* pp. 10-696, Nov. 2005.

[9] E. Normand, "Single Event Upset at Ground Level," *IEEE Trans. Nuclear Science,* vol. 43, no. 6, pp. 2742-2750, Dec. 1996.

[10] R. Baumann, "Soft Errors in Commercial Semiconductor Technology: Overview and Scaling Trends," *Proc. IEEE Reliability Physics Tutorial Notes, Reliability Fundamentals,* pp. 121.01.1-121.01.14, Apr. 2002.

[11] S. Krishnamohan and N.R. Mahapatra, "Analysis and Design of Soft-Error Hardened Latches," *Proc. 15th ACM Great Lakes Symp. VLSI,* pp. 328-331, 2005.

[12] M. Hosseinabady, P. Lotfi-Kamran, G. Di Natale, S. Di Carlo, A. Benso, and P. Prinetto, "Single-Event Upset Analysis and Protection in High Speed Circuits," *Proc. IEEE 11th European Test Symp. (ETS '06),* pp. 29-34, May 2006.

[13] H. Ando, Y. Yoshida, A. Inoue, I. Sugiyama, T. Asakawa, K. Morita, T. Muta, T. Motokurumada, S. Okada, H. Yamashita, Y. Satsukawa, A. Konmoto, R. Yamashita, and H. Sugiyama, "A 1.3ghz Fifth Generation Sparc64 Microprocessor," *Proc. 40th Ann. Design Automation Conf.,* pp. 702-705, June 2003.

[14] A. Benso, S. Di Carlo, G. Di Natale, and P. Prinetto, "A Watchdog Processor to Detect Data and Control Flow Errors," *Proc. IEEE Ninth On-Line Testing Symp. (IOLTS),* pp. 144-148, July 2003.

[15] S. Di Carlo, G. Di Natale, and R. Mariani, "On-line Instruction-Checking in Pipelined Microprocessors," *Proc. 17th Asian Test Symp. (ATS '08),* pp. 377-382, Nov. 2008.

[16] A. Benso, S. Di Carlo, G. Di Natale, P. Prinetto, and L. Tagliaferri, "Control-Flow Checking via Regular Expressions," *Proc. 10th Asian Test Symp.,* pp. 299-303, Nov. 2001.

[17] A. Benso, S. Di Carlo, G. Di Natale, P. Prinetto, L. Tagliaferri, and C. Tibaldi, "Promon: A Profile Monitor of Software Applications," *Proc. IEEE Eighth Int'l Workshop Design and Diagnostics of Electronic Circuits and Systems (DDECS),* pp. 81-86, Apr. 2005.

[18] A. Benso, S. Di Carlo, G. Di Natale, and P. Prinetto, "Seu Effect Analysis in A Open-Source Router via a Distributed Fault Injection Environment," *Proc. Conf. and Exhibition on Design, Automation and Test in Europe,* pp. 219-223, Mar. 2001.

[19] A. Benso, S. Di Carlo, G. Di Natale, L. Tagliaferri, and P. Prinetto, "Validation of a Software Dependability Tool via Fault Injection Experiments," *Proc. Seventh Int'l On-Line Testing Workshop, 2001,* pp. 3-8, July 2001.

[20] A. Benso, S. Di Carlo, G. Di Natale, P. Prinetto, and L. Tagliaferri, "Software Dependability Techniques Validated via Fault Injection Experiments," *Proc. Sixth European Conf. Radiation and Its Effects on Components and Systems,* pp. 269-274, Sept. 2001.

[21] M. Omana, G. Papasso, D. Rossi, and C. Metra, "A Model for Transient Fault Propagation in Combinatorial Logic," *Proc. IEEE Ninth On-Line Testing Symp.,* pp. 111-115, 2003.

[22] A. Maheshwari, I. Koren, and W. Burleson, "Techniques for Transient Fault Sensitivity Analysis and Reduction in VLSI Circuits," *Proc. IEEE Int'l Symp. Defect and Fault-Tolerance in VLSI Systems,* p. 597, 2003.

[23] H. Nguyen and Y. Yagil, "A Systematic Approach to Ser Estimation and Solutions," *Proc. IEEE 41st Ann. Int'l Reliability Physics Symp.,* pp. 60-70, Mar./Apr. 2003.

[24] K. Mohanram and N. Touba, "Partial Error Masking to Reduce Soft Error Failure Rate in Logic Circuits," *Proc. IEEE 18th Int'l Symp. Defect and Fault-Tolerance in VLSI Systems,* p. 433, 2003.

[25] K. Mohanram and N. Touba, "Cost-Effective Approach for Reducing Soft Error Failure Rate in Logic Circuits," *Proc. Int'l Test Conf.,* vol. 1, pp. 893-901, 2003.

[26] M. Sonza Reorda and M. Violante, "Accurate and Efficient Analysis of Single Event Transients in VLSI Circuits," *Proc. IEEE Int'l On-Line Testing Symp.,* pp. 101-105, 2003.

[27] P. Shivakumar, M. Kistler, S.W. Keckler, D. Burger, and L. Alvisi, "Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic," *Proc. Int'l Conf. Dependable Systems and Networks,* pp. 389-398, 2002.

[28] S. Kim and A.K. Somani, "Soft Error Sensitivity Characterization for Microprocessor Dependability Enhancement Strategy," *Proc. Int'l Conf. Dependable Systems and Networks,* pp. 416-428, 2002.

[29] S.S. Mukherjee, C. Weaver, J. Emer, S.K. Reinhardt, and T. Austin, "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor," *Proc. IEEE/ACM 36th Ann. Int'l Symp. Microarchitecture,* pp. 29-40, 2003.

[30] S. Mukherjee, C. Weaver, J. Emer, S. Reinhardt, and T. Austin, "Measuring Architectural Vulnerability Factors," *IEEE Micro,* vol. 23, no. 6, pp. 70-75, Nov./Dec. 2003.

[31] N.J. Wang, J. Quek, T.M. Rafacz, and S.J. patel, "Characterizing the Effects of Transient Faults on a High-Performance Processor Pipeline," *Proc. Int'l Conf. Dependable Systems and Networks,* p. 61, 2004.

[32] C. Weaver, J. Emer, S.S. Mukherjee, and S.K. Reinhardt, "Techniques to Reduce the Soft Error Rate of a High-Performance Microprocessor," *Proc. 31st Ann. Int'l Symp. Computer Architecture,* pp. 264-275, 2004.

[33] X. Li, S.V. Adve, P. Bose, and J.A. Rivers, "Online Estimation of Architectural Vulnerability Factor for Soft Errors," *Proc. 35th Int'l Symp. Computer Architecture,* pp. 341-352, 2008.
[34] X. Li, S. Adve, P. Bose, and J. Rivers, "Softarch: An Architecture-Level Tool for Modeling and Analyzing Soft Errors," *Proc. Int'l Conf. Dependable Systems and Networks,* pp. 496-505, 2005.
[35] V. Sridharan and D.R. Kaeli, "Using pvf Traces to Accelerate avf Modeling," *Proc. IEEE Workshop Silicon Errors in Logic System Effects,* http://web.me.com/vilas.sridharan/Vilas_Sridharan/Publications_files/3_ Sridharan_P.pdf, Mar. 2010.
[36] T.M. Jones and M.F.P., "Evaluating the Effects of Compiler Optimization on Avf," *Proc. Workshop the Interaction between Compilers and Computer Architecture (INTERACT),* 2008.
[37] A. Benso, S. Di Carlo, G. Di Natale, and P. Prinetto, "Static Analysis of Seu Effects on Software Applications," *Proc. Int'l Test Conf.,* pp. 500-508, 2002.
[38] T. Karnik and P. Hazucha, "Characterization of Soft Errors Caused by Single Event Upsets in Cmos Processes," *IEEE Trans. Dependable and Secure Computing,* vol. 1, no. 2, pp. 128-143, Apr.-June 2004.
[39] S. Mitra, T. Karnik, N. Seifert, and M. Zhang, "Logic Soft Errors in Sub-65nm Technologies Design and Cad Challenges," *Proc. 42nd Design Automation Conf.,* pp. 2-4, June 2005.
[40] NIST/SEMATECH, e-Handbook of Statistical Methods, http://www.itl.nist.gov/div898/handbook/, 2011.
[41] J. Larus, "Efficient Program Tracing," *Computer,* vol. 26, no. 5, pp. 52-61, May 1993.
[42] J. Maebe and B. De Sutter Diablo, http://diablo.elis.ugent.be/, 2011.
[43] HT-LAB, Cpu86 cpu86 8088 fpga ip core, http://www.ht-lab.com/freecores/cpu8086/cpu86.html, 2011.
[44] Univ. of Michigan at Ann Arbor, Mibench Version 1.0, http://www.eecs.umich.edu/mibench/, 2011.
[45] A. Bosio. and G. Di Natale, "Lifting: A Flexible Open-source Fault Simulator," *Proc. IEEE 17th Asian Test Symp.,* pp. 35-40, 2008.

Alessandro Savino received the MS degree in computer engineering and the PhD degree in information technologies from the Politecnico di Torino, Italy, where he has been a postdoc in the Department of Control and Computer Engineering since 2009. His main research topics are microprocessor test and software-based self-test.

Stefano Di Carlo received the MS degree in computer engineering and the PhD degree in information technologies from the Politecnico di Torino, Italy, where he has been an assistant professor in the Department of Control and Computer Engineering since 2008. His research interests include DFT, BIST, and dependability. He is a golden core member of the IEEE Computer Society and a senior member of the IEEE.

Gianfranco Politano received the MS degree in computer engineering and the PhD degree in information technologies from the Politecnico di Torino, Italy, where he has been a postdoc in the Department of Control and Computer Engineering since 2011. His main research topics are system reliability and machine learning techniques. He is a member of the IEEE and the IEEE Computer Society.

Alfredo Benso received the MS degree in computer engineering and the PhD degree in information technologies, both from Politecnico di Torino, Italy, where he is working as a tenured associate professor of computer engineering. His research interests include DFT, BIST, and dependability. He is also actively involved in the Computer Society, where he has been a leading volunteer for several projects. He is a Computer Society Golden Core Member, and a senior member of the IEEE.

Alberto Bosio received the MS degree in computer engineering and the PhD degree in information technologies from the Politecnico di Torino, Italy. He is currently an associate professor in the Laboratoire d'Informatique, de Robotique et de Microelectronique de Montpellier, University of Montpellier II/CNRS, Montpellier, France. His main research activity are methodologies and tools to improve the development of highly dependable systems, at different levels: for basic digital components, for systems on chip, up to microprocessor-based systems. He is a member of the IEEE.

Giorgio Di Natale received the PhD degree in computer engineering from Politecnico di Torino in Italy in 2003. Currently, he is a researcher for the National Research Center of France at the LIRMM laboratory in Montpellier. He has published articles in publications spanning diverse disciplines, including memory testing, fault tolerance, and secure chips design and test. He is a Golden Core member of the IEEE Computer Society, member of the IEEE, and he serves the European Test Technology Technical Council (eTTTC) of IEEE Computer Society as vice chair.

A Comprehensive Framework for Logic Diagnosis of Arbitrary Defects

Alberto Bosio, *Member*, *IEEE*, Patrick Girard, *Member*, *IEEE*,
Serge Pravossoudovitch, *Member*, *IEEE*, and Arnaud Virazel, *Member*, *IEEE*

Abstract—This paper presents a comprehensive framework for logic diagnosis consisting of two main phases. In the first phase, a set of suspected faulty sites is obtained by applying an approach based on an Effect-Cause analysis. Then, in the second phase, a set of realistic fault models is associated with each suspected faulty site by analyzing specific information, called fault evidences, collected during the first phase. The main advantage of the proposed methodology is its capability to deal with several fault models at the same time. Another advantage is that it is able to handle both single and multiple fault occurrences. Experiments on ISCAS85, ISCAS89, and ITC99 benchmark circuits show the efficiency of the proposed method both in terms of diagnosis resolution and accuracy of the predicted fault models.

Index Terms—Diagnosis, critical path tracing, fault modeling, circuit simulation, fault simulation.

1 Introduction

Failure analysis is an important operation for the production of good chips and has a growing role in fast yield ramp-up. Failure analysis first relies on logic diagnosis that aims at providing a set of suspected faulty sites in the defective circuit. In a second time, it resorts to sophisticated physical equipments (e.g., laser beam) in order to precisely locate and identify the defect(s). The objective of logic diagnosis is thus to identify potential faulty sites explaining the Circuit Under Test (CUT) erroneous behavior. Information provided by the logic diagnosis process is, therefore, used to guide the circuit physical observation during failure analysis.

Classical algorithms targeting logic diagnosis are based on two paradigms named Cause-Effect and Effect-Cause [1], [2]. The Cause-Effect paradigm, which is usually based on fault simulation, builds a fault dictionary containing circuit responses for a given test set in presence of a given set of faults [3], [4]. Logic diagnosis is then performed by comparing actual circuit responses with those stored in the dictionary. Unfortunately, the Cause-Effect paradigm exhibits some drawbacks. The first one is the need to have an a priori knowledge of the fault models used to build the fault dictionary. The second drawback is related to the huge amount of data that must be generated by the fault simulator, particularly for large industrial circuits.

The second paradigm, called Effect-Cause, resorts to an error backtracing process, such as the Critical Path Tracing process [5]. It starts from the CUT failing Primary Outputs to reach the CUT Primary Inputs. Each CUT lines traversed by the backtracing process is considered as a possible source of the observed error. The advantages of this approach are twofold. First, it does not require any explicit fault simulation process, and hence, does not need to consider explicitly each fault model during the diagnosis. Second, the required amount of data is negligible compared to the one generated by a Cause-Effect approach [6], [7].

Recent approaches presented in the literature combine the advantages of these two paradigms to obtain a mixed-based approach [8], [9], [10].

Nevertheless, a common feature of all the methods proposed so far is that they handle one single fault model at a time or scarcely two fault models when the induced effects are identical [11], [12], [13]. Moreover, due to the advances in manufacturing technologies and more aggressive clocking strategies used in modern designs, more and more defects lead to failures that can no longer be modeled by classical stuck-at faults. Numerous actual failures exhibit timing or parametric behaviors, which are not represented by stuck-at faults. Such failures have to be taken into account during the test and diagnosis processes in order to reach acceptable Defect per Million (DPM) figures.

The concept of composite fault models has been adopted in order to partially solve this drawback. Basically, it assumes that the behavior of some complex fault models (e.g., bridging fault model) can be modeled as multiple stuck-at faults [14]. It is based on stuck-at fault simulation and analysis of resulting test responses to provide a list of suspected faulty lines. An example is the SLAT paradigm [15] and its extension presented in [16] and [17]. The problem is that when an error is observed during the test application, it does not exist any deterministic information about the defect inducing this error, and hence, there is no knowledge of the fault model to be used a priori for the diagnosis process. As considering each fault model explicitly is not a viable solution, there is a need to develop a comprehensive framework for logic diagnosis that is independent of any fault model.

• *The authors are with the Laboratoire d'Informatique, de Robotique et de Microelectronique de Montpellier, University of Montpellier II/CNRS, 161 rue Ada, 34392 Montpellier Cedex 5, France. E-mail: {alberto.bosio, patrick.girard, serge.pravossoudovitch, arnaud.virazel}@lirmm.fr.*

Manuscript received 22 July 2008; revised 16 Mar. 2009; accepted 19 Aug. 2009; published online 17 Nov. 2009.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2008-07-0364.
Digital Object Identifier no. 10.1109/TC.2009.177.

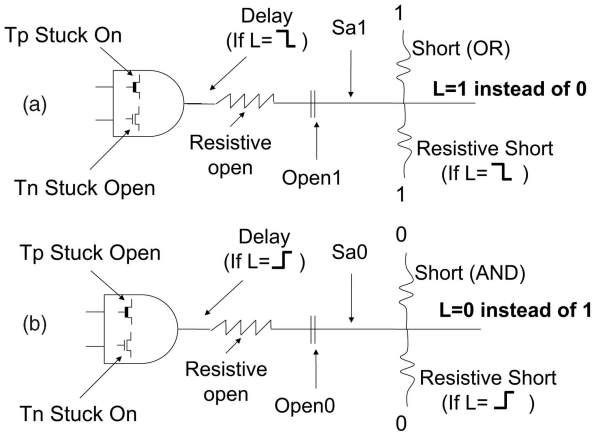


Fig. 1. Potential fault models associated with an error.

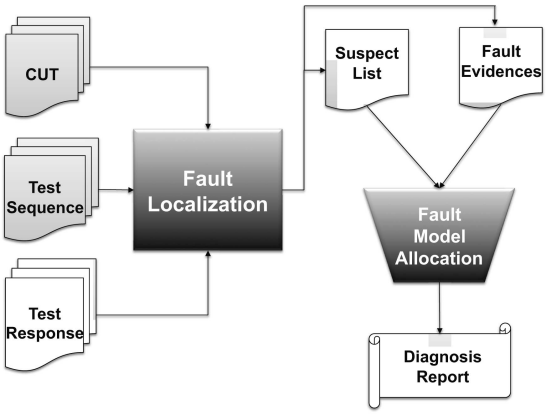


Fig. 2. Structure of the diagnosis approach.

This paper presents a logic diagnosis approach, which assumes that the nature of the defects affecting the CUT is not known a priori before running the diagnosis process. The proposed approach uses the background presented in [18]. Starting from the information provided by the tester (pass/fail), our main goal is to provide a minimal set of suspected lines (sites) in the shortest possible time. The proposed diagnosis solution consists of a two-phase approach: 1) the Fault Localization analyzes the circuit to extract the minimal set of suspected lines that can be the source of the observed error. This phase has been extended with respect to [18] by adding an optimization step. The optimization step aims at narrowing down the list of suspects by considering information provided by the fault-free outputs (coming both from faulty and fault-free test patterns). This information, called fault evidences, is further used to discriminate between possible fault models in order to finally 2) allocate a set of realistic fault models to the suspected lines during the second phase called Fault Model Allocation.

The main advantage of the proposed approach is its capability to deal with several fault models at the same time. Moreover, it is able to distinguish between single and multiple faults affecting the CUT. Compared to our previous work [18] and [21], the advantages are twofold. First, the list of suspects is reduced due to a new optimization step described in Section 3.3, and ranked, thus, achieving a better resolution with respect to [18]. Second, the information extracted during the optimization step is used to reduce the number of fault models associated with each suspect. Thus, the number of fault models associated with each suspect is, on average, lower than [18].

Experimental results show the efficiency and reliability of the method in terms of diagnosis resolution and accuracy of the predicted fault models.

The remainder of the paper is organized as follows: Section 2 gives an overview of the proposed approach. Section 3 details the fault localization phase. Section 4 presents the fault model allocation phase. In Section 5, experimental results are presented. Finally, conclusions are given in Section 6.

2 Overview of the Logic Diagnosis Approach

Errors observed on circuit outputs are the result of physical defects in the circuit. These defects are usually represented by various fault models, which may represent logic deviations (stuck-at, short, open), timing deviations (gate/path delay), or parametric deviations (resistive short, resistive open). Some fault models (stuck-at, open, AND/OR bridging) affect the static behavior and some others (delay, resistive short, resistive open, and stuck-on/open) affect the dynamic behavior, and thus, need a transition to be sensitized and propagated.

Nevertheless, the outcome of these failures on the circuit outputs is always a change of logic value. When an error is located on a given line L, this error can be caused by a defect affecting the line L or is due to the propagation of an error generated by an upstream faulty site. As shown in Fig. 1, the error observed on line L can be $L = 1$ instead of 0 (Fig. 1a) or $L = 0$ instead of 1 (Fig. 1b) depending on the logic value carried out by the line L. This error can be caused by various defects and it can be determined assuming the knowledge of 1) the expected value on line L, 2) the values on other lines (for short faults), and 3) the transition induced on line L by the applied test pattern (for dynamic faults). Therefore, the possible fault models associated with a given logic error can be deduced from the knowledge of the fault-free circuit behavior during the test phase.

The principle of the proposed diagnosis method is thus divided into two main phases as presented in Fig. 2.

The first phase, called Fault Localization, consists in determining the suspect lines. The required inputs are: 1) the circuit description at gate level (CUT), 2) the test vectors (Test Sequence), and 3) the output responses associated with input vectors (Test Response). After this phase, we obtain the set of suspect lines associated with the set of fault evidences. In the second phase, a Fault Model Allocation procedure identifies a reduced set of possible fault models for each suspect line by analyzing the fault evidences and then provides the diagnosis report.

3 Fault Localization

The Fault Localization is done by using the Effect-Cause analyses. It mainly relies on Critical Path Tracing (CPT) and consists of the two following steps: For each faulty test pattern, we perform 1) a fault-free multivalued simulation of the circuit. Then, 2) the CPT process is performed starting from each Failing Primary Output (FPO) caused by the

TABLE 1
Test Pattern Computation

TV Index	Test Vectors	TP Index	Test Patterns
V_0	(1, 1, 1, 0, 0, 1)	T_0	(C1, C1, C1, C0, C0, C1)
V_1	(0, 1, 0, 1, 0, 1)	T_1	(F0, C1, F0, R1, C0, C1)
V_2	(1, 0, 1, 0, 0, 1)	T_2	(R1, F0, R1, F0, C0, C1)
V_3	(0, 1, 1, 1, 0, 0)	T_3	(F0, R1, C1, R1, C0, F0)
V_4	(1, 1, 1, 0, 1, 1)	T_4	(R1, C1, C1, F0, R1, R1)
V_5	(0, 0, 0, 1, 1, 0)	T_5	(F0, F0, F0, R1, C1, F0)
V_6	(0, 0, 0, 0, 1, 1)	T_6	(C0, C0, C0, F0, C1, R1)

current faulty test pattern, and hence, determines the critical lines. The critical lines identified from all the faulty test patterns are ranked in a list of suspects depending on their single or multiple nature.

At this step, the list of suspects provided by the Effect-Cause analysis is further reduced 3) by considering information provided by the fault-free outputs (coming both from faulty and fault-free test patterns).

3.1 Fault-Free Circuit Simulation

For each failing vector of the test sequence, a fault-free circuit simulation is done by applying the vector to the CUT inputs, and subsequently, propagating values trough the circuit toward POs. As previously mentioned (Section 2), we need to know not only the logic value on a given line but also its dynamic behavior. For this purpose, we have to consider not only a single test vector V_i but rather two consecutive vectors (V_{i-1}, the vector preceding V_i during the test application has also to be considered) when defining the fault-free values on each line. To represent signals, we therefore resort to a six-valued logic algebra [19] by considering the couple of logic values ($V_{i-1}V_i$). The signals are therefore defined as follows:

- C0: static $0 = 00$,
- C1: static $1 = 11$,
- R1: rising transition $= 0 \times 1$,
- F0: falling transition $= 1 \times 0$,
- P0: static 0-hazard $= 0 \times 0$, and
- P1: static 1-hazard $= 1 \times 1$,

where:

- C0 (C1): static 0 (1), represents a signal remaining absolutely stable at 0 (1) (whatever the gate propagation delays and timing defects).
- F0 (R1): fall (rise), represents a signal with the initial value 1 (0) and the final value 0 (1) (after circuit stabilization).
- P0 (P1): pulse 0 (pulse 1), represents a signal with the same initial and final value 0 (1), but with possible logic transitions to 1 (to 0) induced by circuit timing parameters or delay faults.

Furthermore, the proposed symbols can be further classified into two different sets used to formalize some steps of the proposed method as follows:

- 0-set $= \{C0, F0, P0\}$,
- 1-set $= \{C1, R1, P1\}$,

where the 0-set (respectively, 1-set) represents symbol with a final value equal to 0(1).

In the rest of the paper, we use the term Test Pattern (T) to refer to the couple of test vectors ($V_{i-1}V_i$), encoded with the six-valued algebra, and computed by considering the following equation:

$$T_i = \begin{cases} (V_{i-1}V_i) & \text{if } i > 0, \\ (V_iV_i) & \text{if } i = 0. \end{cases} \quad (1)$$

Note that when considering the first test vector (the case $i = 0$ in (1)) no transition (both rising and falling) is possible, so the test pattern related to the first test vector is always composed by static symbols (C0, C1) only.

Table 1 gives an example of Test Pattern computation starting from a sequence of Test Vectors. The first column of Table 1 reports the index of each test vector (V_i) while the second column provides each test vector. Column 3 gives the index of the corresponding test pattern (T_i) and the last column shows the test pattern. Each test pattern has been computed by applying (1).

From encoded symbols at all inputs, the fault-free multivalued circuit simulation consists in propagating these values toward the circuit outputs by using propagation tables associated with each logic gate. Propagation tables for classical AND, OR, and NOT gates are shown in Tables 2, 3, and 4, respectively. All other propagation tables can be easily obtained from these three basic ones [18].

For example, let us consider the circuit in Fig. 3. It has six inputs (E_1-E_6) and three outputs (S_1, S_2, S_3). The test pattern applied on the circuit is $T_5 =$ (F0, F0, F0, R1, C1, F0) coming from Table 1. Then, values propagated during the fault-free circuit simulation are shown in Fig. 3 (labels on each line).

TABLE 2
AND Propagation Table

AND	C0	C1	F0	R1	P0	P1
C0	C0	C0	C0	C0	C0	C0
C1	C0	C1	F0	R1	P0	P1
F0	C0	F0	F0	P0	P0	F0
R1	C0	R1	P0	R1	P0	R1
P0	C0	P0	P0	P0	P0	P0
P1	C0	P1	F0	R1	P0	P1

3.2 Critical Path Tracing

The CPT has been developed for stuck-at faults [1] and extended for delay fault in [19]. The CPT starts from a failing

TABLE 3
OR Propagation Table

OR	C0	C1	F0	R1	P0	P1
C0	C0	C1	F0	R1	P0	P1
C1	C1	C1	C1	C1	C1	C1
F0	F0	C1	F0	P1	F0	P1
R1	R1	C1	P1	R1	R1	P1
P0	P0	C1	F0	R1	P0	P0
P1	P1	C1	P1	P1	P1	P1

TABLE 4
NOT Propagation Table

IN	OUT
C0	C1
C1	C0
F0	R1
R1	F0
P0	P1
P1	P0

primary output to reach the primary inputs by tracing each critical line passing trough sensitive gate inputs.

The sensitive inputs of each gate are defined from the rules provided in [5] and [18]. More formally, a gate input i is sensitive if complementing the value of i changes the value of the gate output. In presence of a gate with only nonsensitive inputs, the CPT stops at this gate and restarts at the reconverge fan-out stem, associated with this nonsensitive gate. All details about this process can be found in [19] and lately in [18].

The CPT process is executed by starting from each failing primary output so that the total amount of times that the CPT process is applied is given by (2):

$$\#CPT = \sum_{i=1}^{\#FTP} (\#FPO_i), \tag{2}$$

where #CPT is the number of CPT process application, #FTP is the number of Failing Test Pattern, and $\#\mathrm{FPO}_i$ is the number of Failing Primary Outputs caused by the corresponding ith FTP_i.

Each circuit line traced during the CPT is stored in a so-called list of suspects defined as follows:

$$\begin{aligned} L = \{&(LC_0, CNT_0, S_0), (LC_1, CNT_1, S_1), \ldots \\ &(LC_{N-1}, CNT_{N-1}, S_{N-1})\}, \end{aligned} \tag{3}$$

where LC_i (suspect line) is the name of the critical line, CNT_i and S_i are, respectively, a counter and a symbol associated with the line LC_i. Symbol S and Counter CNT are defined as fault evidence used to further determine the fault candidate. When a line LC_i is traced by the CPT, both its counter (CNT_i) and symbol (S_i) are updated by executing a routine called Trace. Basically, this procedure verifies whether the traced line LC_i is indeed included in the list of suspects L. If not, the CNT is set to 1 and symbol S_i is set to S. In case the line has already been traced (from another failing primary output), the counter is simply incremented ($\mathrm{CNT}_i = \mathrm{CNT}_i + 1$) and the symbol is updated by performing an intersection as follows:

$$S_i = S \cap S_i, \tag{4}$$

where S is the current symbol associated with the traced line and S_i is the symbol associated with the line as stored in the list of suspects. The intersection is done according to the intersection table (Table 5).

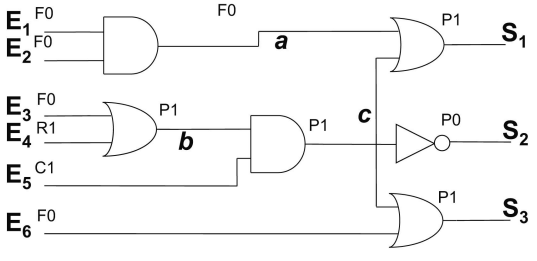


Fig. 3. Fault-free multivalued simulation example.

TABLE 5
Intersection Table

∩	C0	C1	F0	R1	P0	P1
C0	C0	SDW	C0	SDW	C0	SDW
C1	SDW	C1	SDW	C1	SDW	C1
F0	C0	SDW	F0	D	F0	D
R1	SDW	C1	D	R1	D	R1
P0	C0	SDW	F0	D	P0	D
P1	SDW	C1	D	R1	D	P1
D	SDW	SDW	D	D	D	D
SDW	SDW	SDW	SDW	SDW	SDW	SDW

Each column in Table 5 represents the symbol associated with a line LC_k obtained after the current fault-free circuit simulation (S from (4)). Each row in Table 5 represents the symbol stored in the list of suspect L for the same line LC_k (S_i from (4)). When the intersection is performed between two "dynamic" symbols (F0, R1, P0, P1) belonging to two different sets (0-set and 1-set), we obtain the new symbol "D." This new symbol "D," added to the set of symbols used for the fault-free simulation, allows representing a line with a signal possibly delayed for both falling and rising transitions. On the other hand, we need to characterize a line that exhibits different final values during the application of different failing test patterns. We resort to the new symbol "SDW" meaning Strong Driver Wired. The symbol "SDW" allows representing a possible situation, where the logic value of the suspect line depends on the value of another line. In this case, the suspect line will be considered as possible victim of a bridging fault (details are given in Section 4). This symbol is generated when lines exhibit symbols belonging to two different sets (0-set and 1-set) with at least one "static" symbol (C0, C1).

Let us consider an example to better clarify the CPT application. We consider the circuit shown in Fig. 3 affected by a dominant AND bridging fault between lines a and c. After the test phase, one test vector is declared fail. The failing test vector is V_5 and comes from the example of Table 1. From this failing test vector, we obtain the corresponding test pattern, which has been shown in Table 1, $\mathrm{T}_5 =$ (F0, F0, F0, R1, C1, F0). Fig. 4 gives the circuit when the test pattern T_5 is considered. T_5 leads to have three failing primary outputs S_1, S_2, and S_3.

Here, the CPT process starts from S_1 and stops at the primary inputs. The critical lines provided by the CPT (highlighted with a black dot in Fig. 4) are S_1, c, b, E_4, and E_5 stored in the list of suspects L defined as follows:

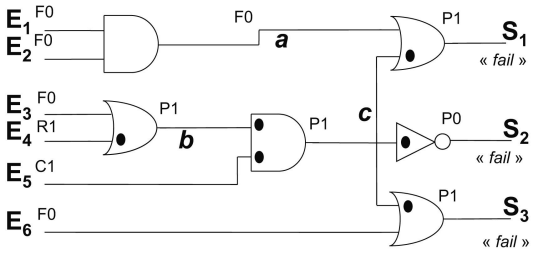


Fig. 4. CPT application example.

$$L = \{(S_1, 1, P1), (c, 1, P1), (b, 1, P1), (E_4, 1, R1), (E_5, 1, C1)\}. \quad (5)$$

Since this is the first application of the CPT, no critical line has been traced before, so the CNT is set to 1 and the associated symbol is the one obtained after the fault-free multivalued simulation.

Now the CPT starts from S_2 and provides as critical lines S_2, c, E_5, b, and E_4. Therefore, the list of suspects L given by (5) is updated as follows:

$$L = \{(S_1, 1, P1), (S_2, 1, P0), (c, 2, P1), (b, 2, P1), (E_4, 2, R1), (E_5, 2, C1)\}. \quad (6)$$

Line S_2 is added to the list, since it is traced by the CPT for the first time. The other lines are updated by incrementing the counter and modifying the symbol by using the intersection table shown in Table 5.

Eventually, the CPT starts again from the failing primary output S_3 and provides the critical lines S_3, c, E_5, b, and E_4. Consequently, the list of suspects L of (6) is further updated as follows:

$$L = \{(S_1, 1, P1), (S_2, 1, P0), (S_3, 1, P1), (c, 3, P1), (b, 3, P1), (E_4, 3, R1), (E_5, 3, C1)\}. \quad (7)$$

Line S_3 is added to the list, while the remaining lines are updated.

At the end of the Effect-Cause analysis, we obtain the list of suspects L (7) that contains only the lines traced at least one time (i.e., having CNT greater than 0) with the corresponding symbol S.

Table 6 reports the list of suspected lines, ranked by descending order of CNT values, obtained after the complete Effect-Cause analysis (7). Lines E_5, E_4, b, and c have a CNT equal to 3. The remaining lines were traced only one time. It can be observed that the actual faulty line (line c as the victim of the bridging fault) is included in the set of lines having the highest CNT ($\mathrm{CNT} = 3$).

Suspect lines having a CNT equal to #FPO were traced during each CPT process application. So, each of these suspect lines can explain all faulty responses. On the other hand, suspect lines having a CNT lower than the number of failing primary outputs were traced a number of times lower than #FPO. Consequently, these suspect lines are not likely to explain all faulty responses when considering the single fault assumption. In this case, only multiple faults may have occurred. Resulting faults can therefore be single or multiple faults. Finally, each suspect line has its associated symbol S summarizing the history of the values carried by the line. At this stage, an important point to note is that by only analyzing the counter associated with each line, we can determine if the CUT is affected by a single or a multiple fault.

TABLE 6
List of Suspects

LC	CNT	S
E_5	3	C1
E_4	3	R1
b	3	P1
c	3	P1
S_1	1	P1
S_2	1	P0
S_3	1	P1

3.3 Optimization of the List of Suspect Lines

At this step, the list of suspects provided by the Effect-Cause analysis can be further reduced by considering information provided by the fault-free outputs (coming both from faulty and fault-free test patterns). A first solution would consist in applying the CPT process by starting from the fault-free outputs [1]. Unfortunately, the total amount of fault-free outputs provided by a given test set is generally much larger than the number of faulty outputs, making this solution computationally unexploitable for realistic circuits. In order to reduce the complexity while taking into account the information available from fault-free outputs, we propose an alternative solution based on fault simulation. This solution consists in performing a stuck-at fault simulation with all test vectors (both faulty and fault-free test vectors) on the set of suspect lines previously identified. In particular, we consider only the suspect lines having the CNT equal to #FPO. The suspect lines with CNT lower than #FPO are associated with multiple faults occurrence, and therefore, the single stuck-at fault simulation does not provide significant information on them. Such stuck-at fault simulation allows to analyze all the possible propagations of the logic error on each line so that it is finally possible, by considering this additional information, to narrow down the list of fault candidates. As the list of suspect lines is generally much smaller than the total amount of lines in the CUT, this solution is suitable for large designs from the CPU time perspective.

Let us now introduce some definitions to better explain the proposed flow:

- Faulty Test Vector (FTV): a test vector causing an output error on the CUT during the test phase.
- Fault-Free Test Vector (FFTV): a test vector that does not cause any error during the test phase.

The type of simulated stuck-at faults during fault simulation depends on the symbol S associated with each line as follows:

- If S $\in$ 0-set, then we simulate a stuck-at 1 (Sa1) fault on the line.
- If S $\in$ 1-set, then we simulate a stuck-at 0 (Sa0) fault on the line.

In case a symbol S is equal to "D" or "SDW," the corresponding line is not considered in the fault simulation

TABLE 7
Fault List Example

L	Fault
b	Sa0
c	Sa0
E_4	Sa0
E_5	Sa0

process. Using these rules, we determine the fault list to be used during the fault simulation process.

Table 7 provides an example of the fault list obtained by applying the above rules on the list of suspects given in Table 6. The first column of Table 7 reports the line (L), the second column gives the type of stuck-at to be used during fault simulation. From Table 6, we consider lines E_5, c, b, and E_4 to be simulated, since they have a CNT equal to #FPO. The considered fault model is the Sa0 since the lines have the symbol S $\in$ 1-set. Lines S_1, S_2, and S_3 are not considered in the fault list since they have counter lower than #FPO and are therefore related to multiple fault occurrences.

Fig. 5 sketches the flow applied during the optimization of the list of suspects.

The fault list FL1 is first defined as mentioned above (according to the suspect list and the associated symbol). Then, fault simulation with FTV is performed. After the fault simulation, we compare the results obtained with those provided by the tester when applying the same FTV. Two cases are possible as follows:

1. The fault simulation provides the same output values than those provided by the tester.
2. The fault simulation provides different output values than those provided by the tester.

In the first case, the simulated fault on line LC_k can explain the observed faulty behavior. In this case, we cannot further reduce the set of suspects, we remove it from FL1 and store it in FL2.

In the second case, the simulated static fault model (stuck-at) does not explain the observed faulty behavior. In this case, we know that the fault is sensitized because errors are observed on POs, and we can conclude that only dynamic fault on line LC_k can be the root cause of the observed errors. To indicate that only dynamic fault has to be considered on suspected line LC_k, we mark it by the symbol "$*$" and remove it from FL1.

In the fault list FL2, we collect all the faults that do not provide any differences between fault simulation of FTV and test phase. To further analyze those faults, a second fault simulation is applied with FL2 and the FFTV. Again we compare the results provided by the tester with those obtained by the fault simulation. Two cases are possible as follows:

1. The fault simulation provides the same output values than those provided by the tester (fault not detected).
2. The fault simulation provides different output values than those provided by the tester (fault detected on a PO).

In the first case, the suspect line can indeed be the cause of the observed errors, since the simulation with the injected stuck-at fault provides the same results than those provided by the tester for both Faulty and Fault-Free test vectors. In the second case, the simulated fault leads to failing primary outputs for an FFTV. We conclude that this line can be associated only with the case of a fault involving at least two lines (e.g., bridging faults) or with a dynamic faults. This consideration is detailed in Section 4, where we analyze these results to associate the fault models on each line. To indicate this situation, the suspect line S is marked by the symbol "$+$."

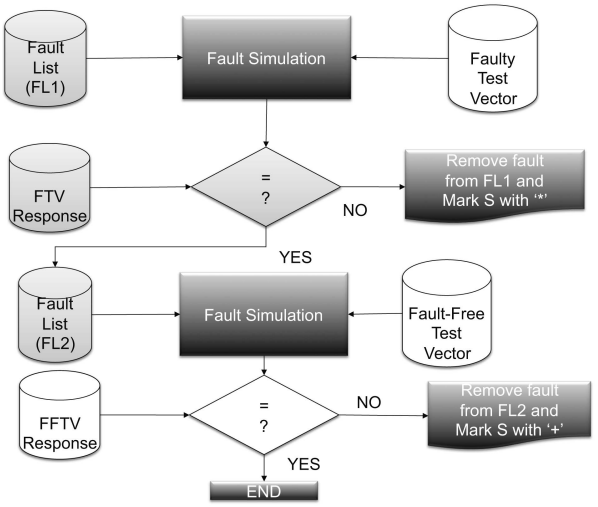


Fig. 5. Optimization flow.

We take this information into account in order to further rank the suspected lines in the suspect list. As the simulation of a stuck-at fault on a line marked by "$*$" does not explain the errors observed during test application, we consider such lines as affected by dynamic faults only. On the other side, lines marked by "$+$" can better explain the observed errors since the simulation with faulty test patterns of a stuck-at fault on such lines has provided the same results than those provided by the tester (a difference only occurs in the case of an FFTV). Hence, we can consider that these lines as more likely to be the cause of the observed errors. Note, however, that suspect lines which are the most likely to explain the observed errors are those having a conventional symbol (marked neither by "$*$" nor by "$+$").

Let us consider again the example circuit described in Fig. 4.

Table 8 reports the set of applied test vectors (defined in Table 1) on the circuit of Fig. 4. For each test vector, we provide the results after the test phase by marking as "F" the failing primary outputs and as "P" the passing outputs (outputs are S_1, S_2, and S_3). We can see that we have one failing test vector and six passing test vectors. To perform the fault simulation, we use the fault list presented in Table 7.

Table 9 reports the results after fault simulation. The first column gives the index of the simulated test vector, the other columns report the detail for each primary output of the circuit. We note with a "-" the case when the simulated fault is not observable at a given primary output for a given test vector. Otherwise, we specify which fault is observable by a given primary output. For example, faults b, c, and E_5 are detectable by test vector V_4 at S_2.

TABLE 8
Applied Test Vectors

Index	Test Patterns	S_1	S_2	S_3
V_0	(1, 1, 1, 0, 0, 1)	P	P	P
V_1	(0, 1, 0, 1, 0, 1)	P	P	P
V_2	(1, 0, 1, 0, 0, 1)	P	P	P
V_3	(0, 1, 1, 1, 0, 0)	P	P	P
V_4	(1, 1, 1, 0, 1, 1)	P	P	P
V_5	(0, 0, 0, 1, 1, 0)	F	F	F
V_6	(0, 0, 0, 0, 1, 1)	P	P	P

TABLE 10
List of Suspects after Fault Simulation

LC	CNT	S
E_4	3	R1
E_5	3	C1+
b	3	P1+
c	3	P1+
S_1	1	P1
S_2	1	P0
S_3	1	P1

We obtain that fault E_4 has the same behavior than that reported by the test phase. Therefore, no marker is added on its symbol. For the remaining faults, we have a difference when FFTVs are applied. Therefore, these faults lead to have a "+" marker.

Table 10 reports the list of suspect lines after the stuck-at fault simulation. In our example, we obtain three lines having a "+" symbol. This information will be used during the fault model allocation for reducing the number of possible fault models associated with each line.

4 Fault Model Allocation

The fault models considered in this study include all the classical single fault models as stuck-at faults, transistor stuck-open faults, transistor stuck-on faults, open and resistive open faults, bridging and resistive bridging faults, delay faults including slow-to-rise, slow-to-fall, and both slow-to-rise and slow-to-fall faults. Moreover, since the proposed method can deal with both single and multiple fault assumptions, we also consider several types of bridging faults (Byzantine, AND-Bridge, OR-Bridge, Strong Driver Wired, etc.). These types of bridging faults show a multiple faulty behavior and we use it as a case study to analyze multiple faults. The whole set of considered fault models is detailed in the first column of Table 11.

As mentioned in Section 2, fault models associated with a suspect line can be deduced from the knowledge of the fault-free circuit behavior during the test phase. This information is known from the knowledge of the symbol associated with the line. Lines marked by a stable symbol (C0 or C1) lead to static faults only, since no transition has appeared on the line, and hence, no dynamic faults can be sensitized. Columns 2 and 3 of Table 11 report the static fault models deduced from these symbols. Lines marked by a symbol representing a transition (F0, P0, R1, P1) can represent either static or dynamic faults. Columns 4 and 5 of Table 11 report both static and dynamic fault models deducted from dynamic symbols. Finally, columns 6 and 7 show the fault models associated with symbols "D" and "SDW," respectively. These two symbols are the results of the intersection procedure. A "D" symbol represents a situation where the suspected line has carried both rising and falling transitions, thus only dynamic faults are possible or Byzantine fault. The "SDW" symbol represents a line that has different values "0" and "1" coming from both dynamic and static symbols. In this case, we vindicate dynamic faults because the line has carried a static symbol (C0 or C1), but we also vindicate static fault models, since the value on the line varies from 0 to 1. In this case, either a strong driver wired fault or bridging fault is considered to be the possible cause of the observed error as detailed in [21].

TABLE 9
Fault Simulation Example

Index	S_3	S_2	S_3
V_0	-	-	-
V_1	-	-	-
V_2	-	-	-
V_3	-	-	-
V_4	-	b, c, E_5	-
V_5	b, c, E_5, E_4	b, c, E_5, E_4	b, c, E_5, E_4
V_6	-	-	-

Let us now analyze the fault model allocation in case of the markers "*" and "+" introduced during the optimization. If the symbol associated with a suspect line is marked by a "*," it means that the selected fault models cannot be static. Therefore, we vindicate these lines to be affected by any static fault. As shown in Table 11, lines marked by $C0^*$ and $C1^*$ (columns 8 and 9) can, therefore, be completely removed from the list of suspect lines, while fault models associated with transition symbols ($F0^*$, $P0^*$, $R1^*$, $P1^*$) are considered as dynamic only (delay and resistive open) as shown in columns 10 and 11.

When symbols are marked by a "+," the fault model allocation is more complex. First of all, the line can be vindicated to be affected only by static faults involving only one line (such as stuck-at, transistor stuck-on/open, and open 0/1) as reported in columns 12-15. This consideration is easily explained because the fault simulation has provided differences with results provided by the tester when applying Fault-Free Test Vectors. Therefore, the stuck-at fault model and other equivalent fault models in terms of logic error can be vindicated. On the other side, during the stuck-at fault simulation of the Fault-Free Test Vectors, the activation condition on the aggressor line (a logic "0" or "1" depending on the OR/AND behavior of the bridge) cannot be ensured (because in the test phase, these patterns did not cause any error). Therefore, we cannot exclude these types of faults as possible root cause of the observed error.

Let us consider again our previous example and the list of suspects provided after the fault localization phase and reported in Table 10. After the fault model allocation, we

TABLE 11
Fault Models Allocation According to Symbols

	C0	C1	F0 P0	R1 P1	D	SDW	C0*	C1*	F0* P0*	R1* P1*	C0+	C1+	F0+ P0+	R1+ P1+
Stuck-at 0		X		X										
Stuck-at 1	X		X											
Tn Stuck-open	X		X											
Tn Stuck-on		X		X										
Tp Stuck-open		X		X										
Tp Stuck-on	X		X											
Open 0		X		X										
Open 1	X		X											
Dominant Or Bridge	X		X								X		X	
Dominant And Bridge		X		X								X		X
Resistive OR			X						X				X	
Resistive And				X						X				X
Byzantine	X	X	X	X	X	X					X	X	X	X
Strong Driver Wired	X	X	X	X	X	X					X	X	X	X
Wired And		X		X								X		X
Wired Or	X		X								X		X	
Resistive open			X	X	X				X	X			X	X
Delay StF			X						X				X	
Delay StR				X						X				X
Delay StF&StR					X									

obtain a set of fault models associated with each suspected line as given in Table 12. Remember that for this example, we have injected a dominant AND bridging fault (between lines a and c). We can observe that the real faulty line (c) with the real type of injected fault appears in this table, thus demonstrating the reliability of the proposed solution.

5 Experimental Results

We have validated the proposed diagnosis method by performing an intensive set of experiments on a set of the ISCAS85 combinational benchmark circuits and the full scan version of ISCAS89 and ITC99 benchmark circuits. In our experiments, we first compare the proposed methodology with a commercial diagnosis tool in order to show the importance of considering several fault models at a time. Moreover, we also compare the proposed methodology with our previous work detailed in [18]. Then, we validate our methodology in terms of diagnosis resolution (i.e., absolute number of suspect lines). Finally, we investigate the capability of our framework to deal with multiple faults (i.e., for the case of bridging faults).

TABLE 12
Diagnosis Report

LC	CNT	S	Fault Models
$\mathbf{E}_4$	3	R1	Stuck-at 0, Tn stuck-on, Tp stuck-open, Open 0, Dominant And Bridge, Resistive And, Byzantine, Strong Driver Wired, Wired And, Resistive Open, Delay StR
$\mathbf{E}_5$	3	C1+	Dominant And Bridge, Strong Driver Wired, Byzantine, Wired And
b	3	P1+	Dominant And Bridge, Resistive And, Byzantine, Strong Driver Wired, Wired And, Resistive Open, Delay StR
c	3	P1+	**Dominant And Bridge**, Resistive And, Byzantine, Strong Driver Wired, Wired And, Resistive Open, Delay StR
$\mathbf{S}_1$	1	P1	Stuck-at 0, Tn stuck-on, Tp stuck-open, Open 0, Dominant And Bridge, Resistive And, Byzantine, Strong Driver Wired, Wired And, Resistive Open, Delay StR
$\mathbf{S}_2$	1	P0	Stuck-at 1, Tn stuck-open, Tp stuck-on, Open 1, Dominant Or Bridge, Resistive Or, Byzantine, Strong Driver Wired, Wired Or, Resistive Open, Delay StF
$\mathbf{S}_3$	1	P1	Stuck-at 0, Tn stuck-on, Tp stuck-open, Open 0, Dominant And Bridge, Resistive And, Byzantine, Strong Driver Wired, Wired And, Resistive Open, Delay StR

TABLE 13
ISCAS85 C432 Diagnosis Result Summary

Scenario	#FP	#FPO	Ref Tool	PA	PDA
L164 - Sa0	19	32	3	4	1
L96 - Sa1	2	5	1	9	1
03 - Sa0	39	39	2	4	3
L94 $\Rightarrow$ L123 OR Bridge	23	76	1	1	1
L140 $\Rightarrow$ L163 OR Bridge	30	62	1	3	3
I2 $\Rightarrow$ L100 AND Bridge	3	7	1 error	21	2
L52 $\Rightarrow$ L84 AND Bridge	22	61	2	2	2
Gate90 - StR	3	4	8	8	5
Gate140 - StF	7	16	5	5	3
Gate46 - StF	15	62	0	1	1
Gate72 - StF	4	8	0	7	2

Let us first define how we instrument the experimental environment. Each test sequence has been generated by using a commercial ATPG tool. The fault model considered during the test generation is the stuck-at fault model, and test patterns were generated with the random fill option. The fault coverage achieved is up to $98 \sim 99$ percent for each circuit. To simulate the behavior of a faulty circuit, we first randomly inject a fault (of various nature) in the structural description of the circuit. Then, the faulty circuit is simulated and the output responses are compared to those obtained with the fault-free circuit simulation.

Let us now compare the diagnosis results provided by the proposed method with those obtained by using a commercial tool targeting stuck-at faults only. The comparison is also done with respect to our previous work [18]. Table 13 summarizes experiments performed on circuit c432 (ISCAS85). For each experiment, we applied a test sequence composed of 71 test vectors. We report in the first column (Scenario) the faulty line and the injected fault model (Stuck-at 0/1, Slow-to-Fall/Rise, and bridging faults Wired AND/OR). In the case of bridging faults, the notation L1 $\Rightarrow$ L2 means that the fault is injected in L1 (aggressor) that attacks L2 (victim). Column 2 reports the number of failing patterns (#FP) and column 3 the number of failing primary outputs (#FPO) provided by the tester after test application. Column 4 gives the number of suspect lines identified by using the reference commercial tool (Ref. Tool). Column 5 gives the number of suspect lines identified by using our previous approach (PA) detailed in [18]. The last column reports the number of lines identified by the proposed unified diagnosis approach (PDA).

A first comment on these results is that the reference tool provides, in some cases, an erroneous diagnosis. As shown in Table 13, in the case where we have injected an AND Bridging fault (I2 $\Rightarrow$ L100, row 6), the reference tool has found one suspect line that is not the line on which the fault has been injected. Another problem appears for the two last injected faults, i.e., Gate46 StF and Gate72 StF (two last rows in Table 13). In these cases, the reference tool fails to find any suspect line while our approach provides a list containing the actual defect. These experiments on circuit c432 show the correctness of the proposed methodology.

A second comment on the results is that our approach (PA and PDA) generally provides more suspect lines than the reference tool. This difference can be simply explained by the fact that the reference tool considers only the stuck-at fault model in its analysis. Only stuck-at fault candidates are therefore provided by the reference tool, which is of course pessimistic. As the proposed approach does not consider a priori a fault model during the diagnosis process, it does not exclude the same lines than the reference tool (note that the same set of stuck-at faults has been found by the two tools). Consequently, the number of suspect lines can be greater than that obtained with the reference tool.

A third comment is about the differences between our PA and PDA. As mentioned before, the optimization of the list of suspect lines, introduced in the presented approach, allows to reduce and rank the list of suspects. So, the last column in Table 13 gives the number of suspects considering the optimization procedure proposed in Section 3.3. As shown in Table 13, the diagnosis resolution is usually lower with the presented diagnosis approach compared to our previous approach. This clearly points out the advantage achieved by introducing the optimization step.

To better explain this situation, we analyze in detail in Table 14 the diagnosis report obtained for the scenario

TABLE 14
Diagnosis Report for L96-Sa1 of c432

LC	CNT	S	Fault Models
L96	5	F0	**Stuck-at 1**, Tn stuck open; Tp stuck on, Open 1, Dominant OR bridge, Byzantine, Strong Driver Wired, Wired OR, Resistive OR, Resistive Open, Delay StF
I10	5	C0+	Dominant OR bridge, Byzantine, Strong Driver Wired, Wired OR
I14	5	C0+	Dominant OR bridge, Byzantine, Strong Driver Wired, Wired OR
I26	5	F0+	Dominant OR bridge, Byzantine, Strong Driver Wired, Wired OR, Resistive OR, Resistive Open, Delay StF
I28	5	C1+	Dominant AND bridge, Byzantine, Strong Driver Wired, Wired AND
L31	5	C0+	Dominant OR bridge, Byzantine, Strong Driver Wired, Wired OR
L49	5	C1+	Dominant AND bridge, Byzantine, Strong Driver Wired, Wired AND
L83	5	F0+	Dominant OR bridge, Byzantine, Strong Driver Wired, Wired OR, Resistive OR, Resistive Open, Delay StF
L84	5	R1+	Dominant AND bridge, Byzantine, Strong Driver Wired, Wired AND, Resistive AND, Resistive Open, Delay StR

TABLE 15
ITC99 Result Summary

Circuit	#TV	#FP	#FPO	#Suspects	Pos.
b17	1931	360	463	28.5	7.03
b20	1246	165	245	33.2	12
b21	1162	215	326	20.5	8.6
b22	1699	199	307	28.4	8.7

reported in the second row of Table 13 (L96-Sa1). In this case, the reference tool reports only one suspect (L96-Sa1) while our approach identifies nine suspects in the diagnosis report (reported in Table 14). Each suspect has the CNT value equal to 5. Therefore, the ranking of the suspects is done by considering the results of the fault simulation. As explained in the previous section, lines with symbol without any marker are the most likely to be the real cause of the observed error. In this example, the most likely suspect is line L96 associated with a stuck-at 1 fault, which is the real cause of the observed error. The other suspect lines in the diagnosis report are associated with a fault that leads the circuit to have the same faulty behavior than that one obtained after injecting Sa1 on line L96. Again, this proves the fact that the proposed approach is not oriented to consider only one type of fault but rather a comprehensive set of faults.

The next set of experiments aims at proving the efficiency of the proposed diagnosis approach when a single fault occurs in the CUT. Table 15 reports the results obtained on ITC99 benchmarks (named in Column 1). Columns 2, 3, and 4 give the number of applied test vectors, the number of failing test vectors, and the number of failing outputs, respectively. Column 5 reports the average number of suspect lines from 100 random scenarios (injected faults). As can be seen, the number of suspects is quite low (less than 33 w.r.t. approximately 20,000 lines). Moreover, the average ranked position of the actual faulty line is always less than 12 (Column 6).

Finally, we investigate the capability of our approach to deal with multiple faults by considering a subset of multiple faults induced by some bridging faults (Byzantine, Wired And, Wired Or, Strong Driver Wired, etc.). These types of bridging faults show a multiple faulty behavior and we use it as a case study to analyze multiple faults.

For each fault model, we performed 100 random fault injections. Table 16 presents the results on ISCAS89 and ITC99 benchmark circuits. Column 2 shows the number of applied test vectors while Column 3 reports the injected fault model. Columns 4 and 5 give the number of failing patterns and the number of failing outputs, respectively. Column 6 (#CL) gives the average number of lines traced at least one time by the CPT. Column 7 (#S) provides the average number of lines having a CNT equal to #FPO.

A first comment is that, again, we always find as suspect the line on which we have injected the fault as well as the type of fault model.

A second comment is related to the absolute number of suspects provided by the tool (#CL), which is always much lower than the total number of CUT lines.

A third comment is about the position of the line on which we injected the fault in the ranked list of suspects (Column 8). We can see that the line is always classified among the most likely suspects (on average, between positions 2 and 19). It is important to note that the resolution (accuracy) strictly depends on the test set used during the test phase. Since for our experiments the test set was generated for stuck-at faults, the coverage, and consequently, the diagnosis resolution on other fault models could be lower than those obtained on the stuck-at faults. Although the resolution may vary with the size of the CUT and the test set, we can consider that we proposed a very efficient diagnosis tool that can provide reliable and fruitful information to be used during the next phase of the failure analysis process.

TABLE 16
ISCAS'89 and ITC'99 Result Summary

Circuit	#TV	Scenario	#FP	#FPO	#CL	#S	Pos.
s38584	866	SDW	590	685	283	7	4
		W-AND	285	340	235	13	7
		W-OR	315	387	172	7	4
		Byzantine	550	641	294	6	5
		Dom AND	95	110	312	9	7
		Dom OR	398	410	215	7	4
b20	1246	SDW	380	464	1261	20	12
		W-AND	350	339	1325	7	8
		W-OR	478	559	1358 2	3	14
		Byzantine	315	641	294	6	5
		Dom AND	274	355	513	12	8
		Dom OR	240	295	540	10	5
b21	1162	SDW	952	396	1224	16	11
		W-AND	110	249	1496	10	10
		W-OR	271	542	1660	12	9
		Byzantine	752	1441	2512	2	7
		Dom AND	745	821	1312	8	3
		Dom OR	319	680	1423	15	2
b22	1699	SDW	411	560	1050	29	19
		W-AND	277	435	1304	7	8
		W-OR	396	483	1318	8	12
		Byzantine	741	1063	2239	3	7
		Dom AND	479	744	939	18	12
		Dom OR	741	764	1251	12	8

In Table 17, we report the performance comparison between the proposed approach and the reference commercial tool in terms of required CPU time (in second) and memory size (in megabyte). The first column gives the circuit used to perform 100 experiments (Column 2). Columns 3 and 4 provide the average number of faulty patterns and failing primary outputs, respectively. Columns 5 and 6 show the average CPU time and memory requirements of the reference commercial tool (Ref. Tool). The last two columns show the average CPU time and memory requirements of the PDA. As

TABLE 17
Performances

Circuit	#Exp	#FP	#FPO	Ref. Tool		PDA	
				CPU Time (s)	Mem (MB)	CPU Time (s)	Mem (MB)
s38584	100	308.3	475.0	360.1	32	54.1	35
b20	100	160.5	284.2	55.9	24	26.6	15
b21	100	247.1	500.5	50.37	24	54.8	15
b22	100	308.3	475	164.2	24	54.1	35

clearly shown in the table, the CPU time and Memory size required by our approach are generally lower than those of the reference tool. Moreover, the CPU time is always lower than 60 s and the memory size used by our approach is at most 35 MB. Again, this proves the feasibility of our approach.

6 Conclusion

The diagnosis approach proposed in this paper relies on two phases. In the first phase (Fault Localization), a set of suspect lines is obtained. Then, in the second phase (Fault Model Allocation), a set of realistic fault models is associated with each suspect line by resorting to fault evidences extracted during the previous step. Compared to other diagnosis solutions, the proposed approach allows to obtain more comprehensive and realistic sets of fault models in a unified manner (i.e., at the same time). Moreover, it is able to manage both single and multiple faults.

Experiments have shown the efficiency of the proposed approach in terms of reliability, diagnosis resolution, and accuracy of the associated fault models. Future work will consider more information provided from the fault simulation in order to achieve a better resolution to obtain a more comprehensive ranking procedure of the suspected lines.

References

[1] J.M. Abramovici, M.A. Breuer, and A.D. Friedman, *Digital System Testing and Testable Design.* IEEE Press, 1990.

[2] J.A. Waicukauski and E. Lindbloom, "Failure Diagnosis of Structured VLSI," *IEEE Design & Test of Computers,* vol. 6, no. 4, pp. 49-60, July 1989.

[3] I. Pomeranz, "On Pass/Fail Dictionaries for Scan Circuits," *Proc. IEEE Asian Test Symp.,* pp. 51-56, 2001.

[4] I. Pomeranz and S.M. Reddy, "On Dictionary-Based Fault Location in Digital Logic Circuits," *IEEE Trans. Computers,* vol. 46, no. 1, pp. 48-59, Jan. 1997.

[5] M. Abramovici, P.R. Menon, and D.T. Miller, "Critical Path Tracing—an Alternative to Fault Simulation," *IEEE Design & Test of Computers,* vol. 1, no. 1, pp. 83-92, Feb. 1984.

[6] K. Shigeta and T. Ishiyama, "An Improved Fault Diagnosis Algorithm Based on Path Tracing with Dynamic Circuit Extraction," *Proc. IEEE Int'l Test Conf.,* pp. 235-244, 2000.

[7] S. Venkataraman and S.B. Drummonds, "Poirot: Applications of a Logic Fault Diagnosis Tool," *IEEE Design & Test of Computer,* vol. 18, no. 1, pp. 19-30, Jan./Feb. 2001.

[8] R. Desineni, O. Poku, and R.D. Blanton, "A Logic Diagnosis Methodology for Improved Localization and Extraction of Accurate Defect Behavior," *Proc. IEEE Int'l Test Conf.,* pp. 1-10, 2006.

[9] M.E. Amyeen, D. Nayak, and S. Venkataraman, "Improving Precision Using Mixed-Level Fault Diagnosis," *Proc. IEEE Int'l Test Conf.,* pp. 1-10, 2006.

[10] J.B. Liu and A. Veneris, "Incremental Fault Diagnosis," *IEEE Trans. Computer Aided Design of Integrated Circuits and Systems,* vol. 24, no. 2, pp. 240-251, Feb. 2005.

[11] X. Fan, W. Moore, C. Hora, and G. Gronthoud, "A Novel Stuck-at Based Method for Transistor Stuck-Open Fault Diagnosis," *Proc. IEEE Int'l Test Conf.,* pp. 1-4, 2005.

[12] X. Fan, W. Moore, C. Hora, and G. Gronthoud, "Stuck-Open Fault Diagnosis with Stuck-at Model," *Proc. IEEE European Test Symp.,* pp. 182-187, 2005.

[13] P. Engelke, I. Polian, M. Renovell, and B. Becket, "Simulating Resistive Bridging and Stuck-at Faults," *Proc. IEEE Int'l Test Conf.,* pp. 1051-1059, 2003.

[14] D.B. Lavo, B. Chess, T. Larrabee, and F.J. Ferguson, "Diagnosing Realistic Bridging Faults with Single Stuck-at Information," *IEEE Trans. Computer Aided Design of Integrated Circuits and Systems,* vol. 17, no. 3, pp. 255-267, Mar. 1998.

[15] L.M. Huisman, "Diagnosing Arbitrary Defects in Logic Designs Using the Single Location at a Time (SLAT)," *IEEE Trans. Computer Aided Design of Integrated Circuits and Systems,* vol. 23, no. 1, pp. 91-101, Jan. 2004.

[16] D.B. Lavo, I. Hartanto, and T. Larrabee, "Multiplets, Models, and the Search for Meaning: Improving Per-Test Fault Diagnosis," *Proc. IEEE Int'l Test Conf.,* pp. 250-259, 2002.

[17] S. Holst and H.-J. Wunderlich, "Adaptive Debug and Diagnosis without Fault Dictionaries," *Proc. IEEE European Test Symp.,* pp. 7-12, 2007.

[18] A. Rousset, A. Bosio, P. Girard, C. Landrault, S. Pravossoudovitch, and A. Virazel, "DERRIC: A Tool for Unified Logic Diagnosis," *Proc. IEEE European Test Symp.,* pp. 13-20, 2007.

[19] P. Girard, C. Landrault, and S. Pravossoudovitch, "Delay Fault Diagnosis by Critical-Path Tracing," *IEEE Design & Test of Computers,* vol. 9, no. 4, pp. 27-32, Oct. 1992.

[20] M. Abramovici and M.A. Breuer, "Multiple Fault Diagnosis in Combinational Circuits Based on an Effect-Cause Analysis," *IEEE Trans. Computer,* vol. 29, no. 6, pp. 451-460, June 1980.

[21] A. Rousset, A. Bosio, P. Girard, C. Landrault, S. Pravossoudovitch, and A. Virazel, "Fast Bridging Fault Diagnosis Using Logic Information," *Proc. IEEE Asian Test Symp.,* pp. 33-38, 2007.

Alberto Bosio received the PhD degree in computer engineering from the Politecnico di Torino in Italy in 2006. Currently, he is an associate professor in the Laboratory of Informatics, Robotics and Microelectronics of Montpellier (LIRMM)—University of Montpellier, France. He has published articles in publications spanning diverse disciplines, including memory testing, fault tolerance, diagnosis, and functional verification. He is a member of the IEEE.

Patrick Girard received the MS degree in electrical engineering and the PhD degree in microelectronics from the University of Montpellier, France, in 1988 and 1992, respectively. He is currently the research director at the French National Center for Scientific Research (CNRS), and works in the Microelectronics Department of the Laboratory of Informatics, Robotics and Microelectronics of Montpellier (LIRMM)—France. He is the vice chair of the European Test Technology Technical Council (ETTTC) of the IEEE Computer Society. He is currently the editor-in-chief of the ASP *Journal of Low Power Electronics* (JOLPE) and an associate editor of the *IEEE Transactions on VLSI Systems* and the *Journal of Electronic Testing Theory and Applications* (JETTA—Springer). From 2005 to 2009, he was an associate editor of the *IEEE Transactions on Computers*. He has served as technical program committee member of the ACM/IEEE Design Automation Conference (DAC), the ACM/IEEE Design Automation and Test in Europe (DATE), the IEEE International Test Conference (ITC), the IEEE International Conference on Computer Design (ICCD), the IEEE International Conference on Design and Test of Integrated Systems (DTIS), the IFIP International Conference on VLSI-SOC, the IEEE VLSI Test Symposium (VTS), the IEEE European Test Symposium (ETS), the IEEE International Online Testing Symposium (IOLTS), the IEEE Asian Test Symposium (ATS), the ACM/IEEE International Symposium on Low Power Electronic Design (ISLPED), the IEEE International Symposium on Electronic Design, Test and Applications (DELTA), and the IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS). He has served as test track chair for DAC 2007, DATE 2007, DATE 2008, ICCD 2008, and DATE 2009. He has also served as program chair for DELTA 2006, DTIS 2006, DDECS 2007, and ETS 2008. He has been involved in several European research projects (ESPRIT III ATSEC, EUREKA MEDEA, MEDEA+ ASSOCIATE, IST MARLOW, MEDEA+ NanoTEST, CATRENE TOETS) and has managed industrial research contracts with major companies like Infineon Technologies, Atmel, STMicroelectronics, etc. His research interests include the various aspects of digital testing and memory testing, with special emphasis on DfT, BIST, diagnosis, delay testing, and power-aware testing. He has supervised 20 PhD dissertations and has published six books or book chapters, 30 journal papers, and more than 110 conference and symposium papers on these fields. He received the Best Paper Award at ETS 2004 and at DDECS 2005. He is a member of the IEEE.

Serge Pravossoudovitch received the PhD degree in electrical engineering in 1983 for his work on symbolic layout for IC design and the doctorat dtat degree in 1987 for his work on switch-level automatic test pattern generation. He is currently a professor in the Electrical and Computer Engineering Department at the University of Montpellier. His research activities are performed in the Laboratoire dInformatique, de Robotique et de Microlectronique de Montpellier (LIRMM). Since 1984, he is working in the testing domain. He is presently interested in memory testing, delay fault testing and diagnosis, design for testability, and power consumption optimization. He has authored and coauthored numerous papers on these fields, and has supervised several PhD dissertations. He has also participated in several European projects (Microelectronic regulation, Esprit, Medea, Catrene). He is a member of the IEEE.

Arnaud Virazel received the MS degree in electrical engineering and the PhD degree in microelectronics from the University of Montpellier, France, in 1997 and 2001, respectively. He is currently an assistant professor at the University of Montpellier II, and works in the Microelectronics Department of the Laboratory of Informatics, Robotics and Microelectronics of Montpellier (LIRMM)—France. He has been on the technical program committee of the International Conference on Embedded and Ubiquitous Computing (EUC) in 2005, the International Symposium and Exhibits in Quality Electronic Design (ISQED) since 2008, the IEEE International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS) in 2008, the ACM/IEEE Design Automation and Test in Europe (DATE) since 2009. He will be on the technical program committee of the IEEE European Test Symposium (ETS) in 2010. He has also served as a reviewer for most of the test conferences (ITC, VTS, DATE, ATS, ETS, DDECS, DELTA, IOLTS) and journals (*JETTA, TCAD, TODAES, JOLPE, IET*). He has been involved in several European research projects (MEDEA+ ASSOCIATE, MEDEA+ NanoTEST, CATRENE TOETS). His research interests include the various aspects of digital testing, with special emphasis on DfT, BIST, diagnosis, delay testing, power-aware testing, and memory testing. He has published more than 10 journal papers and 40 conference and symposium papers on these fields. He is a member of the IEEE.

Are IEEE-1500-Compliant Cores Really Compliant to the Standard?

Alfredo Benso
Politecnico di Torino

Stefano Di Carlo and Paolo Prinetto
Politecnico di Torino

Alberto Bosio
LIRMM

Editor's note:
Functional verification of complex SoC designs is a challenging task, which fortunately is increasingly supported by automation. This article proposes a verification component for IEEE Std 1500, to be plugged into a commercial verification tool suite.

—Erik Jan Marinissen, IMEC

The continual scaling of semiconductor technology represents the basis for SoC integration. Consequently, the race to market high-quality products with an ever shorter design-to-manufacturing cycle compels designers and manufacturers to emphasize the reuse of IP-embedded cores into SoC design. Core reuse also implies the ability to reuse the core test in the SoC test. IEEE Std 1500 for Embedded Core Test is currently one of the most effective solutions for supporting SoC manufacturing test,[1] and it also allows easy interoperability among different cores.[2]

The standard defines a set of guidelines to build a scalable and standard test data interface (*core test wrapper*), completed with an information model describing the implemented test features. The IEEE 1500 information model, described using the IEEE Std 1450.6 Core Test Language (CTL),[3] expedites the transfer of test data from core providers to core integrators.

An entire set of challenges arises when we consider the heterogeneous nature of today's IP market. Core providers must provide IEEE-1500-compliant cores to integrators to facilitate customer test integration into system-level infrastructures. Core integrators, on the other hand, must ensure that IEEE-1500-ready IP blocks properly comply with the standard's required functionalities. The question for both designers and integrators is whether IEEE-1500-compliant cores really are compliant to the standard.

The design of an IEEE 1500 core test wrapper includes several steps whereby bugs might be introduced. To support a wide range of test applications, IEEE 1500 defines only a minimal set of mandatory hardware features, giving designers the freedom to extend the test infrastructure with virtually unlimited sets of registers and instructions.[2,4] A comprehensive approach to thoroughly verify the functionality of IEEE 1500 core test wrappers in a SoC environment is therefore mandatory.

The problem of verifying an IP core's compliance to IEEE 1500 has been poorly addressed in the literature. To solve the problem of verifying an IP core's compliance to IEEE 1500, Globetech Solutions (http://www.globetechsolutions.com) has proposed a commercial answer.[5-7] In part, for example, Diamantidis et al. addressed IEEE 1500 in proposing a unified DFT verification methodology to provide a complete, methodical, and automatic verification flow for SoC DFT infrastructures.[5] The authors showed how to build and manage a database of reusable verification components, targeting different DFT techniques. The database facilitates the implementation of a complete SoC verification plan. The authors only briefly mentioned IEEE 1500 verification as an example of a verification component.

Elsewhere, Diamantidis et al. and Oikonomou et al. introduced an IEEE 1500 compliancy verification

0740-7475/09/$25.00 © 2009 IEEE Copublished by the IEEE CS and the IEEE CASS

component based on dynamic functional verification.[6,7] The component performs verification by comparing the given design with a reference model. One of this solution's key advantages is the ability to consider the verification of both isolated and daisy-chained wrappers. Nevertheless, although the authors claim their verification flow is completely automated, they provide little information on how the verification component is actually configured, and their verification plan is predefined, which means the plan might be not optimal when looking at the overall SoC verification, and therefore it might prevent the reduction of overall SoC verification time. Moreover, they do not completely specify how the reference model is implemented, and it is not clear whether the model can systematically address every aspect of the standard. For example, it is not clear how that model verifies the IEEE 1500 information model, which is a relevant part of the standard. Finally, it is not clear how that model measures overall compliancy to the standard.

Recently, we presented a Unified Modeling Language (UML) abstraction of an IEEE 1500 verification framework.[8] The main contribution of that work was the definition of a rule-based approach for IEEE 1500 compliancy verification that systematically addressed each design rule imposed by the standard. In particular, besides focusing on the implementation alternatives, we proposed a design rule classification on the basis of the methodology required for their verification, and we defined a detailed abstract model of the framework.

In this article, we present a complete implementation of the abstract framework model proposed previously.[8] We show how to map most of the verification tasks required by that model into the functionalities of a commercial verification tool such as Verisity's Specman Elite (http://www.verisity.com/products/specman.html; Verisity is now part of Cadence). In particular, we will show how

- IEEE Std 1647 Functional Verification Language e (*e* for short) can be easily used to write reusable rule verification components;[9]
- the embedded interface of Specman Elite with a commercial simulator can be used to verify a critical class of design rules; and
- coverage metrics the tool provides can be used to measure the quality and completeness of the verification process.

We also present an application of the proposed prototype verification framework to a benchmark SoC's validation. The outcome of this article is a verification component that designers and integrators can insert into a global SoC verification strategy such as that described by Diamantidis et al.[5]

Verification flow

Figure 1 introduces the basic steps of our proposed verification flow. The goal is to see if an IEEE-1500-compliant core fully respects (observes) the design rules defined by the standard. By "IEEE-1500-compliant core" (*core* for short), we mean an IP core that incorporates an IEEE 1500 core test wrapper,

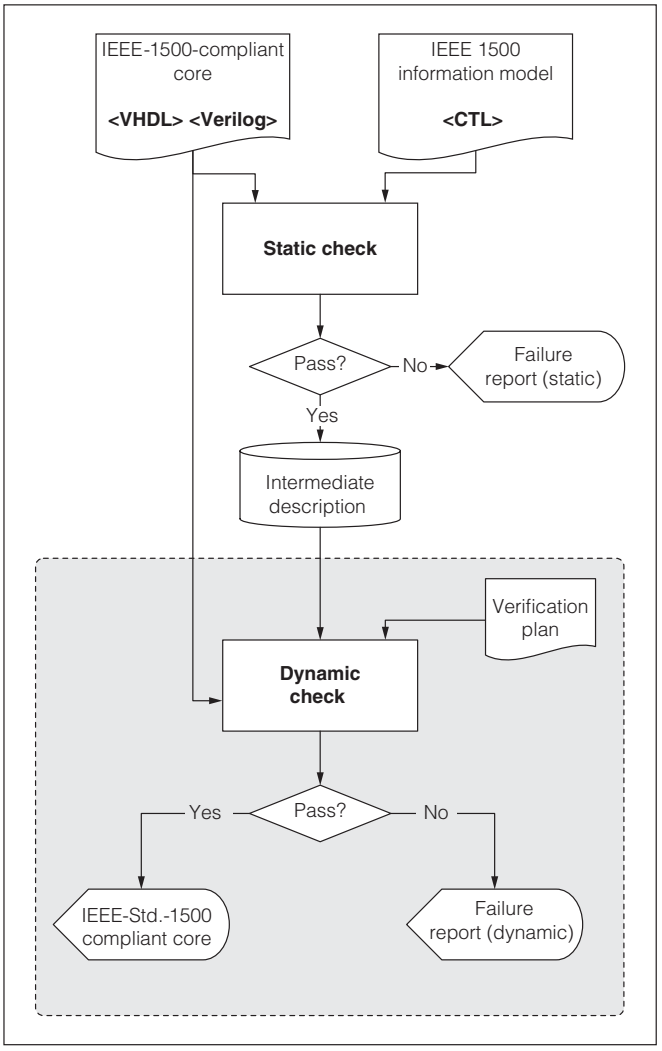


Figure 1. IEEE 1500 verification flow.

and comes with a corresponding CTL file (which is the IEEE 1500 information model) that describes the core test knowledge, including how to operate the wrapper at its external terminals. We consider cores provided using hardware description languages (HDLs) such as Verilog, VHDL, and so on.

The IEEE 1500 verification process consists of two distinct phases: static and dynamic check. The static check is a preliminary verification task to verify the syntax of the core description and the related CTL file. This check is particularly important for the CTL file. Although the core's HDL description is usually automatically synthesized and should not contain syntax errors, CTL files generation is usually not automated. The core designer should therefore identify any incorrect CTL files, which may contain errors or CTL dialects that are not fully compliant with the standard, before moving on to the next verification steps. The syntax analysis can be efficiently performed using special programs called lexers and parsers[10] automatically built over a formal grammar—for example, Vern Paxson's JFlex (http://jflex.de) and Scott Hudson's CUP (http://www2.cs.tum.edu/projects/cup).

Besides verifying the syntax analysis, another goal of the static check is to collect IEEE-1500-related information and convert it to a suitable format for the next verification steps (the "intermediate description" in Figure 1 refers to this process). Information collected during this phase includes the wrapper's signals and register names, implemented instructions, and so on. Because the static check phase is not complex, we don't provide additional details on its implementation but focus instead on the more interesting, dynamic aspects of verification.

To understand the tasks to be performed during the dynamic check phase, we must first recall the characteristics of IEEE 1500 as Benso et al. modeled them.[8] The standard, a collection of design rules for a core test wrapper realization,[1] identifies two classes of wrapper aspects that must be verified: semantic aspects and behavioral aspects.

Semantic aspects concern the CTL IEEE 1500 information model and do not depend on the core behavior. Verifying semantic aspects involves identifying the correct definition of structures such as scan chains, macros, and environments in the information model. These can be easily verified by analyzing the CTL file provided with the core. Although this is a fast and powerful way to verify part of IEEE 1500 compliancy (because it does not require any simulation of the core itself), it is not enough to guarantee the core's complete verification. Semantic aspects are only a relatively small subset of the whole set of rules to verify. Moreover, semantic analysis is performed on information model data supplied by the core provider, so there is no guarantee that the data perfectly matches the actual hardware implementation. An efficient way to implement semantic aspects verification is to store the intermediate description in Figure 1 into a relational database. The verification can then be easily translated into a set of queries performed on the database.

Behavioral aspects target communication protocols and core behavior, and they are the most difficult to verify. Their verification requires core simulation. In fact, simulation is the only effective approach to verify timing, protocols, signal connections, and correct implementation of instructions. ATPG tools perform a similar task, for example, during design rule checking. Nevertheless, these tools must deal with well-known test structures inserted into the circuit, thus reducing the verification activity's complexity. With respect to IEEE 1500, the way the core test wrapper is implemented relates strictly to the target design, which makes verification more complex. Accordingly, in this article we exploit dynamic functional verification for this task.[11]

Finally, an overview of the verification flow would be incomplete without a few considerations on how both semantic and behavioral aspects can be derived from the standard. IEEE 1500 design rules are provided in a natural-language format. By thoroughly analyzing the standard, we can identify the following information for each rule:

- *Rule type.* Does the rule identify a behavioral aspect, a semantic aspect, or both?
- *Involved units.* Which wrapper elements does the rule involve (wrapper instruction register, wrapper bypass register, and so forth)?
- *Dependence.* What dependence does this rule have with other rules?
- *Observation points.* This concerns the wrapper's elements—that is, signals, and units, where a potential violation of the rule can be observed.
- *Actions.* A natural-language description of the actions required to verify the rule.

Starting from this structured description, we have translated each action into the appropriate

formalism—for example, the *e* language for behavioral aspects—to build the verification framework.

Dynamic functional verification

By *functional verification* we mean the task of demonstrating that the intent of a design is preserved in the design's implementation.[11] The most widespread method of functional verification is *dynamic functional verification*. It is called dynamic because input patterns are generated and applied to the design by simulation, and the corresponding results are collected and compared against a reference model for checking their conformance with the specifications.

Verifying all possible behaviors under every possible combination of inputs is, in most cases, unfeasible because the test space is too large to be fully covered in a reasonable amount of time. To overcome this problem, one of the best solutions is to apply *constrained-random* pattern generation. Verification patterns are randomly generated under a set of constraints, limiting the set of legal values on the input signals that drive the design. Moreover, we could also apply *coverage-driven* verification. Functional coverage metrics are automatically stored in real time to ascertain whether, and how effectively, a particular test verifies a given feature. This information can then be fed back into the generation process to drive additional verification effort toward the required goal. Coverage metrics are evaluated on coverage monitoring points defined by the user and specified in the verification plan.

The market offers various tools to support dynamic functional verification—for example, Verisity's Specman Elite and Synopsys' Vera. Specman Elite (Specman for short), which is our reference verification environment, uses the *e* language to capture behaviors defined in the specifications and to automatically generate tests. Designers use the *e* language to write *e* verification cores (eVCs)—that is, software modules modeling the functional behavior of the environment surrounding the system under verification.

IEEE 1500 behavioral-aspects verification

The verification of IEEE 1500 behavioral aspects starts from the definition of an eVC able to fully stress the target core and to compare the obtained results with a core test wrapper reference model. Besides modeling a generic reference IEEE 1500 core test wrapper (which is almost impossible, due to the number of customizations allowed by the standard), we accordingly consider all IEEE 1500 design rules and determine whether the target design actually respects all rules.[8] This will in turn require us to

- translate rule design constraints into portions of *e* code that can generate verification patterns and verify the rules' correctness; and
- identify, for each rule, the related coverage points—that is, wrapper signals or registers used to evaluate the coverage reached during the verification process.

The challenge stems from the definition of rule verifiers and rule coverage points independent of the specific wrapper design. This definition strongly depends on the core internal structure's level of controllability and observability. In a white-box design, which is customarily what core designers develop, the core internal structure is completely accessible. Internal core signals can be fully controlled and observed; therefore, designers can fully verify all IEEE 1500 rules. In a black-box design—the usual situation for core integrators, who buy cores from different vendors—the only available information for IP protection is the core I/O interface. This strongly impacts Specman's ability to generate verification patterns and evaluate the verification coverage. The only rules that can be fully verified are those that do not require direct controllability or observability of core or wrapper internal signals. Full IEEE 1500 compliancy verification can thus be achieved only for white- or black-box designs that implement the standard's basic functionalities.

IEEE 1500 eVC architecture

Here, we highlight the IEEE 1500 eVC architecture used to verify IEEE 1500 behavioral aspects. The full verification is based on a single verification component, according to the recommendations of the *e* Reuse Methodology (eRM; http://www.verisity.com/products/erm.html). We refer to IEEE 1500 rules with their corresponding rule number.[1]

Figure 2 sketches the architecture of the proposed eVC, highlighting for each block the related *e* files. The `IEEE_1500_env` (IEEE 1500 environment) built over the predefined `any_env` unit (a default environment used as the starting point to build eVCs), according to eRM, represents the overall eVC

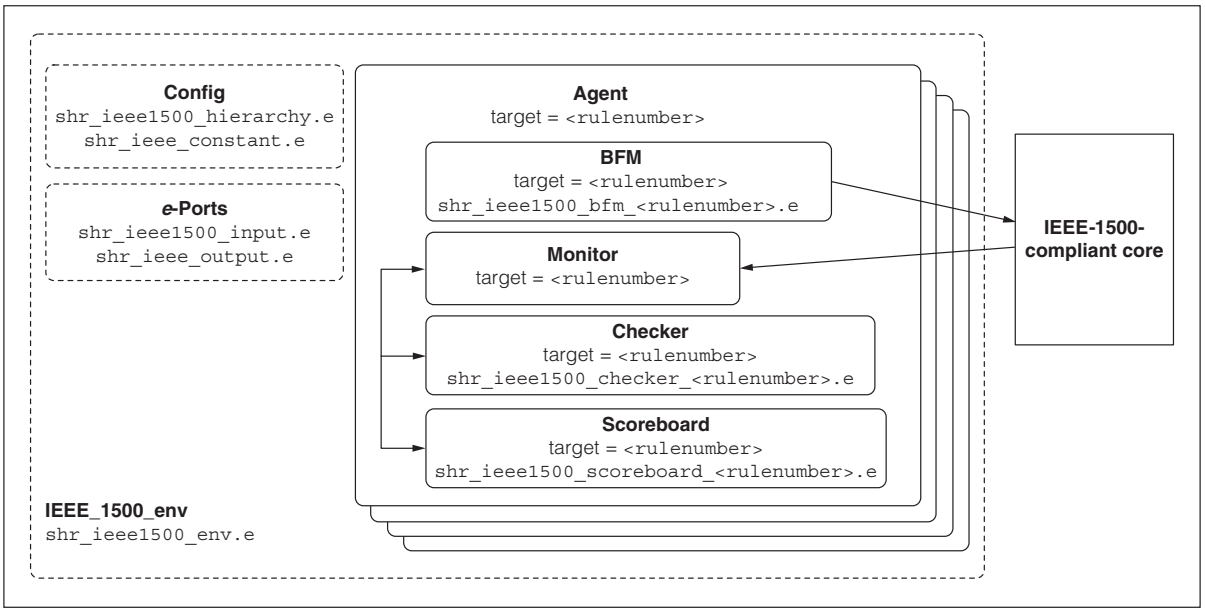


Figure 2. Structure of the IEEE Std 1647 Functional Verification Language *e* verification core (eVC).

environment. The environment defines the verification events required to perform the verification. For each design rule under verification, the environment defines a start event to begin the rule verification and an end event to control the verification result. Figure 3a shows an example of `shr_ieee1500_env.e`, including a checkpoint flag to stop the verification if one of the defined rules is not observed.

The eVC ports are defined through the `shr_ieee1500_input.e` and `shr_ieee1500_ouput.e` files. The core is then configured through the `shr_ieee1500_constant.e` and `shr_ieee1500_hierarchy.e` files. The `shr_ieee1500_constant.e` file provides the eVC enough information to apply verification patterns at the core inputs, monitor the corresponding outputs, and store core-specific information, such as wrapper signal names, register sizes, and so on. The generation of this configuration file can be easily automated, starting from the intermediate description of the target core (see Figure 1). This file provides the basis for guaranteeing the eVC's reusability.

The `shr_ieee1500_hierarchy.e` file is one of the eVC's most important elements. It defines the verification plan—that is, what must be verified and in which order. It describes the scope of the verification problem and serves as the functional specification for the verification environment, highlighting dependencies among results of different verification steps. Moreover, it lets us optimize the verification time, thus reducing overall verification costs. Figure 3b is a small verification plan example. Each time the verification of a given rule ends—that is, the `end_rulenumber` event occurs—the verification plan identifies the next rule to verify. When there is rules dependency, the result of a specific rule verification might modify the verification plan. For example, consider rule `7_4_1_a` in Figure 3b: its verification starts only if rule `10_3_1_j` is correctly verified. Moreover, rule `10_3_1_j` is verified after rule `10_3_1_a3`, regardless of whether the verification of rule `10_3_1_a3` was successful. Users can freely modify this file to build their own verification plan.

The actual verification is then performed by instantiating multiple eVC agents. The agent has a subtype target to identify the target IEEE 1500 design rule. Each agent instantiates a Specman bus functional model (BFM) and a monitor. The BFM generates and simulates the verification patterns for the target rule, and the monitor evaluates the simulation results to understand whether the rule is respected (by using the subtype target). The monitor includes a scoreboard to perform a static analysis of the simulation results (that is, it verifies whether or not the core output signals assume the proper sequence of values regardless of their actual timing), and a

checker to perform the timing analysis. Moreover, the BFM defines the set of coverage items used to evaluate the final verification coverage.

To better understand the proposed verification strategy, consider the following behavioral rule corresponding to Chapter 10.3.1, rule (e), of IEEE 1500[1] (WIR: wrapper instruction register; WRCK: wrapper clock; WRSTN: wrapper reset):

> *The WIR circuitry shall retain its current state (i.e., shift stage values and currently active modes) indefinitely while the WRCK signal is stopped (i.e., WRCK held at a fixed logic value of 1 or 0) and the signal connected to the WRSTN terminal is logic 1.*

The verification of this rule requires that we

1. Fetch one instruction.
2. Force WRCK to 0 (1).
3. Force WRSTN to 1.
4. Drive all the wrapper's input signals except WRCK and WRSTN.
5. Check that the WIR circuitry retains its state.

```
event start_13_1_1_a;
event end_13_1_1_a;
event start_7_2_1_c;
event end_7_2_1_c;
event start_7_2_1_b;
event end_7_2_1_b;
checkpoint: bit;
keep soft checkpoint == 0;
```

(a)

```
on end_10_3_1_a3 {
   emit start_10_3_1_j;
};
on end_10_3_1_j {
   if checkpoint == 1 {
     stop_run;
   } else {
     emit start_7_4_1_a;
   };
};
```

(b)

```
<'
gen_and_drive (sig_size: uint, sig_name: list of string, is_wire:
list of bit) is {

   var i: uint;

   gen input_vector keeping {

      -- Generates a random vector

      .size()==sig_size;
   };

   for i from 0 to sig_size - 1 do {

      -- Assigning generated values to sig_name list of signals

      drive_value(sig_name[i], is_wire[i], pack(packing.low,

                  input_vector[i:i]));
}; '>
```

(c)

Figure 3. eVC code snapshots: eVC environment (a), hierarchy (b), and rule verification example (c).

All operations are repeated for the 11 mandatory or optional instructions defined by the standard.[1] Item 4 resorts to random generation to efficiently drive a not-involved signal. Figure 3c shows the *e* code of the function in charge of randomly generating a vector and of driving it to the device under test. The vector drives all signals specified in `shr_ieee1500_constant.e`. In general, each rule determines a set of signals with a deterministic fixed value, while for the remaining signals, we apply random generation. Because of the proposed eVC's open architecture, users can extend the random generation function while considering the specific core functionality in order to cope with corner cases. Item 5 is finally performed by the checker; it ensures that the core respects the rule.

A complete analysis of IEEE 1500 enabled us to identify a set of 165 mandatory rules and 27 optional rules. Our current prototype eVC component implements the verification of 138 mandatory rules summarized as follows: 40 semantic rules, 25 behavioral rules, and 73 mixed rules, including both semantic and behavioral aspects. Moreover, we can readily extend eVC to address new rules for optional properties of the standard by adding additional agent instantiations for the new rules and by inserting their verification in the overall verification plan.

Experimental results

We have tested the effectiveness of our proposed verification flow on a benchmark SoC consisting of an 8080 8-bit microprocessor, and an 8-bit programmable interrupt controller (PIC) from OpenCores (http://www.opencores.it), complete with a 256 bit × 8 bit

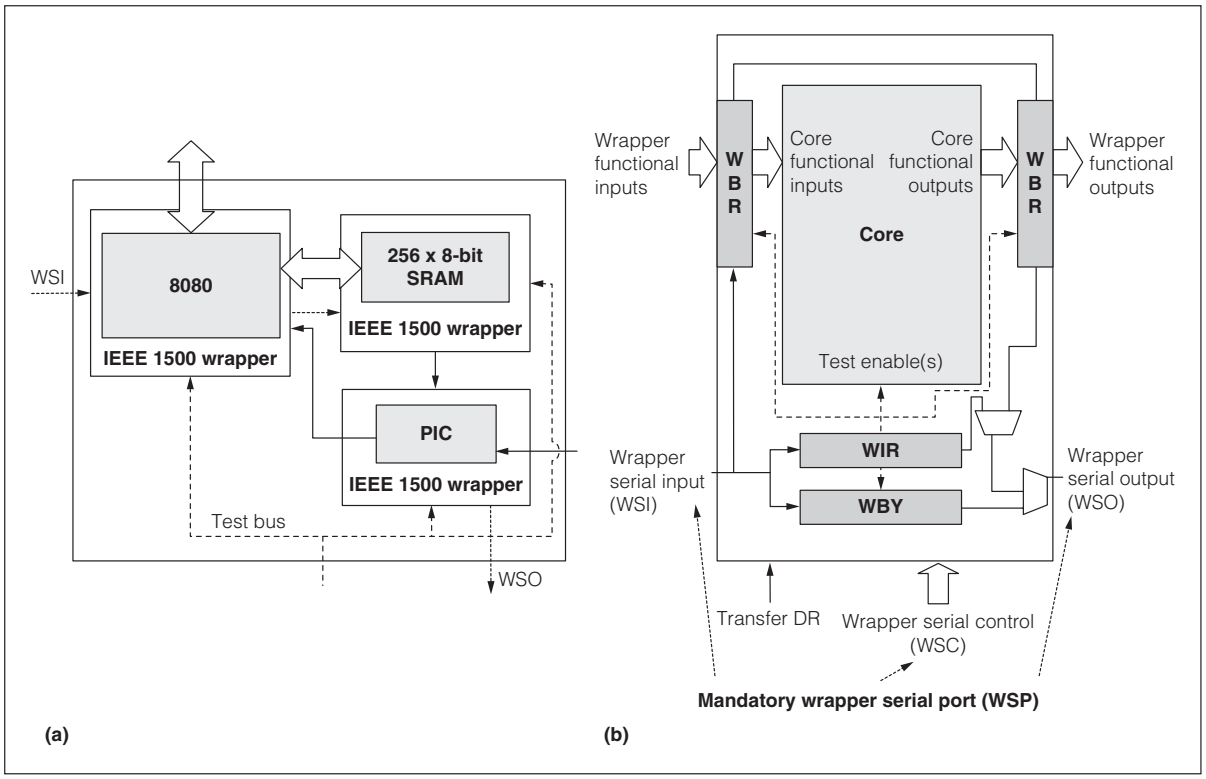


Figure 4. Test case general architecture: SoC architecture (a) and wrapper architecture (b).

SRAM. Each core has an IEEE 1500 core test wrapper with the following characteristics:

- a wrapper instruction register (WIR), 3-bit length;
- a wrapper bypass (WBY) register, 1-bit length;
- a wrapper boundary register (WBR);
- an optional *TransferDR* wrapper serial control; and
- four implemented instructions: `WS_BYPASS`, `WS_PRELOAD`, `WS_INTEST`, and `WS_EXTEST`.

Figure 4a shows the basic SoC architecture, and Figure 4b shows the wrapper architecture. Table 1 shows the SoC complexity in terms of gates.

We applied the complete verification flow to the three SoC cores using a Sun Blade 1000 workstation, and 100% of the implemented rules were correctly verified. Table 2 shows the verification time for each core in terms of clock cycles and CPU time. The simulation time is obviously strictly connected to the core's complexity.

In addition to providing the list of verified rules, Table 3 shows an example of verification coverage analysis on five rules. Verification coverage metrics are an efficient and easy way to assess the result of verification. For each IEEE 1500 rule and defined coverage items that we considered (in column 1), we

Table 1. Synthesized SoC characteristics.

Block type	Number of blocks
Combinational gate	11,292
Nonscan flip-flops	3,092
RAM blocks	1 (256 bits × 8 bits)

Table 2. Simulation time.

Core	Clock cycles	CPU time (s)
8080	207,453	840
SRAM	181,751	730
PIC	18,112	120

* PIC: programmable interrupt controller.

analyzed the actual number of measured hits for the three circuits—8080, SRAM, and PIC—with respect to the number of required hits (RH).

Coverage items and RH depend on the target rule. For example, consider rule 12.1.1.b:

> *The WBR shall have at least one configuration in response to the state of the WIR, allowing serial access to and from all WBR cells between TI and TO.*

This rule implies shifting data through the WBR cells, and requires a number of WBR shift events at least equal to the WBR length. This metric was respected in the three analyzed cores (see Table 3). Table 3 also shows that RH was always reached for all rules, ensuring the reliability of the verification framework.

Table 3. Coverage results.

Rule, coverage Item	No. of hits (8080), RH	No. of hits (SRAM), RH	No. of hits (PIC), RH
10.1.1.c, instruction fetch	4, 4	7, 4	6, 4
12.1.1.b, WBR shift	24, 24	16, 16	8, 8
10.3.1.g, WRSTN transition	4, 4	4, 4	13, 4
13.1.1.c, WRSTN set	17, 4	4, 4	4, 4
11.1.1.b, WBY register select	1, 1	15, 1	1, 1

* WBR: wrapper boundary register; WBY: wrapper bypass; WRSTN: wrapper reset.

Finally, to carefully validate the verification capabilities of the framework, we designed a set of different wrappers, systematically violating different rules of IEEE 1500. Table 4 summarizes the verification results for each of these experiments. The first column reports the target rule number, the second column describes the injected error, and the third column reports the verification result. All violations

Table 4. Rules failure tests.

Rule	Violation	Detected?
7.2.1.c	No register selected during `WS_PRELOAD`	Yes
7.2.1.e	Modified signals configuration for the update WIR register	Yes
7.4.1.c	Inverted wrapper *reset*	Yes
7.4.1.d	WBR not in functional mode during `WS_BYPASS`	Yes
7.4.1.e	WBY register shift signal stuck at 0	Yes
10.1.1.b	Last flip-flop of WBR not connected to WSO	Yes
10.2.1.a	WBR register never selected	Yes
10.2.1.b	WIR mode signal stuck at 0	Yes
10.2.1.c	WRSTN connected to one of the core inputs	Yes
10.2.1.d	Unconnected wire in WIR	Yes
10.2.1.e	Inverted WSI during WIR shift	Yes, 10.3.1.h fails
10.2.1.f	Update operation on every shift	Yes, 10.3.1.a3 and 10.3.1.j fail
10.3.1.a	`WS_BYPASS` not selectable	Yes, 10.3.1.g, 13.1.1.b, 7.3.1.h, and 13.1.1.d fail
10.3.1.b	*updateWR* signal of WIR combined with one or more core inputs	Yes
10.3.1.d	*updateWR* signal of WIR combined with *captureWR*	Yes
10.3.1.e	WIR shifts without clock	Yes
10.3.1.f	*ShiftWR* signal of WIR connected to a core input	Yes
10.3.1.h	*ShiftWR* sampled on both the rise and fall clock transitions	Yes
10.3.1.i	WSI and WSO sampled at the wrong clock transition	Yes
10.3.1.j	*UpdateWR* sampled at both the rise and fall clock transitions	Yes
11.1.1.a	WBY duplicated	Yes
11.1.1.b	Modified WBY register select signal	Yes

* WBY: wrapper bypass; WIR: wrapper instruction register; WSI: wrapper serial input; WSO: wrapper serial output.

have been correctly detected. It is interesting that, in some cases, the violation of a single rule was propagated to other rules, generating multiple design errors.

Our automated environment for IEEE 1500 compliancy verification targets different users, from core designers to core integrators. The environment can therefore guarantee various compliancy levels, depending on the amount of information about the internal core structure that is available to users. Soon, a verification environment such as this will help designers increase productivity, reduce design time, and optimize the test plan of very complex SoCs. Moreover, the same approach used to build the IEEE 1500 verification environment could be exploited to build verification environments for other types of standards such as IEEE 1149.1. ■

References

1. *IEEE Std. 1500, Testability Method for Embedded Core-Based Integrated Circuits,* IEEE, 2005.
2. Y. Zorian and A. Yessayan, "IEEE 1500 Utilization in SoC Design and Test," *Proc. Int'l Test Conf.* (ITC 05), IEEE CS Press, 2005, pp. 543-552.
3. *IEEE Std. 1450.6, Core Test Language,* IEEE, 2005.
4. D. Appello et al., "A P1500 Compliant BIST-Based Approach to Embedded RAM Diagnosis," *Proc. 10th IEEE Asian Test Symp.* (ATS 01), IEEE CS Press, 2001, pp. 97-102.
5. S. Diamantidis, I. Diamantidis, and T. Oikonomou, "A Unified DFT Verification Methodology," *Proc. IP-Based SoC Design* (IP/SOC 05), EE Times, 2005; http://www.us.design-reuse.com/ipsoc2005.
6. I. Diamantidis, T. Oikonomou, and S. Diamantidis, "Towards an IEEE P1500 Verification Infrastructure: A Comprehensive Approach," *Proc. 3rd IEEE Int'l Workshop on Infrastructure IP,* IEEE Press, 2005, pp. 22-30.
7. T. Oikonomou, I. Diamantidis, and S. Diamantidis, "Coverage Driven Verification of IEEE P1500-Compliant Embedded Core Test Infrastructures," Globetech Solutions, 2005; http://www.globetechsolutions.com/index.php?module=uploads&func=download&fileId=40.
8. A. Benso et al., "IEEE Std. 1500 Compliance Verification for Embedded Cores," *IEEE Trans. VLSI Systems,* vol. 16, no. 4, 2008, pp. 397-407.
9. *IEEE Std. 1647, Functional Verification Language 'e',* IEEE, 2006.
10. N. Wirth, *Compiler Construction,* Addison-Wesley, 1996.
11. A. Piziali, *Functional Verification Coverage Measurement and Analysis,* Springer, 2004.

Alfredo Benso is a tenured associate professor in computer engineering at Politecnico di Torino, Italy. His research interests include DFT techniques, BIST, and dependability. Benso has a PhD in information technologies from Politecnico di Torino. He is a Golden Core member of the IEEE Computer Society and a senior member of the IEEE.

Alberto Bosio is an associate professor at LIRMM (Laboratory of Informatics, Robotics, and Microelectronics in Montpellier), University of Montpellier, France. His research interests include dependability, diagnosis, test generation, and memory testing. Bosio has a PhD in information technologies from Politecnico di Torino. He is a member of the IEEE.

Stefano Di Carlo is an assistant professor in the Department of Control and Computer Engineering at Politecnico di Torino. His research interests include DFT techniques, SoC testing, BIST, memory testing, and reliability. Di Carlo has a PhD in information technologies from Politecnico di Torino. He is a Golden Core member of the IEEE Computer Society and member of the IEEE.

Paolo Prinetto is a full professor of computer engineering at Politecnico di Torino and a joint professor at the University of Illinois at Chicago. His research interests include testing, test generation, BIST, and dependability. Prinetto has an MS in electronic engineering from Politecnico di Torino. He is a Golden Core Member of the IEEE Computer Society.

■ Direct questions and comments about this article to Stefano Di Carlo, Department of Control and Computer Engineering, Politecnico di Torino, Corso Duca degli Abruzzi 24, I-10129 Torino, Italy; stefano.dicarlo@polito.it.

March Test Generation Revealed

Alfredo Benso, *Senior Member*, *IEEE*, Alberto Bosio, *Member*, *IEEE*, Stefano Di Carlo, *Member*, *IEEE*, Giorgio Di Natale, *Member*, *IEEE*, and Paolo Prinetto, *Member*, *IEEE*

Abstract—Memory testing commonly faces two issues: the characterization of detailed and realistic fault models and the definition of time-efficient test algorithms. Among the different types of algorithms proposed for testing Static Random Access Memories, march tests have proven to be faster, simpler, and regularly structured. The majority of the published march tests have been manually generated. Unfortunately, the continuous evolution of the memory technology introduces new classes of faults such as dynamic and linked faults and makes the task of handwriting test algorithms harder and not always leading to optimal results. Although some researchers published handmade march tests able to deal with new fault models, the problem of a comprehensive methodology to automatically generate march tests addressing both classic and new fault models is still an open issue. This paper proposes a new polynomial algorithm to automatically generate march tests. The formal model adopted to represent memory faults allows the definition of a general methodology to deal with static, dynamic, and linked faults. Experimental results show that the new automatically generated march tests reduce the test complexity and, therefore, the test time, compared to the well-known state of the art in memory testing.

Index Terms—Automatic test generation, memory test, memory fault modeling, march tests.

1 Introduction

Memories are the predominant majority in semiconductor device production, with also the fastest growing technology [1]. The complex nature of their internal behavior and the very high density of their cell arrays make memories extremely vulnerable to physical defects. The challenge of memory testing stems from the difficulty of defining realistic fault models and designing time-efficient test algorithms [2].

In the last years, the so-called static faults (i.e., faults sensitized by the execution of just a single memory operation) [3] have been the predominant class of fault models addressed by researchers and the industry. Unfortunately, the latest technologies show new faulty behaviors like dynamic [4] and linked faults [5] that need to be considered as well.

March tests [2] are an efficient class of memory tests with low time complexity and high fault coverage. Several hand-designed and automatically generated march tests have been proposed in the literature.

One of the first march test generation algorithms is presented in [6]. It is based on the notion of a transition tree where each path from the root to a leaf corresponds to a certain march test able to cover a target set of faults. The main drawback of this approach is that the transition tree is unbounded and the search process is of exponential complexity in general. Several improvements to this technique have been proposed. Zarrineh et al. [7] restricts the search process to the parts of the tree where a solution exists using the notion of primitive march tests, whereas in [8], a branch-and-bound approach and a fault-collapsing procedure are used to limit the search space. Al-Harbi and Gupta [9] applies the methodology presented in [8] to generate march tests detecting linked faults. The generation process is not detailed, and only one march test is generated.

All these solutions consider a limited set of static fault models, and the extension to new faults is complex. Niggemeyer and Rudnick [10] presents a generation algorithm for the test and diagnosis of memory faults based on a fault description able to model the complete set of single-cell and two-cell static faults. It suffers from the same problems as [6] since it is still based on transition trees, but it theoretically allows covering all possible static faults, even if only stuck-at faults (SAFs), transition faults (TFs), and idempotent and inversion coupling faults (CFid and CFinv) are considered in the paper.

Wu et al. [11] presents a completely different approach, named Test Algorithm Generation by Simulation (TAGS). Several known march tests of different lengths are generated, and their fault coverage is evaluated using the RAMSES fault simulator [12]. The approach allows high flexibility in terms of fault models, but its complexity is still exponential since it requires an exhaustive search. The authors also propose heuristics to overcome the complexity issue, but in this case, they cannot guarantee the optimality of the results.

In [13], we presented a generation algorithm based on a Test Pattern Graph (TPG) to model static memory faults. The generation problem is thus a search problem on the graph. The main contribution of [13] is the extended set of addressed functional faults. Its main drawback is the computational complexity that reduces the number of total faults that can be included in the target fault list.

- *A. Benso, S. Di Carlo, and P. Prinetto are with the Department of Control and Computer Engineering, Politecnico di Torino, Corso Duca degli Abruzzi 24, I-10129 Torino, Italy. E-mail: {alfredo.benso, stefano.dicarlo, paolo.prinetto}@polito.it.*
- *A. Bosio and G. Di Natale are with the Laboratoire d'Informatique, de Robotique et de Microelectronique de Montpellier, University of Montpellier II/CNRS, 161, rue Ada, 34392 Montpellier Cedex 5, France. E-mail: {alberto.bosio, giorgio.dinatale}@lirmm.fr.*

Manuscript received 20 Feb. 2007; revised 11 Dec. 2007; accepted 24 Apr. 2008; published online 10 July 2008.
Recommended for acceptance by C. Metra.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0064-0207.
Digital Object Identifier no. 10.1109/TC.2008.105.

In [14], we introduced a generation algorithm able to manage both static and dynamic unlinked faults. It is based on a graph representation where the set of edges represents the fault models. March tests are generated by traversing each edge of the graph. Despite its effectiveness, the proposed approach lacks of a rigorous formalization, and in the worst case, it has a nonpolynomial complexity.

In this paper, we propose a new approach to automatically generate march tests targeting static, dynamic, and linked faults. The main contributions of this work are, from one side, a formal fault model representation that extends the fault primitive notation proposed in [3] and a completely new march test generation algorithm with a polynomial complexity that strongly reduces the generation time.

The paper is organized as follows: Section 2 introduces the memory model and the fault model used to automatically generate the march test, whereas Section 3 details the steps of the automatic generation process. Section 4 presents the results obtained by using the proposed algorithm. Section 5 proposes some optimizations, and finally, Section 6 summarizes the main contributions of the paper.

2 Formal Models

The generation algorithm proposed in this paper relies on the use of formal models to describe both the good and the faulty memory behaviors.

2.1 Memory Model

This section introduces the formal model adopted to represent the memory under test. In this paper, we focus on bit-oriented memories only. The extension of the obtained march tests to word-oriented memories can be easily done according to the algorithm proposed in [15].

Definition 1. *An N-cell 1-bit memory can be formally defined as a 4-upla:*

$$\langle A, N, M, X_0 \rangle, \tag{1}$$

where

- $A = \{0, 1\}$ *is the set of possible states of a memory cell,*
- N *is the number of cells,*
- $M = (c_0, c_1, c_2, \ldots, c_{N-1}) | c_i \in A, 0 \leq i \leq N-1$ *is the array of N cells, and*
- $X_0 = \{r^i, w_d^i | 0 \leq i \leq N-1, d \in A\}$ *is the set of possible memory operations, where r^i means a read operation on the cell i, whereas w_d^i means a write operation of the value $d \in A$ on the ith cell.*

The behavior of an N-cell 1-bit memory (Definition 1) can be formally represented by an Edge-Labeled Directed Graph (ELDG) G defined as

$$G = (V, E, L_e), \tag{2}$$

where

- V is the set of 2^N vertices representing the possible states of the memory,
- $E = \{(u, v) | u, v \in V\}$ is the set of edges, each one representing one of the possible memory operations that cause the transition from a vertex u to a vertex v, and

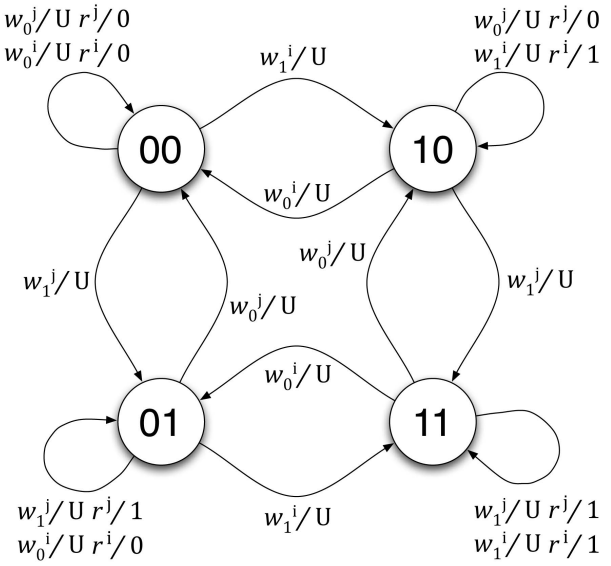


Fig. 1. Two-cell fault-free memory model G_0.

- $L_e : E \rightarrow \{labels\}$ is a label function that maps edges to labels, where given the edge (u, v), the corresponding label is defined as follows:

$$label = x/k, \tag{3}$$

where

  - $x \in X_0$ (Definition 1) is the memory operation able to fire the transition from u to v, and
  - $k \in \{0, 1, U\}$ is the corresponding memory output, where the symbol U denotes the unknown value at the memory data output signals when a write operation is performed.

The proposed model is a modification of the mealy automata model proposed in [16]. The use of the ELDG allows modeling faulty situations where the result of a read operation is incorrect even if the content of the memory is correct. These situations were not representable using the model presented in [16].

As an example, Fig. 1 shows the model of a two-cell memory where the letters i and j are used to identify the first and the second cell, respectively. The ELDG has four states corresponding to the four possible combinations of the values stored into the cells. For the sake of readability, edges having the same initial and final state have been represented as a single edge with multiple labels.

2.2 Fault Model

This section introduces the formalism used to represent the target memory functional faults. Faults are modeled starting from *Faulty Behaviors* (FBs), i.e., deviations of the memory behavior from the expected one. An FB is expressed using the following notation:

- D^i represents a faulty value $D \in A$ (Definition 1) stored in the cell i.
- R_D^i represents an erroneous output $D \in A$ (Definition 1) obtained while reading the content of the cell i. This formalism is needed to represent classes of faults where a read operation returns an erroneous value, while the content of the memory cell is correct.

For example, 0^i means that the ith cell assumes an erroneous value 0, while R_1^i means that a read operation on cell i returns the value 1 even if the content of the cell is 0.

The set of faulty memory cells involved in the faulty behavior is called *f-cells*. Based on the f-cells cardinality ($|f\text{-cells}|$), faults can be clustered into the following classes:

- *Single-cell faults* ($|f\text{-cells}| = 1$).
- *Two-cell faults* ($|f\text{-cells}| = 2$). In this case, we can distinguish between an *aggressor cell* (a-cell) and a *victim cell* (v-cell). The former is the cell that sensitizes the FB, and the latter is the cell that shows the effect of the FB;
- *Multiple-cell faults* ($|f\text{-cells}| \geq 2$). Not all multiple-cell faults are detectable using march tests. Our model allows describing multiple-cell faults, but the proposed generation algorithm will be able to produce results only for those faults detectable using march tests, i.e., faults described by two-cell FBs sharing the same aggressor cell, for example, three-cell linked faults [5].

An FB is sensitized by the application of a sequence of *stimuli* on the f-cells. A *stimulus* S is composed of an *initial condition* C of the f-cells, followed by an optional sequence of *memory operations* $op_1, op_2, \ldots, op_m$:

$$S = (C, op_1, op_2, \ldots, op_m) | op_i \in X, m \geq 0, \tag{4}$$

where

- $C = (s^1, s^2, \ldots, s^k) | s^i \in A \cup \{-\},\ 1 \leq k \leq |f\text{-cells}|$, where s^i identifies one of the f-cells, and "$-$" denotes a don't care condition, i.e., the initial value of that cell does not influence the faulty behavior,
- $op_i \in X = X_0 \cup \{r_d^i | 0 \leq i \leq N-1, d \in A\}$, where X_0 is defined in Definition 1, and r_d^i is a *read-and-verify* operation performed on the ith cell. The value d means "*read the content of the cell i and verify that its value is equal to d.*" The sequence of memory operations can be omitted when an FB is sensitized just by the f-cells being in a certain condition (e.g., State Coupling fault [17]).

According to the number of operations in the sequence (m), the fault is classified as *Static* $(m \leq 1)$ or Dynamic $(m > 1)$.

Examples of possible stimuli are the following:

- $S = 0^i$ corresponds to an FB sensitized by the state of the faulty cell i equal to 0.
- $S = -, w_1^i$ corresponds to an FB sensitized by writing 1 into the faulty cell i, regardless of the current state of the cell.
- $S = 1^i,\ w_1^i r^i$ corresponds to an FB sensitized by a write operation of the value 1 on the faulty cell i, immediately followed by a read operation on the same cell. In this case, the FB is sensitized only if the two operations are applied starting with the cell i containing the value 1.

Definition 2. *A* Functional Fault Primitive *(FFP) represents the difference between an expected (fault-free) and the observed (faulty) memory behavior under a set of performed operations, denoted by*

$$FFP = \langle S/FB \rangle, \tag{5}$$

where S (4) and FB represent a stimulus and a faulty behavior, respectively.

A functional memory fault model is a nonempty set of FFPs. For example, the *Inversion Coupling Fault* (a transition performed on an aggressor cell a causes the inversion of the logic value stored in a victim cell v [2]) can be described by the following two FFPs:

$$\begin{aligned} C_{inv} = \{ & FFP_1 = \langle \underbrace{0^a 0^v, w_1^a}_{S} / \underbrace{1^v}_{FB} \rangle, \\ & FFP_2 = \langle \underbrace{0^a 1^v, w_1^a}_{S} / \underbrace{0^v}_{FB} \rangle \}. \end{aligned} \tag{6}$$

3 Automatic Test Generation Methodology

The proposed march test generation methodology is based on the functional memory model introduced in Section 2.1 and on the definition of functional faults in terms of FFPs (Section 2.2). The main steps of the generation process are summarized as follows:

1. *Fault list representation.* Translate each FFP in the fault list into an "*operational*" representation of the faulty behavior, referred to as *Addressed FFP*, or AFFP (Section 3.1).
2. *Test pattern (TP) generation.* Generate the set of TPs able to cover each AFFP. Each TP is represented by an additional edge on the ELDG modeling the memory (Section 3.2).
3. *March test generation.* Traverse the ELDG to generate the march test.

3.1 Fault List Representation

The FFP formalism describes the conditions to sensitize and detect FBs by considering the f-cells only. It does not consider the actual position of these cells in a generic N-cell memory. To map an FFP into a generic N-cell memory model, we therefore introduce the concept of AFFP. An *AFFP* is an instantiation of an FFP with an explicit indication of the addresses of the involved cells and both the faulty and the fault-free final memory state, after applying the stimulus S defined in the FFP (see Definition 2). The AFFP formalism strictly depends on both the number of memory cells involved in the fault ($|f\text{-cells}|$) and on the size N of the target memory. It can be formalized as

$$AFFP = \langle I, E_s, F_v, G_v \rangle, \tag{7}$$

where

- $I = (i_0, i_1, i_2, \ldots, i_{N-1}) | i_k \in A \cup \{-\}, 0 \leq k \leq N-1$ is the initial state of the memory, i.e., the values stored in the N cells of the target memory as defined by the initial conditions of the FFP. The first value corresponds to the less significant bit (i.e., the memory cell with the lowest address).

- $E_s = (op_1, op_2, \ldots, op_m) | op_i \in X, \ 1 \leq i \leq m$ is the sequence of m operations, performed on the aggressor cell, needed to sensitize the fault, according to the stimulus defined in the FFP. Each operation belongs to the alphabet X defined in (4).
- $F_v = (f_0, f_1, \ldots, f_{N-1}) | f_k \in A \cup \{U\}, \quad 0 \leq k \leq N-1$ are the logical values (faulty state) stored in the memory cells after applying E_s in case of a faulty memory.
- $G_v = (g_0, g_1, \ldots, g_{N-1}) | g_k \in A \cup \{U\}, \quad 0 \leq k \leq N-1$ are the logical values (expected state) stored in the memory cells after applying E_s on a fault-free memory.

Since N does not necessarily corresponds to the number of involved faulty memory cells ($|f$-cells$|$), each FFP can generate several AFFPs. For example, considering the Inversion Coupling fault FFPs defined in (6) applied to the two-cell memory represented in Fig. 1 ($N = 2$, $i =$ address of the first cell, $j =$ address of the second cell), we obtain the following AFFPs:

$$FFP_1 = \langle 0^a 0^v, w_1^a / 1^v \rangle \begin{cases} AFFP_1 = \langle 00, w_1^i, 11, 10 \rangle, \\ AFFP_2 = \left\langle 00, w_1^j, 11, 01 \right\rangle, \end{cases}$$
$$FFP_2 = \langle 0^a 1^v, w_1^a / 0^v \rangle \begin{cases} AFFP_3 = \langle 01, w_1^i, 10, 11 \rangle, \\ AFFP_4 = \left\langle 10, w_1^j, 01, 11 \right\rangle. \end{cases} \quad (8)$$

3.2 Test Pattern Generation

From an AFFP, it is easy to define the sequence of memory operations, TP, used to detect the corresponding faulty behavior as

$$TP = \langle AFFP, O_s \rangle, \quad (9)$$

where AFFP is the target AFFP, and $O_s = \{r_d^i\}$ is the read-and-verify operation (4) performed on the victim cell, needed to observe the fault effect.

For example, the four AFFPs defined in (8) are covered by the following TPs in a two-cell memory:

$$AFFP_1 = \langle 00, w_1^i, 11, 10 \rangle \rightarrow TP_1 = \left\langle \langle 00, w_1^i, 11, 10 \rangle, r_0^j \right\rangle,$$
$$AFFP_2 = \left\langle 00, w_1^j, 11, 01 \right\rangle \rightarrow TP_2 = \left\langle \left\langle 00, w_1^j, 11, 01 \right\rangle, r_0^i \right\rangle,$$
$$AFFP_3 = \langle 01, w_1^i, 10, 11 \rangle \rightarrow TP_3 = \left\langle \langle 01, w_1^i, 10, 11 \rangle, r_1^j \right\rangle,$$
$$AFFP_4 = \left\langle 10, w_1^j, 01, 11 \right\rangle \rightarrow TP_4 = \left\langle \left\langle 10, w_1^j, 01, 11 \right\rangle, r_1^i \right\rangle. \quad (10)$$

Each TP can be represented by an additional directed edge (*faulty edge*) from the state I to the state G_v defined in (7) on the memory model introduced in Section 2.1. The faulty edge label is defined as E_s, O_s, where E_s is the sequence of sensitizing operations, and O_s is the read-and-verify operation required to detect the fault, as defined in (7) and (9), respectively. The ELDG including the faulty edges is named *Pattern Graph* (PG) and is defined as

$$PG = (V, E \cup F, L_e \cup L_f), \quad (11)$$

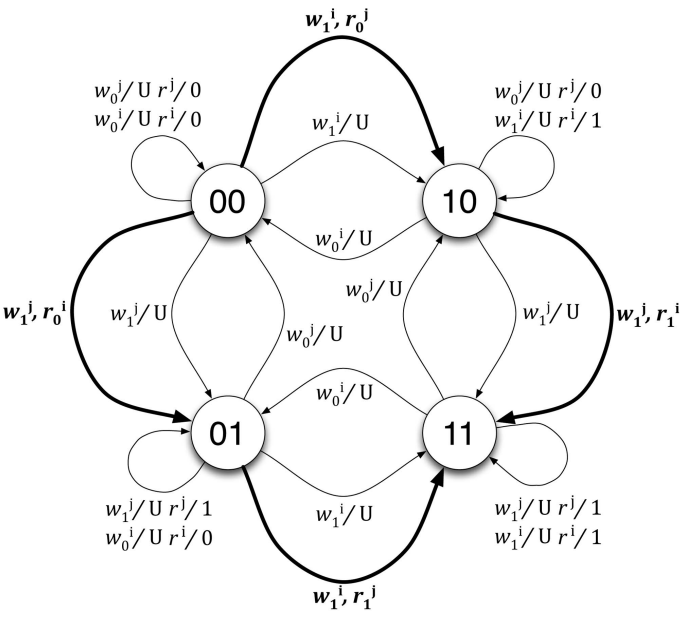


Fig. 2. CF_{inv} PG (PG_{CF}).

where each vertex $v \in V$ is associated to one of the 2^N memory states (1), E is the set of edges modeling the fault-free memory (1), F is the set of faulty edges, L_e is the label function for the fault-free edges (2), and L_f is the label function for the faulty edges.

It is clear that considering a real N-cell memory, the complexity of the model explodes due to the number of nodes of the PG. Anyway, a march test covering an n-cell fault can detect the same fault on any memory of size $N \geq n$ [2]. In general, the minimum number of required states of the PG can be defined as 2^{MaxF}, where $MaxF$ is the maximum number of faulty cells involved in the FFPs defined in the fault list. As stated in Section 2.2, the proposed algorithm is designed to work with faults described by two-cell FBs only, thus requiring a four-state PG.

Fig. 2 shows the PG (named PG_{CF} in the sequel) modeling the four TPs defined in (10). Bold edges represent the faulty edges.

3.3 March Test Generation Algorithm

In this section, the PG introduced in Section 3.2 is used to generate a march test detecting the set of faults described in the graph.

Definition 3. *A* march test *consists of a sequence of march elements (MEs). An* ME *consists of a sequence of operations applied to each cell in the memory based on a given* addressing order *(AO) (⇑ for the up AO, ⇓ for the down AO, and ⇕ for any AO) [2].*

The generation problem consists of finding a sequence of TPs (i.e., memory operations) able to detect the target set of memory faults while respecting the definition of march test (see Definition 3).

Definition 4. *Given an ELDG $G = (V, E, L_e)$, a* directed path *(v_0, v_i) with v_0, $v_i \in V$ is an ordered sequence of vertices and edges $(v_0, e_0, v_1, e_1, \ldots, v_{i-1}, e_{i-1}, v_i)$, where each edge $e_j \in E$*

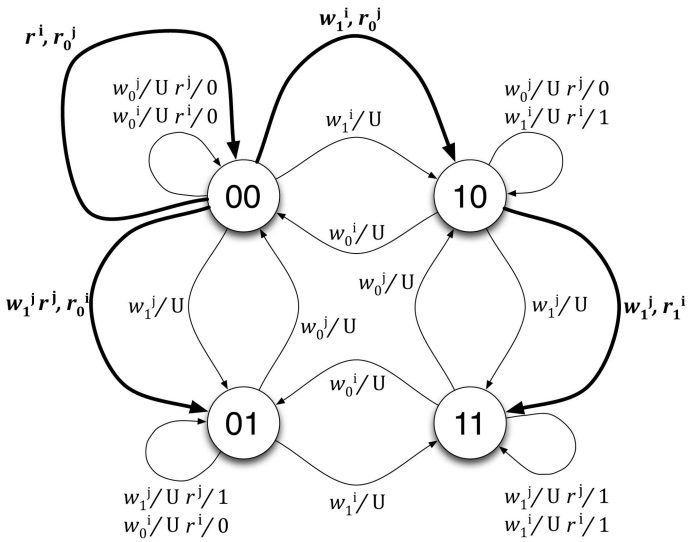


Fig. 3. PG example.

is incident from v_j and incident to v_{j+1} $(0 \leq j < i-1)$. We represent a directed path as

$$DP = (v_0, \text{“}e_0\text{”}, \text{“}e_1\text{”}, \ldots, \text{“}e_{i-1}\text{”}), \tag{12}$$

where v_0 is the starting vertex, and “e_0”, “e_1”, …, “e_{i-1}” is the ordered set of edges that have to be traversed in order to reach the final vertex of the path.

Considering a PG representing a faulty memory (11) and considering that each faulty edge in the PG represents a TP able to detect one of the FFPs in the target fault list, the problem of generating a march test is equivalent to the problem of finding a directed path in PG (i.e., a sequence of TPs) traversing all the faulty edges. The labels on the edges of the path represent the operations needed to sensitize and to observe the target faults. This sequence of operations has to respect the march test definition (see Definition 3).

For example, let us consider the PG in Fig. 3, where faulty edges are in bold.

A directed path starting from the state “00” and including all the faulty edges is

$$DP = (00, \text{“}w_1^j r^j, r_0^i\text{”}, \text{“}w_0^j\text{”}, \text{“}r^i, r_0^j\text{”}, \text{“}w_1^i, r_0^j\text{”}, \text{“}w_1^j, r_1^i\text{”}). \tag{13}$$

The sequence of operations identified by DP does not identify a march test since it is not composed of operations applied on every memory cell before considering the next sequence (MEs). From this consideration, it is clear that only a subset of the possible directed paths traversing the faulty edges actually identifies a march test. The goal of the proposed algorithm is to find a minimal directed path in the PG (i.e., a DP where faulty edges are traversed only once) that can be converted into a march test.

To solve this problem, we have to formalize the conditions that make possible the transformation of a directed path into a march test.

Let us consider the following ME: $\Uparrow (r_0 w_1)$. It identifies the following sequence of memory operations: $(r_0^0, w_1^0, r_0^1, w_1^1, \ldots, r_0^{N-1}, w_1^{N-1})$. The sequence includes the operations composing the ME repeated N times, one per memory cell in ascending order, i.e., starting from cell 0 to cell $N-1$.

The order in which cells are addressed corresponds to the AO of the ME. From this example, the following lemma is clear.

Lemma 1. *A* Sequence of Memory Test Operations *(SMTO) is* march-able *(i.e., it can be transformed into an ME) if and only if it can be described by the same set of operations repeated on every cell of the memory in a given order.*

In order to generate a march test, we try to build one ME at a time. This procedure consists of traversing the edges of the PG in such a way that the labels on the edges identify a march-able SMTO. When a march-able SMTO cannot be further expanded i.e., no other faulty edges can be traversed without violating the march-ability constraint, it is translated into an ME by specifying its AO (“$\Uparrow$” if the sequence of addresses in the SMTO starts from the address 0 and “$\Downarrow$” if the sequence of addresses in the SMTO starts from the address $N-1$). The algorithm exits when all faulty edges are traversed.

In this paper, we consider only classical up and down AOs. For this reason, we cannot generate tests detecting fault models requiring a specific AO to be sensitized (e.g., the fault model proposed in [18]). The extension to additional AOs can be considered by increasing the number of states of the PG in order to be able to model the required sequence of addresses and by modifying the way the AO is determined, thus increasing the complexity of the generation process.

To understand how the algorithm traverses the PG, some additional definitions are required.

Definition 5. *FE_v is the set of faulty edges incident from the state v. $FE_v = \{TP | I = v\}$. It represents the set of TPs (Section 3.2) having the initialization state I (7) equal to the state v.*

Referring to the example in Fig. 3, we have four sets of faulty edges: $FE_{00} = \{\text{“}w_1^j r^j,\ r_0^i\text{”}, \text{“}r^i, r_0^j\text{”}, \text{“}w_1^i, r_0^j\text{”}\}$, $FE_{01} = \{\oslash\}$, $FE_{10} = \{\text{“}w_1^j, r_1^i\text{”}\}$, and $FE_{11} = \{\oslash\}$.

Definition 6. *FE_v^i is the set of faulty edges incident from state v, with an aggressor cell equal to i, $0 \leq i \leq N-1$, $FE_v^i = \{TP | I = v, a\text{-}cell = i\}$ (Section 3.2). It represents the subset of TPs having the initialization state equal to state v and the aggressor cell address equal to i.*

According to Definition 6, we can split the set FE_v into several subsets, each one identified by an aggressor cell, and we can define FE_v in terms of FE_v^i as

$$FE_v = \bigcup_{i=0}^{N-1} FE_v^i. \tag{14}$$

Referring to the example in Fig. 3, we can build the following sets of faulty edges:

$$\begin{aligned} &FE_{00}^i = \left\{\text{“}r^i, r_0^j\text{”}, \text{“}w_1^i, r_0^j\text{”}\right\}, FE_{00}^j = \left\{\text{“}w_1^j r^j, r_0^i\text{”}\right\}, \\ &FE_{10}^i = \{\oslash\}, FE_{10}^j = \left\{\text{“}w_1^j, r_1^i\text{”}\right\}. \end{aligned} \tag{15}$$

Definition 7. *$FE_{v_1 \to v_2}^i$ is the set of faulty edges incident from the state v_1 and incident to the state v_2, with an aggressor cell equal to i, $0 \leq i \leq N-1$, $FE_{v_1 \to v_2}^i = \{TP | I = v_1, G = v_2, a\text{-}cell = i\}$ (Section 3.2).*

1: $V \leftarrow$ select v from possible initial states with $max(\#(FE_v))$.
2: $ME \leftarrow \emptyset, AO \leftarrow NULL$
3: **repeat**
4: $fe \leftarrow get\_fe(FE_V)$ it chooses one fe incident from the current state V
5: **if** $(fe = \emptyset)$ **then**
6: $close\_me(ME)$
7: $print(ME)$
8: $ME \leftarrow \emptyset, AO \leftarrow NULL$
9: $V \leftarrow get\_new\_state()$
10: **else**
11: $put\_fe\_in\_me(fe, ME)$
12: Update V with fe {it determines the new state by moving on the good machine}
13: delete fe
14: **end if**
15: **until** $(FE_v = \emptyset; \forall v)$

Fig. 4. March test generation algorithm.

It represents the set of TPs with the same aggressor cell and for which the application of the sensitizing operations changes the memory state from v_1 to v_2.

We call transition *a TP that belongs to $FE^i_{v_1 \to v2}$ with $v_1 \neq v_2$, while we call* loop *a TP that belongs to $FE^i_{v_1 \to v2}$ with $v_1 = v_2$.*

As an example, let us consider the two TPs, $TP_1 = \langle\langle 00, w^i_1, 11, 10\rangle, r^j_0\rangle$ and $TP_2 = \langle\langle 00, r^i, 01, 00\rangle, r^j_0\rangle$. Both TP_1 and TP_2 have the same aggressor cell (i). After applying TP_1, the memory state changes from 00 to 01. According to Definition 7, TP_1 is a transition, while TP_2 is a loop.

We can define FE^i_v in terms of $FE^i_{v_1 \to v2}$ as

$$FE^i_v = \overbrace{FE^i_{v \to v}}^{Loops} \bigcup \left(\overbrace{\bigcup_{\forall v_j \neq v} \left[FE^i_{v \to v_j} \right]}^{Transitions} \right). \tag{16}$$

We actually split the set of faulty edges into two sets: the former representing the set of faulty edges incident from v and incident to v (the loops set) and the latter being the union of the sets of faulty edges incident from v and incident to a node v_j different from v (the transition set).

Definition 8. *We define the* cardinality *of a set FE_v (written as $\#(FE_v)$) as the number of elements in the set and the* cost *($) of a set FE_v as*

$$\$(FE_v) = \sum_{i=0}^{N-1} \$\left(FE^i_v\right). \tag{17}$$

The cost of the ith component FE^i_v is defined as the cost of the loop set $(FE^i_{v \to v})$ plus the cost of the transition set:

$$\$\left(FE^i_v\right) = \$\left(FE^i_{v \to v}\right) + \sum_{\forall w \in H_1} \$\left(FE^i_{v \to w} | w \neq v\right), \tag{18}$$

where H_1 is the set of nodes reachable from v by traversing a single faulty edge.

The cost of the loop set corresponds to its cardinality: $\$(FE^i_{v \to v}) = \#(FE^i_{v \to v})$. The cost of the transition set is defined as 0 if the set is empty and as 1 plus the cost of the ith component of the destination state if the set is not empty. The value 1 represents the cost of the operation needed to move from state v to state w:

$$\$\left(FE^i_{v \to w}\right) = \begin{cases} 1 + \$\left(FE^i_w\right) \text{ if } \#\left(FE^i_{v \to w}\right) > 0, \\ 0 \text{ if } \#\left(FE^i_{v \to w}\right) = 0. \end{cases} \tag{19}$$

1: **if** (AO not defined) **then**
2: select i where $\$(FE^i_v) = max(\$(FE^i_v))$ with $i = 0 \vee i = N - 1$.
3: **if** (i=0) **then**
4: $AO \leftarrow$ '⇑'
5: **else**
6: $AO \leftarrow$ '⇓'
7: **end if**
8: **else**
9: **if** (AO=⇑) **then**
10: $i \leftarrow 0$
11: **else**
12: $i \leftarrow N - 1$
13: **end if**
14: **end if**
15: **if** $(\#(FE^i_{v \to v}) > 0)$ **then**
16: random select $fe \in FE^i_{v \to v}$
17: $return(fe)$
18: **else**
19: select $fe \in FE^i_{v \to w}$ where $O_s(fe)$ belongs to current ME, if no fe satisy the costraints random select fe.
20: $return(fe)$
21: **end if**

Fig. 5. Function *get_fe* (FE_v): returns the selected fe.

We now have all the elements and the definitions required to present the generation algorithm, whose main steps are summarized in Fig. 4 and described in the following.

The first operation required by the algorithm (step 1 in Fig. 4) is to select the *initial state*. Due to the march test characteristics (see Definition 3), the only valid initial states are the ones with all memory cells initialized with the same value (i.e., all "0" or all "1"). Among them, the algorithm chooses the state V with the maximum number of faulty edges incident from it, i.e., $max(\#(FE_v))\forall v$. An empty ME is initialized (step 2 in Fig. 4). At this point, no AO is specified for the current ME. The algorithm has to select one faulty edge (fe) incident from the current state V (step 4 in Fig. 4).

To select the new fe the *get_fe* function is invoked (Fig. 5). This function identifies the FE^i_V (see Definition 6) with the maximum cost (step 2 in Fig. 5). Since no AO is still selected and since we consider classic up and down AOs only, the only choice the algorithm has to generate a marchable sequence of operations (see Lemma 1) is to select an fe with an a-cell equal to the first or to the last cell of the given memory model.

First, the algorithm tries to select a loop that still needs to be traversed (if more than one loop exists, then the choice is random) (steps 16 and 17 in Fig. 5); otherwise, one of the

```
1: if (E_s(fe) ∉ ME) then
2:    add E_s(fe) to ME {add the sensitizing operation(s)}
3: end if
4: if (O_s(fe) ∉ ME) then
5:    if (O_s(fe) is on the same cell of E_s(fe)) then
6:       add O_s(fe) in ME or, when ME is closed place the read operation at the beginning of the next ME {single-cell}
7:    else if (O_s(fe) < E_s(fe) and AO is down) or (O_s(fe) > E_s(fe) and AO is up) then
8:       insert O_s(fe) at the beginning of the current ME (if not exists before) {two or more cells}
9:    else
10:      insert O_s(fe) at the beginning of the next ME and close the ME {two or more cells}
11:   end if
12: end if
```

Fig. 6. Function *put_fe_in_me (fe, ME)*.

transitions is selected (again, the choice among the transitions is random) (steps 19 and 20 in Fig. 5). The selected faulty edge automatically determines the AO (steps 3-7 in Fig. 5). From this point to the end of the ME generation, only operations performed on the first cell of the identified addressing sequence (the first cell in case of $\Uparrow$ and the last cell in case of $\Downarrow$) can be added (steps 9-13 in Fig. 5).

This constraint guarantees that the resulting ME is a march-able SMTO (see Lemma 1). Every time a faulty edge is traversed, the operations in its label are added to the ME (step 11 in Fig. 4). If not already present in the ME (due to previous operations), also the read-and-verify operation needed to observe the fault is added. At this point, the new memory state is calculated according to the fault-free memory model (step 12 in Fig. 4), and the faulty edge is deleted from the graph (the fault is detected) (step 13 in Fig. 4).

The ME generation ends if no faulty edges can be selected from the current state V (step 5 in Fig. 4). Before completing the generation of the ME (step 7 in Fig. 4), the *close_me* function (Fig. 7) is invoked (step 6 in Fig. 4). It simulates the application of the operations in the ME on each memory cell of the target memory model. Starting from V, the initial state of the current ME, each operation is applied, and the current state is updated according to the memory model of the fault-free memory (step 2 in Fig. 7). If during this operation additional faulty edges are traversed, they are marked as detected and removed from the graph (step 3 in Fig. 7).

At this point, we have reached the final state of the ME. The ME is completely defined, and if there are still faulty edges to traverse (step 15—Fig. 4), the next ME is initialized, and the process is repeated.

Each time a faulty edge is selected the *put_fe_in_me* (Fig. 6) function is invoked to append the faulty edge label to the ME. Before adding the operations needed to sensitize and observe the target fault (step 2 in Fig. 6), the function checks whether the ME already contains the required

```
1: for all memory cells do
2:    apply ME
3:    delete traversed faulty edges (fe)
4: end for
```

Fig. 7. Function *close_me (ME)*.

operations (step 1 in Fig. 6). If it does, then no operations are added.

After the sensitizing operations, the read and verify must be inserted. Two cases may occur:

- The a-cell is equal to the v-cell (i.e., a single-cell fault) (step 5 in Fig. 6). The read-and-verify operation is placed after the sensitizing operations (if it does not already exists) (step 6 in Fig. 6).
- The a-cell differs from the v-cells (i.e., an n-cell fault with a single aggressor cell and a set of victim cells):
  - If a-cell $>$ v-cells and the AO is $\Uparrow$ (or a-cell $<$ v-cells and AO is $\Downarrow$) (step 7 in Fig. 6), the read operation is inserted at the beginning of the ME (step 8 in Fig. 6).
  - Otherwise, the ME is closed, and the read operation is added as the first operation of the next ME according to [6] (step 10 in Fig. 6).

The behavior of the algorithm is greedy since it tries to insert in each ME the highest possible number of faulty edges. A key point is the faulty edge selection (step 4 in Fig. 4). The basic idea is to choose the faulty edges from the FE_v^i having the highest cost (step 2 in Fig. 5), where, in fact, the cost corresponds to the number of loops and transitions. This actually means selecting the FE_v^i allowing the highest number of movements (transitions) on the PG.

When the current memory state has an empty FE_V, the algorithm needs to move to a new memory state v_1 having the highest $\$(FEv_1)$ (step 9 in Fig. 4). The function *get_new_state* (Fig. 8) performs this operation by traversing the fault-free edges of the graph and by appending the corresponding operations (labels) to the ME. In other words, the algorithm builds an initialization ME to reach the target state v_1.

To better understand the generation algorithm, we will show its application on the PG in Fig. 3. The initial selected state is $V = 00$, since $\#(FE_{00}) = 3$ (00 is the state with the highest number of faulty edges incident from it). We have to choose one faulty edge, so we calculate the cost of the different sets of faulty edges: $FE_{00}^i = \{\text{“}r^i, r_0^j\text{”}, \text{“}w_1^i, r_0^j\text{”}\}$, $\$(FE_{00}^i) = 2$, $FE_{00}^j = \{\text{“}w_1^j r^j, r_0^i\text{”}\}$, and $\$(FE_{00}^j) = 1$. The algorithm chooses FE_{00}^i (up AO as defined in steps 3-7 in Fig. 5), and “r^i, r_0^j” (loop set) is selected and added to the ME, which becomes (r_0^i) with the current memory state still equal to “00.” At this point, the algorithm chooses the only

```
1: select v such as FE_v = max(#(FE_v))
2: copy the operation required to reach v into ME
3: return(v)
```

Fig. 8. Function *get_new_state()*: returns the new memory state.

TABLE 1
Generated March Tests for Unlinked Static Faults

Name	Algorithm	O(n)	FaultList
MATS [2]	$\{\Uparrow (w_1) \Downarrow (r_1w_0) \Downarrow (r_0)\ \}$	4n	SAF
MATS+ [2]	$\{\Uparrow (w_1) \Uparrow (r_1w_0) \Downarrow (r_0w_1)\}$	5n	SAF,ADF
X [2]	$\{\Uparrow (w_0) \Uparrow (r_0w_1) \Downarrow (r_1w_0) \Uparrow (r_0)\}$	6n	SAF,TF ADF
C- [2]	$\{\Uparrow (w_1) \Uparrow (r_1w_0) \Uparrow (r_0w_1) \Downarrow (r_1w_0) \Downarrow (r_0w_1) \Downarrow (r_1)\}$	10n	SAF, CFid ADF, TF, CFinv
CLI [13]	$\{\Uparrow (w_1) \Uparrow (r_1w_0w_1) \Downarrow (r_1)\ \}$	5n	CFinv
SS [19]	$\{\Updownarrow (w_0) \Uparrow (r_0r_0w_0r_0w_1) \Uparrow (r_1r_1w_1r_1w_0) \Downarrow (r_0r_0w_0r_0w_1) \Downarrow (r_1r_1w_1r_1w_0) \Updownarrow (r_0)\}$	22n	All static

remaining choice in FE^i_{00} represented by "w^i_1, r^j_0". w^i_1 (sensitizing sequence) is added to the ME, which becomes $(r^i_0w^i_1)$, while the observation is not required since it is already present.

The new state is now $v = 10$. $FE^i_{10} = \{\oslash\}$, so the algorithm closes the ME and generates the corresponding ME: $\Uparrow (r_0w_1)$. It then simulates the operations of the ME on cell j. The simulation shows that also "w^j_1, r^i_1" (from state "10") is traversed. The read-and-verify operation performed on cell i is added to the next ME, which becomes (r^i_1) again with up AO (step 10 in Fig. 6).

The current state is now $v = 11$. Since FE_{11} is empty, the ME is closed, and the corresponding ME is generated: $\Uparrow (r_1)$. We now must change state to "00," which is the only state still having faulty edges. The operation to move to "00" is w^i_0, it is added to the new ME, which becomes (w^i_0) with up AO. The ME is closed, and the corresponding generated ME is $\Uparrow (w_0)$. Now, the algorithm traverses the last faulty edge, and it adds the operations on its label into a new ME, obtaining an ME equal to $(w^j_1r^j_1)$ with down AO. It also inserts the read-and-verify operation, obtaining an ME equal to $(r^j_0w^j_1r^j_1)$. At this point, all the faulty edges are traversed, the ME is closed, and the generated ME is $\Downarrow (r_0w_1r_1)$. The algorithm ends, and the final generated march test is $\Uparrow (w_0) \Uparrow (r_0w_1) \Uparrow (r_1) \Uparrow (w_0) \Downarrow (r_0w_1r_1)$, where the first ME allows initializing the memory in the selected initial state.

4 Experimental Results

This section reports some experimental results obtained by applying the proposed generation algorithm to different fault lists. Tables 1 and 2 report the generated march tests for different sets of target unlinked static and dynamic faults. For each march test, we report the name (column 1), the algorithm (column 2), the complexity in terms of the number of operations (column 3), and the target fault list (column 4).

Table 1 reports march tests targeting different sets of static unlinked faults. Static faults have been deeply studied in the last years, and all the algorithms generated by the tool were already published in previous works.

When we move to the more complex dynamic unlinked faults, the proposed algorithm demonstrates its real value. In this case, we have been able to generate tests shorter than the ones already published. We started from the set of dynamic faults described in [20]. These faults are considered to be some of the most realistic ones for current memory technologies.

TABLE 2
Generated March Tests for Unlinked Dynamic Faults

Name	Algorithm	O(n)	FaultList
ABdrf	$\{\Updownarrow (w_1) \Downarrow (w_0r_0r_0w_1r_1r_1)\}$	7n	Dynamic read faults
AB1 [21]	$\{\Updownarrow (w_0) \Updownarrow (w_1r_1w_1r_1r_1) \Updownarrow (w_0r_0w_0r_0r_0)\}$	11n	Single-Cell dynamic Faults
AB [21]	$\{\Updownarrow (w_1) \Downarrow (r_1w_0r_0w_0r_0) \Downarrow (r_0w_1r_1w_1r_1) \Uparrow (r_1w_0r_0w_0r_0) \Uparrow (r_0w_1r_1w_1r_1) \Updownarrow (r_1)\}$	22n	Two-Cells Dynamic faults
AB2	$\{\Updownarrow (w_1) \Downarrow (w_1r_1w_0r_0w_0 r_0w_1r_1)\}$	9n	Dynamic Read Destructive Fault (dRDF)
AB3	$\{\Updownarrow (w_0) \Downarrow (w_0r_0r_0w_1r_1r_1 w_1r_1r_1w_0r_0r_0)\}$	13n	Dynamic Deceptive Read Destructive Fault (dDRDF)
AB4	$\{\Updownarrow (w_0) \Uparrow (r_0w_0r_0w_1r_1) \Uparrow (r_1w_1r_1w_0r_0) \Downarrow (r_0w_0r_0w_1r_1) \Downarrow (r_1w_1r_1w_0r_0) \Updownarrow (r_0)\}$	22n	Dynamic Read Destructive Coupling Fault (dCFrd)

March AB1 (Table 2), with a complexity of 11n, is able to detect the entire set of single-cell two-operation dynamic faults [20]; compared with March RAW1 (13n), which was manually designed, it guarantees the same fault coverage but reduces the test length by two operations or 18.18 percent. March AB (Table 2), with a complexity of 22n, is able to detect the entire set of realistic two-cell two-operation dynamic faults [20]; compared with March RAW (26n), again manually designed, it provides the same fault coverage but reduces the test length by four operations or 15.38 percent. Finally, we propose three new march tests, March AB2, March AB3, and March AB4, generated for particular subsets of dynamic faults in order to demonstrate the freedom in choosing the target fault list.

We also applied the proposed algorithm, after applying the modification to the PG proposed in [22] to model linked faults, to the set of realistic linked faults presented in [5]. In Table 3, Fault List #1 includes single-, two- and three-cell linked faults proposed in [5], whereas Fault List #2 includes all single-cell linked faults proposed in [5].

TABLE 3
Generated March Tests for Linked Faults

Name	Algorithm	O(n)	FaultList
AB	$\{\Updownarrow (w_1) \Downarrow (r_1w_0r_0w_0r_0) \Downarrow (r_0w_1r_1w_1r_1) \Uparrow (r_1w_0r_0w_0r_0) \Uparrow (r_0w_1r_1w_1r_1) \Updownarrow (r_1)\}$	22n	#1
ABL1	$\{\Updownarrow (w_0) \Updownarrow (w_0r_0r_0w_1) \Updownarrow (w_1r_1r_1w_0)\}$	9n	#2

TABLE 4
BIST Optimized March Test

Algorithm	O(n)	FaultList
$\{\Uparrow (w_0) \Uparrow (r_0w_1w_0) \Uparrow (r_0w_1) \Uparrow (r_1w_0w_1) \Uparrow (r_1w_0) \Uparrow (r_0)\}$	12n	SAF, TF RDF

We compared our new generated march tests with already published algorithms targeting the same fault list. In particular, we considered the following:

- *43n March test* [9]. It is automatically generated and deals only with a subset of faults defined in Fault List #1.
- *41n March SL* [5]. It is the state of the art in the class of hand-generated march tests. It covers Fault List #1.
- *23n March MSL* [23]. It is automatically generated, and it reduces the complexity of March SL. It covers Fault List #1.
- *11n March LF1* [5]. It is one of the most used march tests, and it is able to cover Fault List #2.

March AB of complexity 22n, targeting Fault List #1, reduces the test time by 48.8 percent with respect to the 43n march test, 46.3 percent with respect to March SL, and 4.35 percent with respect to March MSL. Similarly, March ABL1, which targets Fault List #2, reduces the test length by 18.1 percent with respect to March LF1 (the state of the art for the same list of faults).

It is relevant to note that using our algorithm, we have been able to generate a new test, March AB, able to detect both dynamic and linked faults. Moreover, the same algorithm is also able to detect the full list of static unlinked faults detected by the 22n March SS. In all the experiments, the generation process was shorter than 1 second of CPU time. All generated march tests have been verified by fault simulation using the memory fault simulator published in [24] and [25] to validate the correctness of the test with respect to the target fault list.

5 Optimizations

March tests are critical components in any ATE-based or BIST Memory test architecture. In the latter case, it has been shown that the BIST hardware overhead can be reduced if the march test shows some particular characteristics such as uniformity [8] (a constant number of operations in each ME), symmetry [2] (particularly important on transparent march tests), or single AO. The proposed march test generation algorithm already produces, if possible, symmetric march tests (see Table 3). Additional constraints can be very easily added in the generation phase performed by traversing the ELDG. We successfully implemented the possibility of generating single-AO march tests, by adding this additional constraint in the *get_fe* function (see Fig. 5), which is the function defining the AO. Using this optimization, we have been able to generate the single-AO march test proposed in Table 4. Other constraints can be easily implemented; the only drawback is that they can lead to situations where no solutions can be generated.

6 Conclusions

This paper addresses two very important issues usually faced by researchers and test engineers in the field of memory testing. It provides a clear and flexible formalism to model memories and faulty behaviors, and it proposes an efficient algorithm to automatically generate march tests. The flexibility of the fault model formalism allows describing not only traditional static and dynamic faults but also linked and user-defined faults. This feature makes the proposed research very appealing for both memory manufacturers and users. With respect to previously presented approaches, our methodology allows generating shorter march tests in a very low computation time, without exhaustive searches. The paper presents march tests for the complete set of static faults and for most of the known dynamic faults, obtaining both already published and new test algorithms. What emerges from the experimental results is the efficiency of the algorithm, which is able to significantly reduce the march test length and, therefore, the test time for many significant fault lists. Ongoing research activities are focusing on the extension of the model to multiport memory faults and on the possibility of introducing additional constraints on the generated march tests.

References

[1] International Technology Roadmap for Semiconductors, http://www.itrs.net/, 2008.

[2] A.J. van de Goor, "Using March Tests to Test SRAMs," *IEEE Design and Test of Computers,* vol. 10, no. 1, pp. 8-14, Jan.-Mar. 2004.

[3] A.J. van de Goor and Z. Al-Ars, "Functional Memory Faults: A Formal Notation and a Taxonomy," *Proc. 18th IEEE VLSI Test Symp. (VTS '00),* pp. 281-289, Apr./May 2000.

[4] S. Hamdioui, R. Wadsworth, J. Reyes, and A. van de Goor, "Importance of Dynamic Faults for New SRAM Technologies," *Proc. Eighth IEEE European Test Workshop (ETW '03),* pp. 29-34, May 2003.

[5] S. Hamdioui, Z. Al-Ars, A.J. van de Goor, and M. Rodgers, "Linked Faults in Random Access Memories Concept Fault Models Test Algorithms and Industrial Results," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 23, no. 5, pp. 737-757, May 2004.

[6] B. Smit and A.J. van de Goor, "The Automatic Generation of March Tests," *Proc. IEEE Int'l Workshop Memory Technology, Design and Testing (MTDT '94),* pp. 86-91, Aug. 1994.

[7] K. Zarrineh, S. Upadhyaya, and S. Chakravarty, "Automatic Generation and Compaction of March Tests for Memory Arrays," *IEEE Trans. Very Large Scale Integration (VLSI) Systems,* vol. 9, no. 6, pp. 845-857, Dec. 2001.

[8] S. Al-Harbi and S. Gupta, "An Efficient Methodology for Generating Optimal and Uniform March Tests," *Proc. 19th IEEE VLSI Test Symp. (VTS '01),* pp. 231-237, Apr.-May 2001.

[9] S.M. Al-Harbi and S. Gupta, "Generating Complete and Optimal March Tests for Linked Faults in Memories," *Proc. 21st IEEE VLSI Test Symp. (VTS '03),* pp. 254-261, Apr./May 2003.

[10] D. Niggemeyer and E.M. Rudnick, "Automatic Generation of Diagnostic Memory Tests Based on Fault Decomposition and Output Tracing," *IEEE Trans. Computers,* vol. 53, no. 9, pp. 1134-1146, Sept. 2004.

[11] C.-F. Wu, C.-T. Huang, K.-L. Cheng, and C.-W. Wu, "Simulation-Based Test Algorithm Generation for Random Access Memories," *Proc. 18th IEEE VLSI Test Symp. (VTS '00),* pp. 291-296, Apr./May 2000.

[12] C.-F. Wu, C.-T. Huang, and C.-W. Wu, "Ramses: A Fast Memory Fault Simulator," *Proc. IEEE Int'l Symp. Defect and Fault Tolerance in VLSI Systems (DFT '99),* pp. 165-296, Nov. 1999.

[13] S. Di Carlo, G. Di Natale, A. Benso, and P. Prinetto, "An Optimal Algorithm for the Automatic Generation of March Tests," *Proc. Design, Automation and Test in Europe Conf. and Exhibition (DATE '02),* pp. 938-939, Mar. 2002.

[14] A. Benso, A. Bosio, S. Di Carlo, G. Di Natale, and P. Prinetto, "Automatic March Tests Generation for Static and Dynamic Faults in SRAMs," *Proc. 10th IEEE European Test Symp. (ETS '05)*, pp. 122-127, May 2005.
[15] A. van de Goor and I.B.S. Tlili, "A Systematic Method for Modifying March Tests for Bit-Oriented Memories into Tests for Word-Oriented Memories," *IEEE Trans. Computers*, vol. 52, no. 10, pp. 1320-1331, Oct. 2003.
[16] J. Brzozowski and H. Jurgensen, "A Model for Sequential Machine Testing and Diagnosis," *J. Electronic Testing Theory and Application*, vol. 3, no. 3, pp. 219-234, Aug. 1992.
[17] R. Dekker, F. Beenker, and L. Thijssen, "A Realistic Fault Model and Test Algorithms for Static Random Access Memory," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, no. 6, pp. 567-572, June 1990.
[18] A. Ney, P. Girard, C. Landrault, S. Pravossoudovitch, A. Virazel, and M. Bastian, "Slow Write Driver Faults in 65 nm SRAM Technology: Analysis and March Test Solution," *Proc. Design, Automation and Test in Europe Conf. and Exhibition (DATE '07)*, pp. 1-6, Apr. 2007.
[19] S. Hamdioui, A.J. van de Goor, and M. Rodgers, "March SS: A Test for All Static Simple RAM Faults," *Proc. IEEE Int'l Workshop Memory Technology, Design and Testing (MTDT '02)*, pp. 95-100, July 2002.
[20] S. Hamdioui, Z. Al-Ars, and A.J. van de Goor, "Testing Static and Dynamic Faults in Random Access Memories," *Proc. 20th IEEE VLSI Test Symp. (VTS '02)*, pp. 395-400, Apr./May 2002.
[21] A. Benso, A. Bosio, S.D. Carlo, G.D. Natale, and P. Prinetto, "March AB, March AB1: New March Tests for Unlinked Dynamic Memory Faults," *Proc. IEEE Int'l Test Conf. (ITC '05)*, pp. 834-841, Nov. 2005.
[22] A. Benso, A. Bosio, S. Di Carlo, G. Di Natale, and P. Prinetto, "Automatic March Tests Generations for Static Linked Faults in SRAMs," *Proc. Design, Automation and Test in Europe Conf. and Exhibition (DATE '06)*, pp. 1-6, Mar. 2006.
[23] G. Harutanyan, V. Vardanian, and Y. Zorian, "Minimal March Test Algorithm for Detection of Linked Static Faults in Random Access Memories," *Proc. 24th IEEE VLSI Test Symp. (VTS '06)*, pp. 120-125, Apr./May 2006.
[24] A. Benso, S.D. Carlo, G.D. Natale, and P. Prinetto, "Specification and Design of a New Memory Fault Simulator," *Proc. 11th IEEE Asian Test Symp. (ATS '02)*, pp. 92-97, Nov. 2002.
[25] A. Benso, A. Bosio, S.D. Carlo, G.D. Natale, and P. Prinetto, "Memory Fault Simulator for Static-Linked Faults," *Proc. 15th IEEE Asian Test Symp. (ATS '06)*, pp. 31-36, Nov. 2006.

Alfredo Benso currently holds a tenured associate professor position in computer engineering in the Department of Control and Computer Engineering, Politecnico di Torino, Torino, Italy, where he teaches microprocessor systems and advanced programming techniques. In his scientific career, mainly focused on hardware testing and dependability, he coauthored more than 60 publications in books, journals, and conference proceedings. He is also actively involved in the IEEE Computer Society, where he has been the leading volunteer for several projects such as the TEchnical Committees Archives (TECA) database and the Conference Information Management Application (CIMA). He is an IEEE Computer Society Golden Core member and a senior member of the IEEE.

Alberto Bosio received the PhD degree in computer engineering from the Politecnico di Torino, Italy, in 2006. Since 2002, he worked in the area of digital systems dependability for safety-critical applications at the Politecnico di Torino. He is currently an associate professor in the Laboratoire d'Informatique, de Robotique et de Microelectronique de Montpellier, University of Montpellier II/CNRS, Montpellier, France. His research activity mainly focused on the definition of new methodologies and the implementation of tools able to improve the development of highly dependable systems, at different levels: for basic digital components, for systems on chip, up to microprocessor-based systems. He is a member of the IEEE.

Stefano Di Carlo received the MS degree in computer engineering and the PhD degree in information technologies from the Politecnico di Torino, Torino, Italy, in 1999 and 2003, respectively. Since 2008, he has been an assistant professor in the Department of Control and Computer Engineering, Politecnico di Torino. In his scientific career, mainly focused on DFT techniques, SoC testing, BIST, and memory testing, he coauthored more than 40 publications in journals and conference proceedings. He is a Golden Core member of the IEEE Computer Society and a member of the IEEE.

Giorgio Di Natale received the PhD degree in computer engineering from the Politecnico di Torino, Torino, Italy, in 2003. Currently, he is a senior researcher in the Laboratoire d'Informatique, de Robotique et de Microelectronique de Montpellier, University of Montpellier II/CNRS, Montpellier, France. He has published articles in publications spanning diverse disciplines, including memory testing, fault tolerance, and secure chip design and test. He also serves the Test Technology Technical Council (TTTC) of the IEEE Computer Society as the webmaster. He is a member of the IEEE.

Paolo Prinetto received the MS degree in electronic engineering from the Politecnico di Torino, Torino, Italy. He is a full professor of computer engineering in the Department of Control and Computer Engineering, Politecnico di Torino, and a joint professor at the University of Illinois, Chicago. His research interests include testing, test generation, BIST, and dependability. He is a Golden Core member of the IEEE Computer Society, and he served the IEEE Computer Society Test Technology Technical Council (TTTC) as the elected chair. He is a member of the IEEE and a member of the IEEE Computer Society.

Bibliography

[1] B. Kruseman, A. Majhi, C. Hora, S. Eichenberger, and J. Meirlevede. Systematic defects in deep sub-micron technologies. *IEEE International Test Conference*, pages 290–299, 2005.

[2] K.M. Thompson. Intel and the myths of test. *IEEE Design Test of Computers*, 13 (1):79–81, Spring 1996.

[3] S. Eichenberger, J. Geuzebroek, C. Hora, B. Kruseman, and A. Majhi. Towards a world without test escapes: The use of volume diagnosis to improve test quality. *IEEE International Test Conference*, pages 1–10, 2008. doi: 10.1109/TEST.2008.4700604.

[4] F. Hapke, R. Krenz-Baath, A. Glowatz, J. Schloeffel, H. Hashempour, S. Eichenberger, C. Hora, and D. Adolfsson. Defect-oriented cell-aware atpg and fault simulation for industrial cell libraries and designs. *IEEE International Test Conference*, pages 1–10, 2009. doi: 10.1109/TEST.2009.5355741.

[5] F. Hapke, W. Redemund, J. Schloeffel, R. Krenz-Baath, A. Glowatz, M. Wittke, H. Hashempour, and S. Eichenberger. Defect-oriented cell-internal testing. *IEEE International Test Conference*, pages 1–10, 2010. doi: 10.1109/TEST.2010.5699229.

[6] S. Venkataraman, S. Sivaraj, E. Amyeen, Lee Sangbong, A. Ojha, and Guo Ruifeng. An experimental study of n-detect scan atpg patterns on a processor. *IEEE VLSI Test Symposium*, pages 23–28, April 2004. doi: 10.1109/VTEST.2004.1299221.

[7] Youn Cho Kyoung, S. Mitra, and E. J. McCluskey. Gate exhaustive testing. *IEEE International Test Conference*, pages 7 pp.–777, Nov 2005. doi: 10.1109/TEST.2005.1584040.

[8] M. Bruce and V. Bruce. Abcs of emission microscopy. *Electronic Device Failure Analysis*, pages 13–20, 2003.

[9] R. Desplats, A. Eral, F. Beaudoin, P. Perdu, A. Chion, K. Shah, and T. Lundquist. Ic diagnostic with time resolved photon emmison and cad auto-channeling. *International Symposium for Testing and Failure Analysis*, pages 45–54, 2003.

[10] X. Fan, W. R. Moore, C. Hora, and G. Gronthoud. Extending gate-level diagnosis tools to cmos intra-gate faults. *IET Computers Digital Techniques*, 1(6):685–693, Nov 2007. doi: 10.1049/iet-cdt:20060206.

[11] A. Ladhar, M. Masmoudi, and L. Bouzaida. Efficient and accurate method for intra-gate defect diagnoses in nanometer technology and volume data. *Design, Automation Test in Europe Conference Exhibition*, pages 988–993, April 2009. doi: 10.1109/DATE.2009.5090808.

[12] J. Savir. Skewed-load transition test: Part i, calculus. *IEEE International Test Conference*, pages 705–715, Sep 1992. doi: 10.1109/TEST.1992.527892.

[13] J. Savir and S. Patil. Broad-side delay test. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(8):1057–1064, Aug 1994. doi: 10.1109/43.298042.

[14] Xu Gefu and A. D. Singh. Low cost launch-on-shift delay test with slow scan enable. *IEEE European Test Symposium*, pages 9–14, May 2006. doi: 10.1109/ETS.2006.29.

[15] J. Saxena, K. M. Butler, V. B. Jayaram, S. Kundu, N. V. Arvind, P. Sreeprakash, and M. Hachinger. A case study of ir-drop in structured at-speed testing. *IEEE International Test Conference*, 1:1098–1104, Sept 2003. doi: 10.1109/TEST.2003.1271098.

[16] K. Arabi, R. Saleh, and Xiongfei Meng. Power supply noise in socs: Metrics, management, and measurement. *IEEE Design Test of Computers*, 24(3):236–244, May 2007. doi: 10.1109/MDT.2007.79.

[17] S. T. Arasu, C. P. Ravikumar, and S. K. Nandy. A low power and low cost scan test architecture for multi-clock domain socs using virtual divide and conquer. *IEEE International Test Conference*, pages 9 pp.–377, Nov 2005. doi: 10.1109/TEST.2005.1583995.

[18] Patrick Girard, Nicola Nicolici, and Xiaoqing Wen. *Power-Aware Testing and Test Strategies for Low Power Devices*. Springer, 2010. ISBN 978-1-4419-0928-2.

[19] K. M. Butler, J. Saxena, A. Jain, T. Fryars, J. Lewis, and G. Hetherington. Minimizing power consumption in scan testing: pattern generation and dft techniques. *IEEE International Test Conference*, pages 355–364, Oct 2004. doi: 10.1109/TEST.2004.1386971.

[20] K. Agarwal, S. Vooka, S. Ravi, R. Parekhji, and A. S. Gill. Power analysis and reduction techniques for transition fault testing. In *IEEE Asian Test Symposium*, pages 403–408, Nov 2008. doi: 10.1109/ATS.2008.86.

[21] E. K. Moghaddam, J. Rajski, S. M. Reddy, Lin Xijiang, N. Mukherjee, and M. Kassab. Low capture power at-speed test in edt environment. In *IEEE International Test Conference*, pages 1–10, Nov 2010. doi: 10.1109/TEST.2010.5699275.

[22] E. K. Moghaddam, J. Rajski, M. Kassab, and S. M. Reddy. At-speed scan test with low switching activity. In *IEEE VLSI Test Symposium*, pages 177–182, April 2010. doi: 10.1109/VTS.2010.5469580.

[23] Andrea Calimera, Enrico Macii, Danilo Ravotto, Ernesto Sanchez, and Matteo Sonza Reorda. Generating power-hungry test programs for power-aware validation of pipelined processors. In *Proceedings of the 23rd Symposium on Integrated Circuits and System Design*, pages 61–66, 2010. doi: 10.1145/1854153.1854171.

[24] Itrs 2013. URL http://www.itrs.net/home.html.

[25] A. Bosio, L. Dilillo, P. Girard, A. Virazel, and S. Pravossoudovitch. *Advanced Test Methods for SRAMs Effective Solutions for Dynamic Fault Detection in Nanoscale Technologies*. Springer, 2009. ISBN 978-1-4419-0937-4.

[26] S. K. Goel, M. Meijer, and J. P. de Gyvez. Testing and diagnosis of power switches in socs. In *IEEE European Test Symposium*, pages 6 pp.–150, May 2006. doi: 10.1109/ETS.2006.47.

[27] S. Khursheed, Yang Sheng, B. M. Al-Hashimi, Huang Xiaoyu, and D. Flynn. Improved dft for testing power switches. In *IEEE European Test Symposium*, pages 7–12, May 2011. doi: 10.1109/ETS.2011.63.

[28] Viswani D. Agarval and Michael L. Bushnell. *Essentials of Electronic Testing for Digital Memory and Mixed Signal VLSI Circuit*. Kluwer Academic Publications, 1999.

[29] Xiaoyu Huang, Jimson Mathew, Rishad A. Shafik, Subhasis Bhattacharjee, and Dhiraj K. Pradhan. A fast and effective dft for test and diagnosis of power switches in socs. In *Design Automation Test in Europe Conference Exhibition*, pages 1089–1092, March 2013. doi: 10.7873/DATE.2013.229.

[30] R. Desineni, O. Poku, and R.D. Blanton. A logic diagnosis methodology for improved localization and extraction of accurate defect behavior. In *IEEE International Test Conference*, pages 1–10, 2006.

[31] M.E. Amyeen, D. Nayak, and S. Venkataraman. Improving precision using mixed-level fault diagnosis. In *IEEE International Test Conference*, pages 1–10, 2006.

[32] M. De Carvalho, P. Bernardi, E. Sanchez, and M. S. Reorda. An enhanced strategy for functional stress pattern generation for system-on-chip reliability characterization. In *1th International Workshop on Microprocessor Test and Verification*, pages 29–34, Dec 2010. doi: 10.1109/MTV.2010.14.

[33] F. Corno, E. Sanchez, M. S. Reorda, and G. Squillero. Automatic test program generation: a case study. *IEEE Design Test of Computers*, 21(2):102–109, Mar 2004. doi: 10.1109/MDT.2004.1277902.

[34] D. Appello, V. Tancorre, P. Bernardi, M. Grosso, M. Rebaudengo, and M. S. Reorda. On the automation of the test flow of complex socs. In *IEEE VLSI Test Symposium*, pages 6 pp.–171, April 2006. doi: 10.1109/VTS.2006.51.

[35] A. Agarwal, S. Mukhopadhyay, C. H. Kim, A. Raychowdhury, and K. Roy. Leakage power analysis and reduction: models, estimation and tools. *IEE Proceedings Computers and Digital Techniques*, 152(3):353–368, May 2005. doi: 10.1049/ip-cdt:20045084.

[36] K. Itoh. Low-voltage memories for power-aware systems. In *Proceedings of the 2002 International Symposium on Low Power Electronics and Design*, pages 1–6, 2002. doi: 10.1109/LPE.2002.146699.

[37] Wang Jiajing and B.H. Calhoun. Techniques to extend canary-based standby vdd scaling for srams to 45 nm and beyond. *IEEE Journal of Solid-State Circuits*, 43 (11):2514–2523, Nov 2008. doi: 10.1109/JSSC.2008.2005814.

[38] Wang Yih, U. Bhattacharya, F. Hamzaoglu, P. Kolar, Ng Yong-Gee, Wei Liqiong, Zhang Ying, K. Zhang, and M. Bohr. A 4.0 ghz 291 mb voltage-scalable sram design in a 32 nm high-k + metal-gate cmos technology with integrated power management. *IEEE Journal of Solid-State Circuits*, 45(1):103–110, Jan 2010. doi: 10.1109/JSSC.2009.2034082.

[39] Said Hamdioui, Rob Wadsworth, JohnDelos Reyes, and AdJ. van de Goor. Memory fault modeling trends: A case study. *Journal of Electronic Testing*, 20(3):245–255, 2004. ISSN 0923-8174. doi: 10.1023/B:JETT.0000029458.57095.bb.