YEREVAN STATE UNIVERSITY

Gurgen Harutyunyan

BUILT-IN TEST SOLUTIONS FOR NANOSCALE MEMORY DEVICES AND SYSTEMS

THESIS

Doctor of Technical Sciences

Specialty E.13.04 «Mathematics and software for computing machines, complexes, systems and networks»

Scientific adviser:
Member of NAS RA, D.Sc., Professor Samvel K. Shoukourian

Yerevan – 2018

Table of Contents

INTRODUCTION

Relevance of the topic

The reduced size of nanoscale memory devices brings to essential design problems, and in parallel the test and repair requirements are becoming more stringent to achieve high quality and yield for nanoscale Systems-on-Chip (SoCs). Built-in test approaches proposed for past designs do not meet the requirements of current designs which are bigger, faster, have more hierarchy levels and are more sensitive to area, performance and power. Similar way, the current test solutions in their turn cannot ensure enough level of quality for future designs, as well as high accuracy for fault diagnosis and repair efficiency since the physical defects and faults are changing due to technology shrinking. Therefore, usually it becomes a necessity to predict those changes beforehand.

Historically, till very recent times the performance improvements of integrated circuits (ICs) had dependence mainly on scaling planar device structures. Meantime nowadays scaling of transistors and other circuit elements is becoming more difficult due to approaching fundamental physical limitations on device size [1]. Three new factors have appeared very recently which, at least, can be considered as ponderable for the performance improvements. Those are:

- 3D transistors;
- 3D ICs;
- Hierarchical test approach.

The first factor is connected to the change of the structure of IC basic element – transistor [2]. In nowadays technology 3D transistors are being used instead of planar (2D) transistors. The main reason is that planar transistors were not able to perform their main functions anymore due to growing leakage and short-channel problems. Currently the most investigated 3D transistors in the literature are known under the term FinFET. It was coined by University of California, Berkeley researchers to describe a nonplanar (3D) transistor. In such transistors the gate covers the Fin from 3 sides therefore increasing the efficient electrical width. FinFET transistors [3] are widely used in 22nm and below technology nodes.

There are other types of 3D transistors as well, such as gate-all-around (GAA) [4] where the gate covers the transistor channel from all sides. These transistors are being considered for use in 5nm and below technology nodes. The investigation showed that the proposed test solution which mainly targets FinFET transistors can be easily adapted to test memories using GAA and other types of 3D transistors.

The second factor is 3D ICs [5], [6] which can address interconnect latency which is also beginning to limit IC performance:

- Vertical connections allow shorter connections (micrometers vs millimeters);
- Expansion of numbers of interconnects through compact vertical connections.

3D ICs promise to overcome barriers in inter-connect scaling by leveraging fast, dense inter-die vias, thereby providing an opportunity for continued higher performance using CMOS. In addition, 3D ICs also enable the integration of heterogeneous fabrication processes on the same chip to make the form factor more compact.

However, the evolution of 3D ICs is accompanied by a lack of having enough efficient mechanisms for their testing. Though the test experts have solved several important issues of 3D ICs (such as test accessibility for each 3D IC die, thermal issues, etc.), the test problems of 3D ICs are remaining as one of the main problems in nowadays IC testing.

The third factor is that beside structural changes in transistors and memories, SoCs are also changing. Those are becoming bigger and by this introducing new test issues [7]. If in the past in a simple SoC only one test system was used, then in nowadays SoCs which are containing many memories and IP cores, multiple test infrastructures are used. Based on those test infrastructures hierarchical test system for an SoC is created. For such SoCs, usually test algorithms from block level are reused at SoC level which allows to essentially reduce and have acceptable test time.

Of course, there are numerous works done in this direction proposing test and repair solutions for different kind of test problems. However, nowadays SoC test problems necessitate to have a common methodology for solving all the above-mentioned problems within a unified test architecture which will be flexible to be tuned to detect faults of upcoming technologies, as well as will be possible to easily integrate it with the existing test systems and applications.

Recently a new built-in test solution was proposed which is based on properties of fault and test algorithm regularity, periodicity and symmetry (see [8]-[11]). It was proposed to describe all the faults by Fault Periodicity Table (FPT) and construct test algorithms by using Test Algorithm Template (TAT). In FPT, each column corresponds to number of cells that a fault is involved with, and each row corresponds to a fault family determined by the number of operations required for fault sensitization. FPT allows analyzing and remembering the properties of a large number of faults in a simpler way, while TAT allows constructing test algorithms without using test algorithm generation tools.

Also, FPT allows to predict faults of future technologies. Thus, a unified BIST, built based on FPT, can allow to detect new faults even after memory and BIST manufacturing. This creates a possibility to avoid doing BIST redesign when moving from one technology to another since it will be possible to adapt the existing BIST to test requirements of new technology. The experiments showed that properties of fault and test algorithm regularity, periodicity and symmetry are preserved also for nowadays ICs, and therefore it will be possible to apply the proposed unified BIST to those ICs for ensuring efficient test outcome [11], [12].

The aim of the thesis

The aim of the work is to create a basis for evolving development of a self-contained test methodology for memories in nanoscale SoCs including not only design and silicon bring-up, but also volume production and in-system test stages. The developed basis should be easily applicable both for the further development of the test methodology and for test solutions in most crucial for the modern lifestyle challenges such as automotive and Internet of Things (IoT).

Trends and challenges of growing memory content - tens of thousands embedded memory instances in modern SoCs, detection of today's defects upon manufacturing and during life time, process variation and FinFET-specific defects for 16-nm and below technology nodes, BIST solutions to address debug, diagnosis, yield optimization and data retention are presented. The thesis also solves the problem related to power management constraints, functional timing implications, test scheduling optimization and area minimization. Today's high reliability and safety-critical chips (for example automotive chips) need multiple in-system self-test modes, such as power-on self-test and repair, periodic in-field self-test, advanced error correction solutions, etc. The proposed solution should allow to create a unified built-in test architecture as well as to provide possibility to predict fault types and test mechanisms for upcoming technology nodes basing on learned from development of the current technology node. Finally, the proposed test solution should be flexible to be integrated with existing systems and applications.

The object of the research

Test challenges for memory devices and systems in nanoscale SoCs are under consideration. Application of the developed solutions to other IP cores in SoCs is also under consideration in the thesis.

The methods of the research

Methods of testing electronic circuits and systems, theory of probability, discrete optimization, Boolean functions, theory of automata, coding theory, theory of reliability and machine learning are used. In addition, simulation and, particularly fault simulation techniques are broadly used to justify the performed experiments.

Scientific novelty

- A unified methodology is proposed for software modeling of faults and test algorithm creation for nanoscale planar and 3D technology based memory systems including:
    - New fault models, their justification and efficient methods for their simulation and new models' generation;
    - Fault classification and diagnosis flow;
    - Efficient test algorithms for detection and diagnosis of new fault types;
    - Extendable and dynamically adaptable BIST architecture formed by a basic triad: test operations, addressing methods and layout aware physical background patterns;
    - A new efficient method for detection and correction of multi-bit soft errors.
- Fault prediction mechanism is given for memory devices of the current and upcoming technology nodes:
    - A systematic evolving view - Fault Periodicity Table (FPT) of possible memory faults with rules of periodicity and regularity as pillars;
    - A Test Algorithm Template (TAT) which allows to construct efficient test algorithms as an alternative to exhaustive or heuristic generation of test algorithms;
    - Special notions and measures to optimize construction of test algorithms.
- A hierarchical test architecture for SoC is created which provides:
    - An efficient method for building an IP structural model for design and verification independently of IP final implementation;
    - A unified solution to test different types of IP cores in the SoCs;
    - An algorithm for scheduling parallel and serial testing of IPs and BIST subsystems.
- Integration and tuning of created approaches to existing test systems and applications are demonstrated and justified:

    - Interfaces with systems providing solutions for chip development cycle including automated test pattern generation, design planning, test time estimation, yield ramp-up, physical failure analysis, fault coverage and yield reports;
    - Solutions to address functional safety and security requirements.

Practical value

The obtained results serve as a basis for a complete multilevel/hierarchical test solution for nanoscale SoCs which increases the test efficiency, reduces the test cost and improves quality of the test. The done work is not just a basis for further research in this area as is already referred by multiple other researchers and practitioners, but it also induces a broad range of diverse applications built on the mentioned basis and covering design, silicon bring-up, volume production and in-system test stages.

Implementation

The results are implemented in Synopsys DesignWare STAR Memory System (SMS) and DesignWare STAR Hierarchical System (SHS) products and are widely used by more than 200 customers (see Appendix A). 10 of top 25 companies in the world using semiconductors [13] are using the obtained results for testing their products. In 2002, SMS product has received the Best in Test "Product of the Year" award. And in 2013, SMS has received a Test & Measurement World Best in Test Award in "Test of Time" category.

The following theses are presented for defense

- Efficient test and repair algorithms constructed for 2D and 3D memories which are based on planar and FinFET transistors;
- Fault Periodicity Table (FPT) and Test Algorithm Template (TAT);
- The mechanism for predicting the faults of future technologies;
- The universal hierarchical architecture for built-in test;

- The efficient method for describing IP structural model;
- The algorithm proposed to schedule parallel and serial testing of IPs;
- The method proposed for protection against multi-bit soft errors.

Presentations

The main results of the work are presented at IEEE International Test Conference (ITC 2012, 2013, 2014, 2017), at IEEE VLSI Test Symposium (VTS 2005, 2006, 2008, 2013, 2014, 2015, 2016), at IEEE European Test Symposium (ETS 2006, 2007, 2017), at IEEE Asian Test Symposium (ATS 2011), at IEEE International On-Line Testing Symposium (IOLTS 2011, 2013, 2015, 2017), at IEEE Workshop on Design and Diagnosis of Electronic Circuits and Systems (DDECS 2006, 2007), at IEEE East-West Design and Test Symposium (EWDTS 2006, 2007, 2010, 2012, 2013, 2014, 2015, 2016), at International Conference on Computer Science and Information Technology (CSIT 2005, 2009, 2011), at the general seminar of YSU IT Educational and Research Center (2017), at the general seminar of Institute for Informatics and Automation Problems (IIAP) of NAS RA (2017), recipient of the TTTC/ITC Commemorative Gerald W. Gordon Award (2008).

Publications

The main results are published in 51 works, 46 of which as scientific papers, 3 granted patents and 2 patent applications. The published works are included in the list of used literature.

The structure and size of the thesis

The thesis consists of introduction, 8 sections, main conclusions, list of used literature and 3 appendixes. The main part of the work has 226 pages, 58 figures and 57 tables. The list of literature has 24 pages and includes 177 papers. The overall work consists of 247 pages.

Acknowledgements

I would like to express coming from heart thankfulness to Foreign Member and Honorary Doctor of the National Academy of Sciences of Armenia Yervant Zorian and my scientific adviser Member of the National Academy of Sciences D.Sc., Professor Samvel Shoukourian for their essential help and continuous advices in forming and elaborating this work.

Source of study

- S. Shoukourian (Armenia), Y. Zorian (USA) – IEEE Design & Test of Computers (2003, 2004);
- A. J. van de Goor, S. Hamdioui (Netherlands) – Testing Semiconductor Memories: Theory & Practice (1991), VTS (2002, 2004), MTDT (2002);
- Ch. W. Wu (Taiwan) – ITC (2001), TCAD (2002);
- M. Nicolaidis (France) – ITC (1992), ETS (2006);
- L. Dilillo, A. Bosio (France) – ATS (2003), ETS (2004), Book - Advanced Test Methods for SRAMs: Effective Solutions for Dynamic Fault Detection in Nanoscaled Technologies (2010);
- P. Prinetto, S. Di Carlo, G. Di Natale (Italy) – ITC (2005), ETS (2006), DATE (2006);
- H.-J. Wunderlich (Germany) – ATS (2012, 2014), TCAD (2015);
- R. Ubar (Estonia) – ETW (1998), ISQED (2002);
- M. S. Reorda, D. Appello (Italy) – DDECS (2011), LATS (2017);
- M. L. Bushnel, V. D. Agrawal (USA) – Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits, Kluwer Academic Publishers (2000).

CHAPTER 1. TEST CHALLENGES IN NANOSCALE MEMORY DEVICES AND SYSTEMS

1.1 Test Basis for Memory Devices and Systems: Faults, Test Operations and Algorithms, Built-In Self-Test (BIST) Infrastructure

1.1.1 Definition of Faults

Fault types. In general, two types of faults can be distinguished: permanent and non-permanent [14]:

- Permanent faults – Presence of a fault that affects the functional behavior of a system (chip, array or board) permanently. For example, it can be incorrect connections, broken component, functional design error, etc.
- Non-permanent faults – These faults are present only part of the time, they occur randomly and affect the system behavior for finite but unknown periods of time. The non-permanent faults can be divided into two groups: transient and intermittent.
  - Transient faults are caused by environmental conditions, such as cosmic rays, alpha particles, pollution, pressure, etc. Transient faults are hard to model due to their random behavior. Transient faults in random access memories are often called as soft errors. Those faults are considered as non-recurring and they do not bring to permanent damage of the memory.
  - Intermittent faults are caused by non-environmental conditions, such as loose connections, aging, critical timing, resistance and capacitance variations, noise, etc. Intermittent faults can be modeled using models of permanent faults. The only difference is the duration of the failure caused by an intermittent fault.

Static and dynamic faults. Many functional fault models (FFMs) for static and dynamic random access memories (RAMs) have been introduced in the past. Specifically, static and dynamic FFMs were introduced in [15], [16]. Further, a concept of linked functional fault models was developed [17]. Based on Inductive Fault Analysis (IFA), their existence and importance in current memories was validated experimentally [18]-[26].

The definition of the fault primitive (FP) concept, used to define faults, can be found in [15], [16]. Single-cell FPs are described by <S/F/R> and two-cell (coupling) FPs are described by $<S_a; S_v/F/R>$. In notations of FPs, S, S_a and S_v are the sequences of operations required for fault sensitization (S is applied to the faulty cell, S_a - to the aggressor cell and S_v - to the victim cell), $F \in \{0, 1\}$ is the observed memory behavior that deviates from the expected one. $R \in \{0, 1, -\}$ is the result of a read operation applied to the faulty cell, in case if the last operation of S is a Read operation. " - " is used when the last operation of S is not a Read operation. Note that if S_a is a sequence of operations then S_v should be a state; if S_a is a state then S_v can be a state or a sequence of operations. For example, if a cell has the fault <0W0/1/->, it means that if it contains value 0, then applying operation W0 on it will flip the cell value from 0 to 1. Or if two cells contain the fault <1; 0W1R1/0/1>, it means the following: if the aggressor cell has value 1, the victim cell has value 0 then applying two sequential operations {W1, R1} will fail. Though the read operation will return the correct value 1, the victim cell value will remain 0.

A functional fault model (FFM) is defined as a non-empty set of FPs. The difference between static and dynamic faults is determined by the number of operations required in S, S_a or S_v. Static faults are the faults sensitized by performing at most one operation. Dynamic faults are the faults that are sensitized by performing more than one operation.

In this work, all unlinked static faults and all dynamic two-operation single-cell faults, as well as the subclasses of two-operation two-cell unlinked dynamic faults where both sensitizing operations are applied either to the aggressor cell (S_{aa}) or the victim cell (S_{vv}) are considered. The faults of these subclasses are considered as the most important dynamic faults in RAMs [16], [19], [27], [28]. The other two subclasses (S_{av} and S_{va}) with two sensitizing operations to be applied sequentially, one to the aggressor (respectively, victim) cell and the other one to the victim (respectively, aggressor) cell are considered less probable and usually are not considered for testing due to need of high complexity test algorithms (see [29]-[31]). If memory scrambling (mapping between memory logical addresses and their corresponding physical positions) is not available then the test algorithm has to consider all possible pairs of memory cells for completeness of testing, and the algorithm will have quadratic complexity with respect to the number of memory cells (words). In such cases, these algorithms are not feasible for testing of large memories.

The unlinked static and dynamic faults are listed in Table 1 (see [15], [16]). The symbol "~" used in the table denotes logical negation, and x, y, z, t $\in$ {0, 1}. All these faults are presented by <S/F/R> and <S_a; S_v /F/R>. For example, TF includes FPs <0W1/0/-> and <1W0/1/->, while CFdrd includes FPs <0; 0R0/1/0>, <1; 0R0/1/0>, <0; 1R1/0/1> and <1; 1R1/0/1>.

In [17], a definition of static linked faults (LFs) is introduced. In [32], the LF definition is generalized by removing some constraints given in [17] and extending the class of static linked faults. In this work, the same definition is used to define dynamic linked faults, as well as the links between static and dynamic faults.

LF definition. Let FP_1=<S_1/F_1/R_1> and FP_2=<S_2/F_2/R_2> be any static or dynamic fault primitives. If FP_1 and FP_2 share the same victim cell, then it is said that there is a linked fault (denoted as $FP_1 \rightarrow FP_2$) if the following three conditions are satisfied:

C_1: Read operations of FP_1 and FP_2 do not detect a fault.

C_2: FP_2 masks FP_1, i.e., $F_2 = \sim F_1$.

C_3: FP_2 is compatible with FP_1, which means that if S_2 is applied immediately after S_1, then the final state of the aggressor or the victim cell after performing S_1 should be the same as the initial state required by S_2.

In [17], the LFs are divided into 5 groups as it is shown in Figure 1:

- LF1 - combination of two single-cell unlinked faults. Both faults have the same faulty (victim) cell.
- $LF2_{av}$ - combination of one two-cell FP_1 and one single-cell FP_2 unlinked faults, where FP_1 is sensitized first. The victim cell of FP_1 coincides with the faulty cell of FP_2.

Table 1. Unlinked static and two-operation dynamic faults

Subclass	Functional fault models	Fault primitives
Unlinked static single-cell faults	State Fault or Stuck-At Fault (SF)	<x/~x/->
	Transition Fault (TF)	<xW(~x)/x/->
	Write Destructive Fault (WDF)	<xWx/~x/->
	Read Destructive Fault (RDF)	<xRx/~x/~x>
	Deceptive Read Destructive Fault (DRDF)	<xRx/~x/x>
	Incorrect Read Fault (IRF)	<xRx/x/~x>
Unlinked static two-cell faults	State Coupling Fault (CFst)	<x; y/~y/->
	Transition Coupling Fault (CFtr)	<x; yW(~y)/y/->
	Write Destructive Coupling Fault (CFwd)	<x; yWy/~y/->
	Read Destructive Coupling Fault (CFrd)	<x; yRy/~y/~y>
	Deceptive Read Destructive Coupling Fault (CFdrd)	<x; yRy/~y/y>
	Incorrect Read Coupling Fault (CFir)	<x; yRy/y/~y>
	Disturb Coupling Fault (CFds)	<xRx; y/~y/->, <xWy; z/~z/->
Unlinked two-operation dynamic single-cell faults	dynamic Read Destructive Fault (dRDF)	<xWyRy/~y/~y>, <xRxRx/~x/~x>
	dynamic Deceptive Read Destructive Fault (dDRDF)	<xWyRy/~y/y>, <xRxRx/~x/x>
	dynamic Incorrect Read Fault (dIRF)	<xWyRy/y/~y>, <xRxRx/x/~x>
	dynamic Transition Fault (dTF)	<xWyW(~y)/y/->, <xRxW(~x)/x/->
	dynamic Write Destructive Fault (dWDF)	<xWyWy/~y/->, <xRxWx/~x/->
Unlinked two-operation dynamic two-cell faults	dynamic Read Destructive Coupling Fault (dCFrd)	<x; yWzRz/~z/~z>, <x; zRzRz/~z/~z>
	dynamic Deceptive Read Destructive Coupling Fault (dCFdrd)	<x; yWzRz/~z/z>, <x; zRzRz/~z/z>
	dynamic Incorrect Read Coupling Fault (dCFir)	<x; yWzRz/z/~z>, <x; zRzRz/z/~z>
	dynamic Transition Coupling Fault (dCFtr)	<x; yWzW(~z)/z/->, <x; zRzW(~z)/z/->
	dynamic Write Destructive Coupling Fault (dCFwd)	<x; yWzWz/~z/->, <x; zRzWz/~z/->
	dynamic Disturb Coupling Fault (dCFds)	<xWyWt; z/~z/->, <xWyRy; z/~z/->, <xRxWy; z/~z/->, <xRxRx; z/~z/->

- $LF2_{va}$ - combination of one single-cell FP_1 and one two-cell FP_2 unlinked faults, where FP_1 is sensitized first. The faulty cell of FP_1 coincides with the victim cell of FP_2.
- $LF2_{aa}$ - combination of two two-cell unlinked faults FP_1 and FP_2. FP_1 and FP_2 have the same aggressor, as well as the same victim cells.
- LF3 - combination of two two-cell unlinked faults FP_1 and FP_2. FP_1 and FP_2 have the same victim cells, but different aggressor cells.

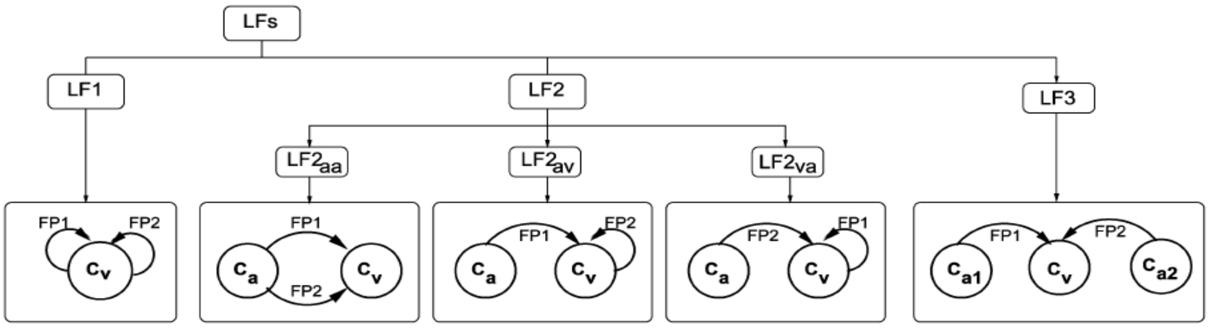


Figure 1. Classification of LFs

2-composite faults. In [32], the notion of 2-composite faults was introduced. This allows defining all the possible links between the considered faults. Let FP_1=<$S_1/F_1/R_1$> and FP_2=<$S_2/F_2/R_2$> be any fault primitives. "2-composite fault", denoted as FP_1*FP_2, is the combination of fault primitives FP_1 and FP_2 having the same victim cell (if both FPs are two-cell faults, then their aggressor cells can be either the same or different) with the only restriction that these FPs cannot be sensitized simultaneously. This limitation is to prevent non-realistic cases. For example, in <1; 0R0/1/0>*<1; 0R0/0/1> the two FPs are sensitized simultaneously by the R0 operation applied to the victim cell if the aggressor cells of both FPs are in state 1. First FP tends to flip the victim cell to state 1 while the second tends to keep the state of the cell in 0. As a result, the victim cell may accept a random value. Similarly, the output may also tend to a random value. In this work, such faults that may lead to random outputs are excluded from the consideration. Such faults are considered as non-realistic (see [14], [17], [33]).

In 2-composite faults there is no special order for fault sensitization, i.e., $FP_1*FP_2=FP_2*FP_1$. The fault sensitization depends on the structure of a March test algorithm, for example, one March test algorithm can first sensitize FP_1, the second one – FP_2. All three conditions (restrictions) defined for linked faults in [17] are not taken into account in the definition of 2-composite faults. Thus, 2-composite fault class is essentially wider than the class of "linked" faults defined in [17].

2-composite faults include unlinked and linked faults, as well as two faults having the same victim cell but not masking each other. For example, <0R0; 0/1/->*<0R0; 1/0/-> and <0R0; 0/1/->*<0R0; 0/1/-> both are 2-composite faults. The first fault is considered as a linked fault since it satisfies all three conditions C_1, C_2 and C_3. The second fault is not considered as a linked fault since condition C_2 is violated. However, the second fault also can be realistic in RAMs. Thus it is expedient to consider all the possible cases of two static and dynamic faults having the same victim cell (none of the conditions C_1, C_2 and C_3 is taken into account).

In FP_1*FP_2, in general, FP_1 and FP_2 can be any types of faults. In this work, as FP_1 and FP_2 only static and dynamic faults will be considered (see Table 1). It is allowed that $FP_1=\varnothing$ or $FP_2=\varnothing$ ($\varnothing$ is empty FP) but not simultaneously $FP_1=\varnothing$ and $FP_2=\varnothing$. The following equalities are true, $FP*\varnothing=\varnothing*FP=FP$.

The whole space of 2-composite static and dynamic faults can be divided into 5 classes:

- Unlinked static faults: $FP*\varnothing$, where FP is an unlinked static fault;
- Unlinked dynamic faults: $FP*\varnothing$, where FP is an unlinked dynamic fault;
- Linked static*static faults: FP_1*FP_2, where FP_1 and FP_2 are both unlinked static faults;
- Linked static*dynamic faults: FP_1*FP_2, where FP_1 is an unlinked static fault and FP_2 is an unlinked dynamic fault;
- Linked dynamic*dynamic faults: FP_1*FP_2, where FP_1 and FP_2 are both unlinked dynamic faults.

Based on the classifications of LFs and 2-composite faults, the universe of static and dynamic faults can be classified (see Figure 2). In the figure, the class $LF2_{va}$ is not depicted since it is the same class as $LF2_{av}$ with respect to 2-composite faults.

The addressing orders of the aggressor cell "a" (or aggressor cells "a_1" and "a_2") and the victim cell "v" of a fault is essential since a test algorithm can detect the same fault if a<v and not detect if a>v, or vice-versa. For a single-cell FP, it will be considered that the aggressor and victim cells coincide, i.e., a=v.

For a 2-composite fault FP_1*FP_2, "a_1" is denoted as the aggressor cell of FP_1, "a_2" - as the aggressor cell of FP_2 and "v" - as the victim cell. All the possible orders of the aggressor-victim cells for 2-composite faults are listed below:

- Unlinked 1-cell fault: a=v;
- Unlinked 2-cell fault: a>v and a<v;
- LF1: $a_1=a_2=v$;
- $LF2_{av}$: $a_1=a_2=a$, a>v, a<v;
- $LF2_{aa}$: $a_1=a_2=a$, a>v, a<v;
- LF3: $a_1>a_2>v$, $a_2>a_1>v$, $a_1>v>a_2$, $a_2>v>a_1$, $v>a_1>a_2$, $v>a_2>a_1$.

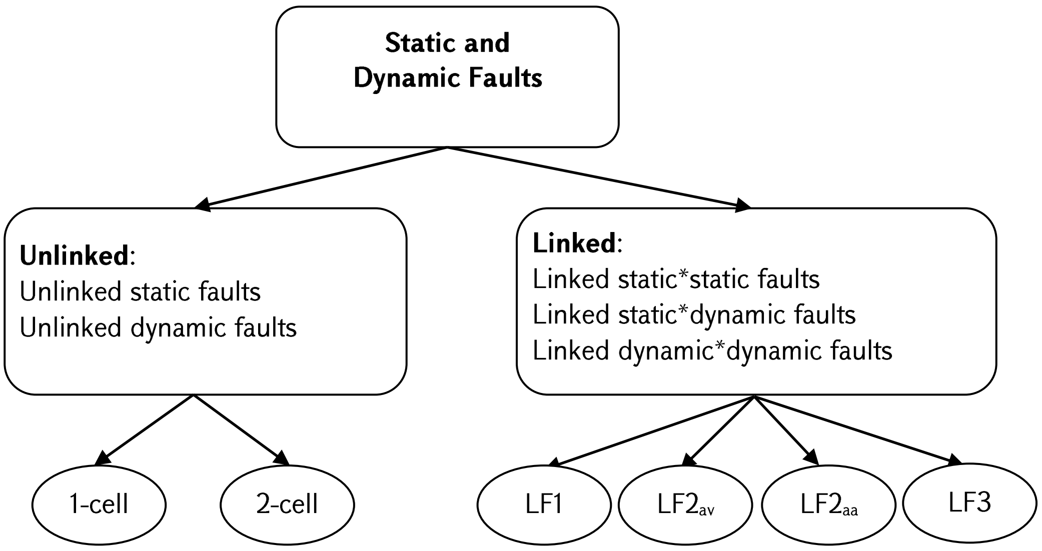

Static and Dynamic Faults
Unlinked:
Unlinked static faults
Unlinked dynamic faults
Linked:
Linked static*static faults
Linked static*dynamic faults
Linked dynamic*dynamic faults
1-cell
2-cell
LF1
$LF2_{av}$
$LF2_{aa}$
LF3

Figure 2. Classification of static and dynamic faults

Figure 3 shows an example of a linked static*static LF3 fault <0; 0W1/0/->*<1; 0R0/1/0>. This fault is composed of two unlinked static coupling faults having different aggressor cells. The fault is sensitized in two different conditions:

C_1. If aggressor cell "a_1" has value 0, victim cell "v" has value 0, then operation W1 applied to the victim cell will fail and the victim cell value will remain 0.

C_2. If aggressor cell "a_2" has value 1, victim cell "v" has value 0 then operation R0 applied to the victim cell will flip the victim cell value returning the correct result 0.

Figure 4 shows an example of a linked static*dynamic $LF2_{av}$ fault <0W1R1; 1/0/->*<1W0/1/->. This fault is composed of an unlinked dynamic coupling fault and an unlinked static single-cell fault. This fault is sensitized in two different conditions:

C_1. If aggressor cell "a" has value 0, victim cell "v" has value 1, then sequence of operations {W1, R1} applied to the aggressor cell will change the victim cell value from 1 to 0.

C_2. If victim cell "v" has value 1, then operation W0 applied to the victim cell will fail and the victim cell value will remain 1.

Figure 5 shows an example of a linked dynamic*dynamic $LF2_{aa}$ fault <1R1W0; 0/1/->*<1; 1W1W0/1/->. This fault is composed of two unlinked dynamic coupling faults having the same aggressor cell. This fault is sensitized in two different conditions:

C_1. If aggressor cell "a" has value 1, victim cell "v" has value 0, then sequence of operations {R1, W0} applied to the aggressor cell will change the victim cell value from 0 to 1.

C_2. If aggressor cell "a" has value 1, victim cell "v" has value 1, then sequence of operations {W1, W0} applied to the victim cell will fail and the victim cell value will remain 1.

In [32], it was shown that the number of all linked static*static fault primitives (FPs) is 600. The classes of linked static*dynamic and linked dynamic*dynamic faults contain a huge number of FPs compared with the class of linked static*static faults. For example, the class of linked dynamic*dynamic faults contains 12459 FPs:

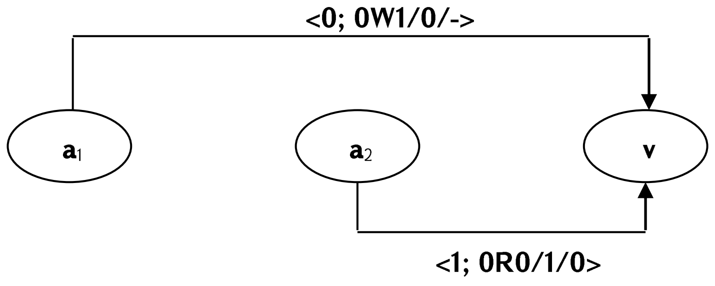


Figure 3. Example of a linked static*static LF3 fault

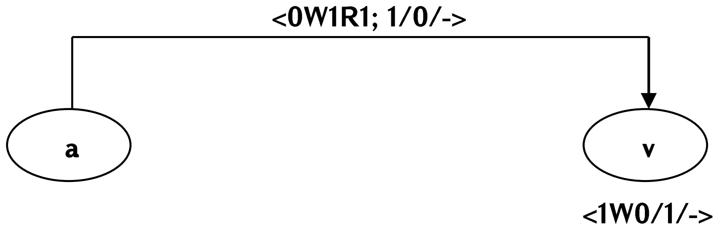


Figure 4. Example of a linked static*dynamic $LF2_{av}$ fault

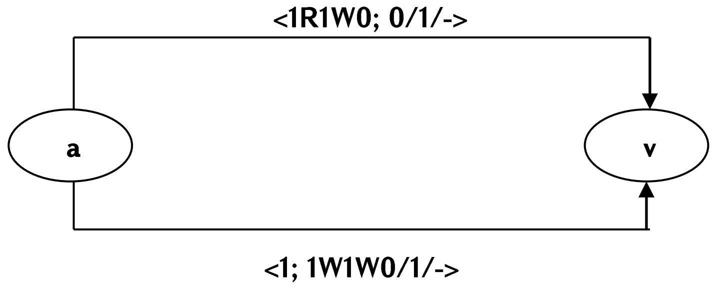


Figure 5. Example of a linked dynamic*dynamic $LF2_{aa}$ fault

- $LF1=FP_1*FP_2$, FP_1 and FP_2 are single cell dynamic FPs. The number of single cell dynamic FPs is 30 (see [28]). Since $FP_1*FP_2=FP_2*FP_1$, and there are 18 non-realistic dynamic*dynamic LF1 faults (see Table 2), then the number of FPs in LF1 is 30(30+1)/2-18=447.
- $LF2_{av}=FP_1*FP_2$, FP_1 is a single cell and FP_2 is a two-cell dynamic FPs. The number of two-cell dynamic FPs is 96 (see [28]). Since there are 72-non-realistic dynamic*dynamic $LF2_{av}$ faults (see Table 2), then the number of FPs in $LF2_{av}$ is 30x96-72=2808.
- $LF2_{aa}=FP_1*FP_2$, FP_1 and FP_2 are two-cell dynamic FPs. Since $FP_1*FP_2=FP_2*FP_1$, and there are 36 non-realistic dynamic*dynamic $LF2_{aa}$ faults (see Table 2), then the numbers of FPs in $LF2_{aa}$ is 96(96+1)/2-36=4620.

- LF3=FP_1*FP_2, FP_1 and FP_2 are two-cell dynamic FPs. Since FP_1*FP_2=FP_2*FP_1, and there are 72 non-realistic dynamic*dynamic LF3 (see Table 2), then the numbers of FPs in LF3 is 96(96+1)/2-72=4584.
- The class of linked dynamic*dynamic faults contains 447+2808+4620+4584=12459 FPs.

A total of 198 dynamic*dynamic non-realistic faults are described in Table 2. Note that the symbol "~" used in Table 2 denotes logical negation, and x, y, z, t $\in$ {0, 1}.

The same way, it is easy to calculate also the number of realistic FPs from the class of linked static*dynamic faults.

Table 2. Non-realistic dynamic*dynamic faults

Subclass	Non-realistic faults
LF1	<xWyRy/~y/~y>*<xWyRy/y/~y>, <xWyRy/~y/~y>*<xWyRy/~y/y>, <xWyRy/y/~y>*<xWyRy/~y/y>, <xRxRx/~x/~x>*<xRxRx/x/~x>, <xRxRx/~x/~x>*<xRxRx/~x/x>, <xRxRx/x/~x>*<xRxRx/~x/x>
LF2_{av}	<x; yWzRz/~z/~z>*<yWzRz/z/~z>, <x; yWzRz/~z/~z>*<yWzRz/~z/z>, <x; yWzRz/z/~z>*<yWzRz/~z/~z>, <x; yWzRz/z/~z>*<yWzRz/~z/z>, <x; yWzRz/~z/z>*<yWzRz/~z/~z>, <x; yWzRz/~z/z>*<yWzRz/z/~z>, <x; zRzRz/~z/~z>*<zRzRz/z/~z>, <x; zRzRz/~z/~z>*<zRzRz/~z/z>, <x; zRzRz/z/~z>*<zRzRz/~z/~z>, <x; zRzRz/z/~z>*<zRzRz/~z/z>, <x; zRzRz/~z/z>*<zRzRz/~z/~z>, <x; zRzRz/~z/z>*<zRzRz/z/~z>
LF2_{aa}	<x; yWzRz/~z/~z>*<x; yWzRz/z/~z>, <x; yWzRz/~z/~z>*<x; yWzRz/~z/z>, <x; yWzRz/z/~z>*<x; yWzRz/~z/z>, <x; zRzRz/~z/~z>*<x; zRzRz/z/~z>, <x; zRzRz/~z/~z>*<x; zRzRz/~z/z>, <x; zRzRz/z/~z>*<x; zRzRz/~z/z>
LF3	<x; yWzRz/~z/~z>*<t; yWzRz/z/~z>, <x; yWzRz/~z/~z>*<t; yWzRz/~z/z>, <x; yWzRz/z/~z>*<t; yWzRz/~z/z>, <x; zRzRz/~z/~z>*<t; zRzRz/z/~z>, <x; zRzRz/~z/~z>*<t; zRzRz/~z/z>, <x; zRzRz/z/~z>*<t; zRzRz/~z/z>

Three-operation single-cell dynamic faults. This kind of faults are sensitized by applying three operations sequentially to the same cell [34]. Table 3 describes all three-operation single-cell dynamic FPs.

Those FPs are compiled into the following 5 FFMs.

1. Three-operation Dynamic Read Destructive Faults (ddRDF);
2. Three-operation Dynamic Incorrect Read Faults (ddIRF);
3. Three-operation Dynamic Deceptive Read Destructive Faults (ddDRDF);
4. Three-operation Dynamic Transition Faults (ddTF);
5. Three-operation Dynamic Write Destructive Faults (ddWDF).

Note that the symbol "~" used in Table 3 denotes logical negation, and x, y, z ∈ {0, 1}.

Table 3. Three-operation single-cell dynamic FPs

FFM	Fault primitives
ddRDF	< xRxRxRx / ~x / ~x >, < xRxWyRy / ~y / ~y >, < xWyRyRy / ~y / ~y >, < xWyWzRz / ~z / ~z >
ddIRF	< xRxRxRx / x / ~x >, < xRxWyRy / y / ~y >, < xWyRyRy / y / ~y >, < xWyWzRz / z / ~z >
ddDRDF	< xRxRxRx / ~x / x >, < xRxWyRy / ~y / y >, < xWyRyRy / ~y / y >, < xWyWzRz / ~z / z >
ddTF	< xRxRxW(~x) / x / - >, < xRxWyW(~y) / y / - >, < xWyRyW(~y) / y / - >, < xWyWzW(~z) / z / - >
ddWDF	< xRxRxWx / ~x / - >, < xRxWyWy / ~y / - >, < xWyRyWy / ~y / - >, < xWyWzWz / ~z / - >

Aging faults. There is another class of faults which mainly appear during the system operating mode (in the field). Those are called aging faults. Long-term performance degradations may activate physical defects in the system, due to transistors aging. The main aging effects cause NBTI (Negative Bias Temperature Instability) and PBTI (Positive Bias Temperature Instability) [35], [36]. These effects shift the threshold voltage of a transistor, which causes bit-cell stuck-at, bit-cell transition, coupling, delay faults and sense amplifier failures.

Random telegraph noise induced faults. Random telegraph noise (RTN) is caused by trapping and de-trapping of individual carriers, due to active traps in the gate oxide. This brings to variations in threshold voltages of the SRAM cell transistors. The variations of these threshold voltages can cause failures in SRAM cell randomly, flipping the cell value. The RTN induced failures can appear in idle state (no operation on the cell) or/and during read/write operations. The occurrence probability of RTN induced failures reaches its maximum value at low voltages and/or at high temperatures [37].

Figure 6 illustrates an example where RTN induced failure is observed. After several

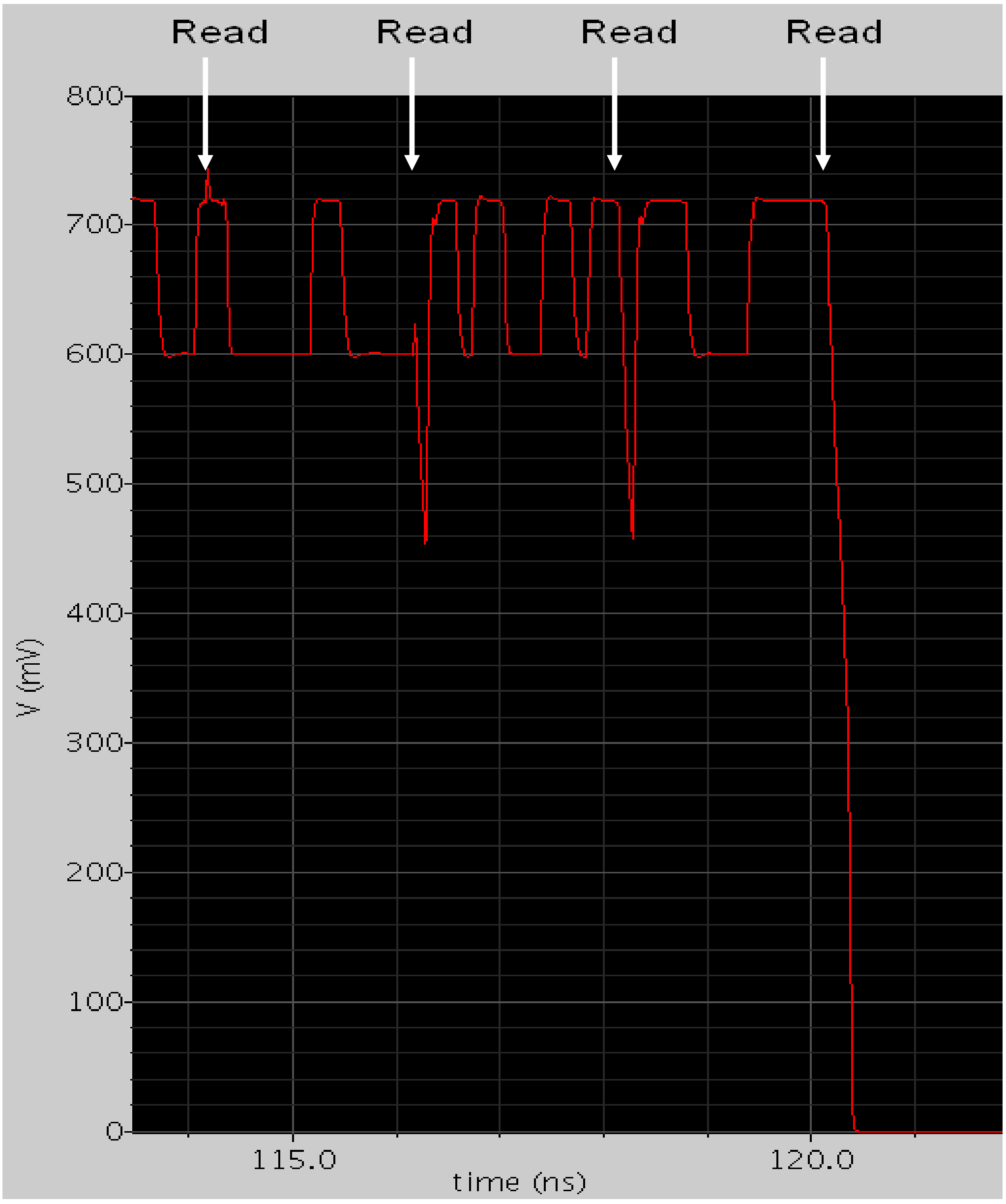


Figure 6. RTN induced failure

Read operations the failure is detected. The first Read operation is passing since it is performed at a time when RT voltage has its maximal value. The second and third Read operations are also passing (but the fluctuations are more than in the first case) since they are performed in the middle voltage value. The forth Read operation fails since it is performed at a time when RT voltage has its minimal value.

The investigations show that the failures caused by RTN can bring to the following types of static faults, but in random manner:

- Stuck-at fault (SAF);
- Transition fault (TF);
- Read destructive fault (RDF);
- Deceptive read destructive fault (DRDF);
- Data retention faults (DRF) - A cell value flips after some time when no operation is applied on the cell.

1.1.2 Test Algorithms

March test *M* [14] is a test algorithm with a finite number of March elements M=M_1; M_2; ...; M_k, where each March element M_i=A_i(O_1, O_2, ..., O_m) consists of:

- $A_i \in \{\Uparrow, \Downarrow, \Leftrightarrow\}$ – addressing direction;
- $O_j \in$ {R0, R1, W0, W1} – finite number of read/write operations.

Example: March C- [14]: $\Leftrightarrow$(W0); $\Uparrow$(R0, W1); $\Uparrow$(R1, W0); $\Downarrow$(R0, W1); $\Downarrow$(R1, W0); $\Leftrightarrow$(R0).

Memory devices can also be tested by other types of test algorithms (GALPAT, Butterfly, etc.), but March tests are considered as efficient test algorithms since they have linear complexity w.r.t. memory cells (O(N)). And this is important since nowadays memory devices consist of huge amount of memory cells and even test algorithms with O(NlogN) complexity are not applicable to those memory devices.

The main problems of testing memory devices are: fault detection, fault localization and fault diagnosis [14], [38]. Fault localization problem is solved by applying test operation jumping between victim cell and candidate aggressor cells. Besides, usually there is a necessity to apply different sequences of test operations to victim and aggressor cells. March tests do not have such capability. Thus, extended class of March tests is used for solving fault localization problems which is called class of March-based tests [38]. March-based tests have the ability to apply test operations only to a subset of memory cells, apply any kind of jumps from one memory cell to another as well as apply different sequence of operations to different cells of the memory.

The above definition of March tests is not sufficient to test nowadays nanoscale memories.

There are special memory faults that require specific test mechanisms (addressing directions, test operations, data background patterns) for their sensitization and detection. If only one operation can sensitize a given fault, then any generated March test that does not use that operation will not detect the fault. Thus, it is very important to develop accurate and full set of test mechanisms to detect the given set of faults. Another aspect is that multi-port memories require concurrent test operations (when applying test operations through different access ports in parallel) to sensitize some inter-port faults [39]. Mainly three types of access ports are considered for multi-port memories:

- Read-only port – Only operation Read can be applied through this port;
- Write-only port – Only operation Write can be applied through this port;
- Read/Write (dual) port – Both operations Read and Write can be applied through this port.

Thus, in this work the class of March tests is extended which can provide the necessary fault coverage for nowadays memories. The extension was done for address directions, test operations and data background patterns:

- Address direction types:
    - Fast column: accessing all addresses of one row before moving to the next row;
    - Fast row: accessing all addresses of one column before moving to the next column;
    - Single address: accessing only one address of the memory (usually it is address 0);

  - Ping-pong (known also as Address Complement): accessing the addresses 0, then N-1, then 1, then N-2, then 2, then N-3, ..., where N is the number of addresses);
  - H1 addressing: In this mode the addresses are changing in the following way:

    0000, 000**1**, 0000, 00**1**0, 0000, 0**1**00, 0000, **1**000, 0000

    0001, 000**0**, 0001, 00**1**1, 0001, 0**1**01, 0001, **1**001, 0001

    ...

    1111, 111**0**, 1111, 11**01**, 1111, **1011**, 1111, **0**111, 1111
- Test operations applicable for single-port and multi-port memories:
  - W: Operation Write is applied to the current address through one of Write-only or Read/Write port.
  - R: Operation Read is applied to the current address through one of Read-only or Read/Write port.
  - RSHR: Operation Read is applied through one of Read-only or Read/Write port (active port) to the current address and simultaneously Read operations are applied through the rest of Read-only and Read/Write ports (inactive ports) to the same address.
  - RSHRW: Operation Read is applied through one of Read-only or Read/Write port (active port) to the current address and simultaneously Read/Write operations are applied through the rest of inactive ports (operations Read from Read-only ports, operations Write from Write-only and Read/Write ports) to neighboring address (same row adjacent column or same column adjacent row). The results of Read operations from inactive ports are ignored.
  - WSHRW**:** Operation Write is applied through one of Write-only or Read/Write port (active port) to the current address and simultaneously Read/Write operations are applied through the rest of inactive ports (operations Read from Read-only ports, operations Write from Write-only and Read/Write

ports) to neighboring address (same row adjacent column or same column adjacent row). The results of Read operations from inactive ports are ignored.

- Data background patterns (see [40]):
  - Solid: data is all 0s (0000) or all 1s (1111);
  - Column Stripe: odd columns contain 0000…00 data, even columns contain 1111…11 data or vice-versa;
  - Row Stripe: odd rows contain 0000…00 data, even rows contain 1111…11 data or vice-versa;
  - Checkerboard: odd columns contain 0101…01 data, even columns contain 1010…10 data or vice-versa;
  - Double Column Stripe: columns 1, 2, 5, 6, 9, 10, … contain 0000…00 data, columns 3, 4, 7, 8, 11, 12, … contain 1111…11 data or vice-versa;
  - Double Row Stripe: rows 1, 2, 5, 6, 9, 10, … contain 0000…00 data, rows 3, 4, 7, 8, 11, 12, … contain 1111…11 data or vice-versa;

Figure 7 shows the above mentioned background patterns.

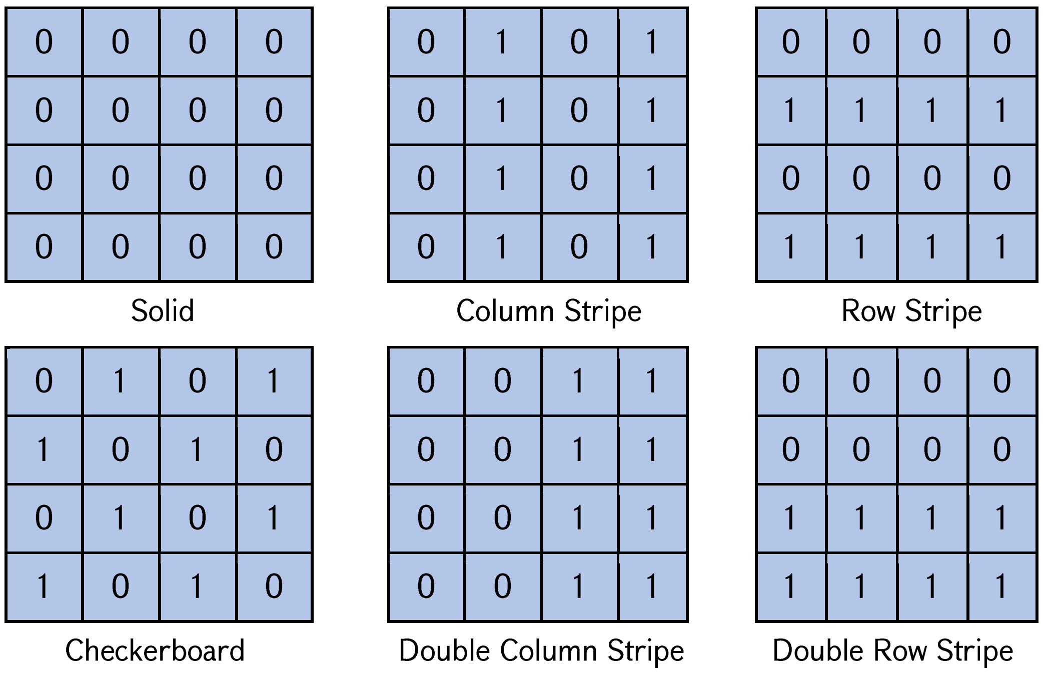


Figure 7. Background patterns

1.1.3 Built-In Self-Test (BIST)

Built-in self-test (BIST) systems are widely used for testing memory devices. During the production, the BIST applies a sequence of read/write operations to the memory device (these sequences are called test algorithms) in order to check if the memory functions correctly. There are different BIST architectures [41]-[46] and one of the most commonly used is the microcode-based programmable BIST architecture ([42], [47]). It contains Test Algorithm Register (TAR) which stores test algorithms in it in a predefined format. TAR is externally accessible which allows to program new test algorithms from outside. Typically, the programmable BIST systems provide enough flexibility for programming new test algorithms. At the same time, the test algorithm is composed of special test mechanisms (test operations, background patterns and addressing methods) which are usually hardwired in the BIST infrastructure. In this context, the programmability of BIST means that only the existing test mechanisms can be used when programming a test algorithm. In some cases, this limitation can lead to impossibility of programming the required test algorithm in TAR. Therefore, there is usually a necessity to make the test mechanisms programmable as well by this providing maximum level of flexibility for programming new test algorithms.

In [47], a programmable microcode-based BIST architecture is extended in order to provide additional programmability for test operations and background patterns. The modifications are done in two units of BIST scheme:

- Data Generator Unit which generates background patterns (Solid, Checkerboard, etc.);
- Operation Decoder Unit which generates test operations (Write, Read, etc.).

Data Generator Unit consists of the list of predefined background patterns (not consuming any storage elements and defined at BIST generation), user-defined background patterns (programmable register of predefined format) as well as from pattern decoders per each memory instance. Multi-port memories have two instances of pattern decoder, one for direct port (port performing main operation) and one for shadow ports (performing shadow operations). The Data Generator Unit is shown in Figure 8.

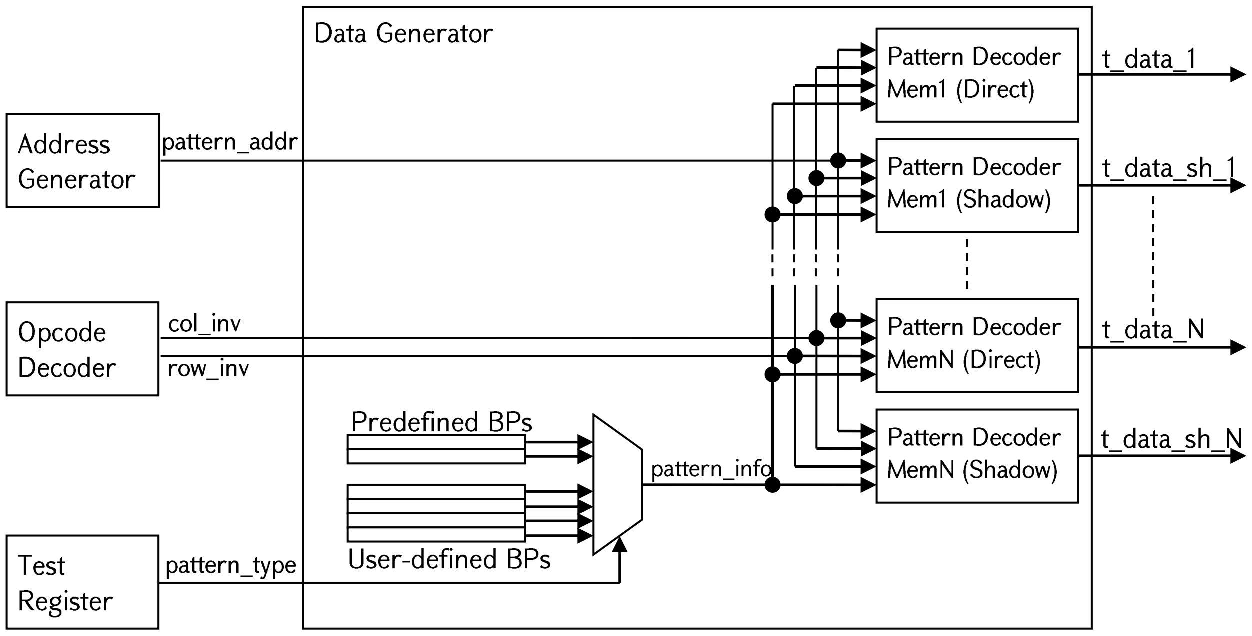


Figure 8. Data Generator Unit

For each operation test register defines the pattern type, used by data generator to select one of the predefined/user-defined background patterns, in its turn address generator provides test address on which selected operation must be performed and operation decoder decodes current operation to define column and row inversion bits.

Based on the selected pattern (pattern_info), test address (pattern_addr), column and row LSB (least significant bit) inversion bits, as well as pattern inversion bit decoded from test register, pattern decoder generates test data for direct and/or shadow ports for selected memory instance. The Test data decoder is shown in Figure 9, where:

pattern_data = ((pattern_addr[i] ^ pattern_info.not_bit[i]) & pattern_info.xor_bit[i] ^ (col_inv ^ pattern_info.col_inv) ^ (row_inv ^ pattern_info.row_inv)) ? pattern_true : pattern_false;

t_data = pattern_inv ? ~pattern_data : pattern_data.

Programmable background pattern register consists of predefined number (defined at generation time) of pattern descriptors. For multi-port memories two pattern descriptors are required per each memory instance, one for direct and the other for shadow ports. The format for pattern descriptor is shown in Figure 10, where:

- xor_bit is an address size field that defines whether the selected address bit must participate in background pattern generation or not;

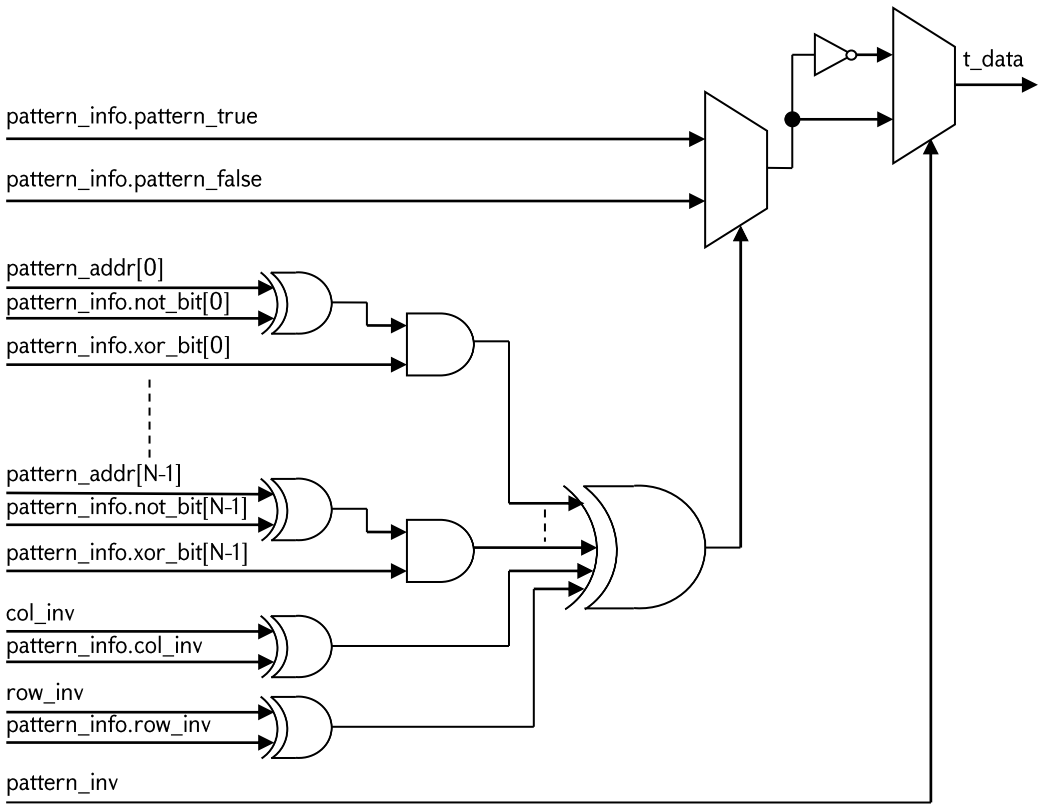


Figure 9. Test data decoder

- not_bit is an address size field that defines the inversion of the selected address bit specified in xor_bit;
- row_inv is one-bit field decoded from operation descriptor, that defines the row LSB inversion for the specified test operation;
- col_inv is one-bit field decoded from operation descriptor, that defines the column LSB inversion for the specified test operation;
- pattern_true is a two-bits field (t_data size) that defines the pattern data in case if equation evaluates true;
- pattern_false is a two-bits field (t_data size) that defines the pattern data in case if equation evaluates false;
- MSB – most significant bit;
- LSB – least significant bit.

MSB					LSB
xor_bit[N-1:0]	not_bit[N-1:0]	row_inv	col_inv	pattern_true[1:0]	pattern_false[1:0]

Figure 10. Pattern descriptor

Operation Decoder Unit consists of the list of predefined test operations (not consuming any storage elements and defined at BIST generation), user-defined test operations (programmable register of predefined format) as well as from operation decoder. For each operation code defined in test algorithm register operation decoder selects one operation from the predefined/user-defined test operations list. The programmable operation register consists of predefined number (defined at generation time) of operation descriptors. For

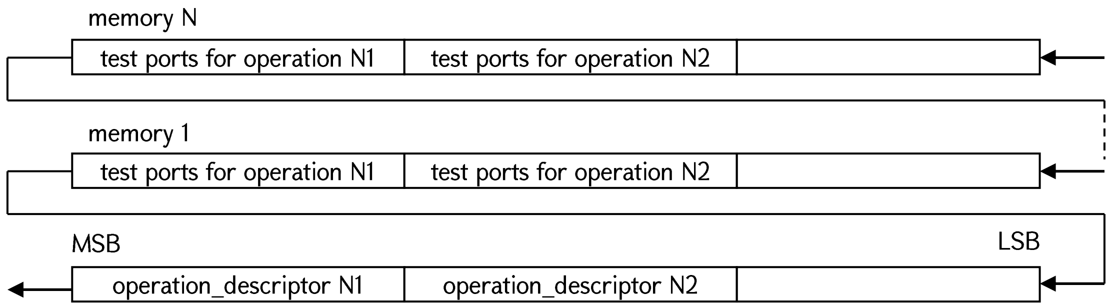

memory N
test ports for operation N1
test ports for operation N2
memory 1
test ports for operation N1
test ports for operation N2
MSB
LSB
operation_descriptor N1
operation_descriptor N2

Figure 11. Test operation chain

multi-port memories two operation descriptors are required per each memory instance, one for direct and the other for shadow ports (see Figure 11).

The format of operation descriptor is shown in Figure 12, where:

- base_operation is a 2-bit field that defines the base operation that must be performed on memory:
  - 00 - No operation;
  - 01 – Read operation;
  - 10 – Write operation;
  - 11 – Reserved.
- data_capt field defines that output data must be captured and compared after operation execution;
- row_inv field defines that row lsb bit in test address must be inverted for selected operation;

- col_inv field defines that col lsb bit in test address must be inverted for selected operation.

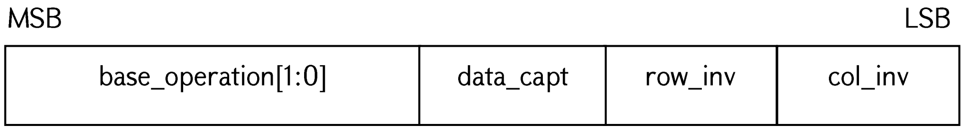


Figure 12. Test operation descriptor

Experiments are done to estimate the modified BIST area overhead against the original BIST. For this purpose, a memory with the following configuration is selected: number of memory words - 32, number of bits per word – 8, column multiplexing - 4. The BIST test algorithm complexity is 8N, where N is the number of memory words. Table 4 shows the obtained results. As it is seen, the modified BIST hardware area increase is ~1%, instead it provides possibility to program test operations and background patterns in the BIST.

Table 4. Test operation programmability results

BIST type	Area (μm^2)	Occupancy (%)
Original BIST without test mechanism programmability	10031.55	100.00
Modified BIST with test mechanism programmability	10132.65	101.01

Existing BIST systems do not guarantee their usage in a new technology without making design changes in the BIST. Usually the existing BIST systems require to be extended in order to consider faults specific to new technologies. Thus, there is a necessity to create a unified BIST architecture which will allow:

- Maximum flexibility for test algorithm programmability;
- To test not only faults of the existing but also of the future technologies.

But in order to build such a BIST architecture first it is needed to make possible the prediction of faults of future technologies. This problem is solved in this work through Fault Periodicity Table (FPT) which allows to predict faults of future technologies and through Test Algorithm Template (TAT) which allows to construct efficient test algorithms for testing those faults.

1.2 Nanoscale Challenges

1.2.1 Evolution of memory devices

Figure 13 shows the evolution chain of nanoscale memory devices. As it is seen from the figure, from 90nm to 28nm nodes 2D transistors were used in the memories and moving from current technology node to next node it was possible to reuse the existing test solutions. The main reason was that during the transition mainly the sizes of the memories were being changed and no structural changes were done. But reuse of those test solutions became impossible for memories of 22nm and smaller technologies since those memories started to use 3D (FinFET) transistors. The evolution chain of memory devices shown in Figure 13 meets Moor's Law. In 1965, Moor predicted that the number of transistors per IC unit will be doubled every 18 months [48]. Interestingly, till today this law is true.

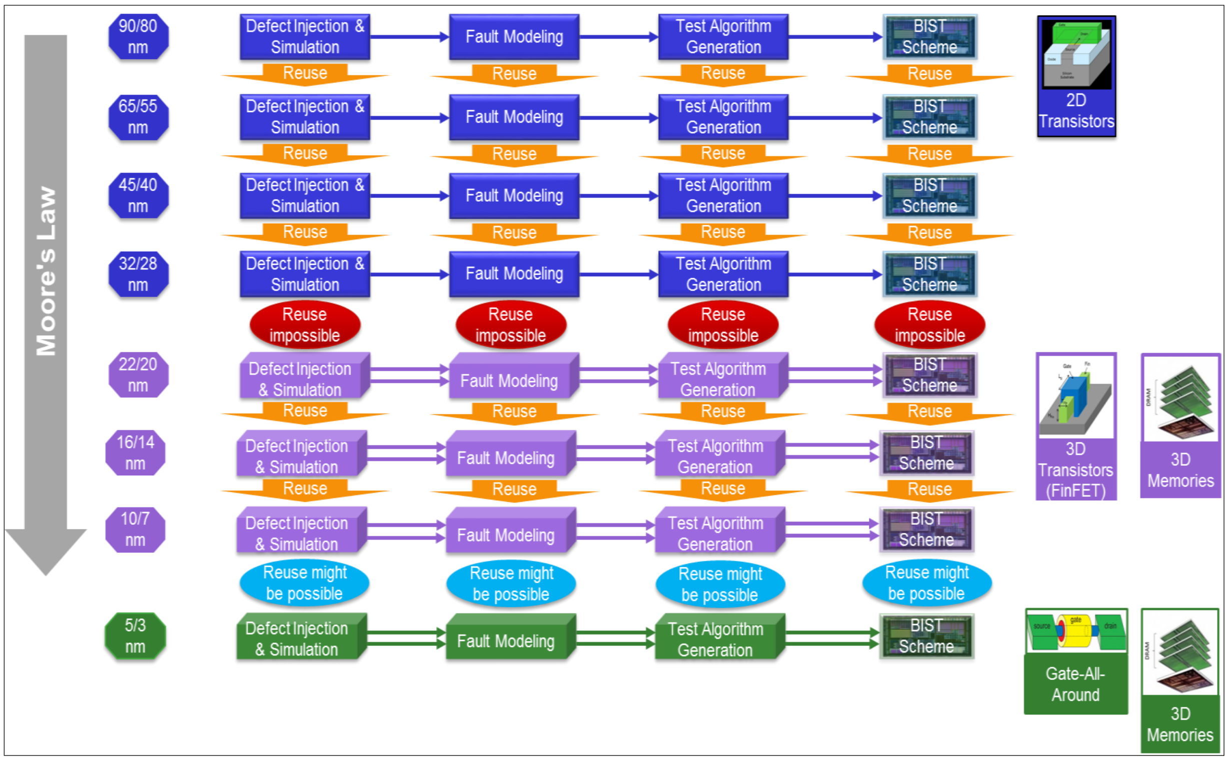


Figure 13. Evolution of nanoscale technology

As it was mentioned above, three new factors (3D transistors, 3D ICs, Hierarchical test approach) have appeared very recently which are playing important role in evolution of nowadays nanoscale memories. Figure 14 shows 3-way evolution of integrated circuits. In addition to evolution of transistors and memories, the complexity of SoCs is also actively evolving. If the initial SoCs contained several memories with one centralized built-in self-test scheme, then current SoCs consist of multiple sub-chips where each sub-chip may contain thousands of memories and multiple built-in self-test schemes.

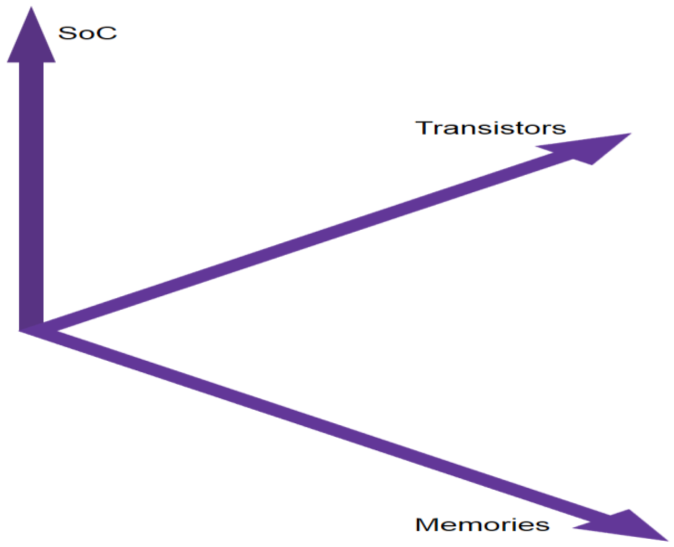


Figure 14. 3-way evolution of integrated circuits

1.2.2 Necessity of a Common Backbone for Various Test Mechanisms

Modern memory built-in self-test (BIST) controllers come either with hardwired standard test algorithm or with a programmability option (see [42], [43], [47], [49]-[51]). In the first case, a user cannot modify or add his/her own test algorithm to the hardwired BIST. Thus, such a test algorithm is typically not optimized for a novel or proprietary memory design; whereas, the programmable option uses an architecture that usually requires an externally accessible register (Test Algorithm Register) of predefined format for storing a micro-program that will perform a given test algorithm. Typically, the programmable BIST controller provides enough flexibility for test algorithm definition. At the same time, the test algorithm is composed of special test mechanisms (test operations, background patterns and addressing methods) which are usually hardwired in the BIST infrastructure. In this context, the programmability of BIST means that only the existing test mechanisms can be used when programming a test algorithm. This limitation specifically impacts the flexibility of test coverage and test time. In some cases, this limitation can even lead to the impossibility of programming the required test algorithm. Afterwards, solutions with programmability of separate test mechanisms are proposed (e.g., [47], [51]). However, all the solutions mentioned above do not have a common backbone which will allow considering these different issues within a unified infrastructure.

1.2.3 Hierarchy of Test Subsystems

Modern nanoscale SoCs besides memories include different IP cores (e.g., analog-mixed signal IPs or digital cores) which need to be tested as well. From the other side, nowadays nanoscale SoCs have become so big and complex that require new solutions for their testing. If in the past in a simple SoC only one test system was used, then in nowadays SoCs which are containing many memories and IP cores, multiple test infrastructures are used.

Figure 15 shows the evolution of the SoC test infrastructure during the last several years. The part (a) of the Figure 15 shows the stage when there was only one BIST scheme per SoC, while in Figure 15 (b) there are multiple BIST schemes and, the test configuration for the SoC has only one Server. This means that though multiple BIST schemes are used but SoC does not require a hierarchy of Servers. The part (c) of the Figure 15 shows modern SoCs with many memories and IP cores as well as containing a hierarchy of Servers, where there is a Server at top and there are Sub-Servers at the second level of hierarchy.

1.3 Problem Statement

Summarizing considerations in previous paragraphs we come to the following. Trends and challenges of growing memory content where tens of thousands embedded memory instances are available in modern SoCs, detection of today's defects upon manufacturing and during life time (such as process variation and FinFET-specific defects for 16-nm and below technology nodes), as well as BIST solutions to address debug, diagnosis and yield optimization insistently need to be addressed in a systematic way as embedded in chip solutions. Moreover, due to a drastic reduction of time when moving from one technology node to the next node it becomes necessary to address the mentioned problems not only ad hoc, but to have mechanisms for prediction of challenges that arise due to increasing variety and complexity of used memories and IP cores, shrinking technologies and design complexity increasing in nanoscale SoCs. These requirements make it crucial to have embedded in SoC test and repair solutions kept up with the advances in order to consistently and continuously provide chip quality and yield optimization.

Given the memory content, covering power management constraints, functional timing implications, test scheduling optimization, and area minimization options, today's automotive safety-critical chips need multiple in-system self-test modes, such as power-on self-test and

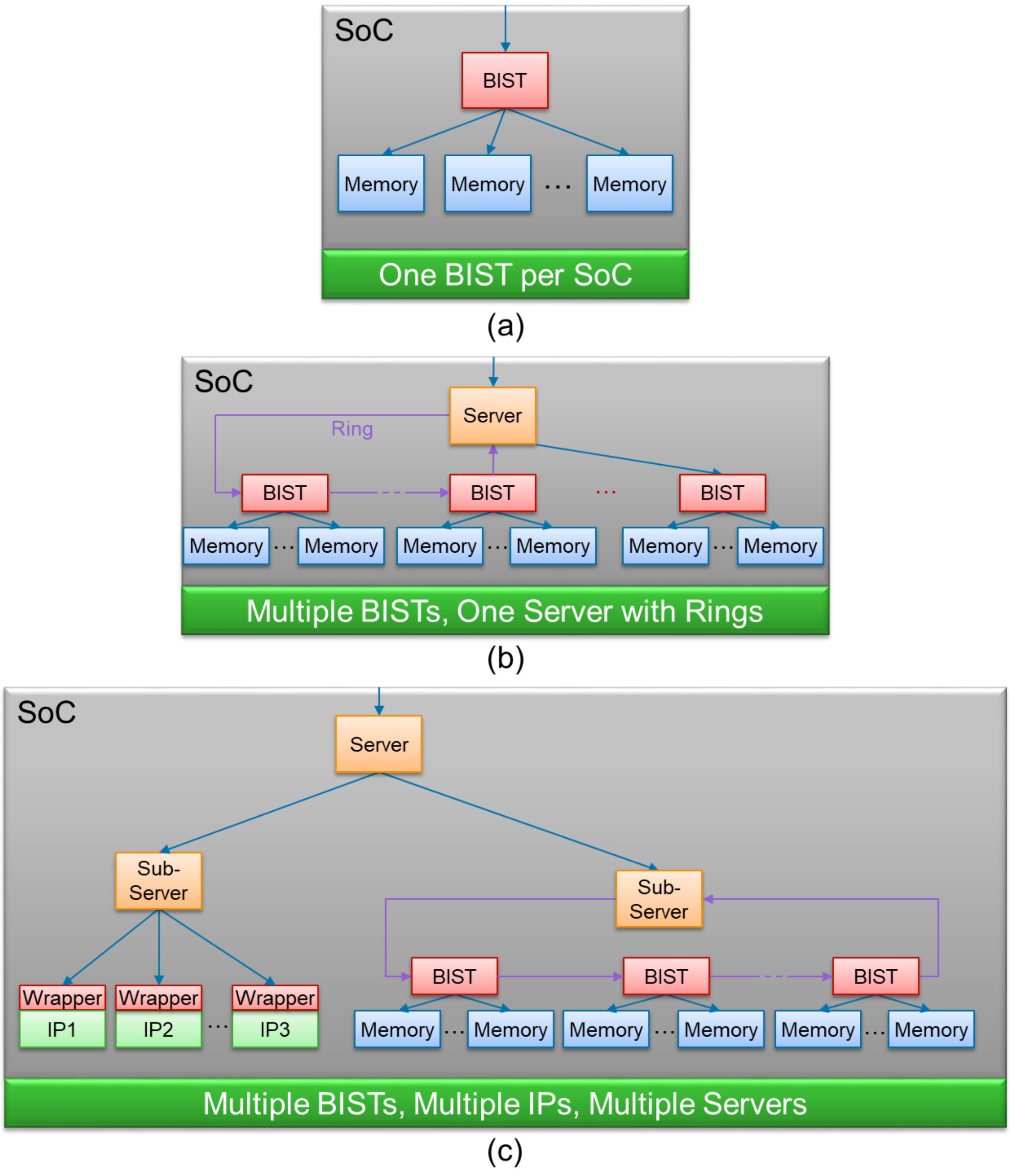


Figure 15. SoC and its evolution

repair, periodic in-field self-test, advanced error correction solutions, etc. At the same time there are certain limitations in SoC which should be considered too when scheduling the test. Common types of SoC limitations are design, test and resource limitations.

Basing on the above these are the two aims to be solved in the current thesis:

1. To provide a basis for evolving development of a self-contained test methodology for memories in nanoscale SoCs that covers not only design and silicon bring-up, but also volume production and in-system test stages. The developed basis should be easily applicable both for the further development of the test methodology and for test solutions in most crucial for the modern lifestyle challenges such as automotive and Internet of Things. Particularly, it should support a possibility to predict fault types and test mechanisms for upcoming technology nodes basing on learned consistent patterns from development of the current technology node.
2. The proposed solution should allow to create a unified hierarchical built-in test architecture as well as it should be flexible enough to be integrated with existing systems and applications. In general, the hierarchical test system will give designers a flexibility to schedule test of memories, individual IP cores and other cores for parallel and serial testing to optimize test time and power consumption during test.

Conclusions

1. Specifics of nanoscale memories as well as faults, test algorithms and main approaches for built-in self-test solutions are described.
2. Evolution of nanoscale technology and 3-way evolution (3D transistors, 3D ICs, Hierarchical test approach) of integrated circuits are presented.
3. It is shown that existing built-in test solutions are not sufficient to solve the test problems of nowadays memory devices. That is why the necessity of having integral test methodology to solve the existing test problems is justified.
4. The problem statement of the work is formulated.

CHAPTER 2. FAULT MODELING AND TEST PATTERN GENERATION FOR NANOSCALE MEMORY DEVICES

2.1 Fault Modeling and Test Pattern Generation for Planar-Based Memory Devices

2.1.1 Main Approaches for Test Algorithm Generation

The main purpose of fault modeling activity is to define the fault models of a given technology in order to create test algorithms for their detection. The existing fault modeling methods mainly are based on injecting models of physical defects into structural model of a memory device. Then, in the presence of a physical defect, a memory model simulation is done to find out the impact of physical defect on the memory device. In the memory, different deviations (e.g., the memory cell value remains 1 even if write 0 operation is applied to that cell) can occur depending on the type and location of a physical defect. Then for each deviation due to which memory starts working incorrectly the corresponding fault model is created.

Different methods for test algorithm generation were proposed in the past (e.g., [52]-[56]). Some of them proposed to perform an exhaustive search of test algorithms starting from shortest test algorithm and then increasing its length. The advantage of such flow is that the found test algorithm is minimal while the main disadvantage of it is the huge run-time (days, months).

The other approach is based on heuristic algorithms, i.e., searching test algorithms non-exhaustively taking into account some conditions extracted from the fault types. The advantage of such flow is the speed, i.e., test algorithms can be found in seconds or milliseconds. But, such a flow has a disadvantage that the found test algorithm can be “inefficient”, i.e., its complexity is far from the complexity of the minimal test algorithm.

2.1.2 Generation of Test Algorithms Based on Necessary and Sufficient Conditions

In [57], necessary and sufficient conditions are proposed for constructing test algorithms for detection of static unlinked faults. This approach allows to avoid modeling faults in a memory model and apply a given test algorithm on the memory model to find out if the test algorithm detects the given set of faults or not. It allows to estimate the fault coverage of a given test algorithm by evaluating its structure. Let us bring a simple example: since DRDF-0 fault is detected by two consecutive Read-0 operations then if a test algorithm contains two consecutive Read-0 operations (e.g., $\Uparrow$(W0, R0, R0) then it will detect the DRDF-0 fault, otherwise (e.g., $\Uparrow$(W0, R0)) it will not detect the fault. Thus, there is no need to model DRDF-0 in a memory model and then run a given test algorithm on the memory model to find out whether the test algorithm detects the fault or not. The same approach is used in [28] to generate test algorithms for detection of two-operation unlinked dynamic faults.

The following propositions and conditions are proposed for static unlinked faults [57].

Proposition 1. In order for a March test M to detect the fault primitives $< 0; R0 / 1 / 0 >$ and $< 1; R0 / 1 / 0 >$ it is necessary for the corresponding March-like sequence $\{M\}$ to contain at least two entries of the sequence {R0, R0}.

A sequence $\{O\} = O_1, O_2, \ldots, O_k$, of Write and / or Read operations will be called a March-like sequence, if O_1 = W0 or W1 and for each j, $1 \leq j \leq k-1$, the following conditions are satisfied:

- If O_j = R0 or W0 then $O_{j+1} \neq$ R1;
- If O_j = R1 or W1 then $O_{j+1} \neq$ R0.

Obviously, from each March test M the corresponding March-like sequence $\{M\}$ can be obtained uniquely. Note that, in general, for any March-like sequence many March test algorithms can be constructed by partitioning the March-like sequence into a finite number of subsequences in different ways and for each subsequence constructing a March element by inserting an address direction for it.

Proposition 2. In order for a March test M to detect the fault primitives < 0; R1 / 0 / 1 > and < 1; R1 / 0 / 1 > it is necessary for the corresponding March-like sequence {M} to contain at least two entries of the sequence {R1, R1}.

Proposition 3. In order for a March test M to detect the fault primitives < 0; 0W0 / 1 / - > and < 1; 0W0 / 1 / - > it is necessary for the corresponding March-like sequence {M} to contain at least two entries of the sequence {W0, [R0]*, W0, R0}.

Here and everywhere below, [X1, ..., Xn]* denotes 0 or more number of the sequence of operations X1, ..., Xn.

Proposition 4. In order for a March test M to detect the fault primitives < 0; 1W1 / 0 / - > and < 1; 1W1 / 0 / - > it is necessary for the corresponding March-like sequence {M} to contain at least two entries of the sequence {W1, [R1]*, W1, R1}.

Proposition 5. In order for a March test M to detect the fault primitives < 0; 0W1 / 0 / - > and < 1; 0W1 / 0 / - > it is necessary for the corresponding March-like sequence {M} to contain at least two entries of the sequence {W0, [R0]*, W1, [W1]*, R1}.

Proposition 6. In order for a March test M to detect the fault primitives < 0; 1W0 / 1 / - > and < 1; 1W0 / 1 / - > it is necessary for the corresponding March-like sequence {M} to contain at least two entries of the sequence {W1, [R1]*, W0, [W0]*, R0}.

Proposition 7. For detection of each of the FPs < 0 / 1 / - >, < R0 / 1 / 1 > and < R0 / 0 / 1 > it is necessary and sufficient for the March-like sequence corresponding to the March test to contain the entry "..., W0, R0, ...".

Here and everywhere below, "..." denotes any number of Rx or/and Wx operations, x ∈ {0,1}, or is empty.

Proposition 8. For detection of each of the FPs < 1 / 0 / - >, < R1 / 0 / 0 > and < R1 / 1 / 0 > it is necessary and sufficient for the March-like sequence corresponding to the March test to contain the entry "..., W1, R1, ...".

Proposition 9. For detection of the FP < 0W1 / 0 / - > it is necessary and sufficient for the March-like sequence corresponding to the March test to contain the entry "..., Op0, ..., W1, R1, ...".

Proposition 10. For detection of the FP < 1W0 / 1 / - > it is necessary and sufficient for the March-like sequence corresponding to the March test to contain the entry "..., Op1, ..., W0, R0, ...".

Proposition 11. For detection of the FP < 0W0 / 1 / - > it is necessary and sufficient for the March-like sequence corresponding to the March test to contain the entries from one of the following cases:

1. "..., R0, W0, R0, ...";
2. "..., Op1, W0, W0, R0, ...".

Proposition 12. For detection of the FP < 1W1 / 0 / - > it is necessary and sufficient for the March-like sequence corresponding to the March test to contain the entries from one of the following cases:

1. "..., R1, W1, R1, ...";
2. "..., Op0, W1, W1, R1, ...".

Proposition 13. For detection of the FP < R0 / 1 / 0 > it is necessary and sufficient for the March-like sequence corresponding to the March test to contain the following entry "..., W0, R0, R0, ...".

Proposition 14. For detection of the FP < R1 / 0 / 1 > it is necessary and sufficient for the March-like sequence corresponding to the March test to contain the following entry "..., W1, R1, R1, ...".

Proposition 15. For detection of each of the FPs $< 0; 0 / 1 / - >_{a < v}$, $< 0; R0 / 1 / 1 >_{a < v}$, $< 0; R0 / 0 / 1 >_{a < v}$ respectively, it is necessary and sufficient for the March test to contain the entries from one of the following cases:

1. $\Uparrow$(..., R0, [..., Op0]*);
2. $\Leftrightarrow$(..., Op0); $\Downarrow$(..., R0, ...).

Proposition 16. For detection of each of the FPs $< 1; 0 / 1 / - >_{a < v}$, $< 1; R0 / 1 / 1 >_{a < v}$, $< 1; R0 / 0 / 1 >_{a < v}$ respectively, it is necessary and sufficient for the March test to contain the entries from one of the following cases:

1. $\Uparrow$(..., R0, ..., Op1);

2. ⇔(..., Op1); ⇓(..., R0, ...).

Proposition 17. For detection of the FP < 0; 0W1 / 0 / - >$_{a<v}$ it is necessary and sufficient for the March test to contain the entries from one of the following cases:

1. ⇑(..., Op0, ..., W1, R1, ..., Op0);
2. ⇔(..., Op0); ⇑(..., W1, R1, ..., Op0);
3. ⇔(..., Op0); ⇓(..., W1); ⇔(R1, ...);
4. ⇔(..., Op0); ⇓(..., W1, R1, ...).

Proposition 18. For detection of the FP < 1; 0W1 / 0 / - >$_{a<v}$ it is necessary and sufficient for the March test to contain the entries from one of the following cases:

1. ⇑(..., Op0, ..., W1); ⇔(R1, ...);
2. ⇑(..., Op0, ..., W1, R1, [..., Op1]*);
3. ⇔(..., Op0); ⇑(..., W1, R1,[..., Op1]*);
4. ⇔(..., Op0); ⇑(..., W1); ⇔(R1, ...);
5. ⇔(..., Op1); ⇓(..., Op0, ..., W1, R1, ...);
6. ⇔(..., Op1); ⇓(..., Op0, ..., W1); ⇔(R1, ...).

Proposition 19. For detection of the FP < R0; 0 / 1 / - >$_{a<v}$ it is necessary and sufficient for the March test to contain the entries from one of the following cases:

1. ⇑(R0, ...);
2. ⇓(..., R0, [..., Op0]*); ⇓(R0, ...).

Proposition 20. For detection of the FP < R0; 1 / 0 / - >a < v it is necessary and sufficient for the March test to contain the entries from one of the following cases:

1. ⇑(R1, ..., R0, ...);
2. ⇓(..., R0, ..., Op1); ⇔(R1, ...).

Proposition 21. For detection of the FP < 0W1; 0 / 1 / - >a < v it is necessary and sufficient for the March test to contain the entries from one of the following cases:

1. ⇑(R0, ..., W1, ...);
2. ⇓(..., Op0, W1, ..., Op0); ⇔(R0, ...);
3. ⇔(..., Op0); ⇓(W1, ..., Op0); ⇔(R0, ...).

Proposition 22. For detection of the FP < 0W1; 1 / 0 / - >$_{a<v}$ it is necessary and sufficient for the March test to contain the entries from one of the following cases:

1. ⇑(R1, ..., Op0, W1, ...);
2. ⇓(..., Op0, W1, [..., Op1]*); ⇔(R1, ...);
3. ⇔(..., Op0); ⇓(W1, [..., Op1]*); ⇔(R1, ...).

Proposition 23. For detection of the FP < 0; 0W0 / 1 / - >$_{a<v}$ it is necessary and sufficient for the March test to contain the entries from one of the following cases:

1. ⇑(..., R0, W0, R0, [..., Op0]*);
2. ⇑(..., R0, W0); ⇔(R0, ...);
3. ⇑(..., Op1, W0, W0, R0, [..., Op0]*);
4. ⇑(..., Op1, W0, W0); ⇔(R0, ...);
5. ⇔(..., Op0); ⇓(..., R0, W0, R0, ...);
6. ⇔(..., Op0); ⇓(..., R0, W0); ⇔(R0, ...);
7. ⇔(..., Op0); ⇓(..., Op1, W0, W0, R0, ...);
8. ⇔(..., Op0); ⇓(..., Op1, W0, W0); ⇔(R0, ...);
9. ⇔(..., Op1); ⇑(W0, W0, R0, [..., Op0]*);
10. ⇔(..., Op1); ⇑(W0, W0); ⇔(R0, ...);
11. ⇔(..., R0); ⇑(W0, R0, [..., Op0]*);
12. ⇔(..., R0); ⇔(W0); ⇔(R0, ...);
13. ⇔(..., Op1, W0); ⇑(W0, R0, [..., Op0]*);
14. ⇔(..., Op1, W0); ⇑(W0); ⇔(R0, ...);
15. ⇔(..., Op1); ⇔(W0); ⇑(W0, R0, [..., Op0]*);
16. ⇔(..., Op1); ⇔(W0); ⇔(W0); ⇔(R0, ...);
17. ⇔(..., R0); ⇓(W0, R0, ...);
18. ⇔(..., Op1, W0); ⇓(W0, R0, ...);
19. ⇔(..., Op1); ⇔(W0); ⇓(W0, R0, ...).

Proposition 24. For detection of the FP < 1; 0W0 / 1 / - >$_{a<v}$ it is necessary and sufficient for the March test to contain the entries from one of the following cases:

1. ⇑(..., R0, W0, R0, ..., Op1);
2. ⇑(..., Op1, W0, W0, R0, ..., Op1);
3. ⇔(..., Op1); ⇑(W0, W0, R0, ..., Op1);
4. ⇔(..., R0); ⇑(W0, R0, ..., Op1);
5. ⇔(..., Op1, W0); ⇑(W0, R0, ..., Op1);
6. ⇔(..., Op1); ⇔(W0); ⇑(W0, R0, ..., Op1);
7. ⇔(..., Op1); ⇓(..., R0, W0, R0, ...);
8. ⇔(..., Op1); ⇓(..., Op1, W0, W0, R0, ...);
9. ⇔(..., Op1); ⇓(W0, W0, R0, ...);
10. ⇔(..., Op1); ⇓(..., R0, W0); ⇔(R0, ...);
11. ⇔(..., Op1); ⇓(..., Op1,W0,W0); ⇔(R0, ...);
12. ⇔(..., Op1); ⇓(W0, W0); ⇔(R0, ...).

Proposition 25. For detection of the FP $< 0; R0 / 1 / 0 >_{a < v}$ it is necessary and sufficient for the March test to contain the entries from one of the following cases:

1. ⇑(..., R0, R0, [..., Op0]*);
2. ⇑(..., R0); ⇔(R0, ...);
3. ⇔(..., Op0); ⇓(..., R0, R0, ...);
4. ⇔(..., Op0); ⇓(..., R0); ⇔(R0, ...).

Proposition 26. For detection of the FP $< 1; R0 / 1 / 0 >_{a < v}$ it is necessary and sufficient for the March test to contain the entries from the following cases:

1. ⇑(..., R0, R0, ..., Op1);
2. ⇔(..., Op1); ⇓(..., R0, R0, ...);
3. ⇔(..., Op1); ⇓(..., R0); ⇔(R0, ...).

Proposition 27. For detection of the FP $< 0W0; 0 / 1 / - >_{a < v}$ it is necessary and sufficient for the March test to contain the entries from one of the following cases:

1. ⇑(R0, [..., Op0]*, W0, ...);
2. ⇓(..., Op0, W0, [..., Op0]*); ⇔(R0, ...);
3. ⇔(..., Op0); ⇓(W0, [..., Op0]*); ⇔(R0, ...).

Proposition 28. For detection of the FP < 0W0; 1 / 0 / - >$_{a < v}$ it is necessary and sufficient for the March test to contain the entries from one of the following cases:

1. ⇑(R1..., Op0, W0, ...);
2. ⇓(..., Op0, W0, ..., Op1); ⇔(R1, ...);
3. ⇔(..., Op0); ⇓(W0, ..., Op1); ⇔(R1, ...).

Based on the proposed propositions a test algorithm generation tool is developed [57], [58]. Using the tool, March MSS1, March MSS2, March MSS3 and March MSS4 minimal test algorithms are generated (see Tables 5-8) for detection of all unlinked static faults.

Table 5. March MSS1 (18N)

Address direction	Operations
⇔	W0
⇑	R0, R0, W1, W1
⇑	R1, R1, W0, W0
⇓	R0, R0, W1, W1
⇓	R1, R1, W0, W0
⇔	R0

Table 6. March MSS2 (18N)

Address direction	Operations
⇔	W0
⇓	R0, R0, W1, W1
⇓	R1, R1, W0, W0
⇑	R0, R0, W1, W1
⇑	R1, R1, W0, W0
⇔	R0

Table 7. March MSS3 (18N)

Address direction	Operations
⇔	W0
⇑	R0, R0, W1, W1
⇓	R1, R1, W0, W0
⇓	R0, R0, W1, W1
⇑	R1, R1, W0, W0
⇔	R0

Table 8. March MSS4 (18N)

Address direction	Operations
⇔	W0
⇓	R0, R0, W1, W1
⇑	R1, R1, W0, W0
⇑	R0, R0, W1, W1
⇓	R1, R1, W0, W0
⇔	R0

2.2 New Challenges in Fault Modeling for FinFET and 3D Memory Technologies

2.2.1 FinFET Technology and Its Specific Faults

Due to growing leakage and short-channel problems of conventional planar MOSFET transistors, it is not possible to continue further scaling down the feature sizes of the planar transistors. FinFET transistors have been introduced as an alternative solution, which has the necessary characteristics to further shrink the technology. The term FinFET was first mentioned as early as in 1999 [59] to describe the non-planar double-gate transistor. It was demonstrated as a possible replacement for conventional planar technology. Later, FinFETs were used in many publications to describe transistors built with new non-planar multi-gate architecture (see [2], [3], [60]-[63]). The distinguishing characteristic of the FinFET transistor is that its conducting channel consists of thin vertical silicon Fins surrounded by gate electrodes. This leads to a better control of the channel and better electrostatic properties, thus diminishing leakage current in the off state. This greatly reduces short-channel effects.

From the structural point of view FinFETs are called tied-gate (TG) or shorted-gate (SG) memories since there is a single gate covering the Fins from 3 sides (see Figure 16). Sometimes in literature independent-gate (IG) memories having two gates in front and back sides of the Fins are also referred as FinFETs (e.g. [64], [65]).

Several of the most important FinFET parameters are: its height (H_{Fin}), its width or body thickness (T_{Fin}), and its channel length (L_g). The effective electrical width of a FinFET is $T_{Fin}+2H_{Fin}$, where T_{Fin} has a fixed size. Thus, in order to increase Fin width, the only way is to use multiple Fins. Due to their 3D structure, FinFETs have several advantages including: controlled fin body thickness, low threshold-voltage variation, reduced variability and lower operating voltage. All of this does help enable designs that can operate faster with less power. Despite the significant power and performance benefits, FinFET design and manufacturing introduce additional challenges. The novelty of the technology and the expected increase in manufacturing cost make it worth taking a closer look at defects that could affect product quality and yield. This is a nontrivial task, given that even memories designed in advance planar technologies can be affected by different defect types. Each of which is categorized as a unique fault model. It is critical to develop a comprehensive yet optimized suite of test algorithms that detect these defects, while keeping test cost low.

Although many of the defect types affecting memories based on planar transistors also affect FinFET-based memories, new defects do occur due to the unique FinFET structure. FinFET-based memories are also less susceptible to certain defects that would cause failures in planar-based memories. Given these differences, the fault models and detection techniques developed for planar transistors are not sufficient to cover FinFET defects in embedded memories [66].

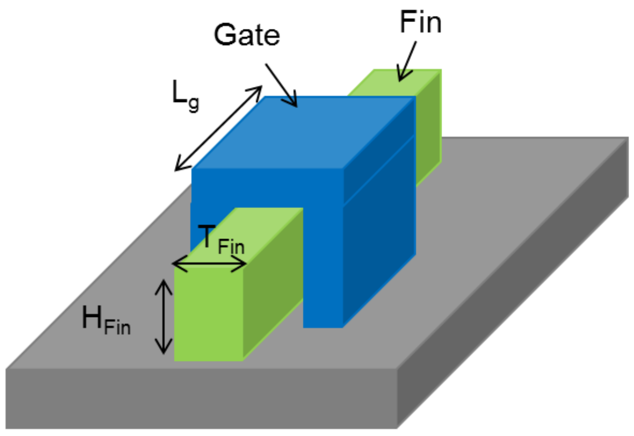


Figure 16. FinFET structure

With production of FinFET-based memories (memories using FinFET transistors), the problem of embedded memory test and repair is critical, as the fault models and test algorithms used for conventional memories may not cover the whole aspect of possible defects in FinFET-based memories. Despite the importance of the problem, relatively a small number of research studies have been conducted in this area during the recent years.

In [67] and [68], the authors have investigated different types of open and short defects in FinFET logic circuits. They showed that even though most of the opens and shorts have their corresponding fault models in planar technology, an open defect on the back gate for IG memories causes delay and leakage problems unique to FinFETs. The same problem arises when TG FinFET is accidentally etched into IG structure.

In [69], the authors examined stuck-open faults (SOF) for small nanometer technologies, including FinFET, and showed that the hold time for SOFs decreases significantly due to increased sub-threshold leakage and gate leakage making the fault detection more difficult. To improve the SOF detection, a test vector strategy is proposed trying to produce lower values at the transistor drain-source voltages of the fan-out gates. The problem of testing SOF faults in small nanometer technologies is also discussed in [70] and two new vector strategies were proposed to increase the possibility of detection of SOF defects.

In [64], the authors examined stuck-open, stuck-on and gate oxide short defects on different number of Fins within one FinFET transistor. According to the results, when the number of defective Fins is relatively small in proportion, then the transistor can be treated as fault-free. Otherwise, if the number is large enough, the defect can be modeled with stuck-open or delay faults. The authors also have investigated the case when a single defect, such as a back gate open or Fin stuck-on, affects multiple gates in IG FinFETs. This type of defect brings to a delay fault, which cannot be detected with traditional delay tests and new test mechanisms are necessary.

In [65], the authors investigate Gate Oxide Shorts (GOS) in FinFETs. According to the authors, GOS defects in FinFET-based memories lead to a more complex fault behavior than in the case of planar-based memories (memories using planar transistors), so traditional test algorithms fail to detect them. For this purpose, two new test techniques were introduced for IG and TG types of FinFETs, in order to detect GOS defects.

Most of the discussed papers show that the existing fault models are not sufficient to represent all possible types of defects in FinFETs. Each of them concentrates on a specific class of defects considering possible testing solutions but, to the best of our knowledge, there is no comprehensive study of FinFET-specific defects and generic test solution for their detection.

Taking into account all these facts a comprehensive study was conducted to investigate the effect of moving to FinFET technology on test and repair aspects of memories in terms of new defect types, fault models, recommended test algorithms and conditions and so forth (see [71]-[75]). For this purpose, an automated flow is developed for SPICE (Simulation Program with Integrated Circuit Emphasis) simulation of FinFET-specific defects that are injected into memory layout or in memory SPICE net-list. Based on simulation results, the corresponding FinFET-specifc faults are modeled (see Table 9) [8] and for their detection a minimal test algorithm is developed (see Table 28). The proposed methodology is validated on existing real FinFET-based memories taken from different foundries.

Table 9. FinFET-specific faults

FFM	FP	FFM	FP	FFM	FP	FFM	FP
dRDF1	<R1R1/0/0>	dRDF0	<R0R0/1/1>	dDRDF1	<R1R1/0/1>	dDRDF0	<R0R0/1/0>
dRDF1-3	<$R1^3$/0/0>	dRDF0-3	<$R0^3$/1/1>	dDRDF1-3	<$R1^3$/0/1>	dDRDF0-3	<$R0^3$/1/0>
dRDF1-4	<$R1^4$/0/0>	dRDF0-4	<$R0^4$/1/1>	dDRDF1-4	<$R1^4$/0/1>	dDRDF0-4	<$R0^4$/1/0>
dRDF1-5	<$R1^5$/0/0>	dRDF0-5	<$R0^5$/1/1>	dDRDF1-5	<$R1^5$/0/1>	dDRDF0-5	<$R0^5$/1/0>
dRDF1-6	<$R1^6$/0/0>	dRDF0-6	<$R0^6$/1/1>	dDRDF1-6	<$R1^6$/0/1>	dDRDF0-6	<$R0^6$/1/0>
dRDF1-7	<$R1^7$/0/0>	dRDF0-7	<$R0^7$/1/1>	dDRDF1-7	<$R1^7$/0/1>	dDRDF0-7	<$R0^7$/1/0>
dRDF1-8	<$R1^8$/0/0>	dRDF0-8	<$R0^8$/1/1>	dDRDF1-8	<$R1^8$/0/1>	dDRDF0-8	<$R0^8$/1/0>

Figure 17 shows the defect types which were considered for TG FinFET:

(a) Fin Open – Full and resistive open defects on Fin;

(b) Gate Open – Full and resistive open defects on Gate;

(c) Fin Stuck-On – Full and resistive short defects between Source and Drain;

(d) Gate-Fin Short – Full and resistive short defects between Gate and Fin;

(e) Fin-VDD/VSS Short – Full and resistive short defects between Fin and VDD or Fin and VSS.

(f) Process Variation – Variations in FinFET parameter values.

In addition to the defects described in Figure 17, other types of defects that are typical to planar-based memories were also considered for FinFET-based memories as well. The most important classes of such defects which have been investigated include resistive/full shorts and opens placed on/between memory bit-lines, word-lines and memory cell internal nodes, as well as defects of the surrounding blocks of memory array, such as address decoder, write driver and sense amplifier.

As a next step, for the obtained set of FinFET-specific faults efficient test conditions for fault detection have been examined. Those test conditions include but are not limited to frequency, temperature, voltage.

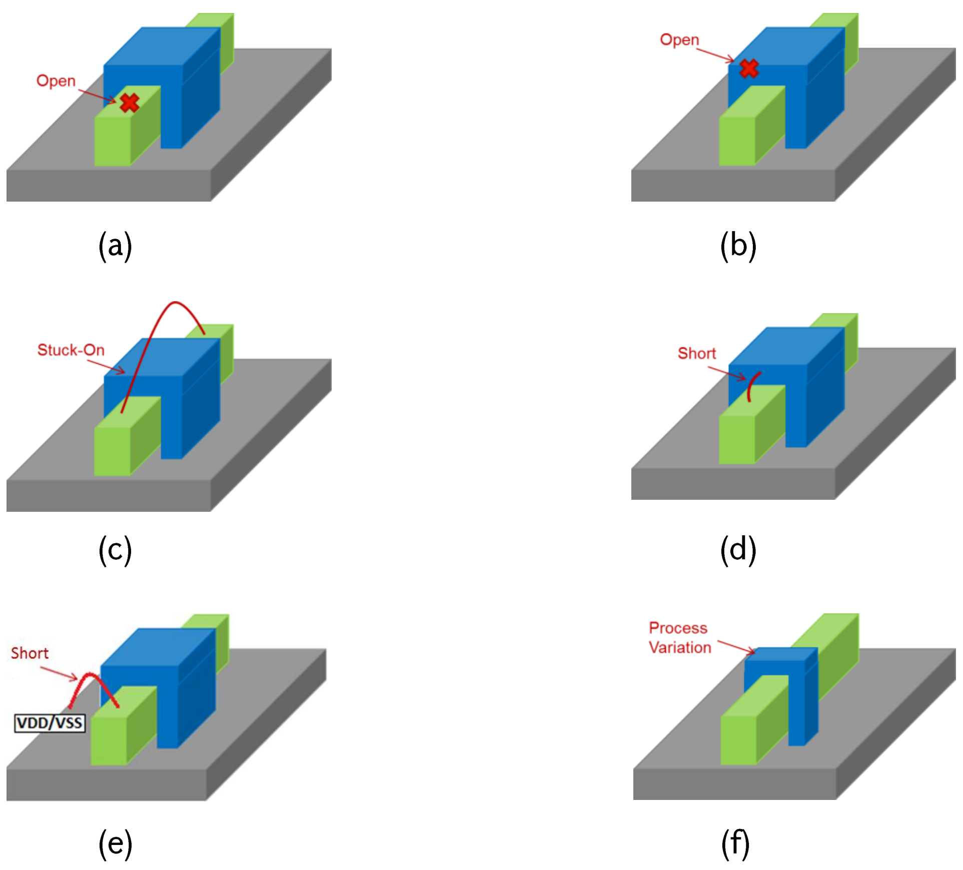


Figure 17. Defect models considered for FinFETs

As it was discussed the transistors are evolving from planar to FinFET and very recently a new candidate for next generation transistors is introduced called Gate-All-Around (GAA) [4]. In GAA transistor the gate covers the transistor channel on all sides (see Figure 18).

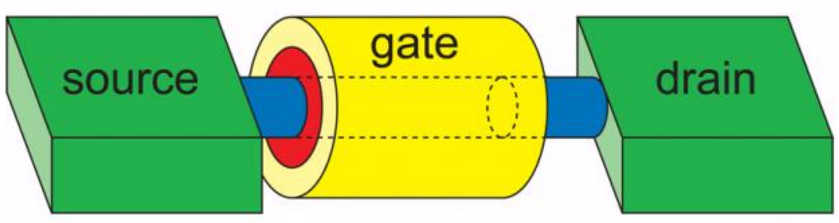


Figure 18. Gate-All-Around (GAA) transistor

2.2.2 3D Memory Technology and Its Specific Faults

From a structural viewpoint, an external memory linked to SoC via high speed I/Os is typically composed of one or more memory dies/chips that interact with SoC using high bandwidth. Testing an external memory alongside with high speed interconnects was always a challenging task. It became even more challenging with the increased usage of 2.5D and 3D packages.

Effective test, diagnosis and repair schemes for 3D packages remain among the key factors for their successful deployment. One of the most popular compositions of 3D packages is the one comprised of a logic die along with one or more stackable external memory dies that communicate with the logic die via Through-Silicon Vias (TSV) interconnects (see [76]-[78]). Of course, in addition to testing individual logic and memory dies stand-alone, each die and interconnect need to be tested as a stack. In case of memory dies, beside the memory array, interconnects and TSVs should also be tested and diagnosed. For 3D external memory stacking, some industry standards have already been developed (e.g., JEDEC Standard for Wide I/Os) and some test mechanisms were proposed [78].

There are many published papers (see [79]-[84]) addressing defects, corresponding faults, and strategies for testing external memories, including stackable memory dies in 2.5D and 3D packages. Some papers suggest test approaches for pre-bond testing (e.g., [79], [80]) or post-bond testing (e.g., [81]-[83]), while others provide architectural and built-in self-test solutions for memory dies in 2.5D and 3D packages [77]-[78].

The most common 3D ICs are 3D Memories which structure is given in Figure 19.

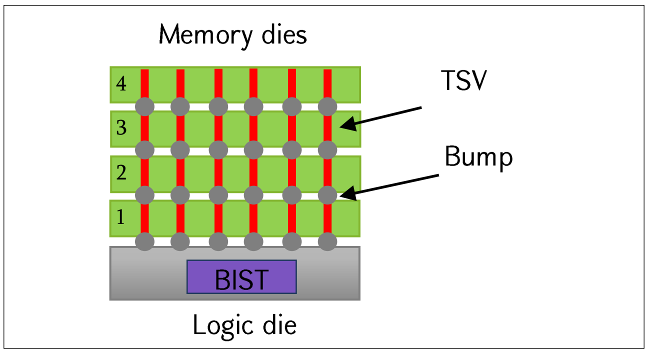


Figure 19. 3D Memory

In the figure four Memory dies and one Logic die are stacked together. Signals from one die to another are passed through TSVs. TSVs of different dies are connected to each other through Bumps.

For the memory array static and dynamic single-cell and two-cell (coupling) faults are considered while for interconnects stuck-at faults, transition faults and bridge faults are considered. Based on discussed fault types and the place where they can appear in external memories the fault space is divided into the following 4 groups: interconnect single-line faults, interconnect two-line (bridge) faults, array single-cell faults and array two-cell faults.

Interconnect single-line faults: <f; d1; d2; i>:

- f – Fault type;
- d1, d2 – Fault is between dies d1 and d2. If d1=d2, then the fault is considered as an intra-die fault, otherwise it is an inter-die fault. If d1 or d2 is equal to 0 then it means that the logic die is considered;
- i – Fault occurred at interconnect i.

Example: <SA0; 1; 1; WE> means that there is a SA0 fault at WE (Write Enable) interconnect on Memory die 1.

Interconnect two-line faults: <f; d1; d2; i1; i2>:

- f – Fault type;
- d1, d2 – Fault is between dies d1 and d2. If d1=d2, then the fault is considered as an intra-die fault, otherwise it is an inter-die fault. If d1 or d2 is equal to 0 then it means that the logic die is considered;
- i1, i2 – Fault occurred between interconnects i1 and i2.

Example: <Wired-AND; 0; 1; Addr[4]; Addr[5]> means that there is a Wired-AND bridge fault between Addr[4] and Addr[5] interconnects occurred between Logic die and Memory die 1.

Array single-cell faults: <f; d; a; b>:

- f – Fault type;
- d – Memory die number;

- a – Faulty address;
- b – Faulty data bit.

Example: <WDF0; 2; 7; 4> means that there is a Write destructive fault WDF0 on Memory die 2 with faulty address 7 and faulty data bit 4.

Array two-cell faults: <f; d; a1; b1; a2; b2>:

- f – Fault type;
- d – Memory die number;
- a1 – Aggressor address;
- b1 – Aggressor data bit;
- a2 – Victim address;
- b2 – Victim data bit.

Example: <CFst-00; 3; 7; 4; 7; 5> means that there is a State coupling fault <0; 1 / 0 / -> on Memory die 3 with aggressor address 7, aggressor data bit 4 and victim address 7, victim data bit 5.

2.3 A New Method for Fault Modeling and Test Algorithm Generation

2.3.1 Fault Modeling and Test Algorithm Synthesis Flow

Since FinFET-based memories may contain new faults that are specific only to FinFETs, the existing test solutions (consisting of various types of test operations, addressing methods, background patterns and other stressing conditions) may be insufficient for investigation of those faults. Thus, the known techniques, such as running SPICE simulations with predefined set of test algorithms may fail to detect FinFET-specific faults. This is the main reason that a new methodology was developed (see [71], [72], [74]) for investigation of FinFET-specific faults and for assisting to model new faults which were not possible to do by the existing known techniques. Figure 20 shows an automated flow for finding appropriate test sequences for detection of defects, as well as for construction of the corresponding fault models. The flow consists of the following steps:

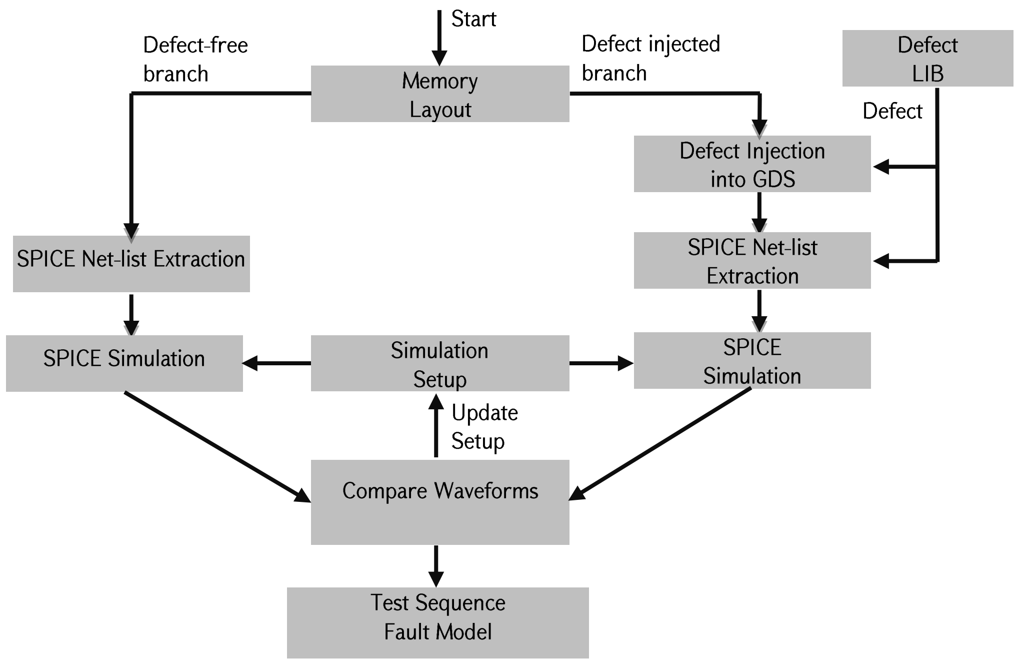


Figure 20. Test Sequence Identification and Fault Modeling Flow

1. A defect can be injected either in GDS (Graphic Database System) or in SPICE Net-list [85]. In some cases, it is worth or convenient to inject a defect into GDS and for other cases SPICE Net-list is more preferable. Thus, two choices are provided for defect injection. The defects are injected from Defect LIB which is enriched periodically based on new technological structures, such as FinFETs. At the moment, it includes all defect models described in Figure 17.

2. Two SPICE Simulations (defect-free and defect injected) are run with specific Simulation Setup which can contain a set of different test sequences, different test conditions (frequency, voltage, temperature), if a resistive defect is injected then the range of resistance magnitude, etc. Based on the setup if multiple simulations are needed, then all simulations are performed automatically and for each simulation PASS/FAIL information and Waveforms of applied test operations are provided.

3. If FAIL is obtained for defective SPICE Net-list then it means that current Simulation Setup and therefore at least one of the used Test Sequences is satisfactory to detect the injected defect. Otherwise, if PASS is obtained for all simulations meaning that the defect is not detected then comparison of Waveforms for defect-free and defect injected cases should be done. Based on the comparison results new test sequences should be provided

which can be candidates for detecting the defect. This part is done by the user (test engineer or someone else) following some special rules to construct new Test Sequences. For example, for this purpose frequently Fault Periodic Table (see [8]-[11]) is used that helps to find appropriate Test Sequences using a systematic approach instead of guessing Test Sequences which, in some sense, can lead to multiple useless and endless iterations.

4. The iteration mentioned in Step 3 should be repeated until finding a satisfactory Test Sequence. If a Test Sequence is found, then the corresponding Fault Model is automatically extracted from the Test Sequence. For example, if detecting Test Sequence = {W0, R0, R0} and only the second R0 detects the fault then DRDF0 = <0R0/1/0> fault [17] is extracted.

When fault models and the corresponding test sequences are identified the next step is to construct a test algorithm for detection of a given set of faults. There are several tools for test algorithm generation (e.g., [55], [86]) that take as an input a set of faults and generate a satisfactory test algorithm. In most of the cases they require to be enhanced each time when new types of faults appear (such as FinFET-specific faults). It means that test algorithm generation tool strongly depends on the set of faults that currently are known by the tool. In order to make the tool independent of the fault type, it is decided to take as an input not a set of faults, but a set of Test Sequences. The advantage of this is that the flow becomes more generic since there is no dependency on fault types. Also, since per the flow described in Figure 20, Test Sequence is identified by the simulation while Fault Model is obtained based on Test Sequence it is more efficient to use the direct output (Test Sequence) in Test Algorithm Synthesis Flow (see Figure 21) instead of using Fault Model that is derived from Test Sequence. In Figure 21, Test Algorithm Generator synthesizes optimal Test Algorithms. Moreover, the experiments show that if the given Test Sequences have minimal lengths in terms of detecting the given defects/fault models, then Test Algorithm Generator will synthesize minimal test algorithms.

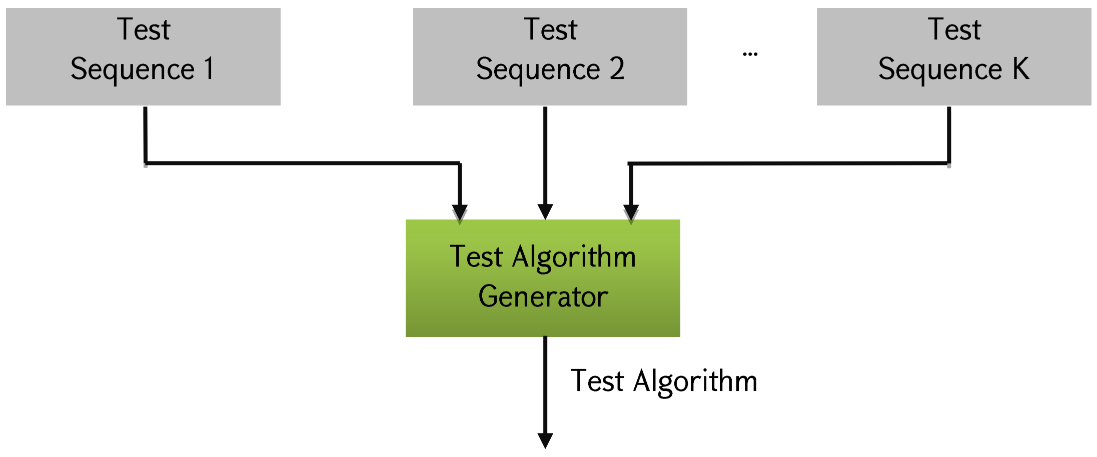


Figure 21. Test Algorithm Synthesis Flow

2.3.2 Experimental Results

In the first phase of the conducted study a massive run of SPICE simulations was performed on several FinFET-based memories from different foundries. The simulations were done for all considered defects using the fault modeling and test algorithm synthesis flows. Each defect was injected into pass-gate (PG), pull-down (PD) and pull-up (PU) transistors one at a time and moreover multiple defects were injected simultaneously in the same memory cell. In parallel to this activity the same defects were injected into planar 28nm and 45nm memories in order to compare the obtained results. Based on the results of the experiments the following observations can be made:

1. FinFET-based memories are more prone to dynamic faults than planar-based memories. Dynamic faults are observed when special types of defects are injected into FinFET-based memories while for the same defects in planar-based memories only static fault behavior is observed.
2. FinFET-based memories are more stable to process variation faults. Changes in transistor parameter sizes (up to 50%) do not lead to any fault in FinFET-based memories while 20%-40% parameter size change in planar-based memories lead to different types of faults [87].
3. Static single-cell and coupling faults are typical for both FinFET- and planar-based memories. Some defects resulting in static faults (either single-cell or coupling) are

injected and similar behavior is observed in both FinFET- and planar-based memories.

The below figures present some of the results obtained by running SPICE simulations done on FinFET-based memories. They are obtained for different types of defects injected in different places in the memories using different test conditions.

Figure 22 shows the simulation output for the case when a resistive Fin Open defect is injected into a pull-down transistor of a FinFET-based memory cell. It results in seven-operation dynamic Deceptive Read Destructive Fault dDRDF0-7 [71] where the 7th R0 operation is flipping the content of the cell without reporting a mismatch while the 8th R0 operation detects the fault. So Test Sequence = {W0, R0, R0, R0, R0, R0, R0, R0, R0} and the corresponding fault model is dDRDF0-7 = <0R0^7/1/0>. This type of dynamic fault was observed for wide range of resistance values 10-25MΩ.

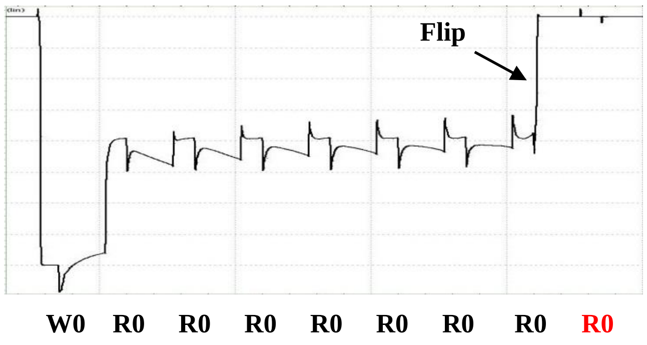


Figure 22. Resistive Fin Open defect in PD transistor results in dDRDF fault

Since there is no Fin in planar-based memories and therefore Fin Open defect cannot be injected, in planar-based memories Channel open defect is examined to obtain a behavior closer to Fin Open defect. A simulation showed that in this case only static fault behavior is observed for all resistance values from [0;∞] range.

Figure 23 shows the result of the case when a resistive Gate Open defect is injected into a pass-gate transistor of a FinFET-based memory cell. It results in a well-known transition fault i.e., Test Sequence = {W1, W0, R0} or {W1, W0, W0, R0} and the corresponding fault model is transition fault TF0 = <1W0/1/->. This fault is observed for resistance values ≥5MΩ. The same results are obtained when the defect is injected into planar-based memories. Similar results are obtained also for the cases when Gate Open defect is injected into pull-down and pull-up transistors, thus it can be considered that Gate Open defects do not lead to FinFET-specific faults.

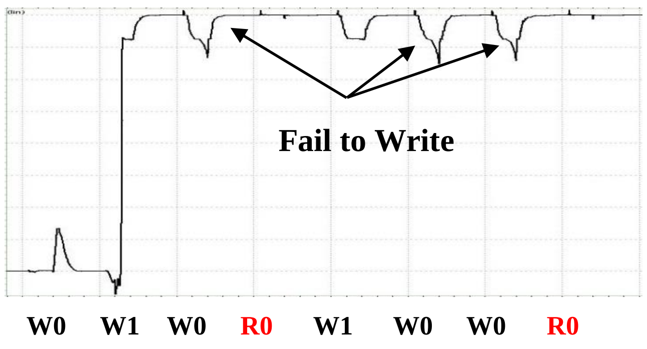


Figure 23. Gate Open defect in PG transistor results in TF fault

Figure 24 shows the result in the case when a resistive Fin Stuck-On defect is injected into a pull-down transistor of a FinFET-based memory cell. It results in four-operation dynamic Deceptive Read Destructive Fault dDRDF1-4 where the 4th R1 operation is flipping the content of the cell without reporting a mismatch while the 5th R1 operation detects the fault. So the optimized Test Sequence = {W1, R1, R1, R1, R1, R1} and the corresponding fault model is dDRDF1-4 = <1R1^4/0/1>. This fault was observed for the resistance range of 120-130KΩ. This example shows that in some cases the satisfactory Test Sequence can contain redundant test operations and those should be eliminated while reporting Test Sequence or extracting the corresponding Fault Model. In this case first two operations (W0

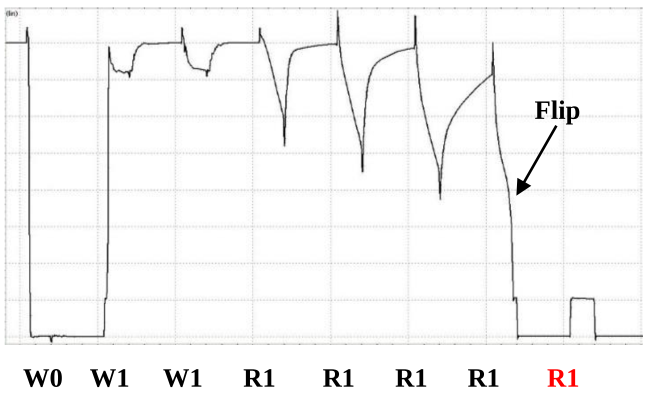


Figure 24. Fin Stuck-On defect in PD transistor results in dDRDF fault

and first W1) can be removed. Again, since there are no Fins in planar-based memories for comparison of the results Channel Stuck-On defect is injected into planar-based memories. As a result, only static fault behavior is observed for all resistance values from [0;∞] range.

Figure 25 shows the result in the case when a resistive Gate-Fin Short defect is injected into a pass-gate transistor of a memory cell. It results in linked fault TF1 = <0W1/0/-> * IRF = <xRx/0/1>, x ∈ {0, 1}, where IRF stands for Incorrect Read Fault (see [17], [32]). It means that W1 operation fails to write value 1 to a faulty cell when the initial value of the cell is 0.

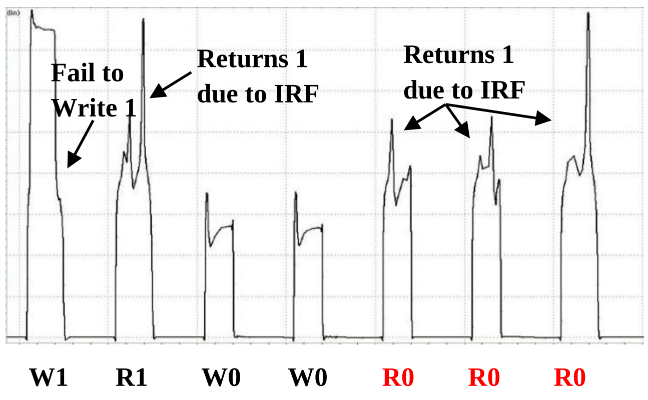


Figure 25. Gate-Fin Short defect in PG transistor results in linked TF*IRF fault

Next, R1 operation returns correct value due to an IRF fault which means that fault is masked at this point. Then by applying R0 operation the fault is detected. So here more than one Test Sequences are used in order to extract a correct fault model though for detection only one Test Sequence is enough. For such cases, if there is a doubt that a fault is linked or some other complex fault, additional module in the flow is enabled to consider more than one Test Sequences for fault model extraction. This fault is observed for resistance values ≤ 17KΩ. In planar-based memories Gate-Channel Short defect is injected and only TF1 = <0W1/0/-> fault is observed for all resistance values from [0;∞] range.

Figure 26 shows the simulation results in case of resistive Fin-VSS short defect injected into a pull-down transistor. As can be seen in figure it also leads to dynamic fault, particularly, dynamic Read Destructive Fault dRDF0-6 where the 6th R0 operation is flipping the content of the cell and reporting a mismatch. So Test Sequence = {W0, R0, R0, R0, R0, R0, R0} and the corresponding fault model is dRDF0-6 = <0R0^6/1/1>.

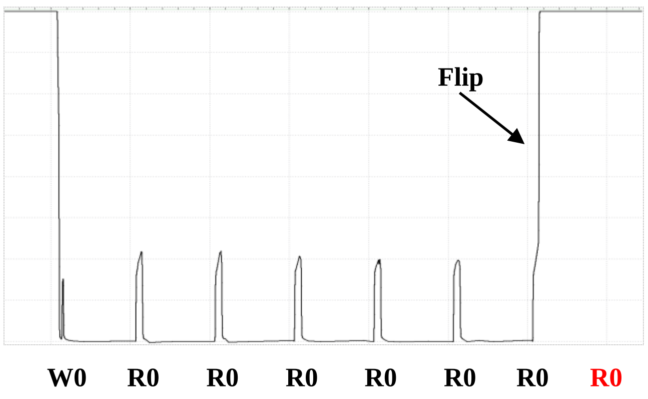


Figure 26. Fin-VSS Short defect in PD transistor results in dDRF fault

The above mentioned cases are some examples of a huge number (more than 1000) of SPICE simulations done on different types of FinFET-based and planar-based memories from different foundries. In addition to FinFET-specific defects, the other types of defects that are typical to planar-based memories are injected into FinFET-based memories and in most cases the same or similar results are obtained as in the case when the defect is injected into planar-based memory. Most of these defects lead to static single-cell and static coupling faults.

Above it was already stated three major conclusions that are based on the simulation results. In addition to that some interesting facts are outlined below:

- Defects injected in pull-up transistors of FinFET- and planar-based memories in most of the cases give the same results.
- dDRDF is observed mainly in pull-down and linked fault TF*IRF in pass-gate transistors of a FinFET-based memory. These faults are not watched in planar-based memories. Though papers [18], [88] state that dDRDF faults were obtained for 0.13um memories, it was not observed in 28nm and 45nm memories.
- Gate open defects in FinFET-based memories do not lead to FinFET-specific faults.
- Test conditions (frequency, voltage, temperature) are playing important role for defect detection since for a given range of a resistive defect nominal test condition may not reveal the defect while running the same Test Sequence with other test condition (e.g., high frequency, low voltage, high temperature) will result in

detecting the defect.

Fault coverage for a defect can be different depending on the selected test condition, i.e., it directly affects the efficiency of the selected test algorithm used in memory test. The fault coverage here is considered in terms of the range of resistance values, for which the defect is detected. This is especially true with regard to the family of special dynamic faults, which were confirmed above to be FinFET-specific faults. Several test conditions, such as voltage, temperature and frequency, were chosen and the level of their influence was investigated.

The influence of voltage, temperature and frequency was examined for different corner values (min, max and nominal) [72], [73]. Tables 10-12 present the experimental results, which were obtained for Fin Open defect. As it can be seen from Table 10 in nominal case (Temperature=25oC, VDD=0.8V) the received fault model is dDRDF0-3, which implies that four consecutive read operations are needed to detect the fault. With temperature increase the fault model changes towards RDF (Read Destructive Fault) meaning that number of read operations needed to detect the fault is decreased to one. On the other hand, with temperature decrease number of consecutive read operations, needed to detect the fault,

Table 10. Temperature Impact on Dynamic Faults

Temperature/VDD	Fault
T = 25, V = 0.8	dDRDF0-3
T = 26, V = 0.8	dDRDF0
T = 27, V = 0.8	dDRDF0
T = 28, V = 0.8	DRDF0
T = 29, V = 0.8	DRDF0
T = 125, V = 0.8	RDF0
T = 24, V = 0.8	dDRDF0-3
T = 23, V = 0.8	dDRDF0-4
T = 22, V = 0.8	dDRDF0-9
T = 21, V = 0.8	dDRDF0-16
T = 20, V = 0.8	No Fault

increases up to the point when the fault is not detected anymore. Similar inverse correlation was observed also in case of voltage change. Table 11 shows that with the voltage increase number of consecutive read operations decreases and when decreasing the voltage, the fault is not detected anymore. Finally, Table 12 shows the results for frequency analysis, where Fin Open defect was injected in Pull Down and Pull Up transistors. It can be seen from the table that in both cases again inverse correlation was observed between frequency and number of consecutive read operations needed to detect the fault.

Table 11. Voltage Impact on Dynamic Faults

Temperature/VDD	Fault
T = 25, V = 0.8	dDRDF0-3
T = 25, V = 0.81	DRDF0
T = 25, V = 0.82	DRDF0
T = 25, V = 0.83	DRDF0
T = 25, V = 0.84	DRDF0
T = 25, V = 0.85	RDF0
T = 25, V = 0.79	No Fault
T = 25, V = 0.78	No Fault

The experiments showed that the best corner value of test conditions vary for different defects. For example, as considered above for Fin Open defect the best fault coverage (maximum covered resistance range) was observed at maximum temperature, maximum voltage and maximum frequency while for Fin Stuck-On defect the best one is minimal temperature, minimal voltage and maximal frequency.

Table 12. Frequency Impact on Dynamic Faults

Defective Transistor	Frequency	Fault
Pull Down	400 MHz	dDRDF0-7
	800 MHz	dDRDF0-5
	1200 MHz	dDRDF0-3
Pull Up	400 MHz	dDRDF0-17
	800 MHz	dDRDF0-4
	1200 MHz	dDRDF0-3

Based on the experiments results, the following observations can be highlighted for dRDF/dDRDF faults:

- In case of open defects: Temperature increase (decrease) leads to decrease (increase) in number of consecutive read operations needed to sensitize dRDF/dDRDF faults.
- In case of short defects: Temperature increase (decrease) leads to increase (decrease) in number of consecutive read operations needed to sensitize dRDF/dDRDF faults.
- In case of open defects: Voltage increase (decrease) leads to decrease (increase) in number of consecutive read operations needed to sensitize dRDF/dDRDF faults.
- In case of short defects: Voltage increase (decrease) leads to increase (decrease) in number of consecutive read operations needed to sensitize dRDF/dDRDF faults.
- In case of open/shorts defects: Frequency increase (decrease) leads to decrease (increase) in number of consecutive read operations needed to sensitize dRDF/dDRDF faults.

To summarize, the observations showed that all corner values of test conditions (minimum and maximum values) are equally important in order to make sure that for each individual defect the highest possible coverage is obtained.

2.3.3 Structure-oriented method for generation of March test algorithms

The experiments showed that the existing test algorithm generation methods are not applicable for generating a March test algorithm for detection of all 2-composite static and dynamic faults. The reason is that the exhaustive search to detect a minimal March test algorithm is time-consuming, while the existing heuristic methods either cannot generate the required March test algorithm (since they support only static faults or only unlinked static and dynamic faults, etc.) or generate inefficient March test algorithms.

The structure-oriented method proposed in this work is based on searching of symmetric March test algorithms, where the same March element is usually repeated with opposite addressing orders or with opposite background patterns. For example, in March C-: {M_0:

$\Leftrightarrow$(W0); M_1: $\Uparrow$(R0, W1); M_2: $\Uparrow$(R1, W0); M_3: $\Downarrow$(R0, W1); M_4: $\Downarrow$(R1, W0); M_5: $\Leftrightarrow$(R0)}, March elements M_1 and M_3 have the same sequence of operations (with the same background patterns) but opposite addressing orders, while March elements M_1 and M_2 have the same addressing order, the same sequence of operations but opposite background patterns.

The idea of the proposed method is the following: since usually for detection of a special set of faults most of the well-known March test algorithms have a symmetric structure, then during the search only a specific class of symmetric March test algorithms will be considered. In this case instead of searching the whole March test algorithm, it will be enough to search only the unique instances of March elements.

Let us assume that $S(x, y) = O_1, O_2, \ldots, O_k$, where $O_j \in \{R0, R1, W0, W1\}$, $1 \leq j \leq k$. The sequence $S(x, y)$ satisfies the following conditions:

- If O_j = W0 or R0, then $O_{j+1} \neq$ R1;
- If O_j = W1 or R1, then $O_{j+1} \neq$ R0.
- O_1 = Rx;
- O_k = Wy or Ry.

Let us consider the following two structures of March test algorithms:

MT_1 = $\Leftrightarrow$(W0); $\Uparrow$(S_1); $\Uparrow$($\sim S_1$); [$\Leftrightarrow$(R0);] $\Downarrow$(S_2); $\Downarrow$($\sim S_2$); $\Leftrightarrow$(R0)

MT_2 = $\Leftrightarrow$(W0); $\Uparrow$(S_1); $\Uparrow$($\sim S_2$); [$\Leftrightarrow$(R0);] $\Downarrow$(S_2); $\Downarrow$($\sim S_1$); $\Leftrightarrow$(R0)

where:

- $S_1, S_2 \in S(0, 1)$;
- [$\Leftrightarrow$(R0);] – means that the March test can contain this March element optionally.

For example March MSS1 can be obtained from MT_1 or MT_2, if S_1=S_2={R0, R0, W1, W1} and the optional March element $\Leftrightarrow$(R0) is absent.

Figure 27 describes the proposed method for March test algorithm generation. The flow starts to construct sequences S_1 and S_2 (of length L_1 and L_2) from class S(0, 1). Note that {R0, W1} is the shortest element of class S(0, 1). Each time next S_1 or S_2 is constructed from class S(0, 1). If there is no other sequence of length L_1 (or L_2) to generate, then the L_1 (respectively, L_2) is increased by 1 and a new sequence of length L_1 (respectively, L_2) is constructed. L_{MAX} is a user specified value, which limits the complexity of the searched March test by number $2*L_{MAX}+3$ (see structures MT_1 and MT_2). If the flow responds "No solution", then the reason is the small value of L_{MAX}. This means that L_{MAX} should be increased and run the flow again.

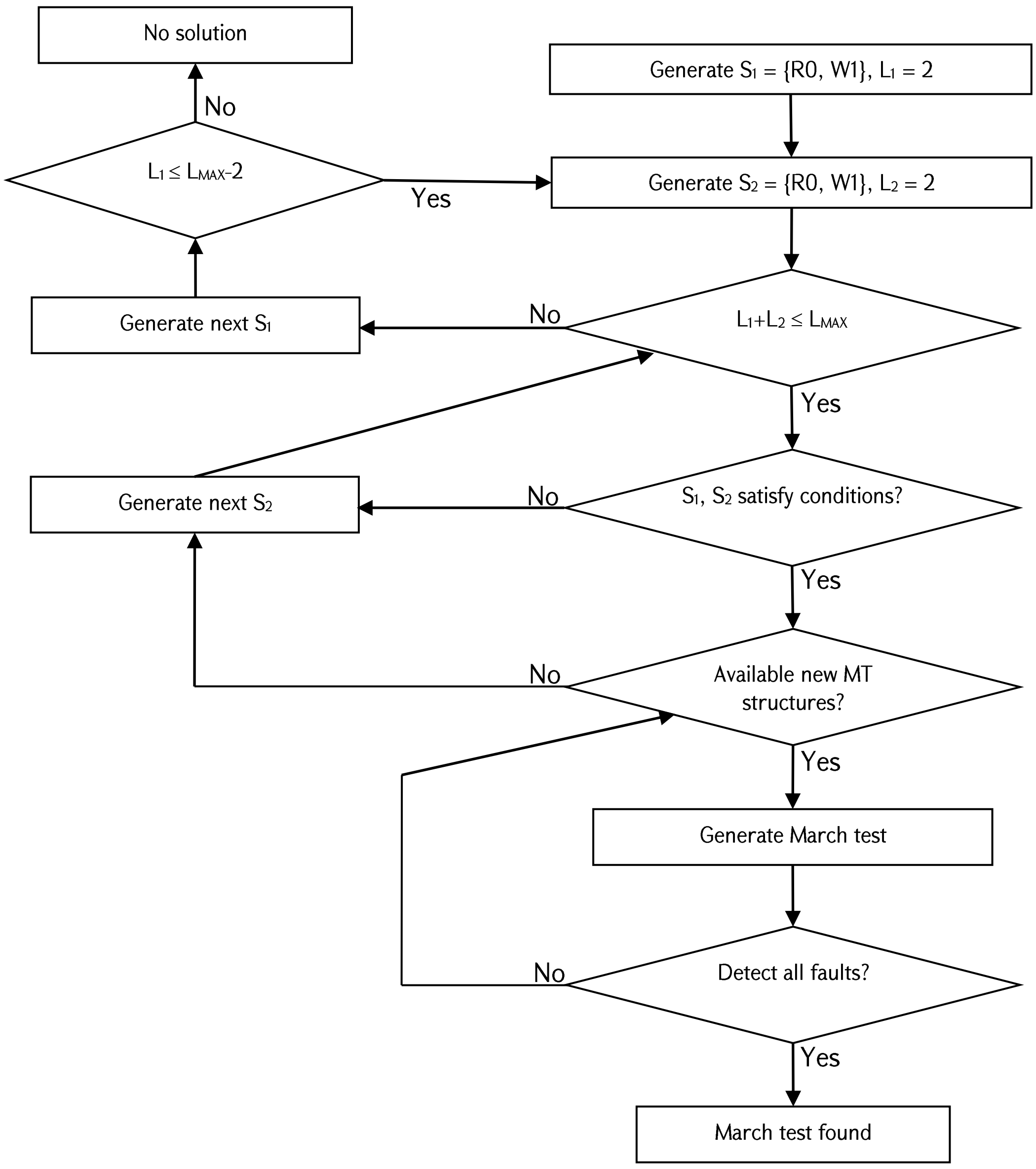


Figure 27. March test algorithm generation method

In [28], [32], [57], several necessary and/or sufficient conditions are described for static and dynamic fault detection. Using these conditions, one can conclude that the given March test algorithm does not detect the given set of faults without running the March test algorithm on the memory model with injected faults (e.g., if a March test does not contain two sequential R0 operations then it does not detect DRDF fault). This speeds up the March test algorithm generation flow essentially. Thus, a March test algorithm is generated only if S_1 and S_2 satisfy the given conditions (see Figure 27). The generated March test algorithm is run on the memory model with injected faults to find out whether it detects all the considered faults or not. The whole space of 2-composite static and dynamic faults is taken into account. If the current March test algorithm detects all the considered faults, then it is considered that the March test algorithm is found, otherwise the next March test algorithm is generated. If structure MT_1 is fully used, then structure MT_2 can be used for March test algorithm generation. Only the structures MT_1 and MT_2 are considered since they give a satisfactory solution for the problem considered in this work. Besides MT_1 and MT_2, other structures of March test algorithms also can be used in the proposed method.

Table 13 shows the March test algorithms obtained by applying the proposed method. Test algorithms March MSS1 and March MD2 are minimal correspondingly for detection of all unlinked static and all unlinked dynamic faults. The minimal test algorithm for detection of all linked and unlinked static faults is March MSL of complexity 23N [32], while a new test algorithm March SL24 of complexity 24N is obtained. Finally, for detection of all linked and unlinked static and dynamic faults, test algorithm March LSD is obtained. From Table 13, it can be noted that it took only 198 seconds to generate March LSD. Currently it is unknown whether March LSD is a minimal March test algorithm or not with respect to the considered faults.

Table 14 shows the corresponding operations for sensitizing and detecting for some of 2-composite faults (since the number of the considered faults is very large to include all those in the table). If a fault is detected more than once, then the first detection scenario is presented in the table. If the sensitized FP is a static fault, then the column "Sensitization" contains one operation and if the FP is a dynamic fault, then accordingly there are two operations. In Table 14, $M_i(j)$ denotes j^{th} operation of i^{th} March element, $i \geq 0$, $j \geq 0$.

Table 13. March test algorithms generated by the proposed method

Faults	March test algorithm	Complexity	Minimal	Structure	Run time

All unlinked static faults	March MSS1 [57]:
	⇔(W0);
	⇑(R0, R0, W1, W1); ⇑(R1, R1, W0, W0);
	⇓(R0, R0, W1, W1); ⇓(R1, R1, W0, W0);
	⇔(R0)
		18N	Yes	MT1	0.12s

All linked and unlinked static faults	March SL24 [86]:
	⇑(W0); ⇑(R0, W1, W0, W0, R0, W1, R1);
	⇑(R1, W0, W0, R0); ⇓(R0, W1, W1, R1);
	⇓(R1, W0, W1, W1, R1, W0, R0); ⇓(R0)
		24N	No	MT2	0.74s

All unlinked dynamic faults	March MD2 [28]:
	⇔(W0);
	⇑(R0, W1, W1, R1, W1, W1, R1, W0, W0, R0, W0, W0, R0, W0, W1, W0, W1);
	⇑(R1, W0, W0, R0, W0, W0, R0, W1, W1, R1, W1, W1, R1, W1, W0, W1, W0);
	⇓(R0, W1, R1, W1, R1, R1, R1, W0, R0, W0, R0, R0, R0, W0, W1, W0, W1);
	⇓(R1, W0, R0, W0, R0, R0, R0, W1, R1, W1, R1, R1, R1, W1, W0, W1, W0);
	⇔(R0)
		70N	Yes	MT1	15s

All linked and unlinked static and dynamic faults	March LSD [86]:
	⇔(W0);
	⇑(R0, W1, R1, W1, W1, R1, W1, W0, R0, W1, W1, R1, W0, W1, R1, W1, R1, R1);
	⇑(R1, W1, W1, R1, W1, W0, R0, W1, W1, R1, W0, W1, R1, W1, R1, R1, R1, W0); ⇑(R0);
	⇓(R0, W0, W0, R0, W0, W1, R1, W0, W0, R0, W1, W0, R0, W0, R0, R0, R0, W1);
	⇓(R1, W0, R0, W0, W0, R0, W0, W1, R1, W0, W0, R0, W1, W0, R0, W0, R0, R0);
	⇓(R0)
		75N	Unknown	MT2	198s

Table 14. Some of 2-composite faults detected by test algorithm March LSD

#	Fault	Sensitization	Detection
1	Unlinked static fault: <0W1; 0/1/-> (a<v)	$M_1(1)$	$M_1(0)$
2	Unlinked dynamic fault: <1: 0W0R0/1/0> (v<a)	$M_4(13)$, $M_4(14)$	$M_4(15)$
3	LF1: <1W1/0/->*<0W1/0/->	$M_1(1)$	$M_1(2)$
4	LF2av: <0W1R1; 0/1/->*<0W1W0/1/-> (a<v)	$M_1(1)$, $M_1(2)$	$M_1(0)$
5	LF2av: <0W1R1; 0/1/->*<0W1W0/1/-> (v<a)	$M_2(11)$, $M_2(12)$	$M_3(0)$
6	LF2aa: <0R0R0; 1/0/->*<1: 1R1R1/0/1> (a<v)	$M_1(16)$, $M_1(17)$	$M_2(0)$
7	LF2aa: <0R0R0; 1/0/->*<1: 1R1R1/0/1> (v<a)	$M_2(14)$, $M_2(15)$	$M_2(16)$
8	LF3: <0W1R1; 1/0/->*<1R1R1; 0/1/-> ($a_1<a_2<v$)	$M_1(16)$, $M_1(17)$	$M_1(0)$
9	LF3: <0W1R1; 1/0/->*<1R1R1; 0/1/-> ($a_2<a_1<v$)	$M_2(11)$, $M_2(12)$	$M_2(0)$
10	LF3: <1W1; 0/1/->*<0: 0W1W1/0/-> ($a_1<v<a_2$)	$M_1(3)$	$M_1(0)$
11	LF3: <1W1; 0/1/->*<0: 0W1W1/0/-> ($a_2<v<a_1$)	$M_2(7)$, $M_2(8)$	$M_2(9)$
12	LF3: <0R0R0; 0/1/->*<0R0R0; 1/0/-> ($v<a_1<a_2$)	$M_4(14)$, $M_4(15)$	$M_4(0)$
13	LF3: <0R0R0; 0/1/->*<0R0R0; 1/0/-> ($v<a_2<a_1$)	$M_5(16)$, $M_5(17)$	$M_5(0)$

Let us consider fault LF3: <0W1R1; 1/0/->*<1R1R1; 0/1/-> ($a_1<a_2<v$) (see Table 14, #8). This fault is a LF3 fault, where "a_1" is the aggressor cell of FP <0W1R1; 1/0/->, "a_2" is the aggressor cell of FP <1R1R1; 0/1/-> and "v" is the victim cell for both FPs. The fault is sensitized by two sequential operations {R1, R1} which correspond to operations {$M_1(16)$, $M_1(17)$} in March LSD. The fault is detected by the first operation (operation R0) of March element M_1, i.e., $M_1(0)$.

Table 14 shows also the sensitization and detection scenario for the same fault but with the given order of the aggressor and victim cells $a_2<a_1<v$ (see #9). It is seen that these faults are sensitized and detected by different operations. This example shows that the order of the aggressor-victim cells of a fault is essential and all the possible orders of the aggressor-victim cells should be considered. The experiments show that the proposed method is an efficient method since the generated March test algorithms are either minimal or very close to the minimal one, as well as the generation times are acceptable (e.g., generation time for March LSD is 198 seconds). Since the generation of a test algorithm is a one-time task, then it is assumed that 198s is an acceptable time (even one day, or one week can be acceptable).

Thus, the conclusion is that the method allows to easily generate complex March test algorithms with symmetric structures.

In Table 15, test algorithm March LSD is compared with other well-known March test algorithms. As it is seen from the table, only March LSD covers 100% of all the considered faults. March LSD is an efficient March test algorithm since its complexity is more than the complexity of March MD2 only by 5N, while additionally March LSD detects all linked static*dynamic and dynamic*dynamic faults (as mentioned above the number of only the linked dynamic*dynamic FPs is 12459). Complete comparison with other methods is impossible since no one of those has 100% fault coverage with respect to the considered faults.

Though this approach is considered optimal, anyway for complex test algorithms (e.g., for test algorithms of complexity more than 100N or 200N) it will require big amount of time to generate test algorithms (since it still contains in it an exhaustion). Hence there is a need to think of a method which is not based on generating/searching test algorithms, but rather constructing those test algorithms based on some rules. Test Algorithm Template (TAT) proposed in Section 3.5 solves this problem since it allows to construct test algorithms without doing any generation or search operation.

Table 15. Comparison of test algorithm March LSD with other March test algorithms

March Test algorithm	Complexity	Unlinked static faults	Linked static*static faults	Unlinked dynamic faults	Linked static*dynamic faults	Linked dynamic*dynamic faults
March C- [14]	10N	-	-	-	-	-
March MSS1 [57]	18N	+	-	-	-	-
March SS [15]	22N	+	-	-	-	-
March AB [89]	22N	+	-	-	-	-
March MSL [32]	23N	+	+	-	-	-
March SL [17]	41N	+	+	-	-	-
March MD2 [28]	70N	+	+	+	-	-
March LSD [86]	75N	+	+	+	+	+

Conclusions

1. Main approaches and existing methods for fault modeling and test algorithm generation are presented. It is shown that exhaustive and heuristic methods for test algorithms generation are not applicable for nowadays nanoscale memory devices.
2. Challenges connected to FinFET and 3D memory technologies and faults specific to those technologies are introduced. New fault models, their justification and efficient methods for their simulation are proposed.
3. A new structure-oriented method for generation of test algorithms is proposed which is able to generate minimal and efficient test algorithms for various classes of static and dynamic faults.

CHAPTER 3. FAULT PREDICTION MECHANISM FOR MEMORY DEVICES

3.1 Faults, Fault Groups, Fault Families

In this work, a new simple notation is proposed for describing the faults. It is obtained from FP=<S/F/R> in the following way:

- Separate from S the cell initial value and the sensitizing sequence of test operations (SSTO). Example: if S=1W0R0, then the initial value is 1 and SSTO is W0R0;
- F and R are not used since they can be easily obtained from S (See Notation 1).

Notation 1. (x_v, S) denotes a single-cell fault, where S=OP_1D_1, …, OP_nD_n, n≥0, is a SSTO applied to the victim cell, when the victim cell value is x_v, $OP_i \in \{W, R\}$, $D_i \in \{0, 1\}$.

- if S ≠ ∅, the victim cell value becomes ~D_n after fault sensitization. "~" means the opposite value;
- if S = ∅, the victim cell value becomes ~x_v after fault sensitization.

The corresponding values for F and R (from FP=<S/F/R>) are obtained by S in the following way:

- if S ≠ ∅, F = ~D_n and R = D_n if OP_n = R and R = - if OP_n = W;
- if S = ∅, F = ~x_v and R = -.

With this notation, some faults are not considered, like RDF (Read destructive faults) and IRF (Incorrect read faults) since these faults are detectable by March tests that detect SF (State faults) and DRDF (Deceptive read destructive faults) [57]. Note that SF and DRDF can be described by the proposed notation.

The idea here is that there are certain fault types that from fault detection point of view are redundant and hence we suggest to describe only non-redundant faults. Let us consider the following example with the below four faults:

1. SF-0 when cell state is always 0, so Read 1 operation will return value 0;
2. RDF-1 when after Read 1 operation the cell value flips to 0 and returns that incorrect value 0;

3. IRF-1 when after Read 1 operation the cell value remains 1 but it returns incorrect value 0.
4. DRDF-1 when after Read 1 operation the cell value flips to 0 but it returns correct value 1. So another Read 1 operation is needed to detect this fault.

As it is described, for the first three faults Read 1 operation (which means apply Read operation and expect to receive value 1) will detect the given faults since in each case after Read 1 operation value 0 will be returned. Thus, instead of describing faults SF-0, RDF-1 and IRF-1 separately (which will require the third R field from <S/F/R> notation to describe all those faults) here we suggest an optimized notation with two fields which allows to describe only the representatives from each such faults set. Usually when considering fault coverage or when creating fault simulators, only one representative of each such faults set is considered. Anyway, it is obvious that test algorithms generated with the proposed approach will cover not only the representative fault of a given faults set but the whole faults from the set.

Also, since usually test algorithms are developed for a set of faults and not just for a separate fault, then usually from the considered faults set are excluded not only the ones that are equivalent from fault detection point of view, but also those that can be detected by test algorithms covering more complex faults from the set. In this particular example, only DRDF-1 can be considered by excluding from the list the other three faults, since each test algorithm detecting DRDF-1 detects also SF-0, RDF-1 and IRF-1 [57].

The same way, considering two-cell static coupling faults, with this notation CFrd (Coupling read destructive faults) and CFir (Coupling incorrect read faults) are not considered since they are detectable by March tests that detect CFst (State coupling faults) and CFdrd (Coupling deceptive read destructive faults) [57]. Similarly, the CFst and CFdrd can be described by the Notation 2.

Notation 2. (x_a, x_v, S_v) denotes a two-cell (coupling) fault, where S_v=OP_1D_1, …, OP_nD_n, $n \geq 0$, is a SSTO applied to the victim cell, when the victim cell value is x_v and the aggressor cell value is x_a. After fault sensitization, the victim cell value becomes $\sim D_n$, if $S \neq \emptyset$, otherwise it becomes $\sim x_v$.

Notation 3. (x_a, S_a, x_v) denotes a two-cell (coupling) fault, where $S_a = OP_1D_1, \ldots, OP_nD_n$, $n \geq 0$, is a SSTO applied to the aggressor cell, when the aggressor cell value is x_a, and the victim cell value is x_v. After fault sensitization, the victim cell value becomes $\sim x_v$.

Component R from <S/F/R> is applicable when last operation of S is Read operation, i.e., $OP_n = R$ (see Notation 1). This means that there are the following possible fault primitives using <S/F/R> notation:

- $F_1 = <x_vOP_1D_1, \ldots, OP_nD_n /\sim D_n/\sim D_n>$
- $F_2 = <x_vOP_1D_1, \ldots, OP_nD_n /D_n/\sim D_n>$
- $F_3 = <x_vOP_1D_1, \ldots, OP_nD_n /\sim D_n/D_n>$

while (x_v, S) describes only fault F_3 (according to definition of Notation 1: if $S \neq \emptyset$, $F = \sim D_n$ and $R = D_n$ if $OP_n = R$).

Proposition 29. Any test algorithm detecting fault F_3 will detect also faults F_1 and F_2.

Proof. F_3 is sensitized when $OP_1D_1, \ldots, OP_nD_n$ sequence is applied to the faulty cell which has value x_v after which the cell value flips (becomes $\sim D_n$) while it returns correct value D_n. This means that in order to detect the fault an additional Read operation is required to be applied to the faulty cell since at that point the fault is sensitized but not detected yet. From the other side, for faults F_1 and F_2 there is no need to apply the above mentioned additional Read operation since OP_n which is itself Read operation will return the incorrect $\sim D_n$ value and the fault will be detected. This means that any test algorithm detecting fault F_3 will detect also faults F_1 and F_2. The same observations can be done for two-cell faults described using $<S_a; S_v/F/R>$ notation.

This is the main reason why it is suggested to have optimized and simplified notation for description of memory faults, i.e., no need to describe F_1 and F_2 faults since from fault detection point of view those are covered by F_3.

Definition 1. FG(x, S) is a fault group that contains all single-cell and two-cell faults that are sensitized by SSTO S applied to a cell containing value x: FG(x, S)={(x, S), (y, x, S), (x, S, y)}, $x, y \in \{0, 1\}$, $S = OP_1D_1, \ldots, OP_nD_n$, $n \geq 0$.

FG(x, S) is meant to describe most common faults of nanoscale memory devices. In addition, $FG_i(x, S)$ is a fault sub-group which contains all i-cell faults that are sensitized by SSTO S applied to a cell containing value x.

3.2 Fault and Test Algorithm Symmetry

3.2.1 Symmetric Faults

Let us denote:

$F_1 = (x_1, \{OP_{11}D_{11}, \ldots, OP_{1k1}D_{1k1}\})$,

$F_2 = (x_2, \{OP_{21}D_{21}, \ldots, OP_{2k2}D_{2k2}\})$,

$F_3 = (y_1, x_1, \{OP_{11}D_{11}, \ldots, OP_{1k1}D_{1k1}\})$,

$F_4 = (y_2, x_2, \{OP_{21}D_{21}, \ldots, OP_{2k2}D_{2k2}\})$,

$F_5 = (y_1, \{OP_{11}D_{11}, \ldots, OP_{1k1}D_{1k1}\}, x_1)$,

$F_6 = (y_2, \{OP_{21}D_{21}, \ldots, OP_{2k2}D_{2k2}\}, x_2)$.

Definition 2. A pair of faults F_1 and F_2 (respectively, F_3 and F_4, F_5 and F_6) is called a pair of symmetric faults (or symmetric faults) and denoted by $F_1 \leftrightarrow F_2$ (respectively, $F_3 \leftrightarrow F_4$, $F_5 \leftrightarrow F_6$), if it satisfies the following conditions:

- $k_1 = k_2$, $x_1 = \sim x_2$, $y_1 = \sim y_2$;
- $OP_{11} = OP_{21}, \ldots, OP_{1k1} = OP_{2k2}$;
- $D_{11} = \sim D_{21}, \ldots, D_{1k1} = \sim D_{2k2}$.

Examples: $(0, \varnothing) \leftrightarrow (1, \varnothing)$, $(0, 1, W0R0) \leftrightarrow (1, 0, W1R1)$.

Let us denote $FG(x, S) \leftrightarrow FG(\sim x, \sim S)$ and call them as a pair of symmetric fault groups (or symmetric fault groups). This means that:

$\forall F \in FG(x, S), \exists G \in FG(\sim x, \sim S)$, that $F \leftrightarrow G$ and $\forall F \in FG(\sim x, \sim S), \exists G \in FG(x, S)$, that $F \leftrightarrow G$.

3.2.2 Test Algorithm Symmetry

Definition 3. A pair of March elements $M_1=A_1(O_{11}D_{11}, \ldots, O_{1n1}D_{1n1})$ and $M_2=A_2(O_{21}D_{21}, \ldots, O_{2n2}D_{2n2})$ is called a pair of symmetric March elements and denoted by $M_1 \leftrightarrow M_2$, if it satisfies the following conditions:

- $n_1=n_2$;
- $OP_{11}=OP_{21}, \ldots, OP_{1n1}=OP_{2n2}$;
- $(D_{11}=D_{21}, \ldots, D_{1k1}=D_{2k2})$ OR $(D_{11}=\sim D_{21}, \ldots, D_{1k1}=\sim D_{2k2})$.

Definition 4. March test $M=M_1; M_2; \ldots; M_k$ is called symmetric if it satisfies one of the following conditions:

Condition A. March test M consists only of pairs of symmetric March elements. The pairs of symmetric March elements are not necessarily adjacent in the $M_1; M_2; \ldots; M_k$ sequence.

Condition B. $M_1=\Leftrightarrow(WD_1)$ and $M'=M_2; \ldots; M_k$ satisfies Condition A.

Condition C. $M_k=\Leftrightarrow(RD_k)$ and $M'=M_1; \ldots; M_{k-1}$ satisfies Condition A.

Condition D. $M_1=\Leftrightarrow(WD_1)$, $M_k=\Leftrightarrow(RD_k)$ and $M'=M_2; \ldots; M_{k-1}$ satisfies Condition A.

Four symmetric March tests are presented below. Each of them satisfies one of the conditions described in Definition 4.

- $\Uparrow$(W0, R0); $\Uparrow$(W1, R1) - satisfies Condition A.
- $\Leftrightarrow$(W0); $\Uparrow$(R0, W1); $\Downarrow$(R1, W0) (MATS+ [14]) - satisfies Condition B.
- $\Uparrow$(W0, R0, W1); $\Uparrow$(W1, R1, W0); $\Leftrightarrow$(R0) - satisfies Condition C.
- $\Leftrightarrow$(W0); $\Uparrow$(R0, W1); $\Uparrow$(R1, W0); $\Downarrow$(R0, W1); $\Downarrow$(R1, W0); $\Leftrightarrow$(R0) (March C- [14]) - satisfies Condition D.

Definition 5. A March test is called partial-symmetric if it is not a symmetric March test but contains at least one pair of symmetric March elements. For example, the well-known March B test algorithm [14] is a partial-symmetric March test. March B: $\Leftrightarrow$(W0); $\Uparrow$(R0, W1, R1, W0, R0, W1); $\Uparrow$(R1, W0, W1); $\Downarrow$(R1, W0, W1, W0); $\Downarrow$(R0, W1, W0), where March elements $\Uparrow$(R1, W0, W1) and $\Downarrow$(R0, W1, W0) are symmetric.

3.2.3 Canonical View for Test Algorithms

Let us define a new representation of test algorithms. It represents all similar fragments of a test algorithm in a standard form comprised of a basic sequence of March elements and a sequence of its transformations for each March element. This view was called "canonical view" and denoted by M_{CV}: $M_{CV} = M_1, \ldots, M_k$, where $M_i=\{\{\ldots\{A_1(O_{11}, \ldots, O_{1t1}), \ldots, A_d(O_{d1}, \ldots, O_{dtd})\}_{C1}\},\ldots, _{Cp}\}$, $A_t \in \{\Uparrow, \Downarrow, \Leftrightarrow\}$, $O_{j1j2} \in \{R0, R1, W0, W1\}$, $C_1, \ldots, C_p$ - sequences of transformation operations which can be:

- inv_data – invert the data polarity, for example from 00000000 to 11111111;
- inv_dir – invert current addressing direction, for example from $\Uparrow$ to $\Downarrow$;
- change_patt – change current data background pattern, for example from Solid background pattern (all 0s) to Checkerboard (first row has 01010101 data, second row – 10101010, third row - 01010101 data, forth row – 10101010, etc.).

This means that if two consecutive parts in a test algorithm have the same structure but are different only by data backgrounds, addressing directions, data polarities and addressing modes, then only the first part can be stored, as for the second part only its difference compared with the first part can be stored. For example, $M_{CV} = \Leftrightarrow(W0); \{\Uparrow(R0, W1); \Downarrow(R1, W0);\}_{inv\_dir}; \Uparrow(R0)$ is canonical view of test algorithm $M = \Leftrightarrow(W0); \Uparrow(R0, W1); \Downarrow(R1, W0); \Downarrow(R0, W1); \Uparrow(R1, W0); \Uparrow(R0))$.

Let us denote by $|M|$ the complexity of test algorithm M. $|M|$ is the count of bits necessary to code test algorithm M in Test Algorithm Register (TAR) of BIST.

Let us denote by M_{CVM} the minimal canonical view of test algorithm M which satisfies the following condition: $|M_{CVM}| = \min(|M_{CV1}|, \ldots, |M_{CVk}|)$, where $M_{CV1}, \ldots, M_{CVk}$ are all canonical views of test algorithm M. For example, $MSS_{CVM} = \Uparrow(W0); \{\{\Uparrow(R0, W1, W1, R1)\}_{inv\_data}\}_{inv\_dir}; \Downarrow(R0)$ is minimal canonical view of test algorithm March $MSS = \Uparrow(W0); \Uparrow(R0, W1, W1, R1); \Uparrow(R1, W0, W0, R0); \Downarrow(R0, W1, W1, R1); \Downarrow(R1, W0, W0, R0); \Downarrow(R0))$ [90] since it has minimum complexity among all canonical views of test algorithm MSS.

3.2.4 Symmetry Measure

The symmetry level of a test algorithm depends on its structure. Experiments showed that higher symmetry of a test algorithm brings to a smaller number of bits needed to code the given test algorithm in TAR.

The symmetry measure M_S for test algorithm M is calculated in the following way:

$M_S = (1 - |M_{CVM}| / |M|) * 100\%$

Chart 1 shows the BIST area saving and the symmetry measure for different test algorithms. For example, the symmetry measure of March MSS1 is 47% while the corresponding BIST area saving is 28%. At it is seen from the chart, the BIST area saving value is always less than the corresponding test algorithm symmetry measure. It is natural to have this gap since only the BIST TAR in the whole BIST is optimized, while there are other blocks in BIST scheme which area are not dependent on test algorithm symmetry.

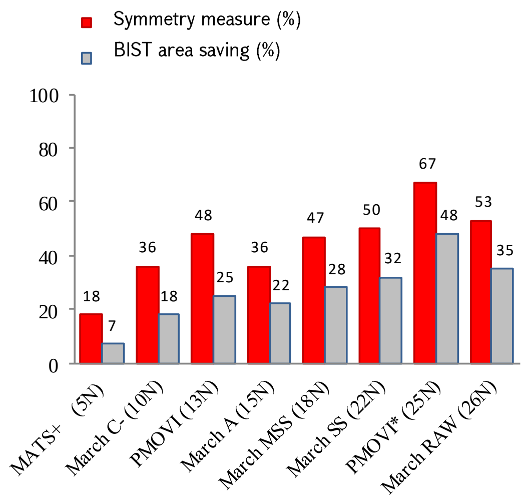


Chart 1. Comparison of test algorithms symmetry

3.3 Regularity of Faults and Test Algorithms

In this work, to study the possible regularities in memory faults and test algorithms, their evolution starting from 90nm to 7nm nodes is investigated. The investigations show that every new technology brings new and more complex faults and similarly test algorithms become more complex. The main regularities observed during the investigations are introduced below.

Regularity 1. The newly discovered faults have similar behavior as the known faults. For example, the same notation is usually used for description of the known and new faults.

Regularity 2. For detection of a new fault, a new test algorithm is usually constructed or extended from an existing test algorithm. Example: March test $\Uparrow$(W0, R0, R0); $\Uparrow$(W1, R1, R1) can be constructed from March test $\Uparrow$(W0, R0); $\Uparrow$(W1, R1) by adding R0 and R1 operations respectively to the first and the second March elements.

Regularity 3. Each fault has its twin fault (e.g., Stuck-At-0 and Stuck-At-1). In other words, for each fault F there is a fault G, that $F \leftrightarrow G$ (symmetric faults).

Regularity 4. Symmetric faults are usually detected by symmetric March tests. Example: the pair (0, R0) and (1, R1) is detected by symmetric March test $\Uparrow$(W0, R0, R0); $\Uparrow$(W1, R1, R1).

During investigation of fault and test algorithm regularity it was noticed that there is a periodicity in evolution of faults and test algorithms.

Fault periodicity. Memory faults are evolved in periodic way. This means that the new faults are the periodic extensions of the existing faults.

Fault families. Based on the length of the SSTO, the faults can be divided into fault families. All faults that are sensitized by SSTO of length k are from FF_k family.

Examples: $(0, \varnothing) \in FF_0$, $(0, 1, W1R1W0) \in FF_3$, $(1, W0R0R0R0, 1) \in FF_4$.

$FF_0=\{FG(0, \varnothing), FG(1, \varnothing)\}$, $FF_1=\{FG(0, W0), FG(1, W1), FG(0, W1), FG(1, W0), FG(0, R0), FG(1, R1)\}$.

Test algorithm periodicity. Test algorithms, like the faults, are evolved in a periodic way. This means that the new test algorithms are the periodic extensions of the existing ones.

Let us compare March C- and March MSS1. March C- is a minimal March test for detection of all traditional faults [14], while March MSS1 is minimal for detection of all static faults. Note that static faults are the superset of traditional faults.

March C- [14]: ⇔(W0); ⇑(R0, W1); ⇑(R1, W0); ⇓(R0, W1); ⇓(R1, W0); ⇔(R0)

March MSS1 [57]: ⇔(W0); ⇑(R0, R0, W1, W1); ⇑(R1, R1, W0, W0); ⇓(R0, R0, W1, W1); ⇓(R1, R1, W0, W0); ⇔(R0)

Both March tests have the same structure with a difference that the Write and Read operations of the 2nd, 3rd, 4th and 5th March elements in March MSS1 are doubled. This means that March MSS1 can be easily extended from March C-.

In memories faults usually occur as pairs of symmetric faults (Regularity 3). Since the symmetric faults are usually detected by symmetric March tests (Regularity 4), a Test Algorithm Template (TAT) is developed to construct symmetric test algorithms.

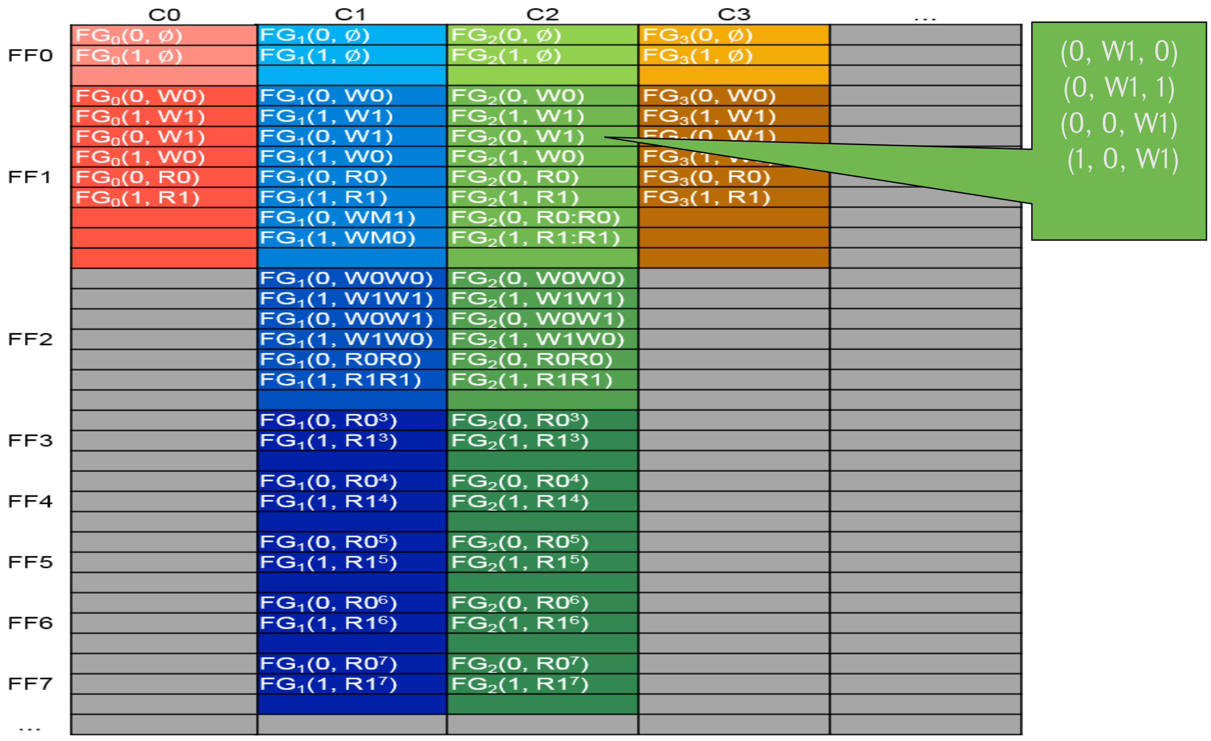
3.4 Fault Periodicity Table

Based on the idea and statement that faults have periodic nature, it is proposed to store all the faults in a Fault Periodicity Table (FPT) (see Figure 28). Each column of FPT corresponds to number of cells that a fault is involved with (C0 includes faults present in memory surroundings and do not have relation with the memory cells, i.e., 0 cells have relation to these types of faults, C1 – all single-cell faults, C2 – two-cell (coupling) faults, etc.), and each row of FPT corresponds to a fault family determined by the number of operations required for fault sensitization (FF0 includes state faults where no operation is needed for fault sensitization, FF1 – faults that are sensitized by one operation, FF2 – faults that are sensitized by two consecutive operations, etc.). In the FPT cells with grey color correspond to unknown (not discovered yet) faults. Each cell of FPT corresponds to a fault sub-group $FG_i(x, S)$. For example, $FG_2(0, W1) = \{(0, W1, 0), (0, W1, 1), (0, 0, W1), (1, 0, W1)\}$.

In the FPT, the following values for S are used ($x, y \in \{0, 1\}$, $n \in N$):

- ∅ - No operation;
- Wx – Write x operation;
- Rx – Read x operation;
- WxWy – consecutive Write x and Write y operations;

- RxRx – two consecutive Read x operations;
- Rx^n – n consecutive Read x operations;
- WMx – Write x operation with all bits masked;
- Rx:Rx – Concurrent (simultaneous) Read x operations applied to different ports of a multi-port memory.



	C0	C1	C2	C3	...
FF0	FG$_0$(0, ∅)	FG$_1$(0, ∅)	FG$_2$(0, ∅)	FG$_3$(0, ∅)	
	FG$_0$(1, ∅)	FG$_1$(1, ∅)	FG$_2$(1, ∅)	FG$_3$(1, ∅)	
FF1	FG$_0$(0, W0)	FG$_1$(0, W0)	FG$_2$(0, W0)	FG$_3$(0, W0)	
	FG$_0$(1, W1)	FG$_1$(1, W1)	FG$_2$(1, W1)	FG$_3$(1, W1)	
	FG$_0$(0, W1)	FG$_1$(0, W1)	FG$_2$(0, W1)	FG$_3$(0, W1)	
	FG$_0$(1, W0)	FG$_1$(1, W0)	FG$_2$(1, W0)	FG$_3$(1, [illegible]	
	FG$_0$(0, R0)	FG$_1$(0, R0)	FG$_2$(0, R0)	FG$_3$(0, R0)	
	FG$_0$(1, R1)	FG$_1$(1, R1)	FG$_2$(1, R1)	FG$_3$(1, R1)	
		FG$_1$(0, WM1)	FG$_2$(0, R0:R0)		
		FG$_1$(1, WM0)	FG$_2$(1, R1:R1)		
FF2		FG$_1$(0, W0W0)	FG$_2$(0, W0W0)		
		FG$_1$(1, W1W1)	FG$_2$(1, W1W1)		
		FG$_1$(0, W0W1)	FG$_2$(0, W0W1)		
		FG$_1$(1, W1W0)	FG$_2$(1, W1W0)		
		FG$_1$(0, R0R0)	FG$_2$(0, R0R0)		
		FG$_1$(1, R1R1)	FG$_2$(1, R1R1)		
FF3		FG$_1$(0, R0^3)	FG$_2$(0, R0^3)		
		FG$_1$(1, R1^3)	FG$_2$(1, R1^3)		
FF4		FG$_1$(0, R0^4)	FG$_2$(0, R0^4)		
		FG$_1$(1, R1^4)	FG$_2$(1, R1^4)		
FF5		FG$_1$(0, R0^5)	FG$_2$(0, R0^5)		
		FG$_1$(1, R1^5)	FG$_2$(1, R1^5)		
FF6		FG$_1$(0, R0^6)	FG$_2$(0, R0^6)		
		FG$_1$(1, R1^6)	FG$_2$(1, R1^6)		
FF7		FG$_1$(0, R0^7)	FG$_2$(0, R0^7)		
		FG$_1$(1, R1^7)	FG$_2$(1, R1^7)		
...					

Figure 28. Fault Periodicity Table

FPT does not limit to have any type of sensitizing operation (such as WMx, Rx:Rx, or it can be other type of operation).

FPT has the following advantages:

- Ability to have all memory faults in a single table;
- Allows to define faults in a systematic way and excludes possibility of fault escapes;
- Is not limited and can be extended, i.e., new rows and columns can be added to FPT according to new faults of future technologies;
- Can help to discover new faults based on its periodicity, e.g., FF$_{i+1}$ faults can be defined based on FF$_i$ faults;
- Can be used for Fault coverage measurement of different test algorithms;
- Allows to build unified BIST architecture which will provide test algorithm

programmability without any limitation.

3.5 Test Algorithm Template

Let us assume that S is a sequence of test operations: $S=OP_1D_1, \ldots, OP_kD_k$, $k\geq 0$ and $x\in\{0, 1\}$. Test Algorithm Template TAT(x, S) has the following structure:

$\Leftrightarrow(W(\sim D_k))$;

$\Uparrow([R(\sim D_k)], [W(x)], S)$;

$\Uparrow([R(D_k)], [W(\sim x)], \sim S)$;

$\Downarrow([R(\sim D_k)], [W(x)], S)$;

$\Downarrow([R(D_k)], [W(\sim x)], \sim S)$;

$\Leftrightarrow(R(\sim D_k))$,

where:

- $\sim S=OP_1(\sim D_1), \ldots, OP_k(\sim D_k)$, if $k\geq 1$. $\sim S=\varnothing$, if $S=\varnothing$;
- $[W(x)]$ and $[W(\sim x)]$ are absent, if $S \neq \varnothing$ and $x=\sim D_k$, otherwise they are present;
- $[R(D_k)]$ and $[R(\sim D_k)]$ are absent, if $S \neq \varnothing$ and $x=\sim D_k$ and $OP_1=R$, otherwise they are present;
- If $S=\varnothing$, then consider $D_k=x$.

All March tests obtained by TAT are symmetric. This is true since all the March tests obtained by TAT satisfy Condition D of Test algorithm symmetry definition.

Table 16 lists some of the well-known test algorithms that can be obtained using TAT. March AB* in Table 16 means that this test algorithm is obtained from original March AB by inverting the addressing directions, while March MSS* means that it is generalized test algorithm of March MSS (first and last March element are using arbitrary address ordering in March MSS*). The investigation showed that test algorithms constructed by TAT are either minimal test algorithms for detection of a specific class of faults or it is an efficient one (its complexity is closer to the complexity of minimal test algorithm). Specifically, for test algorithms from Table 16:

Table 16. Test Algorithms Constructed by TAT

Test algorithm name	Test algorithm description	Fault coverage	Minimal	TAT(x, S)	
				x	S

March C- (10N) [14]	⇔(W0);
	⇑(R0, W1);
	⇑(R1, W0);
	⇓(R0, W1);
	⇓(R1, W0);
	⇔(R0)
		Traditional faults	Yes	1	∅

March MSS1 (18N) [57]	⇔(W0);
	⇑(R0, R0, W1, W1);
	⇑(R1, R1, W0, W0);
	⇓(R0, R0, W1, W1);
	⇓(R1, R1, W0, W0);
	⇔(R0)
		Static unlinked faults	Yes	0	R0, R0, W1, W1

March MSS* (18N) [90]	⇔(W0);
	⇑(R0, W1, W1, R1);
	⇑(R1, W0, W0, R0);
	⇓(R0, W1, W1, R1);
	⇓(R1, W0, W0, R0);
	⇔(R0)
		Static unlinked faults	Yes	0	W1, W1, R1

March SS (22N) [15]	⇔(W0);
	⇑(R0, R0, W0, R0, W1);
	⇑(R1, R1, W1, R1, W0);
	⇓(R0, R0, W0, R0, W1);
	⇓(R1, R1, W1, R1, W0);
	⇔(R0)
		Static unlinked faults	No	0	R0, R0, W0, R0, W1

March AB* (22N) [89]	⇔(W1);
	⇑(R1, W0, R0, W0, R0);
	⇑(R0, W1, R1, W1, R1);
	⇓(R1, W0, R0, W0, R0);
	⇓(R0, W1, R1, W1, R1);
	⇔(R1)
		Subset of dynamic faults (dCFir, dCFrd, dCFdrd, dCFds)	Yes	1	W0, R0, W0, R0

- Marc C- of complexity 10N is minimal test algorithm for detection of traditional faults;
- March MSS* and March MSS1 of complexity 18N are minimal test algorithms for detection of static unlinked faults;

- March SS is efficient test algorithm for detection of static unlinked faults since it has 22N complexity, while minimal one has 18N complexity;
- March AB and March AB* of complexity 22N are minimal test algorithms for detection of subset of dynamic faults (dCFir, dCFrd, dCFdrd, dCFds);

For example, from Table 16 it can be observed that March MSS* and March SS constructed by TAT detect static unlinked faults. March MSS* is minimal, while March SS is considered as efficient since its length (22N) is close to the length of March MSS* (18N). When comparing the S values used for constructing these test algorithms it can be concluded that S of March MSS* - (S = W1, W1, R1) is minimal while S of March SS - (S = R0, R0, W0, R0, W1) is not. This is the reason that March MSS* is minimal test algorithm while March SS is just efficient.

It was observed that most of well-known test algorithms are symmetric and most of those symmetric test algorithms are covered by TAT. TAT is generic enough for covering symmetric test algorithms as well as it is able to construct symmetric test algorithms (well-known or new) for a given set of faults. The investigations showed that at least the following types of faults can be covered by test algorithms constructed by TAT:

- traditional;
- static unlinked;
- dynamic unlinked;
- static linked;
- dynamic linked;
- links between static and dynamic faults, where dynamic faults can be two-operation (most common dynamic faults) and even faults sensitized by more than two operations (e.g., three-operation, four-operation, etc.) which are common to FinFET-based memories (16nm and below nodes).

All these observations and comparisons confirm the generality and effectiveness of the proposed TAT.

Proposition 30. The March test obtained by TAT(x, S) detects all faults from FG(x, S) and FG(~x, ~S).

Proof. Let us show that March test obtained by TAT detects the considered faults. There can be 4 different cases depending on values x and S:

Case A: $S=\varnothing$, $x=\forall$.

MTA=$\Leftrightarrow$(W(~x)); $\Uparrow$(R(~x), W(x)); $\Uparrow$(R(x), W(~x)); $\Downarrow$(R(~x), W(x)); $\Downarrow$(R(x), W(~x)); $\Leftrightarrow$(R(~x)).

Case B: S=OP_1D_1, …, OP_kD_k, $k\geq1$, x=D_k.

MTB=$\Leftrightarrow$(W(~x)); $\Uparrow$(R(~x), W(x), S); $\Uparrow$(R(x), W(~x), ~S); $\Downarrow$(R(~x), W(x), S); $\Downarrow$(R(x), W(~x), ~S); $\Leftrightarrow$(R(~x)).

Case C: S=OP_1D_1, …, OP_kD_k, $k\geq1$, x=~D_k and OP_1=W.

MTC=$\Leftrightarrow$(W(x)); $\Uparrow$(R(x), S); $\Uparrow$(R(~x), ~S); $\Downarrow$(R(x), S); $\Downarrow$(R(~x), ~S); $\Leftrightarrow$(R(x)).

Case D: S=OP_1D_1, …, OP_kD_k, $k\geq1$, x=~D_k and OP_1=R.

MTD=$\Leftrightarrow$(W(x)); $\Uparrow$(S); $\Uparrow$(~S); $\Downarrow$(S); $\Downarrow$(~S); $\Leftrightarrow$(R(x)).

The proof is done in the following way: for each case it is shown that the obtained test algorithms detects all the faults from FG(x, S) and FG(~x, ~S). Table 17 shows the faults and the corresponding operations of March tests for sensitizing and detecting them for Case A. In the table, $MT_{i,j}$ denotes jth component of ith March element in March test MT, $i\geq1$, $j\geq1$. For example, $MTA_{3,1}$ means the first component (i.e., operation R(x)) of the third March element in March test MTA. Table 18 shows the faults and the corresponding operations of March tests for sensitizing and detecting them for Case B. It is easy to construct such tables also for other two cases, i.e., for Case C (March MTC) and for Case D (March MTD). There can be cases when the same fault is sensitized and detected more than once by a given test algorithm. Tables 17 and 18 show only the first sensitizing and detecting scenario for each fault.

TAT(x, S) with its current structure covers all single-cell and two-cell faults (i.e., all faults from FG(x, S) and FG(~x, ~S)) but the same idea can be used to create extended Test Algorithm Template for three or more cells faults.

Table 17. Case A: Fault Sensitization and Detection

Fault Group	Fault	Sensitization	Detection
FG(x, ∅)	(x, ∅)	$MTA_{2,2}$	$MTA_{3,1}$
	$(x, x, \varnothing)_{a<v}$	$MTA_{4,2}$	$MTA_{5,1}$
	$(x, x, \varnothing)_{a>v}$	$MTA_{2,2}$	$MTA_{3,1}$
	$(\sim x, x, \varnothing)_{a<v}$	$MTA_{3,2}$	$MTA_{3,1}$
	$(\sim x, x, \varnothing)_{a>v}$	$MTA_{5,2}$	$MTA_{5,1}$
	$(x, \varnothing, x)_{a<v}$	$MTA_{4,2}$	$MTA_{5,1}$
	$(x, \varnothing, x)_{a>v}$	$MTA_{2,2}$	$MTA_{3,1}$
	$(x, \varnothing, \sim x)_{a<v}$	$MTA_{2,2}$	$MTA_{2,1}$
	$(x, \varnothing, \sim x)_{a>v}$	$MTA_{4,2}$	$MTA_{4,1}$
FG(~x, ∅)	(~x, ∅)	$MTA_{1,1}$	$MTA_{2,1}$
	$(x, \sim x, \varnothing)_{a<v}$	$MTA_{2,2}$	$MTA_{2,1}$
	$(x, \sim x, \varnothing)_{a>v}$	$MTA_{4,2}$	$MTA_{4,1}$
	$(\sim x, \sim x, \varnothing)_{a<v}$	$MTA_{3,2}$	$MTA_{4,1}$
	$(\sim x, \sim x, \varnothing)_{a>v}$	$MTA_{1,1}$	$MTA_{2,1}$
	$(\sim x, \varnothing, x)_{a<v}$	$MTA_{3,2}$	$MTA_{3,1}$
	$(\sim x, \varnothing, x)_{a>v}$	$MTA_{5,2}$	$MTA_{5,1}$
	$(\sim x, \varnothing, \sim x)_{a<v}$	$MTA_{3,2}$	$MTA_{4,1}$
	$(\sim x, \varnothing, \sim x)_{a>v}$	$MTA_{1,1}$	$MTA_{2,1}$

Advantages of Test Algorithm Template. Test Algorithm Template proposes a new approach for constructing test algorithms. It does not use exhaustive and heuristic methods for test algorithm generation, it does not generate but constructs test algorithms by just using the proposed template. Test Algorithm Template does not contain disadvantages of previous methods, i.e., it does not require huge amount of time to be run and it does not produce non-efficient test algorithms. This means that if a new test algorithm is needed to be used in the BIST, then Test Algorithm Template can allow constructing the test algorithm within the given BIST, and the user can specify only the set of faults that need to be detected. In traditional programmable BIST approaches, the picture is quite different. If a

Table 18. Case B: Fault Sensitization and Detection

Fault Group	Fault	Sensitization	Detection
FG(x, S)	(x, S)	$MTB_{2,3}$	$MTB_{3,1}$
	$(x, x, S)_{a<v}$	$MTB_{2,3}$	$MTB_{3,1}$
	$(x, x, S)_{a>v}$	$MTB_{4,3}$	$MTB_{5,1}$
	$(\sim x, x, S)_{a<v}$	$MTB_{4,3}$	$MTB_{5,1}$
	$(\sim x, x, S)_{a>v}$	$MTB_{2,3}$	$MTB_{3,1}$
	$(x, S, x)_{a<v}$	$MTB_{4,3}$	$MTB_{5,1}$
	$(x, S, x)_{a>v}$	$MTB_{2,3}$	$MTB_{3,1}$
	$(x, S, \sim x)_{a<v}$	$MTB_{2,3}$	$MTB_{2,1}$
	$(x, S, \sim x)_{a>v}$	$MTB_{4,3}$	$MTB_{4,1}$
FG(~x, ~S)	(~x, ~S)	$MTB_{3,3}$	$MTB_{4,1}$
	$(x, \sim x, \sim S)_{a<v}$	$MTB_{5,3}$	$MTB_{6,1}$
	$(x, \sim x, \sim S)_{a>v}$	$MTB_{3,3}$	$MTB_{4,1}$
	$(\sim x, \sim x, \sim S)_{a<v}$	$MTB_{3,3}$	$MTB_{4,1}$
	$(\sim x, \sim x, \sim S)_{a>v}$	$MTB_{5,3}$	$MTB_{6,1}$
	$(\sim x, \sim S, x)_{a<v}$	$MTB_{3,3}$	$MTB_{3,1}$
	$(\sim x, \sim S, x)_{a>v}$	$MTB_{5,3}$	$MTB_{5,1}$
	$(\sim x, \sim S, \sim x)_{a<v}$	$MTB_{5,3}$	$MTB_{6,1}$
	$(\sim x, \sim S, \sim x)_{a>v}$	$MTB_{3,3}$	$MTB_{4,1}$

new test algorithm is needed, then it should be generated outside the BIST (for example, using a software tool for test algorithm generation) and then program it into the BIST infrastructure. One of the main advantages of the proposed solution is that it does not require from the user understanding of test algorithm generation process or availability of an external software tool for test algorithm generation.

Fault simulator. Though the Proposition 30 proves that test algorithms constructed by TAT detect a given set of faults, a fault simulator in C environment is developed to check the fault coverage of the test algorithms constructed by TAT. The simulator takes as inputs:

- Set of <fault_info>, where <fault_info> contains the following information:
  - fault type, e.g., stuck-at fault or state coupling fault;

    - fault address, e.g., aggressor address = 5, aggressor data bit = 3, victim address 7, victim data bit = 4;
- <test_algorithm> for which the corresponding fault coverage should be calculated.

The output of the fault simulator is:

- Set of <test_syndrome>, where <test_syndrome> is the vector of 0s and 1s indicating which Read operations of the <test_algorithm> detected the fault (1 means is detected, 0 – is not detected). <test_syndrome> provides information not only about fault detection but, since it contains full information on Read operations detecting the given fault, it can be used also for diagnosing fault types.

The current formats of <fault_info> and <test_algorithm> are developed in a way that those are extendable as well as support a wide range of faults and test algorithms respectively:

- <fault_info> currently supports the following fault classes:
    - traditional faults;
    - static unlinked faults;
    - dynamic unlinked faults;
    - static linked faults;
    - dynamic linked faults;
    - links between static and dynamic faults.
- <test_algorithm> currently supports the following types of test algorithms:
    - Class of March tests;
    - Extended class of March tests (different addressing methods, background patterns and test operations);
    - March-based tests (e.g., when a given set of test operations is applied not to the whole address space of the memories but only to specific addresses);
    - GALPAT and Walking 1/0 test algorithms.

The following experiments were done using fault simulator: different sets of fault groups are selected and for each set of fault groups the corresponding test algorithm is constructed using TAT. Then the same set of fault groups and the test algorithm constructed by TAT were given to fault simulator to generate the corresponding test syndromes. Based on the obtained test syndromes the accuracy of TAT is verified to show that indeed the test algorithms constructed by TAT are detecting the given set of fault groups.

SPICE simulations are done by injecting defects in memory layout GDS model on 90nm – 7nm SRAMs to investigate the behavior of the faults and their periodicity nature. For example, it was observed that in 28nm SRAM there are some open defects that require extra read operations to be sensitized compared to the case with 45nm SRAM. Also, with SPICE simulations it was observed that some defects, causing static faults in 45nm SRAM, cause dynamic faults in 28nm SRAM requiring extra test operations (read or write) to be sensitized. The necessity having extra operations to sensitize the given fault becomes more evident when moving to FinFET-based memories.

Thus, the usage of fault regularities and periodicities efficiently solve at least the following problems:

1. Applying deterministic and direct solution for finding new defects in new technologies is expected. Based on this, new defects can be discovered by adding some extra test operations to the current test algorithm.
2. Using symmetric test algorithms to test symmetric faults. It is obvious that the generation and usage of symmetric test algorithms is preferable than the case when dealing with non-symmetric test algorithms.
3. Creating a BIST scheme where test algorithm generation is performed within the BIST instead of generating it outside the BIST by software tools.
4. Predicting realistic faults for the future technologies.

3.6 A Fault Prediction Mechanism

In this work, a new solution for building the mentioned BIST infrastructure is proposed which is based on multidimensional prediction mechanism. It is based on a notion of periodicity and regularity of faults and test algorithms, and their interdependence. These notions are represented in a form of regularity and periodicity rules. This is considered as a basis for building a generic BIST architecture. Some explanations are shown below to clarify the idea.

The known classification of faults, based on the below factors is considered:

- Complexity of fault sensitization, i.e., number of operations required to activate the fault;
- Number of cells that a fault is involved with.

Following this classification, the faults can be grouped into different classes. The number of classes and faults inside them increase along with technology shrinking (see [91]). A systematic investigation of the evolution of these fault classes and their detection algorithms is done, covering a broad range of manufacturing technologies from 90nm to 7nm. Different types of new faults, such as special classes of dynamic faults, process variation and random telegraph noise induced faults are discovered. Results of this investigation led to determination of some regularity and periodicity rules, which exist for the evolution. Moreover, the known property of symmetry (see [14]), which means that each fault usually has its twin, was developed further and a special symmetry measure for March test algorithms was introduced in [92]. The dependency between symmetry measure and BIST optimization is discussed, as well as a new representation of March test algorithms is introduced called canonical view of March tests. Canonical view represents all similar fragments of a March test algorithm in a standard form comprised of a basic sequence of March elements and a sequence of its transformations for each March element. This means that if two consecutive parts in a March test have the same structure but are different only by data backgrounds, addressing directions, data polarities or addressing modes, then only the first part can be described as for the second part only its difference compared with the first part needs to be described. This allows to use less bits to store a test algorithm in Test Algorithm Register (TAR) of a BIST compared to storing the original view of a March test. The experiments showed that the BIST area overhead can be reduced by 30-40% with negligible time overhead when using canonical view of March tests.

Based on the property of fault and test algorithm symmetry, a new efficient method is proposed in [86] to generate symmetric March test algorithms. The method is based on the observation that almost all known minimal or efficient March test algorithms are symmetric.

It is proposed to describe all the observed fault regularity and periodicity rules in form of a special Fault Periodicity Table (FPT). Each column of FPT corresponds to a fault nature (e.g., number of cells that the corresponding fault is involved with), which can be associated with a variety of different test mechanisms covering the fault. Each row of FPT corresponds to a fault family determined by the complexity of fault sensitization. Fault symmetry is also taken into account in the FPT.

The FPT not only allows building of a generic BIST architecture that supports programmability of test algorithms without limitations mentioned above. It also allows detection of new faults which arise in the field after manufacturing within the same BIST. If these faults can be predicted beforehand then due to regularity of the FPT it is possible to include them into the range of faults covered by the BIST architecture and to detect them further via programmability. Another direction of FPT usage is the following. There can be faults that are not realistic in the current technology, but can be predicted as realistic in the future technologies. These faults can also be reflected in the BIST architecture. This becomes urgent in cases when it is planned to port a given design, including the BIST infrastructure, to a new technology basis without making design changes.

Not all dependencies and regularities of the FPT are yet known. Particularly, during the investigation, several periodical dependencies and regularities that were unknown before were found, justified and included in the FPT. They include also a symmetry reflection in test mechanisms.

Based on the discovered fault types and test mechanism periodicity and regularity, a new generic architecture of BIST is defined [11], [12]. It has two levels of BIST programmability: the first level is test mechanism programmability and the second level is test algorithm programmability. The proposed BIST architecture has the following advantages:

- Automated generation of efficient test algorithms based on FPT;
- Complete programmability of test algorithms and test mechanisms;
- Flexible customization of specific application and a possibility of finding the optimal trade-off between the area and BIST functionality.

From the FPT and TAT development history point of view, the investigation started from the empiric prediction model based on results of the experiments, then found some properties which made the model better described formally and further enriched it by new rules and properties covering also the multidimensional dependencies.

In [9], the initial view of FPT was proposed. It was based on investigation done for memory technologies from 90nm to 28nm. As was mentioned above, in recent years FinFET transistors and 3D memories are playing an important role for performance improvement of ICs as well as for meeting Moor's Law. The investigation showed that new faults caused due to FinFETs and 3D memories have impact on FPT.

During investigation of FinFET-based memories (from 16nm to 7nm) it turned out that there are new types of faults missing in the initial FPT. The investigation showed that all the discovered FinFET-specific faults were possible to add into FPT without changing the structure of FPT, just some blank cells were filled with new faults [93]. Those faults in FPT (see Figure 28) are the following:

- $FG_1(0, R0^3)$, $FG_1(1, R1^3)$, $FG_2(0, R0^3)$, $FG_2(1, R1^3)$;
- $FG_1(0, R0^4)$, $FG_1(1, R1^4)$, $FG_2(0, R0^4)$, $FG_2(1, R1^4)$;
- $FG_1(0, R0^5)$, $FG_1(1, R1^5)$, $FG_2(0, R0^5)$, $FG_2(1, R1^5)$;
- $FG_1(0, R0^6)$, $FG_1(1, R1^6)$, $FG_2(0, R0^6)$, $FG_2(1, R1^6)$;
- $FG_1(0, R0^7)$, $FG_1(1, R1^7)$, $FG_2(0, R0^7)$, $FG_2(1, R1^7)$.

Based on fault types and the place where they can appear in 3D (external) memories the fault space is divided into the following 4 groups [94]:

- G1: Interconnect single-line faults
- G2: Interconnect two-line faults
- G3: Array single-cell faults
- G4: Array two-cell faults

When trying to put 3D memory faults into FPT, it turned out that G3 and G4 groups (array faults) are already part of the existing FPT, while G1 and G2 groups (interconnect faults) required an additional column to be added to FPT and put those faults in that column [94]. The newly added column that incorporates interconnect faults is C0 (see Figure 28) which was not part of initial FPT. C0 means no cells are participating in the fault, i.e., the fault does not have relation to memory cells since it is connected to memory interconnects and memory surroundings (sense amplifier, address decoder, write driver, etc.). Those faults in FPT (see Figure 28) are the following:

- $FG_0(0, \varnothing)$, $FG_0(1, \varnothing)$;
- $FG_0(0, W0)$, $FG_0(1, W1)$;
- $FG_0(0, W1)$, $FG_0(1, W0)$;
- $FG_0(0, R0)$, $FG_0(1, R1)$.

3.7 A Unified BIST Architecture with Fault Prediction Infrastructure

Figure 29 shows the proposed universal BIST architecture [11], [12] which consists of the following blocks:

- Programmable Test Algorithm Register (TAR) – Register for storing test algorithm. This register allows to program new test algorithms where the test algorithms are usually generated by external software tool. In the TAR, it is possible also to program canonical views of March test algorithms which allows to essentially reduce the size of TAR, so consequently the overall BIST area.

- Address Register – Meant for programming new addressing methods which later can be used when creating/programming new test algorithms.
- Data Register – Meant for programming new data background patterns which later can be used when creating/programming new test algorithms.
- Operation Register – Meant for programming new test operations which later can be used when creating/programming new test algorithms.
- BIST Processor – Based on test algorithm stored in the TAR, generates sequence of {Address, Data, Operation} to be applied to the Memory.
- Test Algorithm Generation Unit – User has a possibility to define a set of target fault groups and based on FPT and TAT, Test Algorithm Generation Unit constructs a test algorithm directly in the hardware which is stored in TAR. It is done in the following way: based on the provided set of fault groups, the corresponding x and combined SSTO CS are constructed which are later provided to TAT(x, CS) to construct the required test algorithm. Test Algorithm Generation Unit has CS Constructor module which constructs minimal combined SSTO for a given set of fault groups.

When comparing the proposed BIST architecture with other BIST architectures there are improvements in several blocks while Test Algorithm Generation Unit is new and unique to the proposed architecture. The specifics of Test Algorithm Template (TAT) allows to move the test algorithm generation process from software level into hardware level and construct the test algorithms directly within the BIST scheme. The main advantages of the proposed BIST architecture are the following:

- Maximum flexibility for test algorithm programmability providing not only possibility to construct new test algorithms but also to program the elements of the test algorithm (addressing methods, data background patterns, test operations).
- Possibility to extend the BIST functionality (by using FPT and TAT functions) to test faults of upcoming technologies without redesigning the BIST scheme.

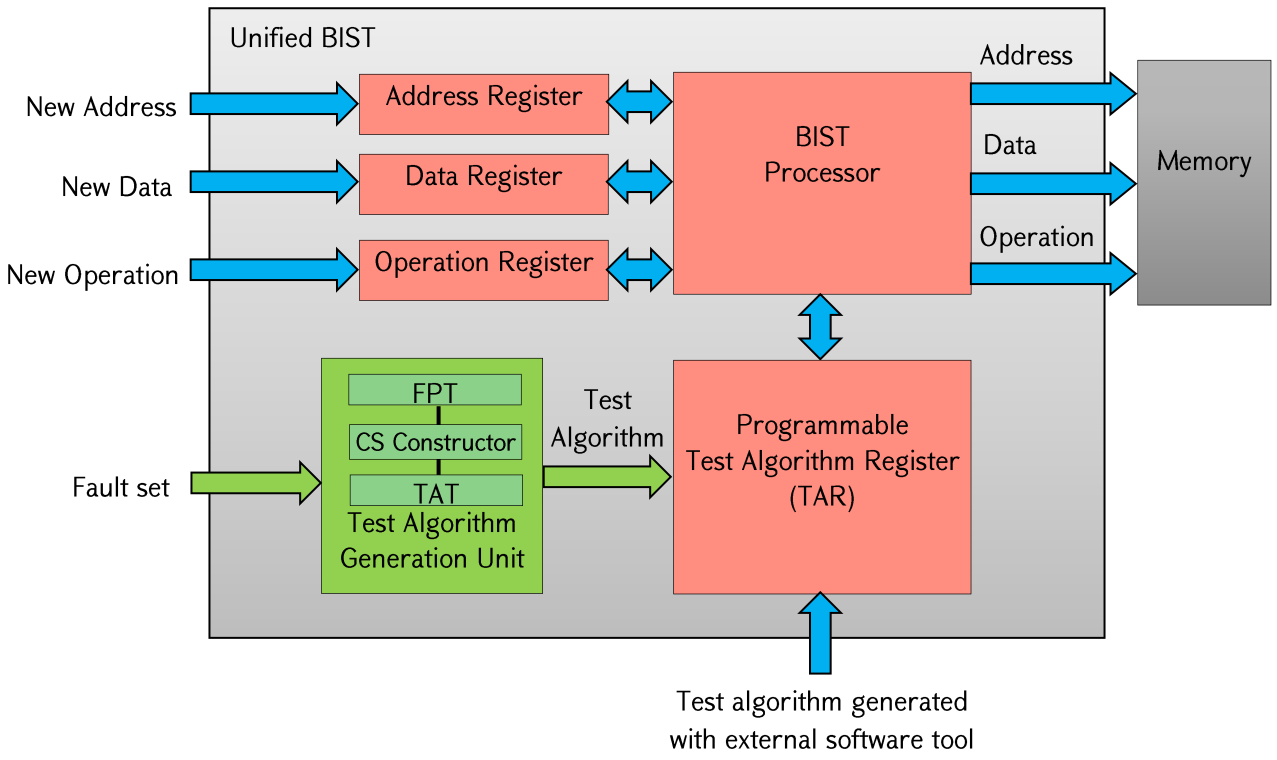


Figure 29. Unified BIST architecture

- User can define a fault set targeted to be covered when using the BIST and does not provide a test algorithm. This is important since the needed test algorithm is not always available or can be generated at user side (a test algorithm generation software tool is needed which is usually at vendor side). Usually the user has knowledge on faults to be covered while has very limited knowledge on how to generate test algorithms. So, Test Algorithm Generation Unit provides BIST users more independence from the vendor and in case a new test algorithm is needed it can be generated at user side.
- Another advantage of Test Algorithm Generation Unit is that in automotive applications usually there is a need to adjust the BIST test algorithm during power-on self-test or in mission mode. In such cases it is easier to adjust the test algorithm using hardware solution (in this case only fault set need to be adjusted and then the corresponding test algorithm will be automatically generated) rather than externally load a new test algorithm generated by software.

- Possibility to program canonical views of March test algorithms in Test Algorithm Register (TAR) which results in reducing the TAR size and consequently reducing the overall BIST area.

Based on the proposed architecture a BIST scheme in Verilog is implemented covering all the features presented. In addition, it has Verilog test bench (TB) environment where a user can do Verilog simulation with or without fault injection. The implemented fault injection mechanism is comprehensive enough since it allows to inject static and dynamic faults and do simulation. The mentioned Verilog TB has the following options:

1. Select a test algorithm to be used by Verilog TB. The test algorithm can be obtained from the following sources:
    a. generated by Test Algorithm Generation Unit using FPT and TAT;
    b. programmed from outside through Programmable Test Algorithm Register (TAR);
    c. selected from the BIST test algorithm library (BIST scheme has a recommended set of test algorithms covering different types of faults).
2. Run Verilog TB without fault injection expecting PASS.
3. Run Verilog TB with fault injection expecting FAIL and a signature matching the injected fault. The following flexibility is provided to do the fault injection:
    a. Select a memory address and data bit where the fault is going to be injected;
    b. Select the fault type (a generic notation is provided where user can define the fault type: traditional, static unlinked, dynamic unlinked, static linked, dynamic linked, links between static and dynamic faults).

As a next step, the BIST Verilog is synthesized and calculated that the Test Algorithm Generation Unit, which is unique for this architecture, adds ~20% to the overall BIST area. Meanwhile, another experiment showed that BIST area without Test Algorithm Generation Unit occupies ~3-5% of the overall SoC area. This means that the 20% increase of BIST hardware should be acceptable resulting in ~3.6-5.5% SoC area increase.

Taking into account that the canonical views allow to reduce the BIST area by 30-40% then the proposed BIST architecture in total will have reduced area overhead compared to traditional BIST architectures meantime providing the above listed additional benefits.

Another experiment is done to estimate the extra time needed for constructing the test algorithm by Test Algorithm Generation Unit. It turned out that this time is negligible compared to the overall BIST execution time. It varies from 1-3% depending on memory size being tested by the BIST.

Conclusions

1. The idea of symmetry, regularity and periodicity of faults and test algorithms is provided.
2. Definitions of fault group, fault and test algorithm symmetry, canonical view of test algorithm and symmetry measure are given.
3. A systematic evolving view - Fault Periodicity Table (FPT) of possible memory faults with rules of periodicity and regularity is proposed.
4. A Test Algorithm Template (TAT) is developed which allows to construct efficient test algorithms as an alternative to exhaustive or heuristic generation of test algorithms.
5. A fault prediction mechanism and unified BIST architecture with flexible programmability options of test operations, addressing methods and layout aware physical background patterns are presented.

CHAPTER 4. ALGORITHMS FOR TESTING MEMORY DEVICES

In this section, the types of nanoscale memory devices and test algorithms created within this work for detecting faults of such memories are presented. There are different types of nanoscale memory devices which are widely used today in SoC production. The most popular ones are: Static Random Access Memories (SRAMs), Content Addressable Memories (CAMs), Read-Only Memories (ROMs), Dynamic Random Access Memories (DRAMs), Flash Memories and 3D Memories.

4.1 Algorithms to Test Static Random Access Memories (SRAMs)

Earlier, an efficient test algorithm March SS of complexity 22N [15] and the minimal test algorithm March MSS of complexity 18N (see Table 19) [90] as well as March MSS1-MSS4 [57] were proposed for detection of all unlinked static faults.

Table 19. March MSS (18N)

Address direction	Operations
⇑	W0
⇑	R0, W1, W1, R1
⇑	R1, W0, W0, R0
⇓	R0, W1, W1, R1
⇓	R1, W0, W0, R0
⇓	R0

For detection of all linked and unlinked static faults test algorithm March SL of complexity 41N [17] was introduced. In [17], the class of all linked static faults was attempted for the first time to classify. However, they assumed erroneously that the number of all static linked faults was 480. In addition to the described linked faults, in [32], extra new 120 linked faults were described. Thus, it was assumed that the number of all static linked faults was essentially higher - 600. This was done based on introduction of the notion of "2-composite faults", thus extending the notion of linked static faults and covering all linked and unlinked static faults. Correspondingly, in [32], a minimal test algorithm March MSL of complexity 23N (see Table 20) is proposed for detection of all “2-composite” (linked and unlinked) static faults.

Table 20. March MSL (23N)

Address direction	Operations
⇑	W0
⇑	R0, W1, W1, R1, R1, W0
⇑	R0, W0
⇑	R0
⇑	R0, W1
⇑	R1, W0, W0, R0, R0, W1
⇑	R1, W1
⇑	R1
⇓	R1, W0

In [19], the authors have shown the importance of dynamic faults for new static random access memory (SRAM) technologies. It has been shown [18], [20], [22], that dynamic faults can manifest themselves in many practical applications [28], [95] and development of test algorithms for their detection is important. In [24], the authors investigated thoroughly, both theoretically and practically, nature and the root cause of dynamic faults based mainly on resistive open defects in memory circuits. Many dynamic faults were validated and the realistic ones were shown.

For the sake of completeness with respect to the space of all linked and unlinked static and dynamic faults, it seems justified to develop test algorithms for the whole space of faults both from theoretical and practical point of view. A non-realistic for the current technology fault may appear to be realistic in a future technology. Thus, excluding some faults from consideration does not seem justified. Of course, it would be ideal to have information on all realistic static and dynamic faults for the designs of certain companies and certain technologies, but in real-life usually such detailed information is missing. In such cases development of universal test algorithms for detection of all linked and unlinked static and dynamic faults would be very useful.

In [55], test algorithm March 100N was proposed for detection of two-operation unlinked dynamic faults. Later, in [28], a minimal test algorithm March MD2 of complexity 70N (see Table 13) is proposed for detection of all dynamic two-operation single-cell and subclass of two-operation two-cell dynamic faults where both sensitizing operations were applied either to the aggressor cell or the victim cell. In [28], [95], only these subclasses of dynamic faults were considered. In [28], there was shown that March MD2 additionally detected all linked and unlinked static faults.

In [28], minimal test algorithms March MD1a and March MD1b (see Tables 21 and 22) of complexity 33N for detection of dynamic two-operation single-cell faults are proposed.

Table 21. March MD1a (33N)

Address direction	Operations
⇔	W0
⇔	W0, W1, W0, W1
⇔	R1, W0, W0
⇔	W0, W0
⇔	R0, W1, R1, W1, R1, R1
⇔	R1
⇔	W1, W0, W1, W0
⇔	R0, W1, W1
⇔	W1, W1
⇔	R1, W0, R0, W0, R0, R0
⇔	R0

Table 22. March MD1a (33N)

Address direction	Operations
⇔	W0
⇔	W0, W1, W0, W1, R1
⇔	W0, W0
⇔	W0, W0
⇔	R0, W1, R1, W1, R1, R1
⇔	R1
⇔	W1, W0, W1, W0, R0
⇔	W1, W1
⇔	W1, W1
⇔	R1, W0, R0, W0, R0, R0
⇔	R0

In [34], minimal March test algorithms are proposed for detection of three-operation single-cell dynamic faults (see Tables 23-26).

Table 23. March ddRDF (39N)

Address direction	Operations
⇔	W0
⇔	W1, W1, R1, R1, R1
⇔	W1, W1, R1, W1, R1
⇔	W0, W0, R0, R0, R0
⇔	W0, W0, R0, W0, R0
⇔	W1, W0, R0, W1, R1
⇔	W0, W1, R1, W0, R0
⇔	W0, W1, R1, R1
⇔	W1, W0, R0, R0

Table 24. March ddDRDF (51N)

Address direction	Operations
⇔	W0
⇔	W0, W0, R0, R0, R0, R0
⇔	W0, W1, R1, R1, R1
⇔	W0, W1, R1, R1, W1, R1, R1
⇔	W0, W0, R0, R0, W1, R1, R1
⇔	W1, W1, R1, R1, R1, R1
⇔	W1, W0, R0, R0, R0
⇔	W1, W0, R0, R0, W0, R0, R0
⇔	W1, W1, R1, R1, W0, R0, R0

Table 25. March ddTF (55N)

Address direction	Operations
⇔	W0
⇔	W0, W0, W1
⇔	W0, W0, W1, R1
⇔	W1, W0, W1
⇔	W1, W0, W1, R1
⇔	W1, R1, W0, W1, R1, W0
⇔	R0, W0, W1, R1
⇔	W0, W1, W0
⇔	W0, W1, W0
⇔	R0
⇔	W0, R0, W1, W0, R0, W1
⇔	R1, W1, W0, R0
⇔	W1, W1, W0
⇔	W1, W1, W0
⇔	R0, R0, W1
⇔	R1, R1, W0
⇔	R0

Table 26. March ddWDF (55N)

Address direction	Operations
⇔	W0
⇔	W1, W1, W1
⇔	W1, W1, W1
⇔	R1, R1, W1, R1
⇔	W0, W0, W0
⇔	W0, W0, W0
⇔	R0, R0, W0, R0
⇔	W0, W1, W1
⇔	W0, W1, W1, R1
⇔	W1, W0, W0
⇔	W1, W0, W0, R0
⇔	W1, R1, W1, W1, R1, W1
⇔	R1
⇔	W0, R0, W0, W0, R0, W0
⇔	R0, W1, W1
⇔	R1, W0, W0
⇔	R0

Table 27 shows the minimal test algorithms created within this work and comparison of those with other test algorithms targeting the same set of faults.

Table 27. Comparison of test algorithms

Fault class	Non-optimal test algorithms	Minimal test algorithms	Reduction
Static unlinked faults	March SS (22N)	March MSS (18N)	18%
Static linked faults	March SL (41N)	March MSL (23N)	44%
Two-operation dynamic faults	March 100N (100N)	March MD2 (70N)	30%

An efficient test algorithm March LSD of complexity 75N is proposed (see Table 13) that detects all the 2-composite static and dynamic faults, including all linked and unlinked static and dynamic faults. "LSD" stands for "linked static and dynamic". The test algorithm March LSD is considered as "efficient" since its complexity is very close to the complexity of March MD2 [28], while March LSD additionally detects all the faults from the subclasses of linked faults such as static*dynamic and dynamic*dynamic faults.

The test algorithm March LSD is generated by a new method which is sufficiently general and efficient to generate symmetric March test algorithms for different combinations (links) of static and dynamic faults. The method is based on the observation that almost all known minimal or efficient March test algorithms are symmetric. In Table 13, several examples of symmetric March test algorithms generated by the proposed method are adduced. Some of them were known previously. Generation of only symmetric March test algorithms reduces drastically the set of all candidate March test algorithms to be considered.

Minimal March FF test algorithm is generated for detection of FinFET-specific faults (see Table 28) [8]. To the best of our knowledge, this work is the first that proposes test algorithm for detection of FinFET-specific faults.

Table 28. March FF (24N)

Address direction	Operations
⇑	W0
⇑	R0, W1, R1, R1, R1, R1, R1, R1, R1, R1, R1
⇓	R1, W0, R0, R0, R0, R0, R0, R0, R0, R0, R0
⇓	R0

Most of the considered functional fault models are validated based on electrical circuit level analysis and experiments on test chips. Different types of faults, including static and dynamic faults, address decoder faults, inter- and intra-port faults, process variation faults and linked static and dynamic faults were considered.

During test fault validation and test algorithm creation process the following 3 major levels are passed:

- Layout to electrical circuit extraction: the extraction is done using memory structural information.
- Electrical circuit to fault modeling extraction: the extraction is done using circuit level simulation. A comprehensive set of faults are injected on electrical circuits (memory array, address decoder, sense amplifier, write driver) and are validated by the experiments.
- Fault modeling to test algorithm extraction: the extraction is performed using test algorithm generation tools.

4.2 Algorithms to Test Content Addressable Memories (CAMs)

Content-addressable memories (CAMs) [96] are a special type of memories used in high speed searching applications, e.g., in computer networking devices, processor caches, etc. CAMs can be binary (each cell stores 0 or 1) or ternary (besides 0 and 1, each cell can store a third value "x" indicating that it is masked and does not participate in search operations). Several testing methodologies have been proposed for CAMs in [97]-[103].

In general, CAMs have word structure and there is a valid bit associated with each CAM word (see Figure 30). Some CAMs may have a valid bit associated with each half or quarter

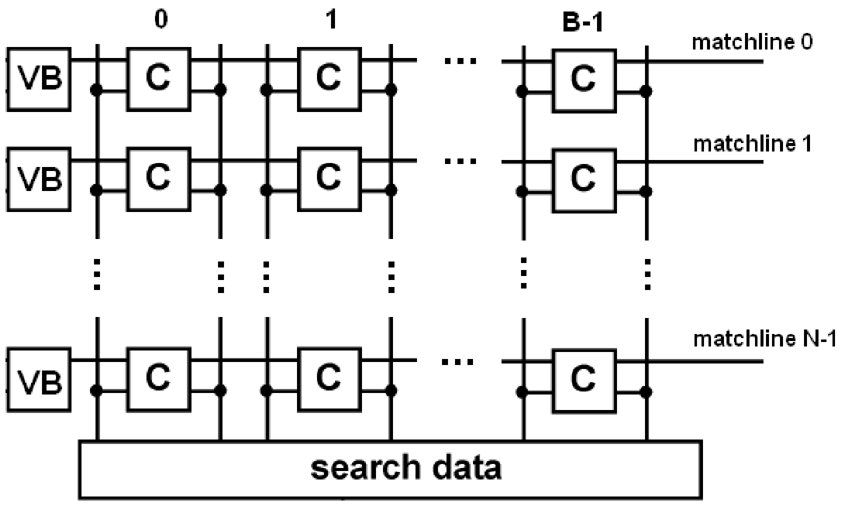


Figure 30. CAM structure

of the CAM word and allow to access/search each sub-word separately. Basic operations of CAMs are write, read, compare and unload operations. Write and read operations are the same as in Static Random Access Memories (SRAMs), except that write operation also sets the valid bit to 1. Unload operation sets the valid bit of a word to 0. Also, there is usually an invalidate operation which unloads all words at once. Compare operation searches a given word within the CAM words. It will report a hit (match) if the word is found in CAM and the corresponding valid bit is set to 1, and a miss in other cases. During compare operation some of the bits of a searched word can be masked. It means that the corresponding cells will not participate during compare operation.

In this work, the list of faults considered in [98]-[103] is extended, as well as a new notation for CAM faults is presented. A comprehensive set of realistic storage, comparison and mask faults of single-port binary and ternary CAMs (BCAMs and TCAMs) is considered. Three minimal March tests for the detection of storage faults, comparison faults and TCAM mask faults are proposed: March MSL, March CCF and March TCMF [104].

Based on the proposed test mechanisms (test algorithms, test operations, background patterns, etc.) a CAM BIST is developed. It is an extension of a BIST architecture dedicated to SRAM testing [104]. The proposed BIST architecture additionally supports the following important CAM specific features: power buffer-zones (sometimes needed due to huge power consumption differences of different test operations), multi-cycle Compare operations (e.g. compare operations with pre-search), half and quarter word operations and walking patterns (e.g. 1000, 0100, 0010, 0001).

The considered CAM faults are described below.

Storage faults. Since the storage part of a CAM cell is similar to an SRAM cell, all unlinked and linked static faults [32] are considered for CAMs.

Comparison faults. In [15], taxonomy for RAM fault notation is proposed. To the best of our knowledge, this work is the first that presents a similar taxonomy for CAM faults.

Let us denote by <S:V/C[:M]/R> a CAM fault primitive, where:

- S – cell/word state: cell state is denoted by $x \in \{0, 1\}$ and word state – by W; S can be also "-" which is applicable only to TCAM indicating that the cell contains the third (masked) value;
- V – valid bit value of the word: 0 or 1;
- C – value of the compared cell/word: cell value is denoted by x, word value – W (W^k denotes the word that differs from W only by kth bit);
- M (optional) – number of the bit that is masked during compare operation. If M is not specified, then no bit is masked;
- R – result of the compare operation on the current cell/word: miss – 0, hit – 1.

For example, <0:1/0/0> means the following: if faulty cell state is 0, valid bit is set to 1 and comparing value is 0, then result of the compare operation is miss.

The following CAM FFMs are considered:

- Always match faults: If a CAM cell contains an always match fault (AMF) then a compare operation will always hit on that cell irrespective of the state of the cell and the compare data.
- Always mismatch faults: If a CAM cell contains an always mismatch fault (AMMF) then a compare operation will always miss on that cell irrespective of the state of the cell and the compare data.
- Partial match faults: If a CAM cell contains a partial match fault (PMF) then a compare operation will always hit on that cell if the cell contains x logic value and will always miss if the cell contains $\bar{x}$ logic value. These types of faults can be divided into two groups: PMF-0 when x = 0 and PMF-1 when x = 1.
- Conditional match faults: If a CAM cell contains a conditional match fault (CMF) then a compare operation will work properly if the cell contains x logic value and will report an incorrect response if the cell contains $\bar{x}$ logic value. These types of faults can be divided into two groups: CMF-0 when x = 0 and CMF-1 when x = 1.
- Stuck valid bit faults: If a CAM word contains a stuck valid bit fault (SVBF) then the word will always participate in a compare operation even if its valid bit is set to 0.

- Stuck invalid bit faults: If a CAM word contains a stuck invalid bit fault (SIVBF) then the word will never participate in a compare operation even if its valid bit is set to 1.
- Mask column faults: If one of CAM columns contains a mask column fault (MCF) then the corresponding cells of every CAM words will always participate in a compare operation even if the corresponding column is masked.
- Mask faults (applicable only to TCAMs): If a CAM cell contains a mask fault (MF) then the corresponding cell will always participate in a compare operation even if the cell contains “x” value. For the encodings, when there are separate data and mask bits, these types of faults can be divided into two groups: MF-0 when 0 value is not successfully masked and MF-1 when 1 value is not masked.
- Cross match faults: It is caused by a short between bit-lines of two neighboring cells. Since single-port CAMs (compare and read operations are performed with the same bit-lines) are considered, cross match faults coincide with storage coupling faults. Thus, further those faults will be considered as storage coupling faults. Note that the proposed fault notation can be extended to describe also these types of faults (aggressor cell state to be included).

For example, <0:1/1/1; 1> would mean the following: if faulty (victim) cell state is 0, the neighboring (aggressor) cell value is 1, valid bit is set to 1 and comparing value is 1, then result of the compare operation is hit.

For CAM testing an extended class of March test algorithms is proposed. $M = \{M_1, M_2, \ldots, M_k\}$, where $M_i = AD_i\,(O_1, \ldots, O_m)$:

$O_i \in$ {R(D), W(D), WC(D), WCM(D), MISS(D), HIT(D), HITM(D, I), INV, UNL}:

- R(D) – Read from the current address expecting D data background pattern;
- W(D) – Write D data background pattern to the current address (for TCAMs, write D to mask cells as well);
- WC(D) – Write D data background pattern to the current address (for TCAMs, write D without masking);

- WCM(D) (applicable only to TCAMs) – Write D data background pattern to the current address with all bits masked;
- MISS(D) – Compare operation with D data background pattern expecting a miss on all words;
- HIT(D) – Compare operation with D data background pattern expecting a hit;
- HITM(D, I) – Masked compare operation with D data background pattern expecting a hit, I^th^ bit is masked;
- INV – Invalidates all CAM words, i.e., sets all valid bits to 0;
- UNL – Unloads the current word, i.e., sets the valid bit of the current word to 0.

$AD_i \in \{\Uparrow, \Downarrow, \Leftrightarrow, \Leftrightarrow_1\}$:

- $\Uparrow$ – ascending address order (0, 1, …, N-1);
- $\Downarrow$ – descending address order (N-1, …, 1, 0);
- $\Leftrightarrow$ – can be $\Uparrow$ or $\Downarrow$;
- $\Leftrightarrow_1$ – single address operation (sequence of operations is performed to the whole address space only once), fwhich is used with MISS, HIT, HITM, INV operations.

In the descriptions of the proposed test algorithms the following notations are used:

- N – Number of words;
- B – Number of bits per word;
- D – data background pattern;
- ~D – Complement of D;
- D_k – Walking background pattern, where all bits are equal to the bits of D, expect for the kth bit;
- ~ D_k – Complement of D_k.

Three minimal March tests are described for detection of CAM faults. For detection of storage faults, the well-known March MSL test algorithm is used (see Table 20). It is a minimal test algorithm for detection of all static unlinked and linked faults.

For detection of comparison faults minimal test algorithm March CCF (see Table 29) is proposed (CCF stands for "CAM Comparison Faults").

For detection of TCAM mask faults minimal March test March TCMF (see Table 30) is proposed (TCMF stands for "Ternary CAM Mask Faults").

Table 29. March CCF (4N+3B+2)

Address direction	Operations
⇔	WC(D)
$⇔_1$	MISS(D_0), MISS(D_1), …, MISS(D_{B-1})
$⇔_1$	HITM(D_0, 0), HITM(D_1, 1), …, HITM(D_{B-1}, B-1)
$⇔_1$	INV
$⇔_1$	MISS(D)
⇔	WC(D), HIT(D), WC(~D)
$⇔_1$	MISS(~D_0), MISS(~D_1), …, MISS(~D_{B-1})

Table 30. March TCMF (5N+1)

Address direction	Operations
$⇔_1$	INV
⇔	WCM(D), HIT(~D), WCM(~D), HIT(D), UNL

For BCAM testing March MSL and March CCF should be applied. For TCAM testing additionally March TCMF should be applied.

4.3 Algorithms to Test Read-Only Memories (ROMs)

Since the content of ROM memories are usually stable and only read operation can be applied (no Write operation is allowed), then usually these kind of memories are being tested by applying XOR based functions using LFSRs or MISRs [105].

The investigation showed that there are some types of ROM faults that cannot be detected by LFSRs and MISRs and a special class of March test algorithms is needed to detect those faults.

The mentioned ROM faults are the following:

- Multiple read faults – Consecutive read operations flip the content of ROM cell;
- Address decoder delay faults – Due to a fault in ROM address decoder there is a delay when transitioning from one ROM address to another.

For detection of ROM faults March VLROM is proposed (Table 31).

Table 31. March VLROM (16N)

Address direction	Operations
y⇑	RS, RI, RC, RC
y⇓	RS, RI, RC, RC
x⇑	RS, RI, RC, RC
x⇓	RS, RI, RC, RC

where:

- RS – Read the current address of the memory and store it in a special BIST register (ROM_REG);
- RI – Read from the address which is obtained by inverting all the bits of the current address and ignore the read result (this operation is needed just to provide jump operation to another address to check if the ROM address decoder is fast enough to make such transitions);
- RC – Read the current address of the memory and compare the read data with the content of ROM_REG.
- y⇑ - fast column address increment;
- y⇓ - fast column address decrement;
- x⇑ - fast row address increment;
- x⇓ - fast row address decrement.

4.4 Algorithms to Test Dynamic Random Access Memories (DRAMs)

DRAM is a type of memory that stores each bit of data in a separate capacitor within an integrated circuit. DRAM is denser than SRAM but needs to be refreshed periodically when in use. From other side, SRAM has better performance than DRAM.

The following faults of DRAMs are considered [106]:

- Retention faults;
- Word-line coupling faults;
- Bit-line toggling faults;
- Stuck-open faults;
- Bank faults.

March DMFD test algorithm is developed to test DRAM faults (Table 32), where "interval" is defined by user.

Table 32. March DMFD (29N)

Address direction	Operations
⇑	W(D)
⇑	R(D), R(D), W(~D), W(~D), R(~D)
⇑	R(~D), R(~D), W(D), W(D), R(D)
⇓	R(D), R(D), W(~D), W(~D), R(~D)
⇓	R(~D), R(~D), W(D), W(D), R(D)
⇑	R(D)
⇑	W(D)
DELAY(interval)	
⇑	R(D), W(~D)
DELAY(interval)	
⇓	R(~D), W(D)
⇑	WM(~D), R(D)

4.5 Algorithms to Test Flash Memories

Since flash memories have different cell structures and operations, there are flash specific faults, that do not occur in RAMs. The reliability issues which occur in floating gate semiconductor arrays are data retention and endurance. Data retention is a critical end-of-life parameter, it is the ability of a non-volatile memory to properly maintain and provide on demand the programmed state of any data bit in the array for a minimum period of time. Endurance is the parameter that specifies the cumulative program/erase cycling capability of any individual sector within a memory. Each sector-erase operation has the capability of introducing defects into the memory cell structure that may accumulate over time. At some point, these defects may prevent a cell from programming, erasing or reading. When such a failure occurs in a cell that contains critical data, the loss of data may cause system failure. The sector in which the failure occurs is effectively end-of-life. These two reliability issues need stress test [107].

Besides, flash memories can experience disturbances that do not conform to any of the traditionally known fault models used in testing RAMs, even though these faults might have certain characteristics similar to some of the known fault models. There are several types of faults common to flash memories [108], [109]:

- Word-line erase disturbance (WED) - The selected cell under program causes the affected cell on the same word-line to be erased.
- Bit-line erase disturbance (BED) – The selected cell under program causes affected cell on the same bit-line to be erased.
- Word-line program disturbance (WPD) - The selected cell under program can cause the affected cell on the same word-line to be programmed.
- Bit-line program disturbance (BPD) - The selected cell under program can cause the affected cell or on the same bit-line.
- Source-line program disturbance (SPD) - The selected cell under program causes the affected cells on the same source-line to be programmed.

- Gate read erase disturbance (GRE) - Reading a cell results in another cell on the same word-line to be erased.
- Channel read-program disturbance (CRP) - Reading a cell results in another cell on the same word-line to be programmed.
- Read program disturbance (RPD) - An erased cell undergoing multiple consecutive read operations is programmed.
- Read erase disturbance (RED) - A programmed cell undergoing multiple consecutive read operations is erased.
- Over erase disturbance (OED)- The threshold voltage of cell decreases, turning the cell into a depletion-mode transistor, because of erasing. In other words, over erase is the situation when there is bit-line leakage. The over erased cell cannot be programmed normally. Only after consecutive program operations the cell may be recovered.
- Over program disturbance (OPD) - A cell is overly programmed such that it cannot be erased in a normal way. The over programmed cell may be recovered after consecutive erase operations.
- Read disturb (RD: RD0, RD1) - Consecutive read operation on a cell impacts the threshold voltages of affected flash cells and causes them to flip their values.

There are some RAM specific faults, that occur in flash memories also. These common faults are stuck-at faults, transition faults, state coupling faults, address decoder faults, stuck-open faults [14].

In Table 33, well-known flash test algorithms are described [110]. Each of these algorithms covers a specific set of flash faults. The one that covers wider set of disturbance (except for disturbances that need consecutive Read operations) and common faults with RAMs is March-FT algorithm [108].

Table 33. Well-Known Tes Algorithms

Test Algorithm Name	Description
MSAF	(E); ⇑(R1); ⇑(P); ⇑(R0)
MTF	(E); ⇑(R1); ⇑(P); ⇑(R0); (E); ⇑(R1)
MSOF	(E); ⇑(R1, P, R0)
MAF	(E); ⇑(R1, P, R0); (E); ⇓(R1, P, R0)
MDF	(E); ⇑(R1, P); ⇑(R0); (E); ⇓(R1, P); ⇓(R0)
March-FT	(E); ⇑(R1, P, R0); ⇑(R0); (E); ⇓(R1, P, R0); ⇓(R0)

In this work the March-FT algorithm has been extended [111]. A set of testing mechanisms are added to it, in order to perform complete testing of embedded flash memories. Test operations used in the test algorithms are:

a. READ – Performs read operation;
b. PROGRAM – Performs program operation;
c. SECTOR_ERASE – Erases the specified sector;
d. CHIP_ERASE – Erases whole memory;
e. DELAY(interval) – No operation is performed within specified interval time.

The proposed types of physical background patterns are the followings:

a. Solid: all 0s (0000…00) or all 1s (1111…11);
b. Checkerboard: odd rows contain physically 0101…01 data, even rows contain 1010…10 data or vice-versa;
c. Row stripe: odd rows contain 0000…00, even rows contain 1111…11 or vice-versa;
d. Column stripe: odd columns contain 0000…00, even columns contain 1111…11 or vice-versa.

There are 4 types of addressing methods in proposed solution:

a. Address increment (0, 1, 2, …, N-1), where N is the number of addresses of a flash memory;
b. Address decrement (N-1, N-2, …, 2, 1, 0);
c. Address complement (0, N-1, 1, N-2, …);

d. Single addressing (no specific address is specified, mainly used with erase operation).

As it has been mentioned above, RD, RPD and RED faults need consecutive Read operations to be detected. For this purpose, some read operations of original March-FT have been replaced with consecutive reads in extended algorithm. The number of consecutive read operations is user defined. With this change extended algorithm fully covers all disturbance and common faults with RAMs that can occur in embedded flash memories. Table 34 shows the proposed test algorithm March-FTE (extended March-FT) [111], where:

Table 34. March-FTE (8N+2)

Address direction	Operations
$\Leftrightarrow_1$	CHIP_ERASE
$\Uparrow$	READ(~D)^n, PROGRAM(D), READ(D)^n
$\Uparrow$	READ(D)
$\Leftrightarrow_1$	CHIP_ERASE
$\Downarrow$	MISS(D)
$\Downarrow$	READ(~D)^n, PROGRAM(D), READ(D)^n
$\Leftrightarrow_1$	READ(D)

- D is background pattern – checkerboard, solid, row stripe or column stripe;
- n is the number of loops (how many times the operation should be repeated) - ($n \geq 1$);
- $\Uparrow$ - Address increment;
- $\Downarrow$ - Address decrement;
- $\Leftrightarrow_1$ – single address operation (sequence of operations is performed to the whole address space only once).

Since as already mentioned some of the flash faults need special stress conditions for fault sensitization, the proposed solution provides the following test modes:

a. Apply high voltage for some operations;

b. Bake test for data retention;

c. Address decoding test;

d. Endurance stress.

For data retention testing the March RT test algorithm is proposed (see Table 35) that should be run in bake stress mode, where D is background pattern and “interval” is defined by user and specifies the time period during which the cell should be able to stay at programmed state.

Table 35. March RT (2N)

Address direction	Operations
$\Uparrow$	PROGRAM(D)
DELAY(interval)	
$\Uparrow$	READ(D)

For endurance testing consecutive program and erase operations are needed under endurance stress mode. The MSAF test algorithm from Table 33 is best suited for it. The extended test algorithm MSAF Stress applies to all memory sectors under endurance stress mode. Table 36 shows the comparison of embedded flash test algorithms.

Table 36. Fault Coverage of Test Algorithms for Flash Memory Testing

Faults/ Algorithms	WED	BED	WPD	BPD	SPD	GRE	CRP	SAF	TF	CFst	AF	SOF	OED	OPD	RD0	RD1	RPD	RED	Ret.	End.
MSAF	-	-	-	-	-	-	-	+	-	-	-	-	+	-	-	-	-	-	-	-
MTF	-	-	-	-	-	-	-	+	+	-	-	-	+	+	-	-	-	-	-	-
MSOF	-	-	-	-	-	-	-	+	-	-	+	+	+	-	-	-	-	-	-	-
MAF	-	-	-	-	-	-	+	+	+	+	+	+	+	+	-	-	-	-	-	-
MDF	+	+	+	+	+	-	-	+	+	+	+	-	+	+	-	-	-	-	-	-
March-FT	+	+	+	+	+	+	+	+	+	+	+	+	+	+	-	-	-	-	-	-
March-FTE	**+**	**+**	**+**	**+**	**+**	**+**	**+**	**+**	**+**	**+**	**+**	**+**	**+**	**+**	**+**	**+**	**+**	**+**	-	-
March RT	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**+**	-
MSAF Stress	-	-	-	-	-	-	-	+	-	-	-	-	+	-	-	-	-	-	-	**+**

The proposed test algorithms (March-FTE, March RT and MSAF Stress) are in the marked range of the table which together cover all CAM faults discussed in this work.

4.6 Algorithms to Test 3D Memories

March EMFD (EMFD stands for External Memory Fault Detection) test algorithm is proposed (see Table 37) [112] which detects all array and interconnect faults considered in this work. Additionally, this test algorithm provides the necessary level of diagnosis for those faults, where:

- y⇑ - fast column address increment;
- y⇓ - fast column address decrement;
- x⇑ - fast row address increment;
- x⇓ - fast row address decrement;
- $\Leftrightarrow_{MIN}$ – Test operations are applied only to minimum address (usually it is 0);
- $\Leftrightarrow_{MAX}$ – Test operations are applied only to maximum address;
- $\Leftrightarrow_{WALK0}$ – Walking 0 addressing order (e.g., 1110, 1101, 1011, 0111);
- $\Leftrightarrow_{WALK1}$ – Walking 1 addressing order (e.g., 0001, 0010, 0100, 1000);
- SO – Solid 0 pattern (all 0s);
- ~SO – Solid 1 pattern (all 1s);
- L01 – Logical 01 pattern (010101…01);
- ~L01 – Logical 10 pattern (101010…10).

Table 37. March EMFD (26N + 6Log(N)+26)

Address direction	Operations
y⇑	W(L01), R(L01)
y⇑	W(~L01), R(~L01), W(~L01), R(~L01)
y⇑	R(~L01), W(~L01), W(L01), R(L01)
y⇓	W(~L01), R(~L01)
y⇓	W(L01), R(L01), W(L01), R(L01)
y⇓	R(L01), W(L01), W(~L01), R(~L01)
x⇑	W(SO)
x⇑	R(SO), W(~SO)
x⇓	R(~SO), W(SO)
x⇓	R(SO)
⇔MIN	W(L01), R(L01)
⇔MAX	W(~L01), R(~L01)
⇔MIN	W(~SO), R(~SO)
⇔WALK1	W(SO), R(SO)
⇔MIN	R(~SO), W(~SO)
⇔WALK1	R(SO)
⇔MAX	W(SO), R(SO)
⇔WALK0	W(~SO), R(~SO)
⇔MAX	R(SO), W(SO)
⇔WALK0	R(~SO)
⇔MIN	WM(SO), R(~SO)
⇔MIN	W(~SO), WM0(SO), RM01(~SO), W(~SO), WM1(SO), RM01(SO)
⇔MIN	W(SO), WM0(~SO), RM01(SO), W(SO), WM1(~SO), RM01(~SO)

Conclusions

1. Different types of nanoscale memory devices are considered and test algorithms created within this work for detecting faults of such memories are presented.
2. Table 38 shows the main test algorithms created within this work.

Table 38. Test algorithms

Memory type	Faults	Test algorithm	Complexity	Minimal complexity
SRAM	All unlinked static faults	March MSS	18N	18N
	All unlinked and linked static faults	March MSL	23N	23N
	All unlinked dynamic faults	March MD2	70N	70N
	All unlinked and linked static and dynamic faults	March LSD	75N	≥ 70N
	FinFET-specific faults	March FF	24N	24N
CAM	All CAM storage faults	March MSL	23N	23N
	All CAM comparison faults	March CCF	4N+3B+2	4N+3B+2
ROM	All address decoder and read destructive faults	March VLROM	16N	16N
DRAM	All memory array and interconnect faults	March DMFD	29N	≥26N
Flash	Read/program/erase disturb faults	March-FTE	8N+2	8N+2
3D Memory	Interconnect and memory die faults	March EMFD	26N+6log(N) +26	26N+6log(N) +26

CHAPTER 5. FAULT DIAGNOSIS AND LOCALIZATION

5.1 Concept of Fault Diagnosis and Localization

Methods of fault detection and diagnosis become increasingly important for the improvement of reliability, safety and efficiency in nanoscale designs. There are numerous works done on diagnosis topic proposing different kind of flows and test algorithms for fault diagnosis [38], [113]-[117].

In [38], definition of March syndrome is provided, as well as efficient March and March-based test algorithms are proposed for classification and localization of traditional faults.

In [113], the concept of test primitives (TPs) is used to diagnose the faulty behavior of memory devices. Test primitives are extensible and flexible such that additional tests can be added to diagnose new faults that are not included originally. The authors presented a procedure to perform fault diagnosis using test primitives for any given set of faults. In order to use Test Primitives to diagnose memory faults, TP dictionary is generated that maps the pass/fail characteristics of a set of TPs to their corresponding faults.

In [114], a diagnosis flow is proposed which performs the physical shape analysis of the defects and the fault modelling based on the obtained failure syndromes. The proposed technique also takes into account possible side effects such as received incomplete data, tester noise and fault intermittent effects. The flow consists of 3 steps, where on the first step the physical failure shape recognition procedure is executed to classify the failed cells (whether it is spot failure, cluster or complete memory column or row failure), on the next step for each failure the fault model is extracted with some probability and finally, the hypothesis is provided over obtained fault type with so called confidence score illustrating the percentage of cells matching the hypothesis criteria. The final results are shown as memory failure bitmap. The efficiency of the proposed technique is illustrated on several test cases done for automotive-oriented 90nm SoCs.

Build-in self-test (BIST) systems usually build test and repair infrastructure for memory devices which provides test mechanisms and background patterns for running test sequences and algorithms [118]. Information on the memory scrambling is necessary for essential improvement of the reliability, safety and effectiveness, e.g. for generating accurate physical background patterns, reporting the exact physical coordinates of failed cells in the memory, etc. [119].

All alternative solutions that do not consider memory scrambling information lead to an inefficient memory test since in that case exhaustive logical data patterns need to be applied to the memory for achieving the desired fault coverage. In [120], it is stated that lack of memory scrambling information can lead up to 35% test escapes.

Memory scrambling information [119], [121] can be considered as comprising of two main parts:

- memory structural information (mapping between memory logical and physical addresses), such as address scrambling, Input/Output (IO) cell scrambling, row scrambling, column scrambling, bit-line twisting, etc.;
- memory topological information (strap distribution) which describes the positions and sizes of blocks in the memory that are not memory bit-cells.

5.2 Memory Structural and Topological Modeling and Its Role in Memory Test, Fault Diagnosis and Localization

Memory scrambling information consists of two main parts: memory structural information (mapping between memory logical and physical addresses) and memory topological information (strap distribution).

The most common memory scrambling types widely used in nowadays memories are adduced below.

- Address scrambling – mapping between logical address and logical column, logical row, logical bank. For example, there can be the following type of address scrambling: counting from LSB of address bus the first 3 bits are logical column bits, next 2 bits are logical bank bits, and the remaining bits are logical row bits;
- Column scrambling – mapping between logical column and physical column;
- Row scrambling – mapping between logical row and physical row;

- Bank scrambling – mapping between logical bank and physical bank;
- IO cell scrambling – mapping between logical data and physical IO cell;
- Column twisting – bit-lines of one column (e.g., when a pair of SRAM 6T cell bit-lines is considered) are twisted with bit-lines of another column when moving from one row to the next row. Figure 31 shows an example of column twisting;
- Bit-line twisting – twisting True and Bar bit-lines within the column when moving from one row to the next row;
- Bit-line mirroring – mirroring True and Bar bit-lines of adjacent bit-cells in a row, i.e., counting from left to right the first cell contains True bit-line, then Bar bit-line, second cell contains Bar bit-line and then True bit-line, etc. In other words, the bit-line distribution of bit-cell in a row is the following: True, Bar, Bar, True, True, Bar, Bar, True, …

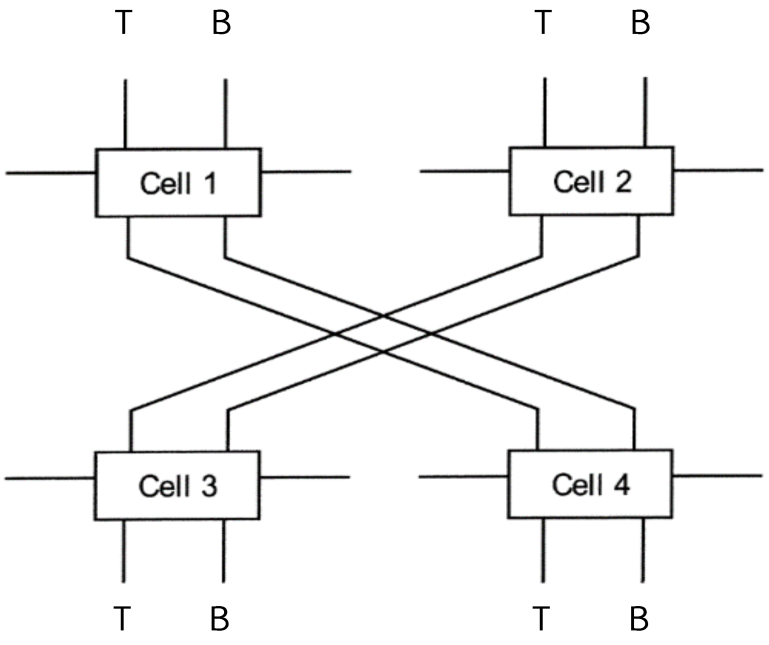


Figure 31. Column twisting

Strap distribution describes positions and sizes of blocks in the memory that are not memory bit-cells. It usually includes the surrounding blocks of the memory as well as blocks that are between in memory array banks. Example of straps are IOs, address decoder, sense amplifier, bank separator, etc.

Figure 32 shows an example of strap distribution. It contains the following straps:

- Vertical strap "Row Decoder";
- Vertical strap "Right Border";
- Horizontal strap "Column Decoder";
- Horizontal strap "Bank Separator";
- Horizontal strap "Sense Amplifier".

Usage of memory scrambling information is necessary for generating accurate physical background patterns as well as for calculating the exact physical coordinates of failed cells in the memory. If for a given logical address (Address, Bit) the corresponding mapping into physical address (Row, Column) is not known then it is not possible to:

- generate physical background patterns (e.g., physical Checkerboard pattern);
- calculate exact physical coordinate of failed cells (e.g., if width of Row Decoder in Figure 32 is not known, then it is impossible to calculate the physical coordinate of a failed cell relative to memory (0, 0) coordinate).

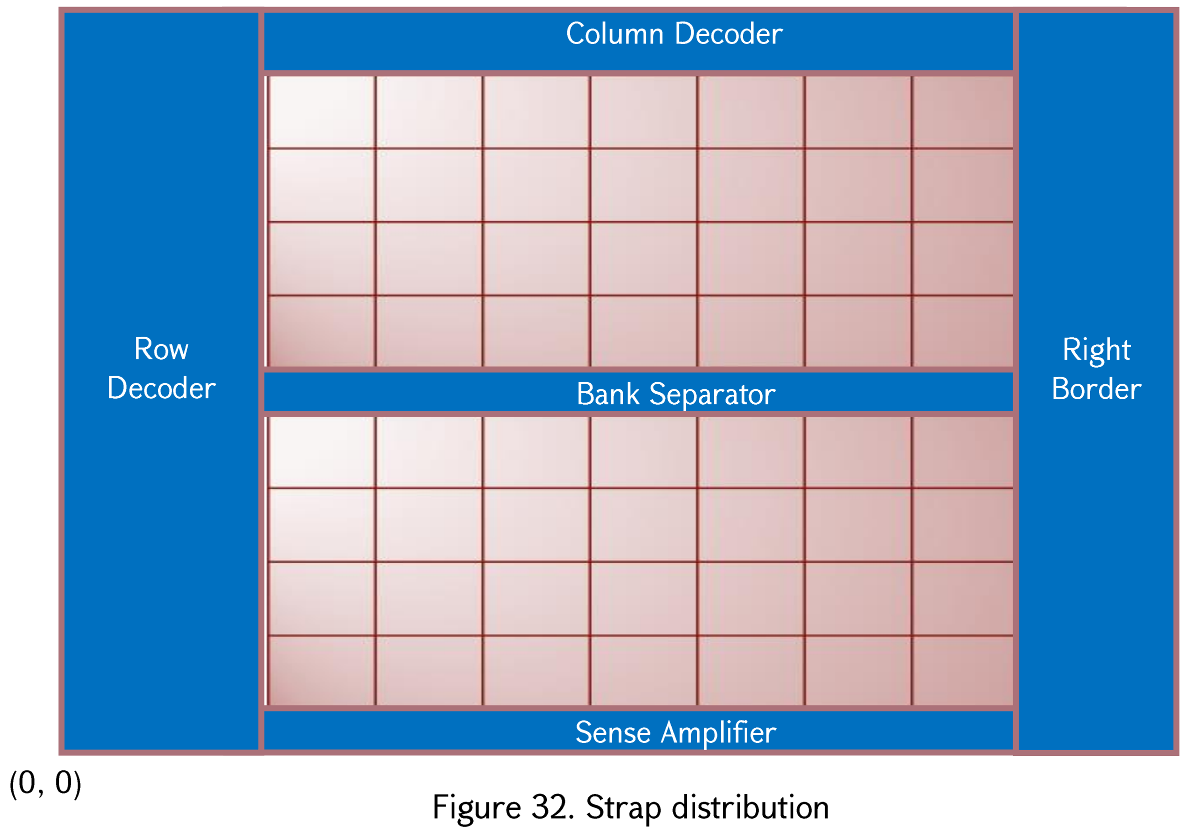


Figure 32. Strap distribution

5.3 Physical-Aware Multi-Level Fault Diagnosis and Localization Flow

The problem of fault diagnosis in memories is of prime importance in connection with the increasing density of embedded memories and their dominating portion in SoCs. In this work, a multi-level fault diagnosis flow is proposed [122] which provides comprehensive information about the fault including its type, the corresponding defect classification, physical location of the fault as well as aggressor cell information in case of coupling faults.

Figure 33 shows the levels of multi-level fault diagnosis flow. Depending on how many levels of diagnosis is needed to be achieved, it might be required to apply additional test algorithms or use memory scrambling information. The experiments showed that in order to obtain complete information on a fault, all the seven levels of diagnosis need to be applied.

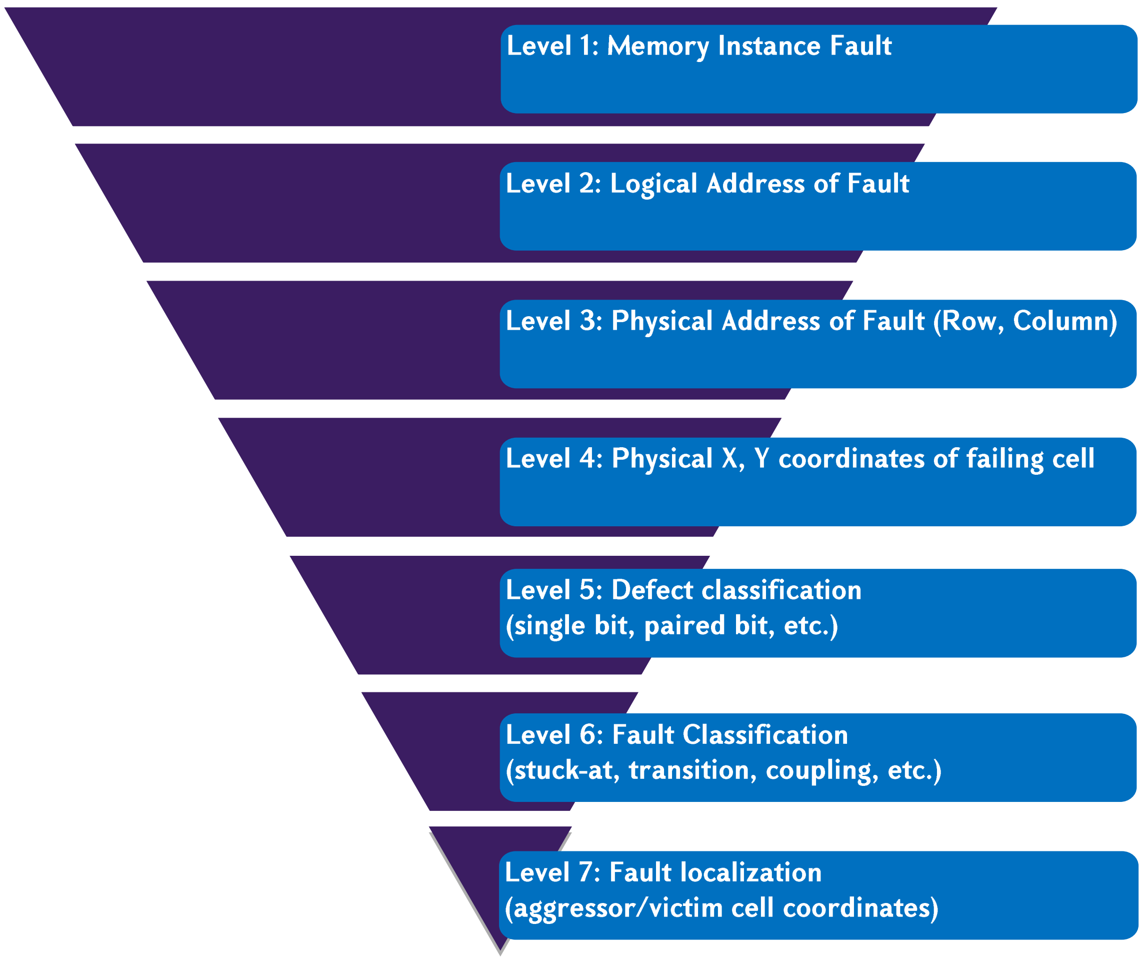

Level 1: Memory Instance Fault
Level 2: Logical Address of Fault
Level 3: Physical Address of Fault (Row, Column)
Level 4: Physical X, Y coordinates of failing cell
Level 5: Defect classification (single bit, paired bit, etc.)
Level 6: Fault Classification (stuck-at, transition, coupling, etc.)
Level 7: Fault localization (aggressor/victim cell coordinates)

Figure 33. Multi-level fault diagnosis flow

Level 1: Identifies if the memory instance has a fault or not. One of the most common ways to identify whether the memory has a fault or not is to run built-in self-test (BIST) system of the chip.

Level 2: In case of faults, identifies the logical address of the fault (e.g., address 7, data bit 4). Logical address of the fault is usually obtained by running the test system in diagnostic mode. One of the common approaches to obtain diagnostic data is "Stop On Nth Error (SONE)" mechanism. It runs the BIST and stops on the specified Nth fault (by ignoring the detected first N-1 faults) and reports the corresponding diagnostic information of the Nth fault (e.g., logical address of the fault, which March element and which operation in the March element detected the fault, etc.). Using this technique, the diagnostic information of all the faults in the memory can be achieved by running SONE mechanism with value 1, then 2, and so forth until all the faults of the memory are detected and diagnosed.

Level 3: Identifies physical address (i.e., physical row and column position) in the memory. This step can be done only if memory scrambling information is available.

Level 4: Identifies physical X, Y coordinates of the failing memory cell (X, Y shows the center of the memory cell). This step can be done in case memory scrambling information is available including strap distribution and memory cell size information (cell height, cell width).

Level 5: Classifies the defect distribution which can be (see Figure 34):

a. single bit;
b. vertical paired bits;
c. horizontal paired bits;
d. quadro bits;
e. column fail;
f. row fail.

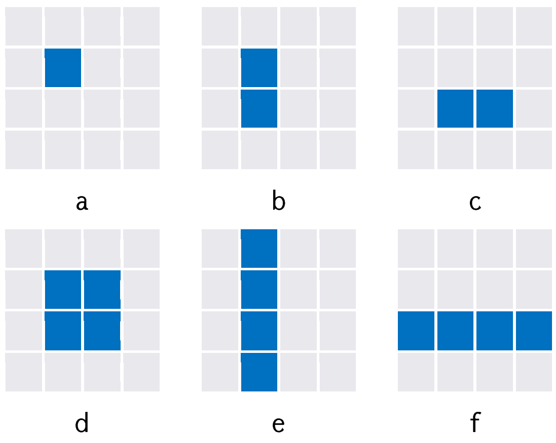


Figure 34. Defect classification types

Level 6: Identifies the fault type, i.e., whether it is stuck-at, transition, coupling or other type of fault. In this step, special March test algorithms are applied, which are able to fully classify the fault types, i.e., distinguish between fault types.

Level 7: In case of coupling faults, identifies the logical and physical addresses of aggressor cells. In this step, special March-based test algorithms are applied to localize the aggressor cell of the fault.

The proposed multi-level fault diagnosis flow allows to solve the below mentioned problems:

1. Defect classification allows to understand the distribution of defects and provides an initial view on the defect types and causes. For example, if there is a row fail, then someone can conclude that the issue might be connected to broken word-line.
2. Identification of physical address (Row, Column) of the fault provides actual place of the fault in the memory. Since if only logical address (Address, Bit) of the fault is known then there is no information where the fault is located in the memory.
3. If physical coordinates of the fault are known, then someone can do physical failure analysis (PFA) with laser cutting to identify the actual cause of the fault.
4. Fault classification (fault type identification) and fault localization (aggressor cell identification) allow to understand the nature and cause of the fault (i.e., the defect type causing the fault).

5.4 Test Algorithms for Fault Classification

This section describes March test algorithms able to provide classification for static, two-operation dynamic and FinFET-specific faults using the proposed fault diagnosis flow.

Note that, the below test algorithms are solving specific problems by applying Level 6 of Multi-level fault diagnosis flow, while the proposed flow is generic and can be applied also with other test algorithms targeting other set of faults.

5.4.1 Classification of Static Faults

Table 39 shows March FD test algorithm of complexity 35N (where N is the number of memory addresses) developed for classification of unlinked static faults [123]. March FD is considered as a full diagnosis algorithm since its corresponding fault dictionary consists of unique March syndromes. This means that if a static fault is detected after running March

Table 39. March FD (35N)

Address direction	Operations
$\Uparrow$	W0
$\Uparrow$	R0
$\Uparrow$	R0, W1, W1, R1
$\Uparrow$	R1, W0, R0, W1
$\Uparrow$	R1, W1
$\Uparrow$	R1
$\Uparrow$	R1
$\Uparrow$	R1, W0, W0, R0
$\Uparrow$	R0, W1, R1, W0
$\Uparrow$	R0, W0
$\Uparrow$	R0
$\Downarrow$	R0, W1, W1, R1
$\Uparrow$	R1
$\Downarrow$	R1, W0, W0, R0
$\Uparrow$	R0

FD then from the obtained March syndrome it will be possible to uniquely identify the exact type of the fault.

5.4.2 Classification of Static and Two-Operation Dynamic Faults

Tables 40, 41 and 42 show respectively, March VLP1 (26N), March VLP2 (26N) and March VLP3 (22N) test algorithms for classification of unlinked static and two-operation dynamic faults [124]. These algorithms must be applied sequentially (first March VLP1, then March VLP2, then March VLP 3) and join the obtained March syndromes. The overall complexity of these test algorithms is 74N. Having three test algorithms instead of one

Table 40. March VLP1 (26N)

Address direction	Operations
⇑	W0
⇑	R0, W1, W1, R1, W1, W1
⇑	R1, W0, W0, R0, W0, W0
⇓	R0, W1, W1, R1, W1, W1
⇓	R1, W0, W0, R0, W0, W0
⇑	R0

Table 41. March VLP2 (26N)

Address direction	Operations
⇑	W0
⇑	R0, W1, R1, W1, R1, R1
⇑	R1, W0, R0, W0, R0, R0
⇓	R0, W1, R1, W1, R1, R1
⇓	R1, W0, R0, W0, R0, R0
⇑	R0

allows to avoid having long Test Algorithm Register (TAR) in programmable BIST systems [92]. If the area is not an issue, then one can concatenate these test algorithms and program it as a single test algorithm.

Table 42. March VLP3 (22N)

Address direction	Operations
⇑	W0
⇑	R0, W0, W1, W0, W1
⇑	R1, W1, W0, W1, W0
⇓	R0, W0, W1, W0, W1
⇓	R1, W1, W0, W1, W0
⇑	R0

The classification of unlinked static and two-operation dynamic faults is done with multiple phases. In Phase 1, after running March VLP1, March VLP2 and March VLP3 test algorithms the faults will be detected and partially diagnosed. Partial diagnosis means that not all the faults are uniquely identified since there are March syndromes that correspond to more than one faults. In other words, the whole set of faults will be divided into groups, where two faults will be in the same group if their corresponding March syndromes are the same.

To reach full diagnosis an additional phase is applied. Since after Phase 1 logical addresses of victim cells of each detected faults are known, then March-based test algorithms [38] are applied instead of March test algorithms. This means that instead of running the test algorithm through the whole space of the memory, only adjacent cells of the victim cell will be accessed. Of course, to identify the logical addresses of those adjacent cells (to apply March-based test algorithms through those addresses), the memory scrambling information is necessary. [124] describes March-based test algorithms for Phase 2 of a constant complexity (no dependency on number of memory addresses, it just applies 325 Write/Read operations) to reach full diagnosis of unlinked static and two-operation dynamic faults. Also, to localize the aggressor cells of the considered faults, Phase 3 is proposed which again applies March-based test algorithms of a constant complexity (181 Write/Read operations).

In addition, in [90], [125]-[128], several March-based and multi-phase test algorithms are proposed for diagnosis and localization of traditional, static and dynamic faults. The test algorithms and their corresponding fault dictionaries can be found in Appendix B (see Tables B1-B13).

5.4.3 Classification of FinFET-Specific Faults

Table 43 describes test algorithm March FFDD (42N) [122] for classification of FinFET-specific faults, i.e., it identifies whether the detected fault is FinFET-specific or not. In other words, the structure of March FFDD is organized in a way that it allows to construct unique March syndromes for FinFET-specific faults. “FFDD” stands for FinFET Detection and Diagnosis. To the best of our knowledge, this work is the first that proposes classification algorithm for FinFET-specific faults. The previously known test algorithms cannot diagnose FinFET-specific faults since those are not capable to detect FinFET-specific faults and consequently cannot distinguish between planar-based and FinFET-specific faults.

Table 43. March FFDD (42N)

Address direction	Operations
⇑	W0
⇑	R0, W0
⇑	R0
⇑	R0, W1, R1, R1, R1, R1, R1, R1, R1, R1, R1
⇑	W0, R0
⇑	R0, W1, R1, R1
⇑	R1, W0, R0
⇑	R0, W1
⇑	R1
⇑	R1, W0, R0, R0, R0, R0, R0, R0, R0, R0, R0
⇑	R0
⇓	W1, R1
⇑	R1

Table 44 shows one part of the corresponding fault dictionary created for March FFDD. In the table, the first column shows the fault family, i.e., whether it is static, two-operation dynamic or FinFET-specific fault. The second column shows the FP, while the third column shows the corresponding March syndrome for a given FP. In the table, the March syndromes are unique and allow to identify FinFET-specific faults.

The selection of which of the proposed fault diagnosis algorithms to use depends on different factors. For example, if dealing with FinFET-based memories (16nm and below nodes), then it is recommended to run March FFDD test algorithm. For planar-based memories (28nm and above nodes), it is recommended to run March FD and March VLP1-VLP3 algorithms. If the target is to address only static faults then March FD should be run, otherwise if static and two-operation dynamic faults are considered then March VLP1-VLP3 algorithms should be run.

Table 44. Fault Dictionary of March FFDD

Fault Family	FP	March syndrome
Static	<R0/1/1>	111000000000110001100111111111100
Static	<R0/1/0>	001000000000010000100011111111100
Static	<1; R1/0/0>$_{a<v}$	000111111111000000011000000000011
Static	<R1; 0/1/->$_{a<v}$	001000000000000000000000000000100
Static	…	…
Two-operation dynamic	<R0R0/1/1>	000000000000000000000011111111100
Two-operation dynamic	<R1R1/0/0>	000011111111000110000000000000000
Two-operation dynamic	…	…
FinFET-specific	<R0^3/1/1>	000000000000000000000001111111100
FinFET-specific	<R1^8/0/1>	000000000001000000000000000000000
FinFET-specific	…	…

5.5 Related Experiments

In the experiments, Synopsys Star Memory System [129] is used that allows to create test patterns which can be applied to the chip in the automatic test equipment (ATE) environment. It can process tester output files providing fault detection, classification and localization.

The steps of the diagnosis flow are the following:

Step 1: Select a test algorithm for diagnosis flow. It can be March FD or March VLP1-VLP3 or March FFDD.

Step 2: Run Diagnosis pattern that does the following steps:

a. Set SONE = 1 and run the BIST. Then register the cycles number on which the BIST is failing. Later this cycle number is used to recover all the necessary information about the fail (i.e., on which address and data bit the BIST failed, which operation of the test algorithm failed, etc.). More information about SONE is provided in description of Level 2.
b. Increment SONE value by one and repeat Step 2.a. until all the faults are detected.
c. After finalizing the run of the BIST for all faults a datalog file is generated with the registered cycle numbers.

Step3: Based on the given datalog file the corresponding fault information is provided:

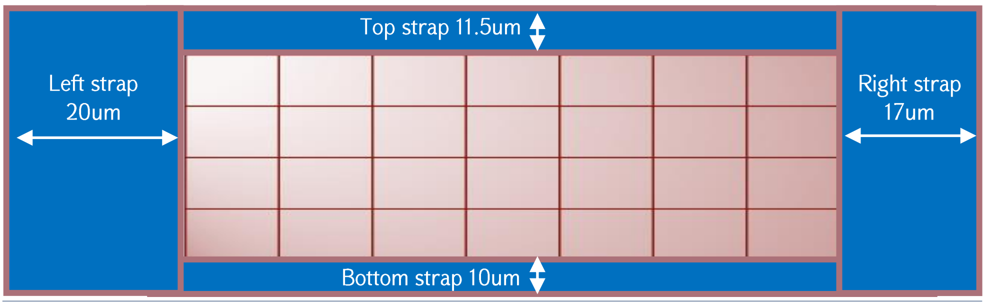
a. Fault signature, fault type, faulty address, faulty data bit.
b. Then using memory scrambling information physical row, physical column, physical X, Y coordinates of the faulty cell are calculated.

Step 4 (optional, applicable only when coupling faults are detected in Step 3): A March-based test algorithm is run for each coupling cell fault and the corresponding datalog file with failing cycle numbers is generated.

Step 5: Based on the given datalog file the corresponding fault information is provided for aggressor cell:

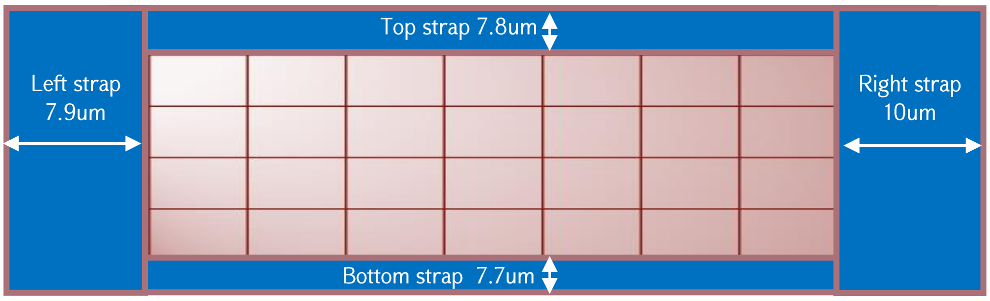
a. address, data bit, physical row, physical column, physical X, Y coordinates of the aggressor cell.

The proposed diagnosis flow has been applied to FPGA board which contains two 16nm (FinFET-based) SRAMs: one single-port (RAM_1p) memory tested by SMS1 system and one 2-port (RAM_2p) memory tested by SMS2. Figures 35 and 36 describe the memory configuration and some of the scrambling types for those memories. As it is shown in Figure 35, RAM_1p has regular Column and Row scrambling, while IO cell scrambling is irregular (i.e., logical data 0 corresponds to physical IO cell 0, … logical data 7 corresponds to physical



- Number of words = 128
- Number of bits per word = 16
- Words in one row (column multiplexing) = 4
- Number of physical rows = 32
- Number of physical columns = 64
- Column scrambling = { 0 1 2 3 }

Figure 35. Configuration and scrambling information of RAM_1p



- Number of words = 32
- Number of bits per word = 12
- Words in one row (column multiplexing) = 4
- Number of physical rows = 8
- Number of physical columns = 48
- Column scrambling = { 0 1 2 3 }
- Row scrambling = { 0 1 3 2 4 5 7 6 }
- IO cell scrambling = { 0 1 2 3 4 5 6 7 8 9 10 11 }
- Bit-cell height = 0.56um
- Bit-cell width = 1.0um

Figure 36. Configuration and scrambling information of RAM_2p

IO cell 7, while logical data 8 corresponds to physical IO cell 15, ..., logical data 15 corresponds to physical IO cell 8). Figure 36 shows that RAM_2p has regular Column and IO cell scrambling, while Row s crambling is irregular (i.e., logical row 0 corresponds to physical row 0, logical row 1 corresponds to physical row 1, logical row 2 corresponds to physical row 3, logical row 3 corresponds to physical row 2, etc.). Both memories have interleaved structure, i.e., the cell distribution in each memory row has the following order: 0th bit of word 0, 0th bit of word 1, ..., 1st bit of word 0, 1st bit of word 1, ..., etc.

First experiment. In this experiment, one fault in each memory of the FPGA is injected. The first fault is stuck-at 0 injected in single-port memory in a cell with coordinates (Row = 2, Column = 19). And the second fault is stuck-at 1 injected in 2-port memory in a cell with coordinates (Row = 1, Column = 15). After fault injection, a fault diagnostic pattern has been generated, which uses March FD (35N) March test algorithm. The pattern is run on both memories, and as a result memory test data have been generated for them. After processing datalog, the fault classification is done. Table 45 shows the obtained fault classification results.

Table 45. Experiment 1. Classification of Stuck-At Faults

#	SMS	Memory	Address	Bit	Row	Column	X	Y	FFM	FP	Fault Signature
1	SMS1	RAM_1p	11	4	2	19	29.75um	10.9um	SF	<1/0/->	00110111100100011100
2	SMS2	RAM_2p	7	3	1	15	23.4um	8.54um	SF	<0/1/->	11001000011011100011

As shown, the faulty cells have been identified, fault types and corresponding fault signatures have been generated. As described in the fault diagnosis flow, first faulty address and faulty data bit are obtained from the test run and then since memory scrambling information is available it is possible also to identify the corresponding physical row, physical column, as well as physical X, Y coordinates of the faults. The steps of the calculations for RAM_1p are the following:

1. From datalog file Address = 11 and Bit = 4 is obtained.
2. Using memory structural information, the corresponding physical row and physical column are calculated, i.e., (Address = 11, Bit = 4) → (Row = 2, Column = 19).
    a. Since "Words in one row" = 4 and Row scrambling is regular, then Address 11 is located in Row 2, i.e., 11 (Address) / div 4 (Words in one row) = 2 (Row).

    b. Similarly, due to interleaved structure of the memory, Bit 4 of Address 11 is located in Column 19, i.e., 4 (Words in one row) * 4 (Bit) + (11 (Address) % 4 (Words in one row)) = 19 (Column).

3. Then using memory topological information, the corresponding X, Y coordinates are calculated, i.e.:
    a. X = 20 (Left strap width) + 19.5 (19 and half cell) * 0.5 (Bit-cell width) = 29.75um;
    b. Y = 10 (Bottom strap height) + 2.5 (2 and half cell) * 0.36 (Bit-cell height) = 10.9um.
4. Defect classification: Both faults are considered as "single bit".
5. March syndrome 00110111100100011100 is obtained for RAM_1p memory which according to fault dictionary of March FD (see [123]) corresponds to SF <1/0/-> fault, while March syndrome 11001000011011100011 obtained for RAM_2p corresponds to SF <0/1/-> fault.
6. No need to apply Fault localization pattern (Level 7) since both faults are single-cell faults and there are no aggressor cells to be detected.

Second experiment. In this experiment, dynamic transition coupling fault has been injected in single-port memory with victim cell coordinates (Row = 17, Column = 2) and aggressor cell coordinates (Row = 18, Column = 2). In this experiment March VLP1-VLP3 test algorithms are run. The obtained fault classification results are shown in Table 46.

In this case, combined March syndrome for March VLP1-VLP3 0000000000000010000000000000000 is obtained which corresponds to dynamic fault

Table 46. Experiment 2. Classification of Dynamic Coupling Fault

#	SMS	Memory	Address	Bit	Row	Column	X	Y	FFM	FP	Fault Signature
1	SMS1	RAM_1p	70	0	17	2	21.25um	16.3um	dCFir	<1;1W0R0/0/1>	0000000000000010 000000000000000

dCFir <1;1W0R0/0/1>. In this case again address, data bit, physical row, physical column, physical X, Y coordinates of the faulty cell are identified.

Since this is a coupling fault it requires Fault localization pattern (Level 7) to be run for identification of aggressor cell. After running Fault localization pattern on the faulty memory, the corresponding datalog is generated as an output. After processing the datalog, the fault localization (i.e., identification of aggressor cell information) is performed. Table 47 shows the obtained fault localization result. Since memory scrambling information is available then logical and physical addresses as well as physical coordinates of aggressor cell are also identified.

As it is seen from Table 47 victim and aggressor cells of the fault are located on the same column (Column 2), and vertically are adjacent (victim cell is on Row 17, aggressor cell is on

Table 47. Experiment 2. Localization of Aggressor Cell

#	SMS	Memory	Victim Row	Victim Column	Aggressor Row	Aggressor Column	Aggressor Address	Aggressor Bit	Aggressor X	Aggressor Y
1	SMS1	RAM_1p	17	2	18	2	74	0	21.25um	16.66um

Row 18). So this is considered as "vertical paired bit" (see Level 5: Defect classification).

Third experiment. In this experiment, one static, one dynamic and one FinFET-specific faults are injected. After injecting the faults, March FFDD test algorithm is run to diagnose the faults. The obtained results are shown in Table 48. Similar to the previous experiments all the diagnosis information about the fault is identified. So, March FFDD was able to differentiate the exact fault type independent of whether it is static, dynamic or FinFET-specific fault. In this experiment, it is important to mention that if memory scrambling information is not considered then for a given faulty address and bit a wrong physical row will be provided (since as it is shown in Figure 36, the Row scrambling is not regular). For example, without using memory scrambling information the physical row of CFdsr fault

Table 48. Experiment 3. Classification of Static, Dynamic and FinFET-Specific Faults

#	SMS	Memory	Address	Bit	Row	Column	X	Y	FFM	FP	Fault Signature
1	SMS2	RAM_2p	14	2	2	10	18.4um	9.1um	CFdsr	<R1; 0/1/->	0010000000000000 00000000000000100
2	SMS2	RAM_2p	23	1	5	7	15.4um	10.78um	dRDF	<R0R0/1/1>	000000000000000 000000011111111100
3	SMS2	RAM_2p	25	0	7	1	9.4um	11.9um	dDRDF-8	<R1^8/0/1>	00000000000100000 0000000000000000

should be reported 3 (according to simple calculations done in First experiment, 14 (Address) div 4 (Words in one row) = 3 (logical row)), but since there is a mapping between logical to physical rows (i.e., logical row 2 corresponds to physical row 3 and logical row 3 corresponds to physical row 2), the final result for physical row will be 2 and not 3.

As it is shown in Table 48, 2 single-cell faults (dRDF and dDRDF-8) and one coupling fault (CFdsr) are identified. For coupling fault, Fault localization pattern (Level 7) needs to be run for identification of aggressor cell. After running Fault localization pattern on the faulty memory, the corresponding datalog is generated as an output. After processing the datalog, the fault localization (i.e., identification of aggressor cell information) is performed. Table 49 shows the obtained fault localization results. As it is seen from the table victim and aggressor cells of CFdsr fault are located on the same row (Row 2), and horizontally are adjacent (victim cell is located in Column 10, aggressor cell is on Column 9). So, this is considered as "horizontal paired bit" (see Level 5: Defect classification). To summarize, in this experiment there are two "single bit" faults (dRDF and dDRDF-8) and one "horizontal paired bit" fault (CFdsr).

Table 49. Experiment 3. Localization of Aggressor Cell

#	SMS	Memory	Victim Row	Victim Column	Aggressor Row	Aggressor Column	Aggressor Address	Aggressor Bit	Aggressor X	Aggressor Y
1	SMS2	RAM_2p	2	10	2	9	13	2	17.4um	9.1um

These experiments proved the validity and correctness of proposed fault diagnosis flow. In addition, the following important observations and conclusions can be done:

1. Without memory scrambling information, i.e., without memory structural information (mapping between memory logical and physical addresses) and memory topological information (strap distribution), it is not possible to identify the physical row, physical column as well as physical X, Y coordinates. So, it is crucial to have accurate and complete memory scrambling information to provide complete diagnosis information, otherwise it will be incomplete or incorrect (e.g., information about Levels 3, 4, 5 cannot be provided).

2. Without having special test algorithms (e.g., March FD, March VLP1-VLP3, March FFDD) it is not possible to detect and moreover to diagnose static, dynamic and FinFET-specific faults. Without special fault diagnosis test algorithms fault detection can be escaped or fault can be classified incorrectly.
3. The advantage of the proposed diagnosis flow is that it is able to provide complete information about the fault. Other known diagnosis flows have the following limitations:
    a. Not all 7-level information about the fault is provided.
    b. The fault scope where the fault can be diagnosed is limited only to traditional faults (e.g., [38], [115]) or static faults (e.g., [114], [123]) or dynamic faults (e.g., [124]). While the proposed flow (based on March FFDD test algorithm) allows to provide information also about FinFET-specific faults which is too important when dealing with nowadays FinFET-based memories.

The proposed flow does not have essential impact on BIST area since the flow implementation is done in the software. The only feature required from the BIST is to have SONE diagnostics capability which itself does not essentially impact the BIST area. Regarding the run time, though the BIST runs in average twice longer test algorithm for fault diagnosis compared to fault detection case, but since the fault diagnosis is done mainly during the PFA (and not during the volume production) then having the mentioned longer run time is mainly acceptable. If datalog parsing and fault diagnosis information generation by the software (which is part of STAR Memory System) are also considered, then the run time for complex projects takes maximum 1 minute.

The proposed multi-level fault diagnosis flow has been applied to different chips of 45nm, 28nm and 16nm (FinFET-based) technology and enabled to do successful PFA. Some FinFET-specific faults in 16nm chips were possible to identify using March FFDD test algorithm.

5.6 Real-Life Case Scenarios

Figure 37 shows Scenario 1 where after applying fault diagnosis flow it turned out that all the failing cells are located in the borders of the memories. Based on this information, PFA showed that those border cells are damaged. One of the solutions to this problem is to put dummy (extra) rows and columns at the borders of memory to protect functional cells from damaging.

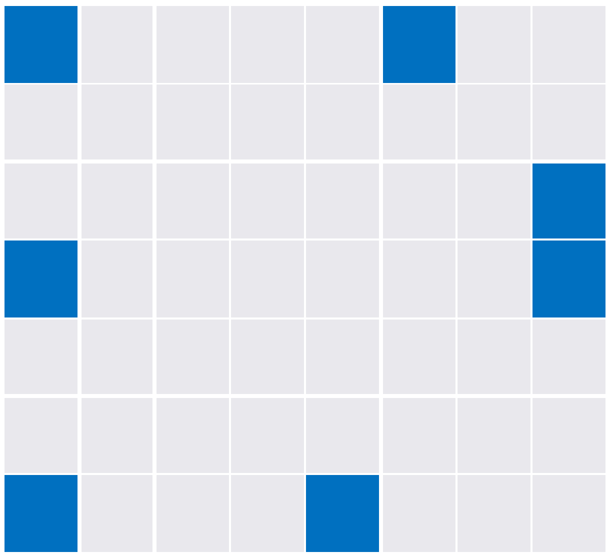


Figure 37. Scenario 1: Cells at border are damaged

Figure 38 shows Scenario 2 where all the cells marked in blue (aggressor cells) in a row were impacting a victim cell marked in red. The fault was sensitized under the following condition: victim cell value was flipping from 0 to 1 when a Write or Read operation was applied to one of the aggressor cells (marked as W/R in Figure 38). The interesting point is that the victim cell was flipping from 0 to 1, while opposite transition from 1 to 0 was not happening. The PFA showed that the victim cell was a weak cell and was impacted when the word-line of that row was active high. This was the reason that there was a faulty flip only from 0 to 1 direction since when the victim cell had value 1, the active word-line (containing value 1) was not impacting the victim cell value. Also, having word-line value impacting the weak cell value clarifies the fact that there was no difference whether adjacent cell of the victim cell (which is common for coupling faults) or another cell of the row far from victim cell was impacting the victim cell value.

Figure 39 shows Scenario 3 where all cells marked by blue (aggressor cells) in a column were impacting a victim cell marked in red. The fault was sensitized under the following condition: victim cell value was flipping from 1 to 0 when several Read operations were applied to the aggressor cells containing value 0 (marked as R0 in Figure 39). The PFA showed that there is a bit-line leakage happening at victim cell. Thus, applying R0 operation to aggressor cells (no matter in which order) was inducing a current leakage at the victim cell and after applying several Read operations, the victim cell value was finally flipping from 1 to 0.

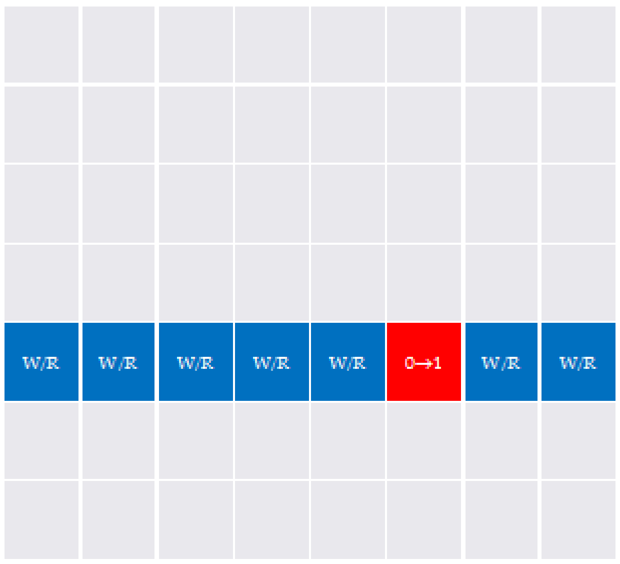


Figure 38. Scenario 2: Weak cell impacted by other cells of the same row

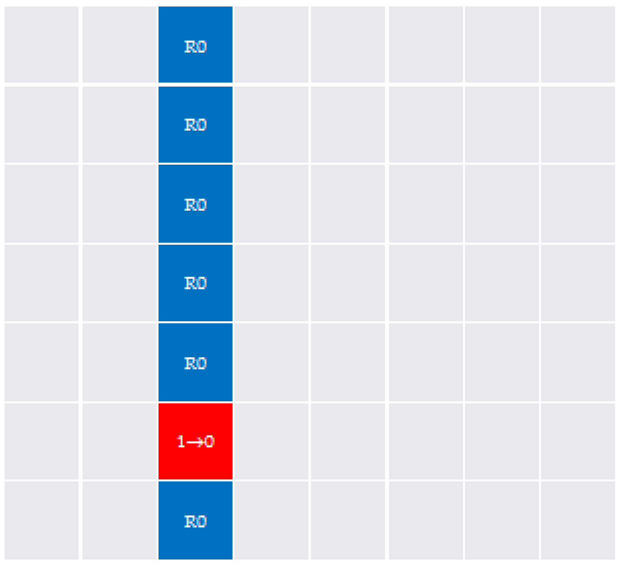


Figure 39. Scenario 3: Bit-line leakage fault

Conclusions

1. Concept of fault diagnosis and localization is provided.
2. The importance of memory scrambling information for fault diagnosis is demonstrated and clarified. Based on that a new generic multi-level fault diagnosis flow is proposed applicable both for planar-based and FinFET-based memories.
3. Special test algorithms for classification of static, dynamic and FinFET-specific faults are developed.
4. The suggested flow is validated on a 16nm (FinFET-based) FPGA board. The flow has also been applied to numerous chips of 45nm, 28nm and 16nm technologies enabling successful physical failure analysis (PFA).
5. Some real-life case scenarios of the flow application are presented illustrating the effectiveness of the approach.

CHAPTER 6. A HIERARCHICAL TEST METHODOLOGY

6.1 A Universal Hierarchical Architecture for Built-In Test

6.1.1 Known Hierarchical Test Approaches and Associated Standards

Usually, different approaches and standards are used for IP testing and integration into SoC [130]-[134]. At the chip level, the total number of test channels is limited such that all core-level test channels cannot be accessed at the same time.

Variety and complexity of used memories and IP cores, shrinking technologies and design complexity increasing in nanoscale SoCs make it crucial to have embedded in SoC test and repair solutions kept up with the advances in order to consistently and continuously provide chip quality and yield optimization. The embedded test approaches developed for designs just a few years ago are not sufficient for today's designs, which are bigger, faster, hierarchical and much more sensitive to area, timing and power [7], [133]. Similarly, the embedded test solutions developed, for example, for 28-nm technology node will not deliver the same level of test quality, diagnosis accuracy and repair efficiency for 14-nm and below technology nodes, as defects and failure mechanisms change with process technologies shrink.

In [131], authors proposed SoC testing with channel sharing/broadcasting methodology. This solution is good for medium-sized designs that can still run ATPG at the chip level.

In [132], authors presented hybrid test methodology incorporating modular test and hierarchical test. A case study is illustrated to compare different test flows used in the EDT (Embedded Deterministic Test) compression based SoC testing context.

In general, hierarchical test gives designers flexibility to schedule test of individual interface IP cores and other cores for parallel and serial testing to optimize test time and power consumption during test [133], [134]. The flexible test schedule can significantly reduce test time, especially for designs with a large number of high-speed I/Os.

From the other side, there are certain limitations in SoC which should be taken into account when scheduling the test. Common types of SoC limitations are the following:

- Design limitation – e.g., limited number of test access mechanisms to test multiple IP cores;
- Test limitation – e.g., precedence constraint (e.g., test of IP2 should be run only after completing the test run on IP1);
- Resource limitation – e.g., power constraint (limitation on SoC consumed power when testing multiple IP cores in parallel).

In this work, a hierarchical test approach is proposed which takes into account all the above mentioned limitations and allows to do efficient test scheduling.

Due to increase of SoC complexities, IP level integration and test pattern porting from IP level into SoC level cannot be done manually anymore and efficient automation techniques are needed to meet time-to-market requirements.

There are three main standards (IEEE 1500 [135], IEEE 1149.1 (JTAG) [136], IEEE 1687 (IJTAG) [137]) which are addressing the requirements and providing solutions for SoC hierarchical test.

IEEE 1500 Standard. The IEEE 1500 hardware architecture [135] is comprised of an Instruction Register (the Wrapper Instruction Register), and two data registers, the Wrapper Bypass Register (WBY) and the Wrapper Boundary Register (WBR). The use of Core Data Registers (CDRs) is also anticipated by the standard. Access to these registers is provided via a set of wrapper interface ports. Figure 40 displays the architecture of IEEE 1500 Standard. There are two categories of wrapper interface ports:

- Wrapper Serial Ports (for serial access to the wrapper);
- Wrapper Parallel Ports (for parallel access to the wrapper).

IEEE 1149.1 (JTAG) Standard. This standard [136] defines test logic that can be included in an integrated circuit to provide standardized approaches to:

- testing the interconnections between integrated circuits once they have been assembled onto a printed circuit board or other substrate;
- testing the integrated circuit itself;
- observing or modifying circuit activity during the component's normal operation.

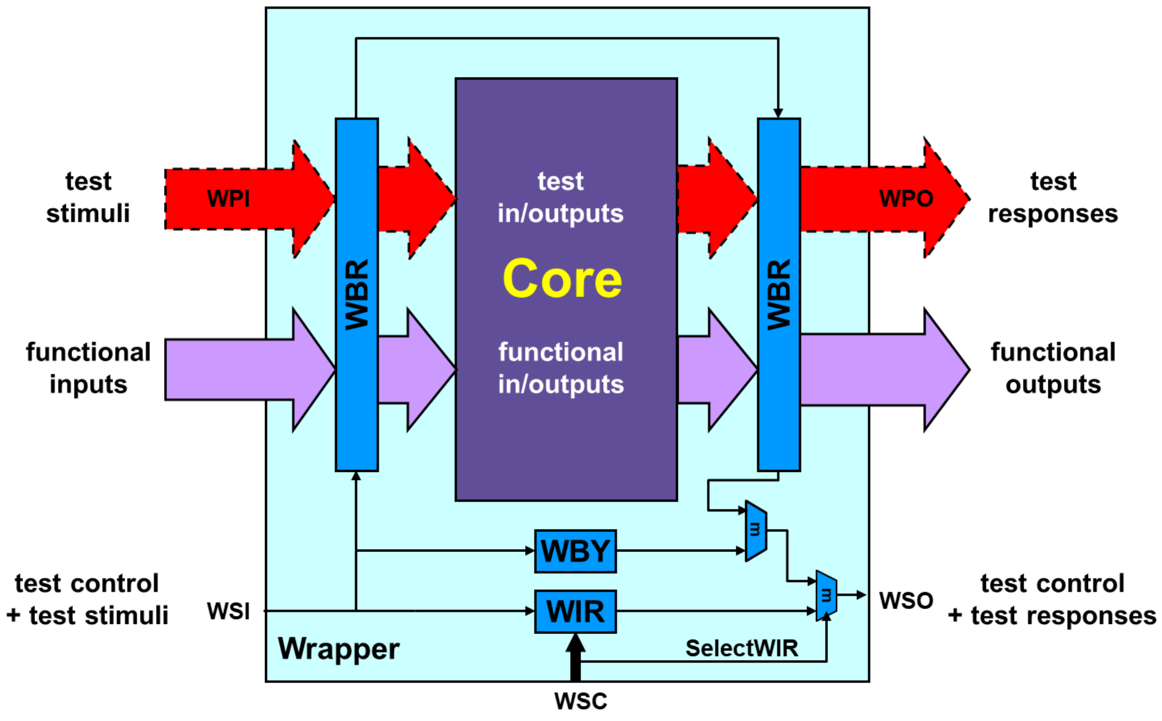


Figure 40. Architecture of IEEE 1500 Standard

The test logic consists of a boundary-scan register and other building blocks and is accessed through a Test Access Port (TAP). The TAP is a general-purpose port that can provide access to many test support functions built into a component, including the test logic defined by this standard. It is composed as a minimum of the three input connections and one output connection required by the test logic defined by this standard. An optional fourth input connection provides for asynchronous initialization of the test logic defined by this standard. The connections that form TAP are the following:

- Test clock input (TCK) - The test clock input (TCK) provides the clock for the test logic;
- Test mode select input (TMS) - The signal received at TMS is decoded by the TAP controller to control test operations.
- Test data input (TDI) - Serial test instructions and data are received by the test logic at TDI.
- Test data output (TDO) - TDO is the serial output for test instructions and data from the test logic;

- Test reset input (TRST) - The optional TRST input provides for asynchronous initialization of the TAP controller.

IEEE 1687 (IJTAG) Standard. This standard [137] develops a methodology for access to embedded instrumentation, without defining the instruments or their features themselves, via the IEEE 1149.1 test access port (TAP) and additional signals that may be required. The elements of the methodology include a description language for the characteristics of the features and for communication with the features, and requirements for interfacing to the features.

IEEE 1687 Standard facilitates access from the chip boundary to instruments embedded within the device. Achieving this access requires at least three portions of an access architecture:

- the device interface;
- the instrument interface;
- the access network that resides between the device interface and the instrument interface.

IEEE 1687 Standard covers the mechanisms to build, describe, and operate those three architectural elements. It can be connected to many types of controllers and interfaces, but pays attention to the very popular serial access networks used in IEEE 1149.1 and IEEE 1500 standards.

The main elements of the standard are:

- Instrument access network – which includes Segment-Insertion-Bit (SIB) to allow the overall scan chain to be of variable length;
- Instrument Connectivity Language (ICL) - which describes the elements that comprise the instrument access network as well as their logical (though not necessarily their physical) connections to each other and to the instruments at the endpoints of the network;
- Procedural Description Language (PDL) – which provides a means to define procedures to operate an instrument. These instrument procedures are documented

by PDL in terms of stimuli and expected responses for the ports and/or registers described in the ICL module for the instrument.

6.1.2 Universal Hierarchical Built-In Test (UHBIT) Architecture

Figure 41 shows the proposed UHBIT architecture [138] which is extension of unified BIST architecture presented in Section 3.7. At IP level it is based on IEEE 1500 standard, while at top level the test system consists of main Server and multiple Sub-Servers placed at the second level of hierarchy. Top level Server is connected to IEEE 1149.1 JTAG interface which is in charge of providing test patterns from the outside world. Usually SoCs are consists of Sub-Chips and the test patterns applied to a Sub-Chip can be reused at top level by porting those test patterns from Sub-Chip level to SoC level. The considered hierarchical test system has the following main capabilities:

- Unified test accessibility for different IP cores in SoC based on existing test standards (IEEE 1500, IEEE 1149.1 (JTAG), IEEE 1687 (IJTAG));

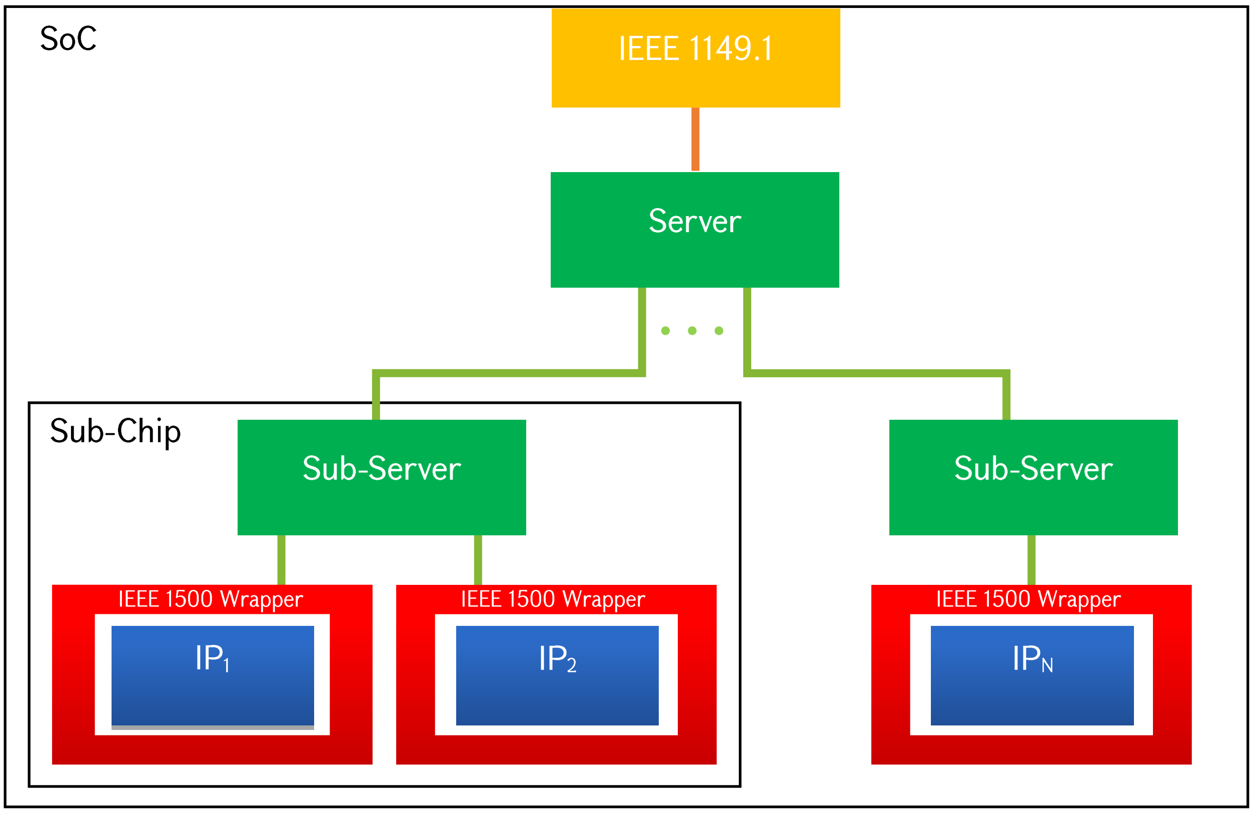


Figure 41. UHBIT architecture

- Pattern porting from IP and Sub-Chip level to SoC level which allows to reduce the design and test times;
- Language for describing the structural models of memories and IP cores (let us call it CSL – core structure language);
- Capability to create effective test scenarios under the presence of limited resources available in SoC.

CSL language. This language provides set of parameters which are necessary to describe the structure of a given IP [139]. It has the following sections:

- Port description – Name of ports and attributes (function, direction, range, etc.);
- Core internal serial test data – CDR and its attributes;
- Isolation chain – WBR or CDR ordering;
- WDR – Wrapper Data Register (WDR) chain;
- Comments – line (//...) and block (/* ... */) comments are supported.

Some examples of format for ports are adduced below:

```
Port {
 clk_1 {
  PortInfo = “Clk from PLL"
  Direction = Input
  Tag = Clock
  MinFreq = 50.0
  MaxFreq = 150.0
 }
 din {
  PortInfo = “Input data"
  Range = "[7:0]"
  ExpandFactor = 1
  Direction = “Input"
  Tag = Static
```

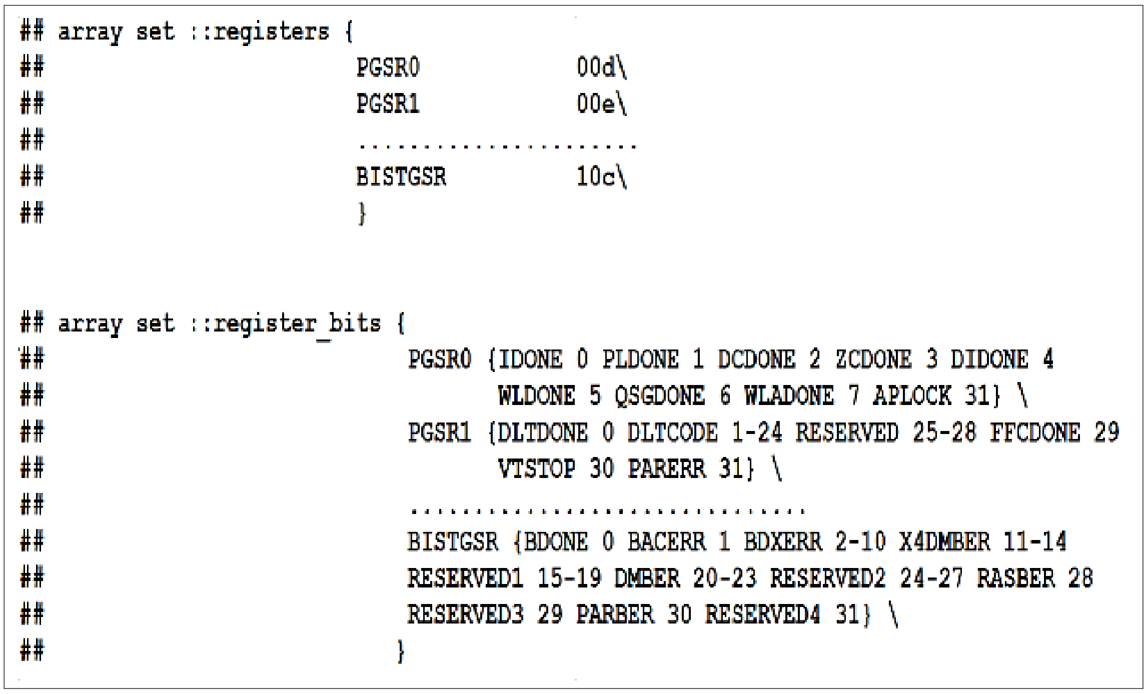
```
      IsolationCell = DynamicTestIn
    }
  }
```

CPL (Core Pattern Language) is used to describe test patterns with high level description. CPL provides:

- Verilog task-like test pattern description flow;
- Supporting of binary, decimal, octal, hexadecimal and string type operands;
- Support of expressions (concatenation, replication and grouping);
- Support of bit-select and range-select;
- Support of comments;
- Tcl-based extension and parameterization;
- Declaration of signal groups;
- Support of include files.

In order to generate IP test patterns in a modular way and avoid massive changes when a single register address or port range in an IP is changed, it is suggested to have 3 separate views:

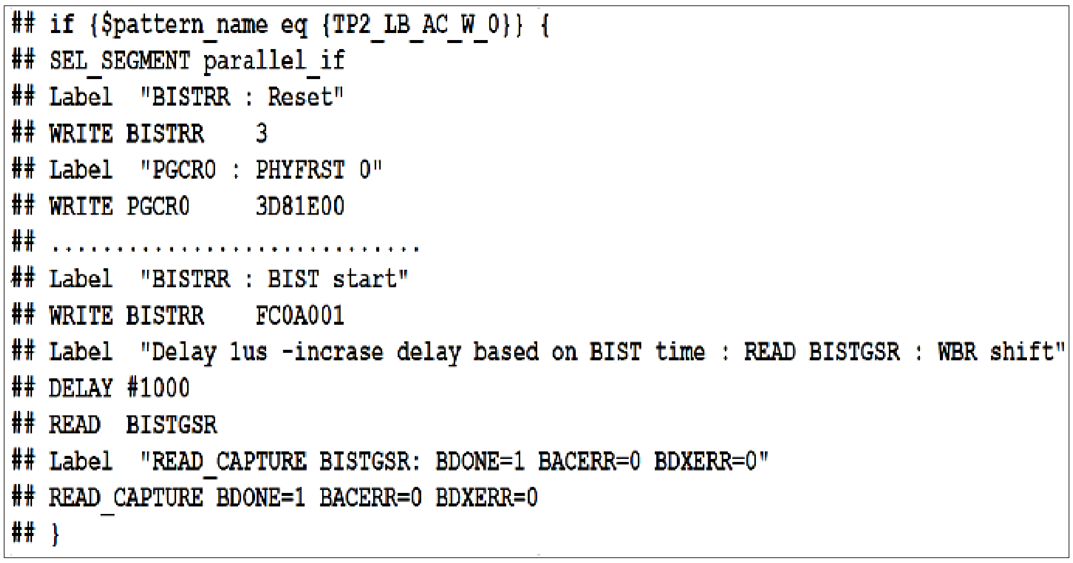
- Register CPL – contains two types of information: 1. Mapping between register names and their addresses, 2. Field names of each registers. An example of Register CPL is shown in Figure 42. For instance, register PGSR0 has address 00d, while fields of PGSR0 are presented in register_bits part (0th bit is IDONE, 1st bit is PLDONE, 2nd bit is DCDONE, etc.).
- Test CPL – Sequence of test operations. Example of Test CPL is shown in Figure 43. As it is seen from the figure, the test patterns are described using high-level operations, like WRITE, READ, DELAY, etc.
- Setup CPL – Contains definitions of high-level operators (READ, WRITE, etc.) used in Test CPL. Implementation of those operations depends on test protocol (e.g., APB protocol). Part of WRITE procedure defined in setup CPL is shown in Figure 44.



```
## array set ::registers {
##                      PGSR0           00d\
##                      PGSR1           00e\
##                      ......................
##                      BISTGSR         10c\
##                      }


## array set ::register_bits {
##                          PGSR0 {IDONE 0 PLDONE 1 DCDONE 2 ZCDONE 3 DIDONE 4
##                                 WLDONE 5 QSGDONE 6 WLADONE 7 APLOCK 31} \
##                          PGSR1 {DLTDONE 0 DLTCODE 1-24 RESERVED 25-28 FFCDONE 29
##                                 VTSTOP 30 PARERR 31} \
##                          ..............................
##                          BISTGSR {BDONE 0 BACERR 1 BDXERR 2-10 X4DMBER 11-14
##                          RESERVED1 15-19 DMBER 20-23 RESERVED2 24-27 RASBER 28
##                          RESERVED3 29 PARBER 30 RESERVED4 31} \
##                         }
```

Figure 42. Example of Register CPL

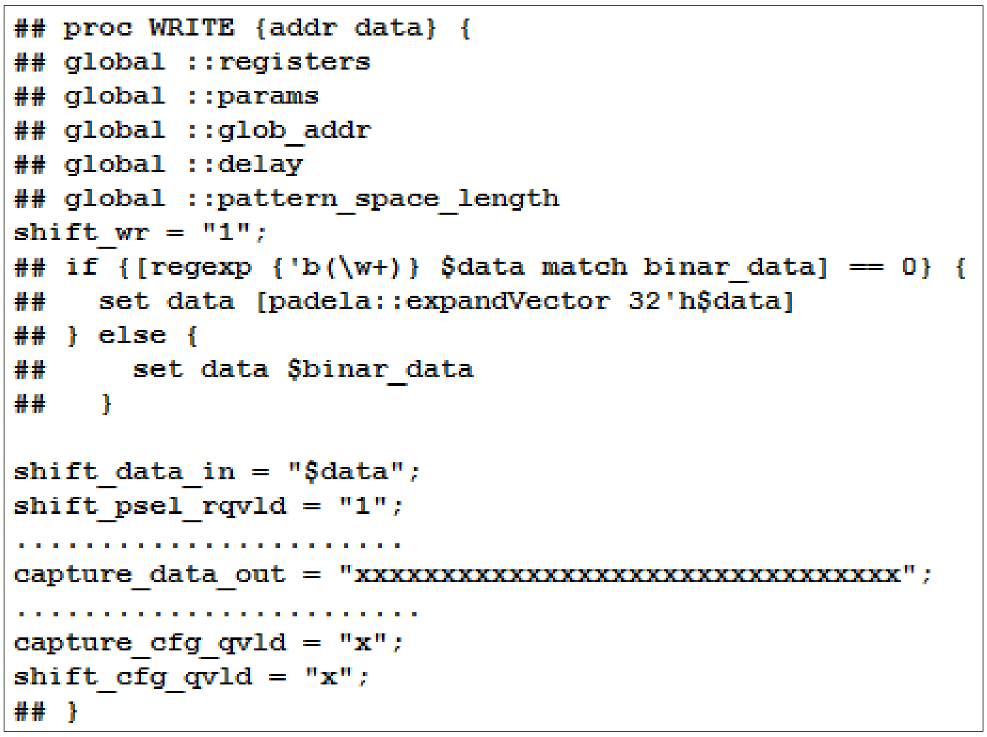


```
## if {$pattern_name eq {TP2_LB_AC_W_0}} {
## SEL_SEGMENT parallel_if
## Label  "BISTRR : Reset"
## WRITE BISTRR    3
## Label  "PGCR0 : PHYFRST 0"
## WRITE PGCR0     3D81E00
## ............................
## Label  "BISTRR : BIST start"
## WRITE BISTRR    FC0A001
## Label  "Delay 1us -incrase delay based on BIST time : READ BISTGSR : WBR shift"
## DELAY #1000
## READ  BISTGSR
## Label  "READ_CAPTURE BISTGSR: BDONE=1 BACERR=0 BDXERR=0"
## READ_CAPTURE BDONE=1 BACERR=0 BDXERR=0
## }
```

Figure 43. Example of Test CPL

A software tool is developed which allows to perform the following steps:

- Generate IP level information (CSL) based on available sources;
- Read CSL information of a given IP and generate corresponding UHBIT scheme;
- Integrate the generated UHBIT scheme into user's SoC design and verify the correctness of the system;
- Port patterns from IP level to SoC level;



```
## proc WRITE {addr data} {
## global ::registers
## global ::params
## global ::glob_addr
## global ::delay
## global ::pattern_space_length
shift_wr = "1";
## if {[regexp {'b(\w+)} $data match binar_data] == 0} {
##   set data [padela::expandVector 32'h$data]
## } else {
##     set data $binar_data
##   }

shift_data_in = "$data";
shift_psel_rqvld = "1";
.......................
capture_data_out = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
........................
capture_cfg_qvld = "x";
shift_cfg_qvld = "x";
## }
```

Figure 44. Example of Setup CPL

- Generate optimal test scenarios.

Automated flow for CSL generation. Figure 45 shows the diagram of the automated flow for CSL generation which ensures the following steps:

- Nowadays IPs may have a huge set of ports and usually it becomes time-consuming task to describe all these ports manually. Since this information is available in IP (e.g. in IP top Verilog view), a conversion utility is developed which translates the port information from IP view (can be in Verilog/SystemVerilog or VHDL format) into CSL format.
- Register CPL – Similar to ports, nowadays IPs may have a huge set of registers and usually IP vendors provide that information using IP-XACT view [140]. IP-XACT is in XML format which sometimes provides difficulties to integrate that information in a SoC test environment where TCL/Python/Perl or other scripting languages are used. Therefore, a conversion utility is developed which translates IP-XACT information into Register CSL.
- Test CPL - Since test pattern of nowadays IPs may have more than 1000 operations per test pattern, generating Test CPL manually is a time-consuming process. Usually

IP test benches provide information on sequences for each test pattern (e.g., Verilog, STIL, VCD, simulation log file, etc.). A conversion utility is developed that converts test pattern information into CPL from IP test bench outputs. One of the preferred views for conversion is simulation log file which usually contains enough high-level information about sequences of a given test pattern.

- Since Setup CPL is test protocol specific, it is not automated. But the development of Setup CPL is not a time-consuming task as well as it is one-time effort for each test protocol.
- Once all the necessary views are ready, the Generator will generate a one common CSL view containing IP level information (ports, chains, test patterns, etc.).

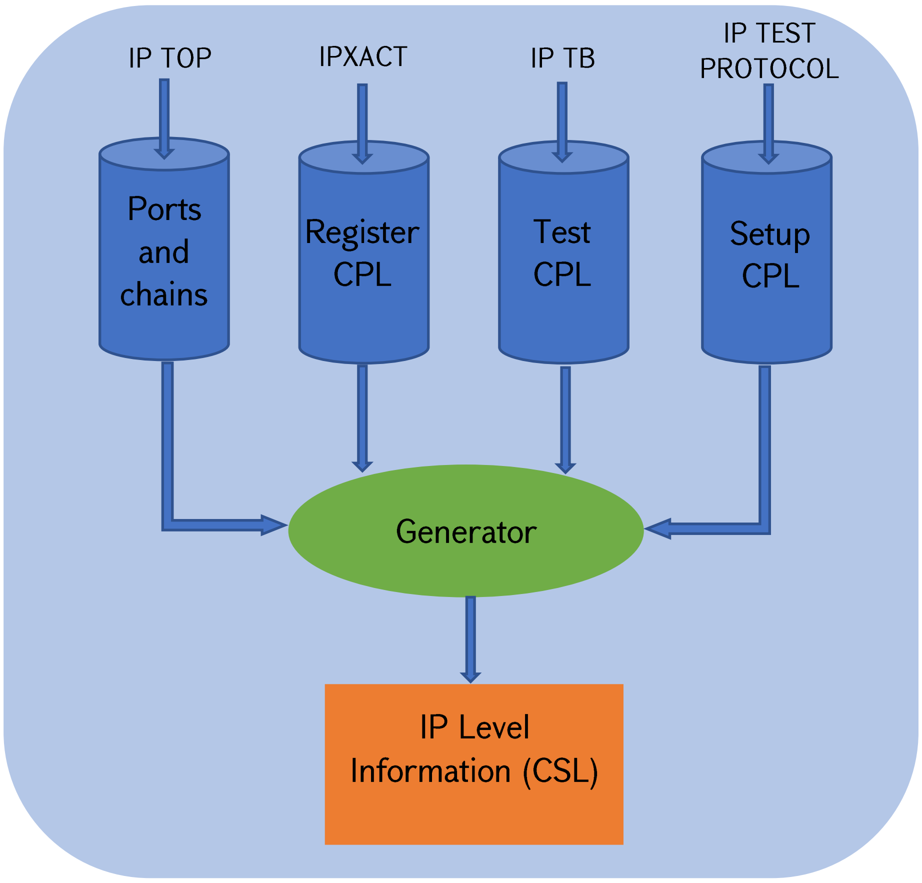


Figure 45. Automated flow for generating IP level information (CSL)

In order to show the effectiveness of the proposed method, experiments on LPDDR4 IP have been performed. This IP has following tests:

1. TEST_DLL_LINEARITY
   - Tries to ensure that the delays generated out of LCDL are linearly increasing or decreasing.
   - Contains a sequence of 2200 test operations.
2. TEST_PLL
   - Checks if PLL initialization is done properly.
   - Contains a sequence of 40 test operations.
3. TEST_PULL_DOWN_DQ
   - Brings up PHY and runs continuous calibration at the background, sets the device in Flyover mode, sets the desired effective resistance and sweeps Pull Down impedance with corresponding values.
   - Contains a sequence of 10000 test operations.
4. TEST_PULL_UP_DQ
   - Brings up PHY and runs continuous calibration at the background, sets the device in Flyover mode, sets the desired effective resistance and sweeps Pull Up impedance with corresponding values.
   - Contains a sequence of 10000 test operations.
5. TEST_VIH_VIL_CHAR
   - Tests BP_ALERT_N Receiver.
   - Contains a sequence of 400 test operations.
6. TEST_VREF_DAC0
   - Tests DQ Receiver by enabling DFE0.
   - Contains a sequence of 300 test operations.
7. TEST_VREF_RxDAC1
   - Tests DQ Receiver by enabling DFE1.
   - Contains a sequence of 300 test operations.

So, overall there are 23240 test operations for testing LPDDR4. In order to manually create CPL view for these tests it would take approximately one week. While the proposed automation allows to generate the same CPL views much faster:

- Port and chain information – 1 second;
- Register CPL – 1 second;
- Test CPL – 10 seconds;
- Generate CSL – 10 seconds.

Since Setup CPL is test protocol specific, it is not automated. From other side, development of Setup CPL is one-time effort for each test protocol and requires about 1-hour effort. So, using the proposed automation it was possible to reduce the development of LPDDR4 IP level test information from 1 week to 1 hour.

Appendix C lists the test patterns of typical 7nm PCIe4 IP core.

6.2 Scheduling of Parallel and Serial Testing of IPs in SoC

Within the concept of the hierarchical test scheduling, in nowadays complex SoCs the test time is one of the important challenges for which usually concurrent test is used to minimize the test time. For thousands of cores in SoC comprised of multiple levels of hierarchy the following problem exists: **determination of an optimal scenario for concurrent test and its implementation in a test infrastructure.**

Figure 46 shows a hierarchical ring architecture of the proposed hierarchical test system which allows to do efficient test scheduling [138]. Group of blocks connected serially is called ring. Sub-Server can have one or more rings (Ring 1, Ring 2, ..., Ring K), and each ring can contain one or more blocks (e.g., Block 11, Block 12, ..., Block $1N_1$), where a block can be an IP core or group of IPs (connected with hierarchical connections) or it can be another Sub-Server.

When several IP cores are being tested in parallel, usually there is a limitation that the total consumed power should not exceed the given number (e.g., MAX_POWER) [141]. Figure 47 shows two scenarios for testing a given set of IP cores. Figure 47 (a) shows a non-efficient (poor) scheduling while in Figure 47 (b) an efficient scheduling scenario is shown.

In both scenarios IP cores are divided to be tested in 3 sequential sessions where in each session the IP blocks are being tested in parallel. In the figure, the «Idle time» shows how well or bad the available resources are used: better scheduling brings smaller «Idle time» which means the available resources are being used more efficiently and thus the overall test time is shorter.

Within the architecture proposed in this work, the overall time for testing all the blocks is calculated in the following way: $T = T_{ring\_access} + T_{ring\_load} + T_{block\_test}$, where:

- T_{ring_access} – time needed to access rings;
- T_{ring_load} – time needed to load information into rings;
- T_{block_test} – time needed to test all the blocks.

Let us assume there are K test sessions determined by one of well-known scheduling algorithms to get effective test concurrency for given design, test and resource limitations, e.g. Greedy/Rectangle Packing algorithm [142]. It is proved that the following proposition is true.

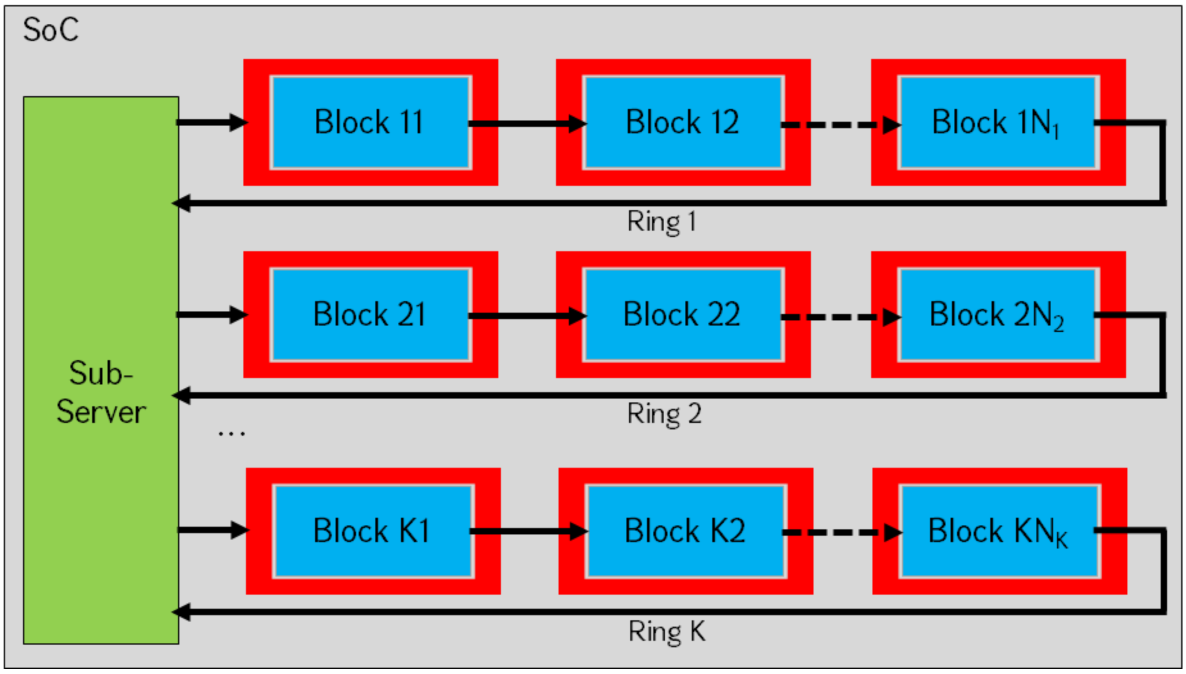


Figure 46. Hierarchical ring architecture

Proposition 31. It is necessary for the proposed test architecture to have K independent rings (one ring per session) to reach optimal test time.

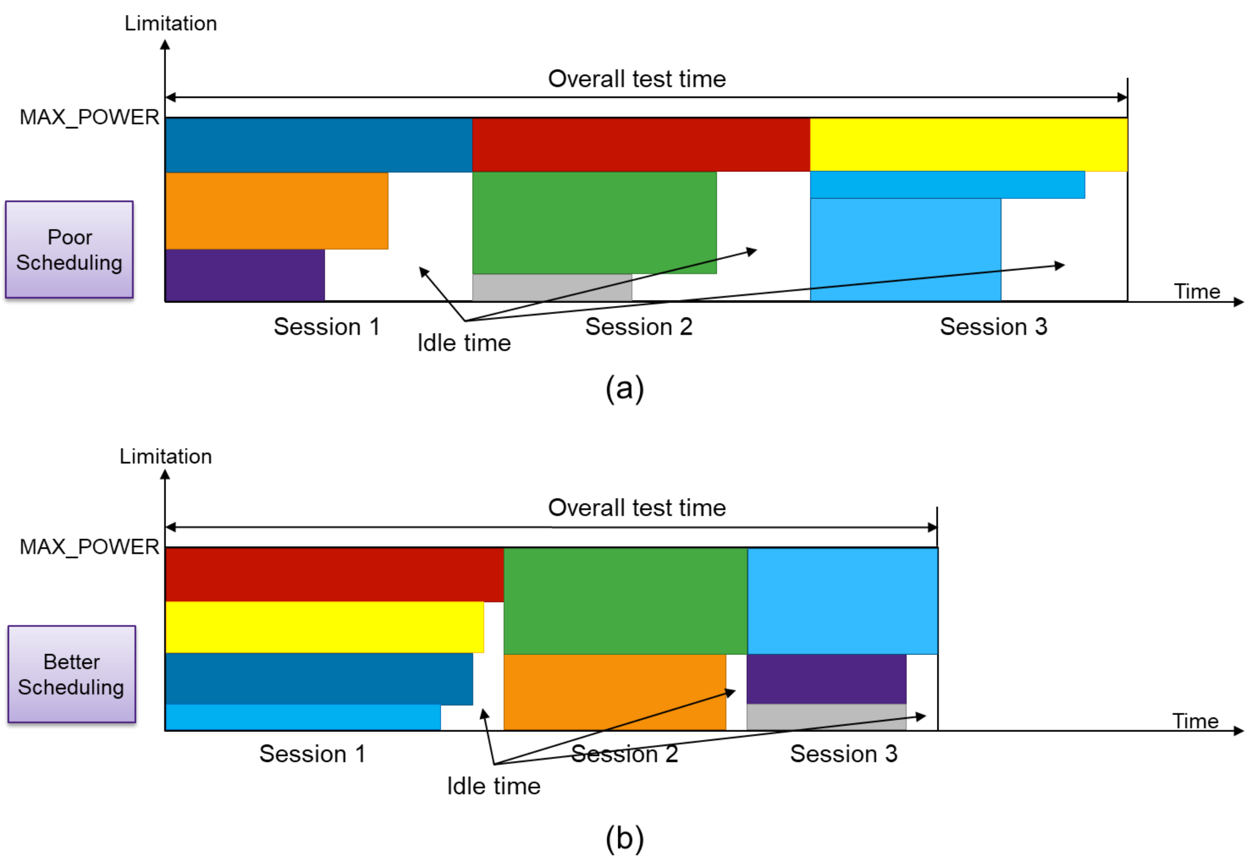


Figure 47. Optimal scheduling scenario

Proof. All other cases will lead to having non-optimal test time. There are the following two cases:

- If blocks of the same session are distributed in different rings, then for testing those blocks in parallel there is a need to access more than one ring, which increases the overall ring access time (T_{ring_access}).
- If a ring contains blocks from different sessions, then for testing blocks of one session there is a need to bypass the blocks that are in that ring but are out of that session, which will increase the overall ring load time (T_{ring_load}).

T_{block_test} depends on the scheduling algorithm and is independent from the test architecture, thus it is not impacted by how the blocks are distributed in the rings.

Experiments showed the same results, i.e., optimal test time is obtained when the blocks of the same session belong to the same ring and there are no other blocks in the ring.

6.3 Test Pattern Template

In Section 3.5 Test Algorithm Template (TAT) is proposed to construct test algorithms which are used to test memories in SoC. But the test algorithm itself does not contain SoC level specifics, such as how many memories should be tested, what is the test scheduling plan and how many test sessions should be applied, when the test should be started, etc. To consider all these specifics there is a need to have a sequence of operations, in other words a test program (usually it is called Test Pattern). Test Pattern can define different types of instructions, such as:

- select a test algorithm to test the memory;
- select set of memories to apply the selected test algorithm to those memories;
- define the test scheduling, i.e., which of the memories should be tested in parallel in the first session, which are in the second session, etc.;
- in which conditions the test should be applied (high voltage, low temperature, etc.).

In this work, a Test Pattern Template (TPT) is developed for constructing Test Patterns which is based on TAT and test scheduling algorithm described in Section 6.2. The steps of Test Pattern generation are following:

Step 1: For a target set of faults construct test algorithm using TAT;

Step 2: Identify the test scheduling, i.e., test sessions using the test scheduling algorithm;

Step 3: Generate test pattern.

Here is an example of a Test Pattern:

Let us assume that in SoC there are 3 IP cores (ip1, ip2, ip3) and 5 memory devices (m1, m2, m3, m4, m5).

Step 1: Construct a test algorithm assuming the target set of faults are static faults (according to Table 16, TAT will construct the March MSS test algorithm);

Step 2: Identify test scheduling – let us assume that based on given SoC limitations the algorithm creates the following scheduling: Session 1: {m1, m2, m5}, Session 2: {ip1, ip3}, Session 3: {ip2}, Session 4: {m3, m4}

Step 3: The generated pattern will have the following sequence:

Session 1:

- GROUP_SEL // Instruction to select m1, m2, m5 memories
- TBOX_SEL // Instruction to select a test algorithm constructed by TAT (March MSS* will be selected)
- BIST_RUN // Run the BIST
- SHIFT // Shift out the obtained results on whether BIST has passed or failed

Session 2:

- GROUP_SEL // Instruction to select ip1, ip3 cores
- WS_INTEST // Enter test mode (IEEE 1500 instruction)
- SHIFT // Shift input/output data (apply necessary test sequence coming with IP core)

Session 3:

- GROUP_SEL // Instruction to select ip2 core
- WS_INTEST // Enter test mode (IEEE 1500 instruction)
- SHIFT // Shift input/output data (apply necessary test sequence coming with IP core)

Session 4:

- GROUP_SEL // Instruction to select m3, m4 memories
- TBOX_SEL // Instruction to select a test algorithm constructed by TAT (March MSS* will be selected)
- BIST_RUN // Run the BIST
- SHIFT // Shift out the obtained results on whether BIST has passed or failed

6.4 Calculation of Optimal Size of E-Fuse Used in Memory Repair Flow

In SoCs, embedded memories usually contain redundant elements (like redundant rows or columns) which are used for repairing faults found during memory test. Since SoC usually contains multiple memories and testing is done in multiple phases (like running the test under different temperatures, voltages and operating frequencies) there is a need to store information about all detected faults in a centralized place which will be further used in the phase of memory repair. Usually Electrical Fuses (E-Fuse) are used for storing information about faulty rows and columns to be repaired (also known as repair signature). During repair phase the repair engine reads repair signatures from E-Fuse and substitutes the faulty rows and columns with redundant rows and columns respectively. Investigation showed that in real projects the probability is low that all redundant elements will be used in repair phase. And since E-Fuse itself adds extra area to SoC the designers usually consider reduced size for E-Fuse instead of considering that redundant elements should be used at 100% rate. From the other hand, having difficulties in calculation of the accurate usage rate for redundant elements, usually approximate usage rate is considered during E-Fuse size determination. This approach has a drawback since it can result in having bigger than needed or smaller than needed E-Fuse size. Bigger than needed E-Fuse results in having extra area in SoC, while smaller than needed E-Fuse can result in impossibility to store information about all faults in E-Fuse and therefore to repair all the memory faults.

In this work, probabilistic models and formulae that calculate exact size of E-Fuse are created [143]. F1 and F2 formulae are proposed for that purpose (see Figures 48 and 49). F1 calculates probabilistic usage rate of redundant elements based on memory parameters

$$u = \sum_{q=1}^{n} \left(\binom{n}{q} y^{(n-q)} (1-y)^{q} \sum_{k=1}^{\min\left(c, \left\lceil \frac{n}{l} \right\rceil, q\right)} \left(k \frac{\binom{n-kl+k}{k}\binom{k(l-1)}{q-k} + \sum_{i=1}^{l-1} \binom{n-(k-1)l+k-1-i}{k-1}\binom{k(l-1)-l+i}{q-k}}{\binom{n}{q}}\right)\right)$$

Figure 48. Formula 1 - Calculation of redundancy usage rate

(number of words, number of bits per word, number of redundant elements, etc.) and memory defect density (based on foundry input or historical data). F2 calculates exact size of E-Fuse based on result obtained from F1. Since this approach allows to calculate the exact size of E-Fuse for a given SoC then from probabilistic point of view it will provide the best

SoC area optimization and repair efficiency in comparison with the existing approaches that are based on approximate consideration of usage rates for redundant elements.

Formula 1 is based on expected value of a random variable, where

- n – number of rows/columns in memory;
- c – number of redundancy groups in memory;
- l – number of redundant elements per redundancy group;
- y – yield of one row/column in memory, which can be calculated with well-known formulae based on memory die area (a) and memory defect density (d_0), e. g. according to Poisson model:

$$y = \sqrt[n]{e^{-ad_0}}$$

$$efuse\_size = \sum_{i=1}^{M} u_i * F$$

Figure 49. Formula 2 – Calculation of optimal size of E-Fuse

F2 calculates optimal size of E-Fuse for group of M memories, where

- u_i – redundancy usage rate of i-th memory;
- F – number of E-Fuse bits needed for one redundant element.

An experiment was done on a SoC design consisting of 5 subchips (see Figure 50) to calculate the chip area saving in case the proposed optimal E-Fuse size calculation method is used.

1. Actual case - with 100% redundancy usage rate 4519 E-Fuse bits were calculated:
   - Therefore 5 instances of E-Fuse macros of 32x32 size (1024 bits) were used;
   - Chip total area was 4200422.649um^2.
2. According to proposed formulae < 71 E-Fuse bits are needed (taking into account different memory defect density numbers):
   - Therefore 1 instance of E-Fuse macro of 32x32 size (1024 bits) is enough;
   - Hence, total area will become 4007924.697um^2 which results in ~4.58% area saving.

3. In case, smaller E-Fuse macro is available, e.g., 16x8 (128 bits), then chip total area saving will be even > 4.58%.

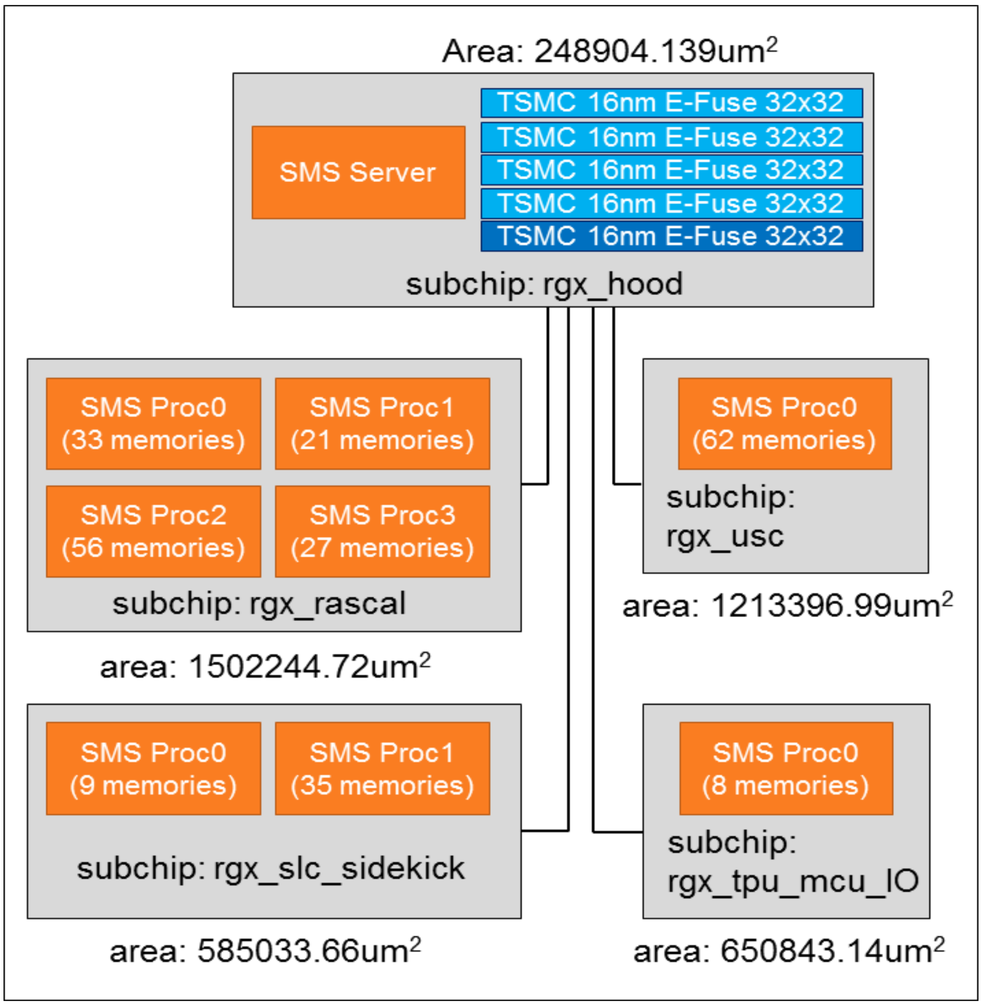


Figure 50. SoC design example

Conclusions

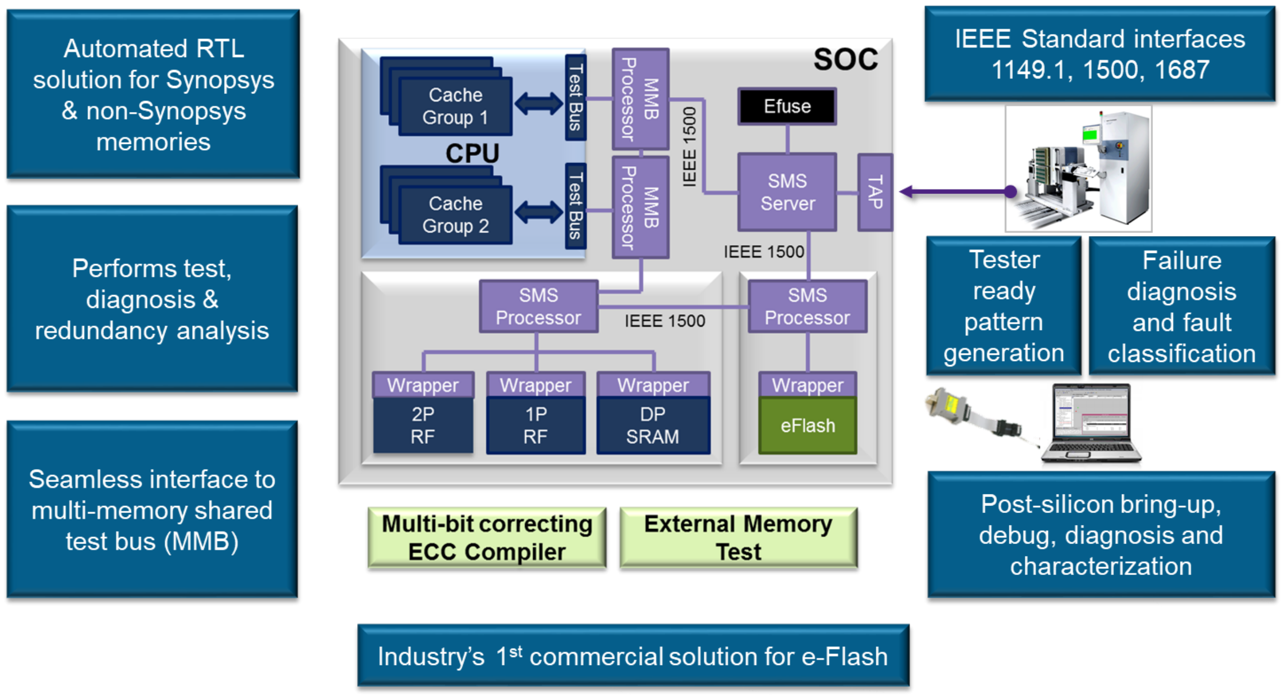
1. Known hierarchical test approaches and associated standards for nanoscale SoCs are provided.
2. An efficient method for building an IP structural model for design and verification is presented. Based on that, a universal hierarchical built-in test (UHBIT) architecture for nanoscale SoC testing is proposed.
3. An approach for scheduling parallel and serial testing of IPs and BIST subsystems is proposed which provides capability for creation of optimal test scenarios under the presence of SoC design, test and resource limitations.
4. Test Pattern Template (TPT) based on Test Algorithm Template (TAT) is proposed to describe SoC level test sequences (test patterns).
5. An idea and specifics of SoC optimization by calculating optimal size of E-Fuse used in memory repair flow is presented.

CHAPTER 7. IMPLEMENTATION OF THE DEVELOPED BUILT-IN TEST SOLUTIONS

The built-in test solutions created within this work form the basis of Synopsys STAR Memory System (SMS) and STAR Hierarchical System (SHS) products.

7.1 STAR Memory System (SMS)

The STAR Memory System (https://www.synopsys.com/dw/ipdir.php?ds=dwc_bist_ip) is a comprehensive, integrated test, repair and diagnostics solution that supports repairable or non-repairable embedded memories across any foundry or process node (see Figure 51). Silicon-proven in over a billion chips on a range of process nodes, the SMS is a cost-effective solution for improving test quality and repair of manufacturing faults found in advanced processes. The SMS highly automated design implementation and diagnostic flow, including Silicon Browser and Yield Accelerator, enables SoC designers to achieve quick design closure and significantly improve time-to-market and time-to-yield in volume production. The SMS includes optimized test algorithms specifically targeted at increasing



Automated RTL solution for Synopsys & non-Synopsys memories
Performs test, diagnosis & redundancy analysis
Seamless interface to multi-memory shared test bus (MMB)
SOC
CPU
Cache Group 1
Cache Group 2
Test Bus
Test Bus
MMB Processor
MMB Processor
IEEE 1500
Efuse
SMS Server
TAP
IEEE 1500
SMS Processor
IEEE 1500
SMS Processor
Wrapper
2P RF
Wrapper
1P RF
Wrapper
DP SRAM
Wrapper
eFlash
Multi-bit correcting ECC Compiler
External Memory Test
Industry's 1st commercial solution for e-Flash
IEEE Standard interfaces 1149.1, 1500, 1687
Tester ready pattern generation
Failure diagnosis and fault classification
Post-silicon bring-up, debug, diagnosis and characterization

Figure 51. STAR Memory System (SMS)

coverage for memory defects like process variation and resistive faults that are prevalent at smaller process nodes, including 28-nm and 14/16-nm FinFET.

The SMS has a set of compilers that generate the corresponding register transfer level (RTL) IPs:

- Wrapper Compiler - merges the differences of the memories and provides an interface between Processor and memories. For example, if there are single-port and two-port memories which should be tested by the same Processor then Wrapper will appropriately interpret the given test operation for each of those memories.
- Processor Compiler - runs the test and repair flow of the system. Test Algorithm Register (TAR) is part of Processor, so the execution of the test algorithms is performed by Processor.
- Server Compiler - provides a top level access to Processors. In case there are multiple Processors in the design then Server allows to hierarchically connect the Processors using Ring architecture discussed in Section 6.2.
- ECC Compiler - provides error correcting codes for protection against soft errors supporting single error detection (SED), single error correction (SEC), double error detection (DED), as well as special class of multiple error correction and detection (when errors occur in adjacent cells) capabilities.

Besides the generated RTL IPs, the SMS provides several design and manufacturing automation tools:

- MASIS Compiler – Generates input files for SMS and SHS (called MASIS – Memory and SMS Interface Standard);
- STAR Planner – Provides possibility to plan the projects for given design, test and resource limitations (how many Processors to use, how to distribute the memories per Processors, how to connect Processors under the Server, etc.);
- Integrator – Environment to generate SMS and SHS infrastructure along with RTL views, database, scripts, etc.;
- STAR Builder – Inserts the generated SMS and SHS RTLs into user's design;

- STAR Verifier – Generates verification test benches for different levels of design hierarchies (Processor, Ring, sub-chip and SoC levels);
- STAR Yield Accelerator – Provides post-silicon debug and diagnosis capabilities (fault classification, coordinate calculation for faulty bit-cells, etc.).
- STAR Silicon Browser – Provides interactive post-silicon debug and diagnosis capabilities.

The SMS test algorithms use memory scrambling information [40] and various test mechanisms. The scrambling information gives possibility to generate accurate background patterns and calculate the exact coordinates of the failed bits. The below list shows some of the important test mechanisms of SMS.

- Test operations:
  - Single-cycle (simple operation, such as Write and Read operations);
  - Concurrent (when multiple operations are applied simultaneously through different ports);
- Physical background patterns generated based on memory scrambling information, such as:
  - Solid, Row stripe, Column stripe, Checkerboard, etc.;
- Addressing methods:
  - Fast column, Fast row, Ping-pong, H1 addressing, Single address, etc.;
- Looping mechanisms
  - Test operation loop;
  - March element loop;
  - Nested loop (when test operation and March element loops are simultaneously enabled);
- Programmability options:
  - Test operation programmability;
  - Background pattern programmability;
  - Test algorithm programmability;

- Possibility to hardwire multiple test algorithms;
- Stop On Nth Error (SONE) diagnosis;
- Fault diagnosis and localization.

7.2 Implementation of Obtained Results in SMS

The test algorithms created within this work and presented in Table 38 are the main test algorithms of SMS product. Table 50 shows the comparison between SMS test algorithms and other well-known test algorithms used in other industrial test products. SMS test algorithms are divided into two groups:

- SMS on-chip - these test algorithms are meant to detect most common faults of memories (considering the range from 90nm to 7nm) and are recommended to be stored in SMS hardware.
- SMS off-chip – these test algorithms are meant to detect less common faults of memories (considering the range from 90nm to 7nm) and are recommended to be provided along with SMS hardware. These test algorithms can be later on loaded into SMS hardware and run on the memory.

As it is seen from Table 50, SMS test algorithms (on-chip + off-chip) are able to detect all common faults of memories while the other test algorithms have less fault coverage.

The results obtained within this work provided the following benefits to SMS product:

1. Increase of test quality and efficiency by using:
    - New test algorithms for detection of newly discovered faults including:
        - March FF for testing FinFET-based memories;
    - Fault diagnosis and localization flows and test algorithms including:
        - The proposed multi-level fault diagnosis and localization flow
        - March FD (for static unlinked faults);
        - March VLP1, March VLP2 and March VLP3 (for dynamic unlinked faults);
        - March FFDD (for FinFET-specific faults);
    - The unified BIST architecture based on:

  - Fault Periodicity Table (FPT);
  - Test Algorithm Template (TAT);
  - Test programmability options.

2. Test time and area reduction by using:
   - Minimal and optimal test algorithms for detection of different sets of faults including:
     - Minimal test algorithm March MSS (for unlinked static faults);
     - Minimal test algorithm March MSL (for unlinked and linked static faults);
     - Minimal test algorithm March MD2 (for unlinked dynamic faults);
     - Optimal test algorithm March LSD (for unlinked and linked static and dynamic faults);
     - Minimal test algorithm March FF (for FinFET-specific faults);
     - Minimal test algorithm March MSL (for CAM storage faults);
     - Minimal test algorithm March CCF (for CAM comparison faults);
     - Minimal test algorithm March VLROM (for ROM address decoder and read destructive faults);
     - Optimal test algorithm March DMFD (for DRAM memory array and interconnect faults);
     - Minimal test algorithm March-FTE (for Flash read/program/erase disturb faults);
     - Minimal test algorithm March EMFD (for 3D Memory interconnect and memory die faults).
   - TAR optimization technique:
     - Based on test algorithm symmetry and canonical view of test algorithms.

Table 50. Comparing SMS test algorithms with other well-known test algorithms

Test algorithms/ Faults	Traditional Faults	Address decode faults	Static unlinked fault	Static linked faults	Bit write enable faults	Bit-line leakage faults	Memory enable faults	Inter-port faults	Dynamic unlinked faults	Delay coupling faults	Data related faults	Retention faults	Process variation faults	FinFET faults	Random telegraph faults
March C- [14]	+	+													
GALPAT [14]	+	+													
March LRD [33]	+	+										+			
March SS [15]	+	+	+												
March SL [17]	+	+	+	+											
March DITEC+ [20]	+	+								+					
SMS on-chip	+	+	+	+	+	+	+	+							
SMS off-chip									+	+	+	+	+	+	
SMS on-chip + off-chip	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+

7.3 STAR Hierarchical System (SHS)

The STAR Hierarchical System (https://www.synopsys.com/dw/ipdir.php?ds=star-hierarchical-test) is an automated hierarchical test solution for efficiently testing SoCs or designs using multiple IP/cores, including analog/mixed-signal IP, digital logic cores and interface IP (see Figure 52). It significantly reduces test integration time by automatically creating a hierarchical IEEE 1500 network to access and control all IP/cores at the SoC level, and increases test quality of results (QoR), including optimizing test time and power through flexible test scheduling of IP and cores. It simplifies SoC test pattern creation by using the IEEE 1500 network to port IP or core-level patterns to the SoC-level, and allows silicon debug and diagnostics by enabling the IP debug test modes from the SoC level. The SHS is compliant with IEEE 1687 Standard, which allows re-use of embedded test instruments for system-level debug. The system's highly automated design-for-test (DFT) implementation and hierarchical IP and core-level test enables engineering teams to cut their test integration time to a matter of days and bring their designs to market faster and with lower design and test costs.

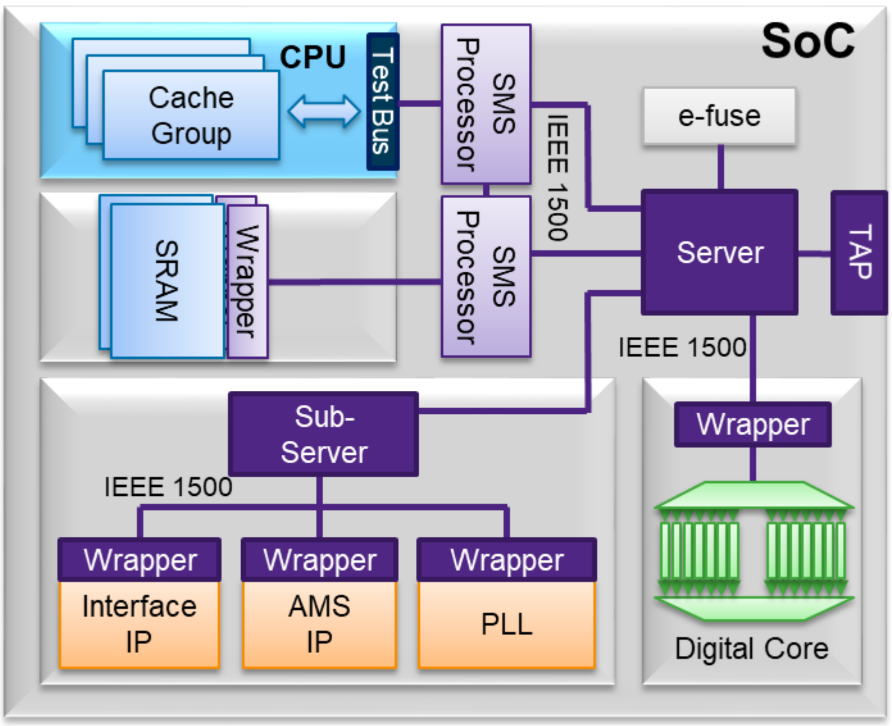


Figure 52. STAR Hierarchical System (SHS)

The SHS product has the following compilers and automation tools:

- RTL IP generation compilers involving:
  - SHS Wrapper Compiler – wraps a given IP with IEEE 1500 compliant RTL;
  - Server Compiler – this is the same Compiler used in SMS product.
- Automation tools are the same as used in SMS product.

The SHS provides the following capabilities:

1. RTL level IP core wrapping
   - Specifies ports of IP core to be wrapped;
   - Specifies test sequences at core level;
   - Generates Wrapper boundary register (WBR), Wrapper instruction register (WIR) and Wrapper bypass register (WBY) components in RTL format, test benches, constraints, and integration data base.

2. RTL level SoC test integration
    - Generates Server RTL;
    - Generates JTAG TAP RTL.
3. Core and SoC level insertion
    - Inserts SHS Wrapper and Server RTL into user's design;
    - Propagates IEEE 1500 signals to the next level of integration;
    - Connects Server to TAP and SHS Wrappers.
4. Pattern porting of IP cores
    - IP level test patterns are ported to high level of hierarchy.
5. Hierarchical Verification
    - Verification capability at IP, ring, sub-chip and SoC level.

7.4 Implementation of Obtained Results in SHS

Table 51 shows the test times (in microseconds) and corresponding test time saving when using SHS concurrent test versus sequential test of IPs in SoC. The experiments showed that when Internal Loopback test is applied on 5 PCIe3 IP cores in parallel, then SHS test time saving versus sequential test of IPs is 70%.

The results obtained within this work provided the following benefits to SHS product:

1. Increase of test quality and efficiency by using:
    - the universal hierarchical built-in test (UHBIT) architecture including the automated flow for CSL generation;
    - Test Pattern Template (TPT).
2. Test time and area reduction by using:
    - the scheduling algorithm for parallel and serial testing of IPs in SoC;
    - the method for calculation of optimal size of E-Fuse.

Table 51. Test time savings of SHS

IP cores	Test Pattern	Test time for one IP core (T_{test})	Test time for accessing one IP core (T_{access})	SHS concurrent test time ($T_{test} + N*T_{access}$)	Sequential IP test - assumes 0 delay for IP access ($N*T_{test}$)	SHS test time saving (%)
Two PCIe3 IP cores (N = 2)	PHY Initialization	1000	300	1600	2000	400 (20%)
Two PCIe3 IP cores (N = 2)	Internal Loopback	3000	300	3600	6000	2400 (40%)
Five PCIe3 IP cores (N = 5)	Internal Loopback	3000	300	4500	15000	10500 (70%)

Conclusions

1. The brief description of Synopsys STAR Memory System (SMS) and STAR Hierarchical System (SHS) products is given.
2. It is shown that the results obtained within this work are widely used and form the basis of SMS and SHS products.
3. The benefits of the obtained results (increase of test quality and efficiency, as well as test time and area reduction of SMS and SHS) are described.
4. The obtained results allowed:
    a. to reduce test time by 18%-44%;
    b. to reduce BIST area by 7%-48% and overall chip area by up to 5%.

CHAPTER 8. APPLICATION OF THE DEVELOPED BUILT-IN TEST SOLUTION TO FUNCTIONAL SAFETY AND SECURITY CRITICAL APPLICATIONS

8.1 Functional Safety and Security Requirements for Automotive SoCs

The automotive is one of the fastest growing sectors in semiconductor industry at the moment. The reasons of such tremendous growth in automotive market are the increasing consumer demands in safety, reliability, and security enhanced applications. The tendency for greater safety and better driving experience is forcing automakers to continually integrate large amount of Electronic Control Units (ECU) like Advanced Driver Assistance Systems (ADAS) and In-Vehicle Infotainment (IVI) into their vehicles. Few examples of such systems are adaptive cruise control, parking assistance, automotive emergency braking, lane change assistance and so forth as this list continues growing.

Automotive electronic systems have traditionally utilized the well-established technology node processes for the benefit of higher yield, reliability, and low cost. However, considering the growing demands automakers had to adopt the solutions with increased computing power and communication performance which are mainly used with emerging technology nodes like FinFET. As the International Technology Roadmap for Semiconductors (ITRS) diagram [144] in Figure 53 shows in the past only the technologies with enough maturity level were permitted to be used in automotive electronics and the technology maturity period was estimated to be about five years in average.

Nevertheless, especially during last years, with the growing consumer market pressure, this margin has significantly decreased reaching up to two years. Such trends pose additional challenges for automotive application designers and original equipment manufacturers (OEMs) to meet the requirements for higher performance and power efficiency along with the native demands for safety, robustness, reliability, and time-to-market. As a result, the modern automotive industry has still several major challenges which need to be addressed, including:

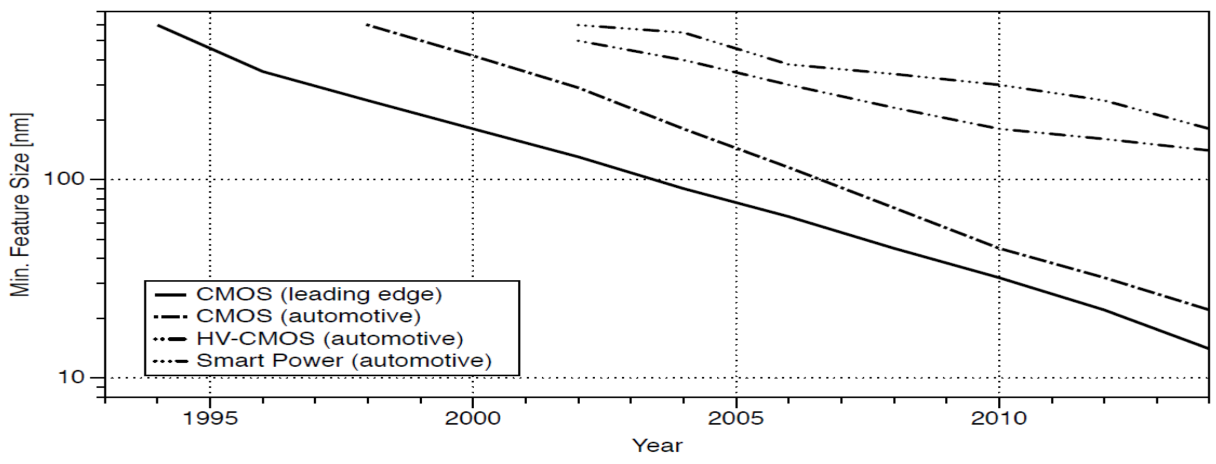


Figure 53. International Technology Roadmap for Semiconductors (ITRS), Update 2012

- functional safety and reliability;
- high quality testing during the production;
- security and privacy.

One of the most important aspects to consider for automotive is the high-test quality requirement. Here special care should be taken to develop a reliable test and repair solution applicable not only at the production stage but through the whole product life-cycle starting from the design stage to silicon bring up and volume production all the way up to field operation.

This is related especially to the embedded memories spanning most of the SoC space and being the major contributors to achieving high yield and low defective parts per million (DPPM) or even defective parts per billion (DPPB) rate as put into circulation recently. Built-in self-test (BIST) solution is traditionally the most preferred approach for testing and repairing embedded memories providing reasonable trade-off between the cost and achievable yield. Until recently existing BIST solutions primarily focused on SoC design stage since test solutions in the mission were neither considered effective nor affordable. However, today the situation has drastically changed considering the recent developments in automotive market. Now mission mode safety and security are regarded as highest priority requirements for automotive thus require comprehensive test capabilities not only during the production stage but also in the field. Many techniques and new methods addressing the post-production testing problems have already been proposed in the literature which consider specific testing aspects or target local testing problems as shown below.

For instance, the concept of transparent BIST is described in [145], which ensures that the execution of test algorithm is transparent to the system, i.e., the content of the memory after the test is the same as the initial content. This allows to run the BIST not only in the production but also during the field operation. The proposed technique is based on transforming BIST test algorithms into transparent ones based on a set of predefined rules meanwhile preserving the fault coverage. However, the area overhead of the proposed transparent BIST as well as the test time are increased compared to traditional BIST. Several other versions of transparent BIST architectures have also been proposed with similar concept which all try to achieve better fault coverage and improved test time, e.g., [41], [146], [147].

Memory and logic BIST design implementation for automotive SoCs described in [148] supports several new features specifically designed for in-field testing. For instance, the idea of non-destructive and destructive self-tests is presented. In the first case the test control units are not included in the test allowing the greater in-field control and scheduling of the test procedures. Meanwhile in the latter case the entire device is tested and after the test full system reset or restart is required since the system state gets lost. Other useful features of the solution are also presented including reset control during the self-test, and performing built-in analysis and self-repair based on the results of the self-test.

There are other works as well on this topic showing the benefit of using structural tests (mostly built-in) for the purpose of in-field test and fault diagnosis. For instance, [149] shows the tradeoffs and advantages that the automotive domain can gain with reusing the production test methods for in-system test. However, this requires development of a strategy for accessing relevant on-chip DfT (Design-for-Testability) and emulation of production test environment in the field at lower cost. The case study in the paper for the industrial ECU demonstrates the implementation of such environment with comparable fault coverage though some design and timing-related constraints are also reported.

In complement to structural tests other alternative techniques proposed recently in the literature can be also adopted for in-field test. This includes software-based self-test techniques as described in [150], [151] and functional test methods for which good overview can be found in [152].

Two other related aspects to in-field testing are the problem of aging of integrated circuits in the mission mode and importance of Operating Life Testing (OLT). With regard to aging problem there are several methods proposed for on-line aging monitoring such as the architecture introduced in [153] specifically developed for safety-critical applications. The proposed built-in sensors are activated only during the car power-up thus are resilient to aging and threshold voltage variations. Aging monitoring is carried out by observing propagation delays in circuit critical parts. Meantime, the importance of adopting functional approach for reliability characterization and OLT of SoCs are discussed in [154]. A novel approach is presented for automatic generation of proper test patterns to be used for OLT phase with experimental results demonstrating the advantages of the proposed technique.

Finally, certain efforts have been put also to perform failure analysis in the field. An optimized memory diagnosis flow for accurate failure analysis mainly targeted for automotive-oriented SoCs is described in [115]. The proposed post-processing flow allows to investigate failure syndromes even in the case of incomplete or noisy data and provides the physical location of the fault as well as preliminary information on the underlying physical defect.

8.2 ISO 26262 Standard and Its Certification Process

One of the most important requirements to take into account while designing any kind of application for modern automotive is functional safety and reliability. This is not a new topic and the fundamental requirements for functional safety-critical applications have already been well-established long time ago in the scope of military, nuclear and aerospace industries and now are being widely adopted by other industries like automotive. With the growth of ADAS and IVI systems in vehicles, the demands for higher safety and reliability are growing at the same rate. Safety primarily focuses on mitigating the risk of systematic as well as random hardware failure occurrence in the system during the production or field operations. Meanwhile, reliability defines the probability that a system performs its assigned functions in a specified period of time.

Increased attention to safety and reliability aspects in automotive raised the need to have common criteria for measuring their compliance level in the system. It motivated the emergence of dedicated ISO 26262 Standard titled "Road vehicles – Functional safety" [155] which establishes functional safety definitions and requirements for automotive equipment applicable throughout the life-cycle of all automotive electronic and electrical (E/E) safety-oriented systems. ISO 26262 prescribes the requirements for achieving acceptable level of functional safety for electrical and/or electronic systems intended to be used in production automobiles. Based on the adherence to these requirements a final product can be qualified with one of available four Automotive Safety Integrity Levels (ASIL) A-D.

ASIL refers to an abstract classification of inherent safety risk in an automotive system or elements of such a system. ASIL classifications are used within ISO 26262 to express the level of risk reduction required to prevent a specific hazard, with ASIL-D representing the highest and ASIL-A - the lowest levels. The ASIL assessed for a given hazard is then assigned to the safety goal set to address that hazard and is then inherited by the safety requirements derived from that goal. ASIL is determined based on the combination of probability of exposure, the possible controllability by a driver, and the possible outcome's severity if a critical event occurs.

The ASIL certification has become a crucial requirement for automotive as a guarantee of necessary safety level. Thus, E/E system providers for automotive are required to obtain corresponding certificates as a proof that their systems are safe and reliable. For this purpose, professional organizations are established and specialized in performing the certification process and verifying the product compliance to ISO 26262 using well-established methodology and safety metrics. Therefore, the product developed on the basis of the proposed test architecture was certified by one of such organizations and as an outcome it received the ASIL-D certificate [156]. Figure 54 shows the common ASIL-X certification process which is used for product certification. It consists of the following major steps:

1. The product is provided to Certifier.
2. Certifier identifies Safety Goal Violations (SGVs) of the product:
   - Safety Goal (SG) is a safety requirement assigned to a product with the purpose of reducing the risk of one or more hazardous events to a tolerable level;
   - Safety Goal Violation (SGV) is a violation of a safety goal due to a fault in the product.
3. Certifier identifies Failure Modes (FMs) of the product:
   - Faults that lead directly to the violation of a safety goal are called Single Point Faults (SPFs);
   - MPFs (Multiple Point Faults) are combination of multiple independent faults leading directly to the violation of a safety goal.

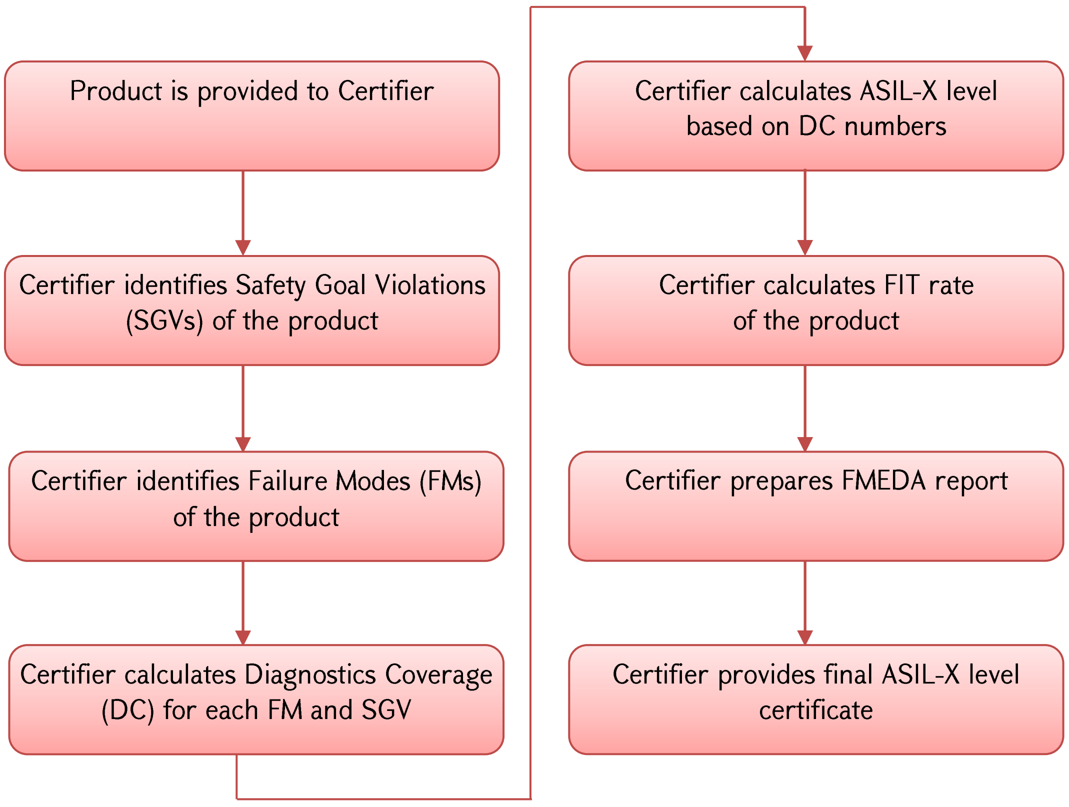

Product is provided to Certifier
Certifier identifies Safety Goal Violations (SGVs) of the product
Certifier identifies Failure Modes (FMs) of the product
Certifier calculates Diagnostics Coverage (DC) for each FM and SGV
Certifier calculates ASIL-X level based on DC numbers
Certifier calculates FIT rate of the product
Certifier prepares FMEDA report
Certifier provides final ASIL-X level certificate

Figure 54. ISO 26262 certification process

4. Certifier calculates Diagnostics Coverage (DC) for each FM and SGV:
   - In this step, Certifier identifies the impact of each FM on each SGV and the corresponding Diagnostic Coverage (DC) of the product able to detect that impact.
5. Certifier calculates ASIL-X level based on DC numbers:
   - Based on obtained DC number and using well-known table (see Table 52) from ISO 26262 Standard, Certifier calculates the ASIL level of the product.
6. Certifier calculates FIT rate of the product:
   - Using well-known formulae, Certifier calculated FIT rate of the product.
7. Certifier prepares FMEDA report:
   - FMEDA report contains all the information obtained during the certification process including SGVs, FMs, DCs, ASIL and FIT numbers.
8. Certifier provides final ASIL-X level certificate.

Table 52. ASIL and FIT requirements

ASIL	SPF	MPF	FIT Rate
ASIL B	≥ 90 %	≥ 60 %	100 (recommended)
ASIL C	≥ 97 %	≥ 80 %	100 (required)
ASIL D	≥ 99 %	≥ 90 %	10 (required)

8.3 Concept of Safety and Security Oriented Test Architectures

The safety requirements do not bypass also the on-chip logic structures responsible for performing test and repair functions in the SoCs which are going to be used in automotive. ISO 26262 has defined certain requirements to BIST module and related test structures which they should meet before being qualified as acceptable for automotive. In the past the main requirement to test was providing the necessary manufacturing fault coverage in a reasonable time frames and with acceptable hardware overhead. In automotive the picture is different since the safety during the SoC life-cycle comes on the scene. To have better understanding of the automotive demands closer look at different phases of SoC life-cycle need to be taken and identify the key requirements to be addressed in each case. In fact, SoC life-cycle can be divided into three major modes: production, power-up and mission modes.

8.3.1 Test requirements for Production, Power-up and Mission Modes

Production Mode: In automotive like in every other high-end application achieving optimal yield of the production is a vital requirement. Thus, having a powerful test and repair mechanism possessing all the required capabilities is highly regarded. For this purpose, at design stage most commonly DfT and memory BIST blocks are incorporated into the chip which are later being used for test, repair, and diagnosis operations. The memory BIST block enables the self-testing capability of the chip reducing the complexity of the test setup as well as decreasing the test cost and time-to-market. First step towards building a comprehensive test infrastructure in BIST is understanding the full spectrum of realistic fault models for different memory architectures which are going to be used in the chip. The memory faults are the abstract representations of physical defects that can occur during the chip manufacturing stage. However, not only faults occurring in memory cell arrays need to be considered but also the faults occurring in memory array surrounding blocks should be taken into consideration including address decoder, write driver, sense amplifier and so on (see Figure 55).

Power-Up Mode: Maintaining the test capabilities active in a SoC also after the production phase is a specific requirement for functional safety-oriented applications. With the passage of time the circuits become worn out and negative consequences can start to show up in a period of few years after the production or even couple of months in worst case. Aging effect is the most common issue which is usually defined as circuit performance degradation over time. The aging effects can include increased power consumption, decreased speed, timing delays and this finally may result into operation failure and emergence of permanent faults. With the rapid technology node shrinking, the aging effects have increased significantly and cannot be ignored anymore, especially in the applications like automotive. The main causes of aging are Negative Bias Temperature Instability (NBTI) and Hot Carrier Injection (HCI) effects, but recently importance of Positive Bias Temperature Instability (PBTI) effect has also increased.

In contrast to production phase, in the field the requirements to test are more stringent due to number of area, power, and time-related constraints. Therefore, the test solution from production mode cannot match these criteria one-to-one, thus usually alternative solutions are proposed. On the other hand, there is no need to target all manufacturing defects during power-up test since only a subset of those defects can show up (e.g., aging, electromigration). The power-up test starts whenever a key is on and the main goal is to quickly test whether all devices are properly functioning before the system is in mission mode or otherwise to report if any issue is found.

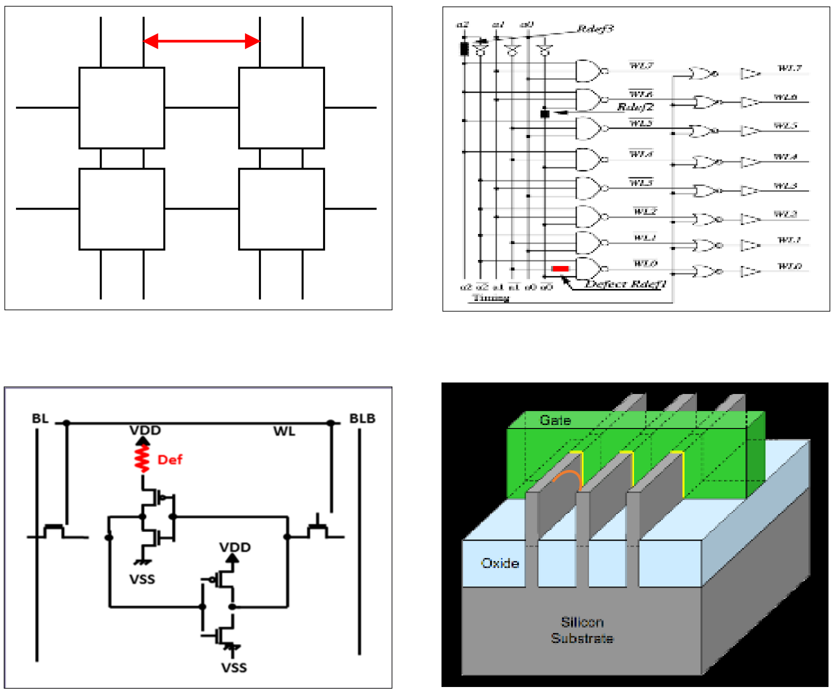


Figure 55. Different fault types in memories

Mission Mode: After the power-up phase is successfully accomplished and the system enters the mission mode. During the mission mode, the faults are commonly occurring due to soft errors though permanent faults mainly due to aging can occur as well. The main cause of soft errors is exposure of integrated circuits to alpha particles and cosmic rays coming from the outer space as discovered in recent studies [157], [158]. Soft errors, compared to hard errors, have transient nature, and do not cause a permanent damage. However, with the growing safety and reliability concerns in automotive, detection and correction of these type of errors in mission mode become critical.

Dealing with the described types of faults requires mechanisms for constant monitoring and prompt reaction but at the same time having minimum impact on system performance. This kind of tests are called periodic tests [148] which are run periodically in the mission mode to check for the occurrence of faults in memory without corrupting its content. The periodic test starts once per safety interval, which is defined in the system specification as a period of time during which failure can occur. The main goal of periodic test is to quickly check if there is an issue in the system. Since the system is in a mission mode there is usually only a small-time interval left to perform the periodic test for memory units which are not being used (are in idle state) at the moment. Figure 56 shows an example of periodic test scenario at the given point of time. In the illustrated case M3, M5, M10, M14 and M16 memory units are under the test while the other memories are being utilized or are in the idle state.

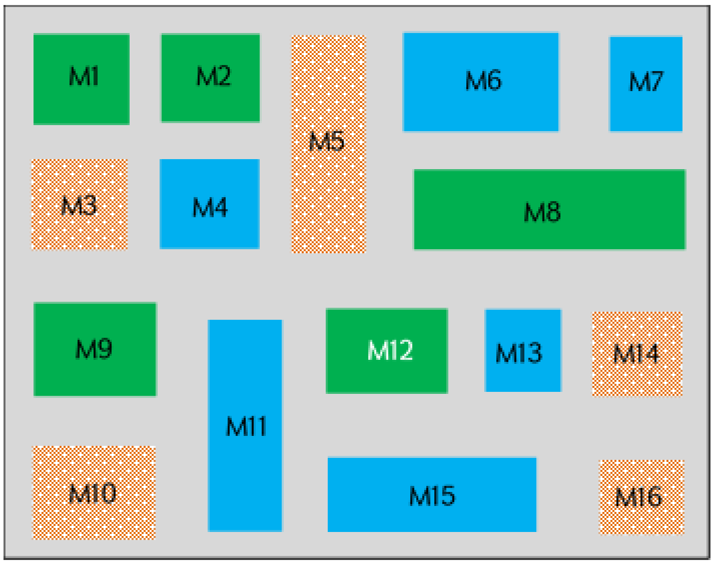


Figure 56. Periodic test scenario in the field

8.3.2 Known Approaches for Secure Testing

In most cases the test infrastructure of the system is constructed using scan architecture built upon known test interface standards developed by the IEEE working groups over the years. This includes both IEEE 1149.1 and IEEE 1500 as well as recently ratified IEEE 1687 (or IJTAG) standards. Nevertheless, all the described test interfaces are not inherently designed with built-in protection mechanisms which makes them the easy target for the intruders. Building secure test infrastructure is relatively new sphere and therefore there is still a plenty of work to be done. However, number of solutions are already proposed in the literature for securing test interfaces as well as scan network consisting of individual scan chains, most common which are discussed below.

TAP (e.g., most frequently JTAG) is usually the test interface used at system level providing access to the internal test structure of the system. One of the light-weight solutions for protecting access to TAP may be permanently disabling the test infrastructure on the chip or disabling the switching between the mission and test modes after the manufacturing. However, this may not be acceptable solution for many applications which require in-field test and debug capabilities. Therefore, several alternative techniques have been proposed in the literature (mainly based on JTAG Standard) which aim to protect the system from unauthorized access. For instance, a solution proposed in [159] suggests adding the feature for resetting the system and performing an initialization process every time the chip switches from test mode to mission mode or vice versa. However more sophisticated solution is based on challenge-response based authentication scheme which suggests that one of the parties presents the challenge while the other should provide a valid response in order to be authenticated. In the simplest case the password-based authentication mechanism is added to the JTAG which requires users to enter valid credentials, in order to gain access to the system (e.g., [160]). A little bit more complex but more advanced approach is based on using the public-key or private-key cryptography for the user authentication (e.g., [161]). More sophisticated approach proposed in [162] suggests utilizing the concept of multi-level hierarchical permission system in order to control the individual users or user's groups access to the system.

There are also several techniques proposed which aim to protect individual scan chains from illegal access via locking mechanisms. Some of the them are "flipped scan tree" [163] and "Lock and Key Technique" [164]. In case if the test scan architecture on the chip is built on the base of IEEE 1687 there are also secure solutions proposed as well. The central idea here is protecting reconfigurable scan networks (RSN) which are realized via the gateways called segment insertion bits (SIB) for dynamically configuring the test paths to the instruments. In [165], the authors proposed inserting the separate authorization instrument while in [166] an alternative approach is proposed suggesting to replace the SIBs with so-called Locking SIBs (LSIB) requiring certain key value to be scanned into the network for unlocking. At last, similar key-based techniques were also proposed to be embedded into 1500 Standard Test Wrapper in order to protect the embedded cores from unauthorized access (e.g., in [167]).

8.4 A Hierarchical Test Architecture for Functional Safety and Security

8.4.1 Hierarchical Test for Automotive SoCs

Based on the above considerations and following the outlined requirements the hierarchical test architecture is developed for automotive SoCs [156], [168], [169]. Two keystones of the proposed architecture are the flexible and programmable memory BIST and ECC engines supplemented with different control modules. The architecture hierarchy has the following structure (see Figure 57):

- On the bottom of the hierarchy the embedded memory units are placed with surrounding read/write logic blocks. The memory units are connected to BIST and ECC controllers which conduct the testing process.
- Memory instances are grouped under one BIST controller or use dedicated BIST controller depending on the configuration. Grouping of memories can be based on certain considerations including maximum allowed power consumption, routing

congestion, timing limitation, or possibly other criteria. ECC engines are allocated one per memory instance where necessary.

- On the top level, the access control module is placed which not only provides interface to access and run the underlying BIST modules but also provides various test scheduling options for reducing power consumption and/or test time.

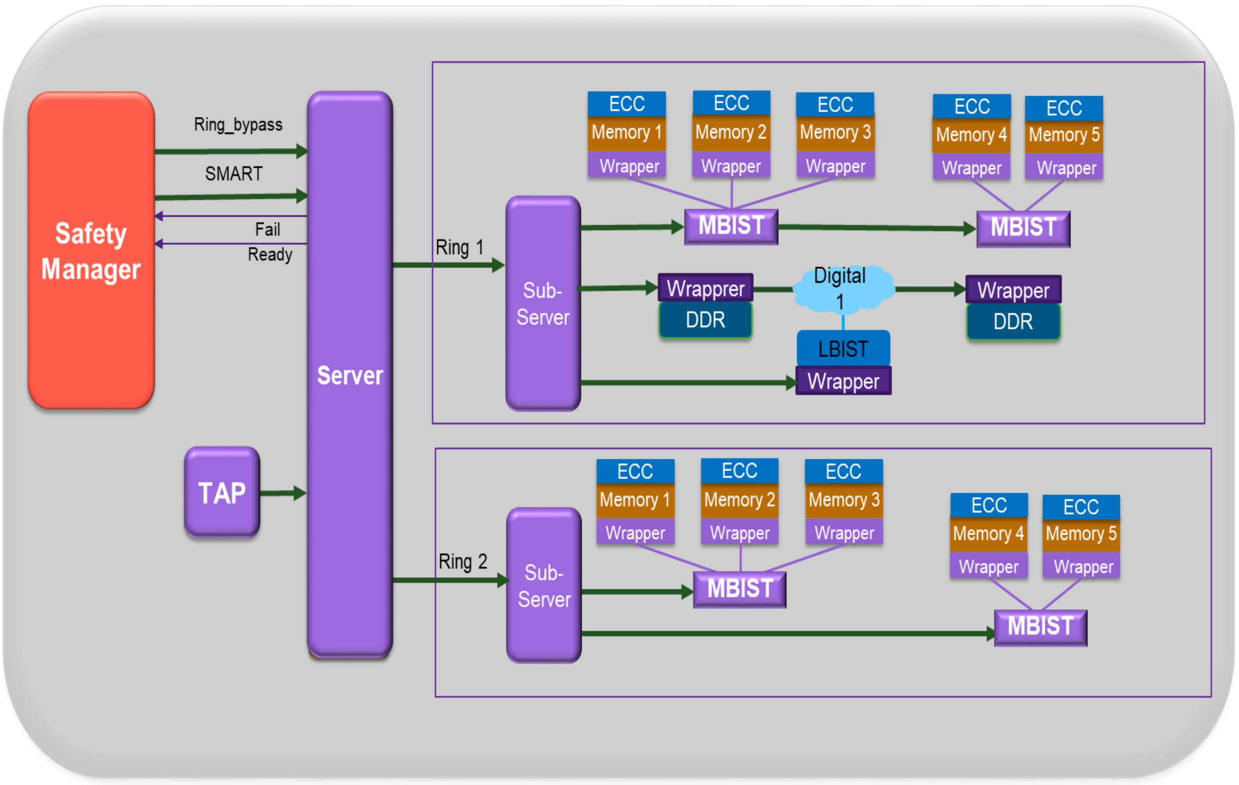


Figure 57. Hierarchical test infrastructure for automotive SoCs

Since the self-test is not only going to be used in production mode but also in the field, two types of access mechanisms are provided:

- Standard IEEE 1149.1 (JTAG) compliant Test Access Port (TAP);
- a special interface (SMART) with few control ports to instantiate and run the self-test and repair procedures in functional mode (i.e., at power-up and mission modes) by Safety Manager during which the JTAG/TAP is usually disabled.

The SMART interface has the following interface ports:

- Ring_bypass – capability to select a ring or rings for the self-test and repair;
- SMART – initiates memory self-test and repair on the selected rings;

- Ready – indicates that the execution of the test run is accomplished;
- Fail – indicates PASS/FAIL result of the test run.

8.4.2 BIST Features Addressing Functional Safety Requirements

The unified BIST architecture presented in Section 3.7 satisfies the functional safety requiremnts [156] since it has the below mentioned capabilities:

- Programmable Test Algorithm Register – The proposed BIST architecture contains a set of predefined test algorithms dedicated to test and provide a high fault coverage for planar- and FinFET-based memories. Specifically the fault coverage includes:
  - static single-cell and coupling faults;
  - dynamic single-cell and coupling faults;
  - links between static and dynamic faults;
  - process variation faults;
  - technology-specific faults (including FinFET-specific faults). For instance, the study conducted recently for sub-20nm technology nodes, namely FinFETs, revealed new subtypes of dynamic faults specific to FinFETs, which were not observed earlier (see [71]-[75]).

  In addition to the comprehensive set of predefined test algorithms, there is still an opportunity to program new test algorithms or to modify the existing ones if it is needed. For this purpose, a programmability interface is provided not only to add new test algorithms but also new addressing methods, new test operations and new data background patterns. In case of automotive, in production mode runtime and performance requirements are usually sacrificed to the benefit of higher fault coverage, meaning that using complex but reliable test algorithms is usually recommended.
- Test Algorithm Container – It allows to store test algorithms during the design phase and then choose the appropriate one to run during the power-up and mission modes. Thereby, by default, along with the predefined comprehensive test algorithm

for production mode two or more test algorithms are included in the test suite with reduced test length intended for testing in power-up and mission modes. In contrast to production mode, the access to built-in test in these modes can be granted through the dedicated SMART interface (SELECT TEST ALGORITHM functionality) since after the production most commonly the access via TAP is usually prohibited for security reasons. Beside the self-test capability, a quick self-repair mechanism is also provided via the same interface to quickly run the repair procedure after the test is complete.

- BIST Controller – Applies the selected test algorithms to Memory as well as provides a number of functionalities that are necessary for testing automotive SoCs:
    - SELF TEST functionality allows to check if the BIST scheme itself is functioning properly or not. This especially refers to the logic units which are responsible for collecting and sending the status information to outer world. The reason of running such test is that in case if they are not functioning properly, then wrong information about the system state may be sent compromising the overall safety of the car and the driver. For this purpose, an extensive fault injection mechanism is provided which allows to inject faults in the memory array and surrounding blocks (can be done by user or automatically initiated by SELF TEST function).
    - Performing periodic test is associated with number of native challenges which are addressed by the proposed solution. First, due to the critical time constraints complex test algorithms cannot be used during the in-field test, however, running simplified test periodically compensates this to some extent and minimizes the risk of fault escapes. Depending on the given interval length either test can be run over the whole memory space or a range of memory addresses (by using the SELECT ADDRESS RANGE functionality).
    - Another provided optional feature is making the test algorithm transparent using the concept similar to the one described in [145]. This can be useful in

case when memory content should be preserved during the periodic test. The periodic test scheduling for the memory units is left to the system main processor to define since it depends on many factors including length of the test interval, availability of memories, power consumption limitations and so forth. For this purpose, capability of invoking periodic test for the selected memories and skipping the other ones is provided.

- Last but not least is the TEST ABORT functionality which allows to stop the test process and return the memory to operational state in case of access request to a specific memory.

The proposed BIST architecture provides capability not only to test but also to repair the memories. For repair purpose, the set of detected faults with their location information is stored in the designated space of the top control module. For this purpose, usually electrical fuse array is used as one- or multiple-time programmable module to store these data. After the BIST run, the self-repair procedure is initiated based on the stored information. In this process first redundancy analysis is performed to read the configuration of redundant elements (redundant rows and columns) and fault locations, and then to identify the best redundancy allocation scheme for repair. If such a scheme is found, then the repair procedure is run and all the faulty rows and columns in the memory array get replaced with redundant ones. Only after that the chip is proclaimed as ready for operation. The whole test flow is designed in a way to minimize the level of user interference and necessary efforts.

Optionally also the powerful diagnostic capability is provided in case if it is required to debug and understand the root cause of the detected faults. The provided efficient test algorithms take as an input the list of detected faults and determine the type of faults from the known set of faults and try to identify common physical defect patterns, for instance, whether it is a single-cell defect, quadruple defect, row/column defect and so forth. The identification procedure is made more efficient by storing the known fault types in the form of periodicity table as proposed in [8]-[11]. After the diagnosis is finished, a detailed report is generated providing information about the types of detected faults and observed physical defect patterns.

8.4.3 Advanced Error Correcting Code (ECC) Solution for Functional Safety

For a long time since FIT (Failure in Time) rate of soft errors was not high enough, they were considered as a noise and therefore ignored. Nevertheless, with the technology node shrinking, technology complexity and the packing density of the memory array have significantly increased leading to higher probability of soft error occurrence. Periodic self-test is not efficient when dealing with such errors given the transient nature of faults. For that purpose, usually ECC codes are utilized in the field. Several variations of ECC codes exist which have different error detection and correction capabilities and provide trade-off between complexity, performance, and area. The most common types of error detecting/correcting codes are the following: Single error detection (SED), Single error correction (SEC), Single error correction & Double error detection (SEC-DED).

ECC is based on the concept of using additional check bits which are determined by even parity checks over the selected information positions and storing or transmitting them along with the information bits. At a high level, ECC can be considered as a two-phase process consisting of encoding and decoding steps (see Figure 58). During the encoding step, all

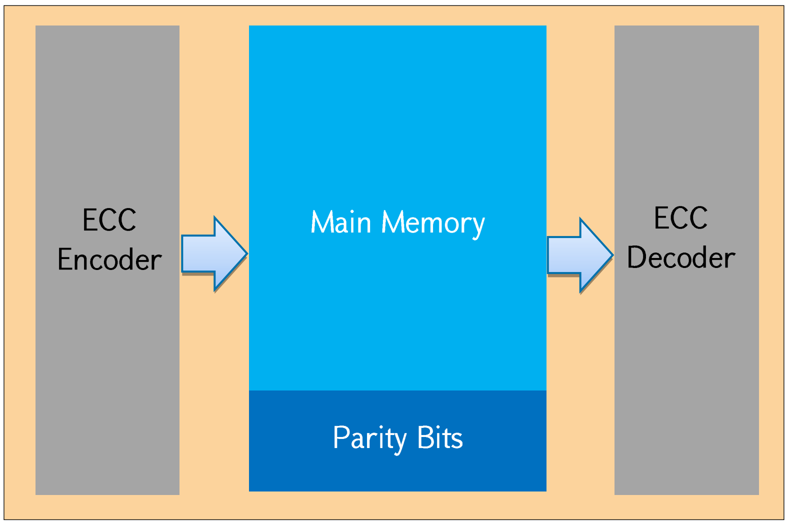


Figure 58. Error Correcting Code (ECC) scheme

the necessary calculations are performed to generate the required number of check bits and keep them along with the information bits in the memory. During the decoding step, it is determined whether the actual retrieved information contains errors or not using check bits' values, and they get fixed when possible. The encoding is done during the memory write operation while the decoding step corresponds to memory read.

Until the recent years Hamming and Hsiao linear SEC-DED codes were de facto most commonly used ECC codes which were proven by the community to be robust codes [170]-[172]. However, with technology shrinking the probability of multi-bit upsets has significantly increased becoming almost equal to single-bit upsets. Thus, in the proposed solution the low overhead multi-bit error detection and correction code is implemented [173]. The proposed solution is further enhanced to protect not only the memory array but also address decoder from permanent and transient faults [174], [175]. The proposed ECC solution adheres the safety requirements and corresponding fault injection mechanism is also provided which is designated for power-up check to ensure that the ECC error reporting mechanism is properly functioning.

The proposed ECC solution has the following important features meeting automotive SoC needs:

- It provides check bits not only on data bits but also on address bits. "Addresses in the code" is one of the most effective techniques to achieve a full coverage of the memory macro. In this way during mission mode ECC increases memory fault coverage by detecting errors on data and address bits.
- It takes into account the structural information of the memory providing ability to correct multi-bit errors typically caused by two or more neighboring bit failures. In addition, it allows to select or tune the multi-bit error correction rate from traditional single bit correction to double, triple or more bit correction rates.
- It provides possibility to target several types of memory size / width, interleaving, etc.
- It comprises in-field error injection capability to imitate errors and test the system response (both testing the HW path from the ECC to the receiving logic, as well as

allowing error handling facilities to be properly validated). A special option is provided to inject errors without corrupting the actual data. The automotive requirement here is that e.g. during POST, the ECC scheme can be checked whether it is functional by injected single-bit and double-bit errors and expect the appropriate values on error output bits. Special design techniques have been used in order not to spoil or corrupt the actual data, and thus the functional behavior of the module connected to the ECC.

- It provides in-system fault injection advanced control by managing all the in-field error injection signals (one per ECC instance) and find out if all ECCs are functioning or there are some that are not working fine, even when in a SoC there are hundreds or thousands of memories and each of the memories is protected by the proposed ECC. This solution makes easier the power on/power-down “check the checkers” test done on the ECC to make sure that safety mechanisms are working fine so that if during mission mode errors are occurred in the memory then ECC will be able to catch those errors. The current solution foresees the creation of the following link between Memory BIST and ECC: at power-up/power-down when Memory BIST tests the memories, it is used to check also the ECC functionality by activating ECC in-field error injection and then based on the observed results report to the system if there are non-working ECC instances.
- It includes a memory content repair for ECC-reported errors (read-modify-write procedure, also known as “memory scrubbing”). For conventional ECCs, ECC error detection and correction are done on ECC data output, and not in the memory itself. And if later on another error will appear in the same memory, it will not be corrected and can result in multi-bit uncorrectable error affecting the safety function. To avoid this, the proposed solution is able to correct the wrong data not only on the ECC data output but also in the memory itself. This again can be done by Memory BIST each time when the ECC will report an error. By this the accumulation of errors in the

memory is prevented, as well as the probability of multiple/uncorrectable errors is decreased.

Table 53 summarizes the possible scenarios and usage of the proposed ECC features related to code calculation [175]. To estimate the memory fault coverage provided by this solution a detailed analysis has been done on a sample 16nm memory. The obtained results in terms of Diagnostic coverage (DC) are reported in Table 54 for permanent faults and in Table 55 for transient faults.

Table 53. ECC features and their usage

Configuration	Realistic	Scenario
Parity data only	YES	Small memory where address decoder is small or protected by other means. In some memories ECC may be too expensive (e.g. memories that are byte-writeable would incur 50%+ area penalty), and thus parity may be suitable.
ECC data only	YES	Small memory where address decoder is small or protected in another way.
ECC address only	NO	-
ECC data + ECC address	YES	Medium sized and big memories, where address decoder has a relevant Functional Safety importance.

Table 54. Diagnostic coverage results on 16nm memory for permanent faults

Permanent Fault model	Diagnostic coverage (DC)
Stuck-at faults on address decoder input	100%
Stuck-at faults on address decoder internal logic or output (word-line)	98.125%
Activation/deactivation delay faults (due to open defects inside address decoder)	100%

The area and gate count overhead for ECC using 16nm technology node is reported in Table 56. It shows that memory area will be increased by 19% in case ECC data is used and in case of using ECC data and ECC address it will be increased by 24%. ECC will add logic area for both cases but based on experiments and use cases the area increase numbers are in acceptable range.

Table 55. Diagnostic coverage results on 16nm memory for transient faults

Transient Fault model	Diagnostic coverage (DC)
Single bit flip on address decoder input	100%
Multiple bit flip on address decoder input	~100% (almost always detected)

Table 56. Area numbers for ECC solution using 16nm technology node

ECC Option	No ECC	ECC data	ECC data + ECC address
Memory configuration	1024x64	1024x72 (8 code bits for data)	1024x76 (8 code bits for data + 4 code bits for address)
Memory area (sq microns)	10528	12514 (~19% increase)	13048 (~24% increase)
Logic area (sq microns)	0	191	218
Logic gate count (eq NAND2)	0	923	1053

8.4.4 Secure Test Solution for SoCs

So far the most relevant approaches in the sphere of test security were discussed, which are targeted mostly to system-level protection of test access port and related test interfaces. Nevertheless, the aspect of individual IP or embedded memories security is yet another important topic which needs separate discussion. This chapter presents a secure test solution for SoC taking into account the specifics of test architecture for embedded memories and IPs. In the typical case the embedded memories (e.g., SRAM, DRAM, Flash) and IPs (e.g., USB, DDR, PLL) placed on SoC are wrapped with Test Wrapper (in this work it is considered as IEEE 1500-compliant) for making the test access uniform.

TAP is the standard port for accessing SoC in order to perform test and debug procedures. Test Controller is responsible for shifting in the test vectors into corresponding cores under the test, scheduling and executing the test process and shifting out the responses. Test Controller controls the test process via several registers including instruction, data, bypass, test algorithm and possibly other user-defined registers. The execution of test-related operations, e.g., selecting the specific test instruction, algorithm in case of memories or specific test pattern in case of IPs are performed via the above mentioned registers.

The solution considered in this work proposes the concept of the flexible test architecture with two possible modes of operations, secure and insecure [176], [177]. In a secure mode a light-weight security logic is embedded into the Test Controller which controls access to the most vulnerable assets of the system identified during the manufacturing. If at least one security feature is enabled during the manufacturing, then the system is automatically placed in secure mode, otherwise no security feature is added in an insecure mode. This architecture gives user the opportunity to select the operation mode as well as the scale of the security features during the manufacturing thereby adjusting them to application needs. In the frame of the discussed typical test architecture the embedded memories, IPs and the user data registers (UDR), where the most sensitive information is usually stored by the users are the primary targets of the attackers. The security logic contains an authorization scheme which ensures that only the eligible users are able to access this type of sensitive information. Otherwise for unauthorized users the access is blocked using the below listed protection mechanisms. In a typical case access to the embedded memory is either via Test Algorithm Register Chain using the test algorithm programmability or via Serial Access Chain using IEEE 1500 WPRELOAD instruction. Therefore, for ensuring the memory content protection in a secure mode the following two countermeasures are applied:

- Memory is prohibited from being accessed during BIST execution by resetting the memory content.

- If one of the Test Algorithm Register Chain or Serial Access Chain are selected using the instruction register, then they are replaced with Bypass Register Chain in order to block memory access.

For IPs there is usually a single serial access path to the IP to shift in the corresponding instructions so in secure mode the access to the protected IPs will be forbidden and IPs will be placed in a bypass mode until the user access permission is not confirmed via authentication process. In a similar way access to protected UDRs, allowing storage of sensitive user data, is disabled for unauthorized users. The eligible users need to enter the correct key combination in order to bypass the mentioned protection mechanisms. The proposed solution also gives an opportunity to establish the flexible multiuser permission system defining different key combinations, e.g., for individual or user-defined groups of memories, UDRs as well as IPs. The system assets which need protection and the corresponding mapping of permissions for them are defined during the manufacturing and later cannot be modified in the field operation.

8.5 Test Time Calculation for Automotive SoC

Synopsys STAR Memory System [129] is enhanced based on the proposed solution which is certified as ASIL-D and provides all the necessary capabilities for testing and repairing automotive SoCs. STAR Memory System is designed to provide trade-off between safety level, test time, area and performance. An experiment is done on a project to calculate the test time for different test algorithms dedicated for production, power-up and mission modes. The project details are adduced below:

- Number of memory instances in the project are ~500.
- The memories are divided into four groups and for each group one BIST block (instance of STAR Memory System) is generated.
- Memories grouped under one BIST block are tested in parallel (assuming that the power budget allows this). Otherwise, there is a way to divide the group into subgroups and run the BIST for each subgroup individually.

- BIST blocks are hierarchically connected to the top-level module which allows to run the BIST blocks in parallel, sequentially (serial mode) or in any custom schedule (e.g., two BIST blocks at a time).
- Clock frequency is 500MHz.
- Three test algorithms are used for the experiments:
  - TestAlgo1 with linear complexity of 8N (for mission mode), where N is the number of memory addresses;
  - TestAlgo2 with linear complexity of 16N (for power-up mode);
  - TestAlgo3 with linear complexity of 55N (for production mode).
- Two run scenarios are considered:
  - Scenario1: All four BIST blocks are run in parallel;
  - Scenario2: The BIST blocks are run sequentially one after another.

Table 57 summarizes the test times (in clock cycles and milliseconds) obtained for the selected three test algorithms applied with the above mentioned two scenarios. As it can be seen from the table, the test time for the worst case is less than 1ms. Meanwhile, automotive SoCs usually have 1-10ms time budget for power-up and mission mode test and more than 10ms time budget for production test. These results show that the proposed test solution is efficient and fully meets the testing requirements of automotive SoCs.

Table 57. Test time calculations

Test algorithm	Scenario1 (Parallel) Clock cycles (run time)	Scenario2 (Serial) Clock cycles (run time)
TestAlgo1	33000 (0.066ms)	80000 (0.16ms)
TestAlgo2	57000 (0.114ms)	123000 (0.246ms)
TestAlgo3	193000 (0.386ms)	383000 (0.766ms)

Conclusions

1. Functional safety and security requirements and solutions for automotive SoCs are discussed. Test requirements for production, power-up and mission modes of the system are described, as well as advanced solutions for built-in self-test, error correcting codes, hierarchical test and security are introduced.
2. ISO 26262 Standard titled "Road vehicles – Functional safety" is presented which establishes functional safety definitions and requirements for automotive equipment applicable throughout the life-cycle of all automotive electronic and electrical safety-oriented systems.
3. Application of the proposed hierarchical test solution to automotive systems are demonstrated and justified that it meets the functional safety and security requirements.

MAIN CONCLUSIONS

The thesis is dedicated to development and its software and hardware implementation of a new unified methodology for fault modeling and test algorithm construction which led to essential progress in area of testing of nanoscale memory devices and systems. The methodology creates possibility to clarify and systemize the current perceptions about existing fault models, as well as to predict sets of possible new faults in future technological nodes.

1. The proposed methodology includes new models of faults and flow for their classification and diagnosis, minimal test algorithms and efficient methods for their construction, as well as a built-in test system which is programmable, extendable and dynamically adaptable to existing test systems and applications.
2. A learning and prediction mechanism for memory faults is developed which is based on periodicity and regularity properties of faults and corresponding test algorithms for their detection. For the implementation of the mechanism, using those rules, fault periodicity table and test algorithm template are constructed which allows to construct efficient test algorithms as an alternative to exhaustive or heuristic methods for generation of test algorithms.
3. From the proposed methodology follows a unified architecture of built-in test infrastructure in system-on-chip (SoC) which is easily adaptable to new faults. The developed hierarchical test architecture provides a unified solution for testing different IP blocks in SoC, as well as for scheduling parallel and serial testing of IP blocks.

The usage of the obtained results led to essential improvement of the test system based on which:

- the test time is reduced by 18%-44%;
- the occupied area of the test system is reduced by 7%-48%
- overall chip area is reduced by up to 5% allowing to put more functional logic in the chip.

By using the developed methodology all these led to essential increase of economic efficiency.

The results are implemented in Synopsys DesignWare STAR Memory System (SMS) and DesignWare STAR Hierarchical System (SHS) products and are widely used by more than 200 customers in their nanoscale SoC designs. Particularly, 10 of top 25 companies in the world from semiconductor industry are using these systems for performing built-in test of their products.

Thus, the obtained results serve as a basis for a complete test solution for nanoscale hierarchical SoCs which increases the test efficiency, reduces the test cost and improves quality of the test.

The done work is not just a basis for further research in this area which is confirmed by multiple links to the obtained results by other authors, as well as it provides a broad range of new applications which is built on the obtained results, and covering all the development phases of nanoscale SoCs: design, silicon bring-up, volume production and in-system test.

The scientific novelty of the work is summarized as follows:

- A unified methodology for software modeling of faults and test algorithm creation for nanoscale planar and 3D technology based memory systems including:
    - New fault models, their justification and efficient methods for their simulation and new models' generation ([32], [56], [58], [71]-[75], [85]-[87], [121]);
    - Fault classification and diagnosis flow ([90], [112], [122]-[128]);
    - Efficient test algorithms for detection and diagnosis of new fault types ([27], [28], [32], [34], [37], [39], [57], [90], [111], [122]-[128]);
    - Extendable and dynamically adaptable BIST architecture formed by a basic triad: test operations, addressing methods and layout aware physical background patterns ([11], [12], [47], [104], [129]);
    - A new efficient method for detection and correction of multi-bit soft errors ([172]-[175]).

- Fault prediction mechanism for memory devices of the current and upcoming technology nodes:
    - A systematic evolving view - Fault Periodicity Table (FPT) of possible memory faults with rules of periodicity and regularity as pillars ([8]-[12], [93], [94]);
    - A Test Algorithm Template (TAT) which allows to construct efficient test algorithms as an alternative to exhaustive or heuristic generation of test algorithms ([8]-[12]);
    - Special notions and measures to optimize construction of test algorithms ([9], [10], [32], [86], [92], [112]).
- A hierarchical test architecture for SoCs:
    - An efficient method for building an IP structural model for design and verification independently of IP final implementation ([130], [139]);
    - A unified solution to test different types of IP cores in the SoCs ([138], [168], [169]);
    - An algorithm for scheduling parallel and serial testing of IPs and BIST subsystems ([138]).
- Integration and tuning of created approaches to existing test systems and applications:
    - Interfaces with systems providing solutions for chip development cycle including automated test pattern generation, design planning, test time estimation, yield ramp-up, physical failure analysis, fault coverage and yield reports ([141], [143]);
    - Solutions to address functional safety and security requirements ([156], [168], [169], [174]-[177]).

LIST OF USED LITERATURE

[1] P. Ranade, Y.-K. Choi, D. Ha, H. Takeuchi, T.-J. King, “Metal Gate Technology for Fully Depleted SOI CMOS”, International AVS Conference on Microelectronics and Interfaces, 2003, pp. 131-133.

[2] H.-W. Cheng, Y. Li, “16-nm Multigate and Multifin MOSFET Device and SRAM Circuits”, International Symposium on Next-Generation Electronics, 2010, pp. 32-35.

[3] M. Jurczak, N. Collaert, A. Veloso, T. Hoffmann, S. Biesemans, “Review of FinFET Technology”, IEEE International SOI Conference, 2009, pp. 1-4.

[4] D.-I. Moon, S.-J. Choi, J.P. Duarte, Y.-K. Choi, “Investigation of Silicon Nanowire Gate-All-Around Junctionless Transistors Built on a Bulk Substrate”, IEEE Transactions on Electron Devices, 2013, Vol. 60, No. 4, pp. 1355-1360.

[5] R. S. Patti, “Three-Dimensional Integrated Circuits and the Future of System-on-Chip Designs”, Proceedings of the IEEE, 2006, Vol. 94, No. 6, pp. 1214–1224.

[6] S. Deutsch, K. Chakrabarty, “Test and Debug Solutions for 3D-Stacked Integrated Circuits”, IEEE International Test Conference, 2015, pp. 1-10.

[7] D. Han, Y. Lee, S. Kang, "A New Multi-site Test for System-on-Chip Using Multi-site Star Test Architecture", ETRI Journal, 2014, Vol. 36, No. 2, pp. 293-300.

[8] Գ.Է. Հարությունյան, “Թեստավորման արդյունավետ մոտեցում նանոչափական հիշող սարքերի համար”, ՀՀ ԳԱԱ Զեկույցներ, 2017, 117 (1), էջեր՝ 35-43.

[9] G. Harutyunyan, S. Shoukourian, Y. Zorian, “Fault and Test Algorithm Periodicity Hypothesis in Memory Devices and Its Application to Memory BIST Processor Architecture”, Reports of National Academy of Sciences of Armenia, 2012, Vol. 112, No. 3, pp. 229-238.

[10] G. Harutyunyan, S. Shoukourian, V. Vardanian, Y. Zorian, “An effective solution for building memory BIST infrastructure based on fault periodicity”, IEEE VLSI Test Symposium, 2013, pp. 71-76.

[11] A. Hakhumyan, G. Harutyunyan, S. Shoukourian, V. Vardanian, Y. Zorian, “Testing Electronic Memories Based on Fault and Test Algorithm Periodicity”, Patent No.: US 9,831,000, Date of Patent: November 28, 2017, Appl. No. 14/484,736, filed on September 12, 2014.

[12] Գ.Է. Հարությունյան, “Համապիտանի ներկառուցված թեստավորման համակարգ՝ հիմնված անսարքությունների պարբերական աղյուսակի և թեստային ալգորիթմների շաքլոնի վրա”, ՀՀ ԳԱԱ և ՀՊՃՀ Տեղեկագիր: Տեխնիկական գիտություններ, 2017, 70 (1), էջեր՝ 64-72.

[13] “Six Top 20 1Q15 Semiconductor Suppliers Show >20% Growth”, Research Bulletin, IC Insight, 2015.

[14] A.J. van de Goor, Testing semiconductor memories: Theory and Practice, John Wiley & Sons, Chichester, England, 1991.

[15] S. Hamdioui, A.J. van de Goor, M. Rodgers, “March SS: a test for all static simple faults”, IEEE Workshop MTDT, 2002, pp. 95-100.

[16] S. Hamdioui, G. N. Gaydadjiev A.J.van de Goor, “A fault primitive based analysis of dynamic memory faults”, IEEE Workshop on Circuits, Systems and Signal Processing, 2003, pp 84-89.

[17] S. Hamdioui, A. J. van de Goor, M. Rodgers, “Linked faults in Random Access Memories: concept, fault models, test algorithms, and industrial results”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 23, No. 5, May 2004, pp. 737-756.

[18] L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, “Dynamic read destructive fault in embedded-SRAMs: analysis and march test solution” IEEE European Test Symposium, 2004, pp. 140–145.

[19] S. Hamdioui, R. Wadsworth, J. D. Reyes, A.J. van de Goor, “Importance of Dynamic Faults for New SRAM Technologies”, IEEE European Test Workshop, 2003, pp. 29-34.

[20] M. Azimane, A. K. Majhi, G. Gronthoud, M. Lousberg, et al, “A New Algorithm for Dynamic Faults Detection in RAMs”, IEEE VLSI Test Symposium, 2005, pp. 177-182.

[21] A. Ney, P. Girard, C. Landrault, S. Pravossoudovitch, A. Virazel, M. Bastian, “Slow Write Driver Faults in 65nm SRAM Technology: Analysis and March Test Solution”, IEEE Design, Automation and Test in Europe, 2007, pp. 528-533.

[22] A. Ney, P. Girard, C. Landrault, S. Pravossoudovitch, A. Virazel and M. Bastian, “Dynamic Two-Cell Incorrect Read Fault due to Resistive-Open Defects in the Sense Amplifiers of SRAMs”, IEEE European Test Symposium, 2007, pp. 97-104.

[23] Z. Al-Ars, S. Hamdioui, G. Gaydadjiev, “Manifestation of Precharge Faults in High Speed DRAM Devices”, IEEE Design and Diagnostics of Electronic Circuits and Systems, 2007, pp. 179-184.

[24] A. Bosio, L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, "Advanced Test Methods for SRAMs: Effective Solutions for Dynamic Fault Detection in Nanoscaled Technologies", Springer, 2009.

[25] A.J. van de Goor, S. Hamdioui, G. Gaydadjiev, Z. Al-Ars, “New Algorithms for Address Decoder Delay Faults and Bit Line Imbalance Faults”, IEEE Asian Test Symposium, 2009, pp. 391-396.

[26] R. A. Fonseca, L. Dilillo, A. Bosio, P. Girard, S. Pravossoudovitch, A. Virazel, N. Badereddine, “Impact of Resistive-Bridging Defects in SRAM Core-Cell”, IEEE International Symposium on Electronic Design, Test & Applications, 2010, pp. 265-269.

[27] G. Harutunyan, V.A. Vardanian, Y. Zorian, “Minimal March Tests for Dynamic Faults in Random Access Memories”, IEEE European Test Symposium, 2006, pp. 43-48.

[28] G. Harutunyan, V. A. Vardanian, Y. Zorian, “Minimal March Tests for Detection of Dynamic Faults in Random Access Memories”, Journal of Electronic Testing: Theory and Applications, Vol. 23, No. 1, February 2007, pp. 55-74.

[29] H. Avetisyan, G. Harutunyan, V.A. Vardanian, “Efficient March-Like Algorithm for Detection of All Two-Operation Dynamic Faults from Subclass Sav”, Mathematical problems of cybernetics and computer science, Armenia, 2008, pp. 18-24.

[30] H. Avetisyan, G. Harutyunyan, V.A. Vardanian, Y. Zorian, “An Efficient March Test for Detection of All Two-Operation Dynamic Faults from Subclass S_{av}”, IEEE East-West Design and Test Symposium, 2010, pp. 310-313.

[31] H. S. Avetisyan, G. E. Harutyunyan, V. A. Vardanian, “Minimal March Test Algorithms for Detection of All Realistic Two-Operation, Two-Cell Dynamic Faults from Subclasses Sav and Sva”, Reports of National Academy of Sciences of Armenia, 2010, Vol. 110, No. 2, pp. 143-150.

[32] G. Harutunyan, V. Vardanian, Y. Zorian, “Minimal march test algorithm for detection of linked static faults in Random Access Memories”, IEEE VLSI Test Symposium, 2006, pp. 120-125.

[33] A.J. van de Goor, G. N. Gaydadjiev, V.G. Mikitjuk, V.N. Yarmolik, “March LR: A Test for Realistic Linked Faults”, IEEE VLSI Test Symposium, 1996, pp. 272-280.

[34] G. Harutunyan, V. A. Vardanian, “Minimal March Tests for Dynamic Faults in Random Access Memories”, IEEE European Test Symposium, 2007, pp. 223 – 227.

[35] Z. Qi, J. Wang, A. Cabe, S. Wooters, T. Blalock, B. Calhoun, M. Stan, SRAM-Based NBTI/PBTI Sensor System Design, Design Automation Conference, 2010, pp. 849-852.

[36] W. Prates, L. Bolzani, G. Harutyunyan, A. Davtyan, F. Vargas, Y. Zorian, "Integrating Embedded Test Infrastructure in SRAM Cores to Detect Aging", IEEE International On-Line Testing Symposium, 2013, pp. 25-30.

[37] K. Amirkhanyan, H. Grigoryan, G. Harutyunyan, T. Melkumyan, S. Shoukourian, A. Shubat, V. Vardanian, Y. Zorian, “Detecting Random Telegraph Noise Induced Failures in

an Electronic Memory", Patent No.: US 8,850,277 B2, Date of Patent: September 30, 2014, Appl. No. 13/183,471, filed on July 15, 2011.

[38] J.-F. Li, K.-L. Cheng, C.-T. Huang, and C.-W. Wu, "March based RAM diagnostic algorithms for stuck-at and coupling faults", IEEE ITC, 2001, pp. 758-767.

[39] G. Harutyunyan, V. A. Vardanian, "An Efficient March Test Algorithm for Detection of Resistive Shorts in Multi-Port SRAMs", Computer Science and Information Technologies, 2009, pp. 435-438.

[40] K. Aleksanyan, K. Amirkhanyan, S. Shoukourian, V. Vardanian, Y. Zorian, "Memory Modeling Using an Intermediate Level Structural Description", US Patent, No. US 7,768,840 B1, Aug. 3, 2010.

[41] S. Boutobza, M. Nicolaidis, K.M. Lamara, A. Costa, "A Transparent based Programmable Memory BIST", IEEE European Test Symposium, 2006.

[42] D. Youn, T. Kim, S. Park, "A microcode-based memory BIST implementing modified march algorithm", IEEE Asian Test Symposium, 2001, pp. 391-395.

[43] K. Zarrineh and S. J. Upadhyaya, "On Programmable Memory Built-In Self Test Architectures," IEEE DATE, 1999, pp. 708–713.

[44] A. A. Ivaniuk, "Optimal Memory Tests Coding for Programmable BIST Architecture", Journal of "Radioelectronics & Informatics", 2008, No. 4, pp. 32-37.

[45] A. Benso, S. Di Carlo, G. Di Natale, M. L. Bodoni, P. Prinetto, “Programmable Built-In Self-Testing of Embedded RAM Clusters in System-on-Chip Architectures”, IEEE Communications Magazine, Vol. 41, No. 9, September 2003, pp. 90-97.

[46] X. Du, N. Mukherjee, W.-T. Cheng, S. M. Reddy, “Full-Speed Field Programmable Memory BIST Supporting Multi-level Looping”, IEEE International Workshop on Memory Technology, Design, and Testing, 2005, pp. 67-71.

[47] A. Hakhumyan, G. Harutyunyan, “Implementation of a Flexible BIST Architecture Based on Programmability of Test Operations, Patterns and Algorithms”, Computer Science and Information Technologies, 2011, pp. 287-290.

[48] G.E. Moore “Cramming More Components Onto Integrated Circuits”, Electronics, Vol. 38, No. 8, 1965, pp. 114-117.

[49] K. Park, J. Lee, S. Kang, "An Area Efficient Programmable Built-In Self-Test for Embedded Memories Using an Extended Address Counter", IEEE International SoC Design Conference, 2010, pp. 59-62.

[50] P. Bernardi, M. Grosso, M. Sonza Reorda, Y. Zhang, "A Programmable BIST for DRAM testing and diagnosis", IEEE International Test Conference, 2010.

[51] S. Boutobza, M. Nicolaidis, K.M. Lamara, A. Costa, “Programmable memory BIST”, IEEE International Test Conference, 2005, pp. 1155-1164.

[52] S.M. Al-Harbi, S.K. Gupta, “An efficient methodology for generating optimal and uniform March Tests”, IEEE VLSI Test Symposium, 2001, pp. 231 -237.

[53] C.-F. Wu, C.-T. Huang, K.-L. Cheng, C.-W. Wu, “Fault Simulation and Test Algorithm Generation for Random Access Memories”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 21, No. 4, April 2002, pp. 480-490.

[54] A. Benso, A. Bosio, S. Di Carlo, G. Di Natale, P. Prinetto, “March Test Generation Revealed”, IEEE Transactions on Computers, Vol. 57, No. 12, December 2008, pp. 1704-1713.

[55] A. Benso, A. Bosio, S. Di Carlo, G. Di Natale, P. Prinetto, “Automatic march tests generation for static and dynamic faults in SRAMs”, IEEE European Test Symposium, 2005, pp. 122-127.

[56] G. Harutunyan, D. Melkumyan, H. Elchyan, V. Vardanian, “An Efficient Method for Generation of March Tests Based on Formulas”, Mathematical problems of cybernetics and computer science, Armenia, 2008, pp. 5-17.

[57] G. Harutunyan, V. A. Vardanian, Y. Zorian, “Minimal March Tests for Unlinked Static Faults in Random Access Memories”, IEEE VLSI Test Symposium, 2005, pp. 53-59.

[58] G. Harutunyan, “A Software Tool for Generation of March Algorithms for Faults in SRAMs”, IEEE East-West Design and Test Symposium, 2007, pp. 444-447.

[59] X. Huang, W.-C. Lee, C. Kuo, et al, “Sub 50-nm FinFET: PMOS”, International Electron Devices Meeting Technical Digest, 1999, pp. 67-70.

[60] T.-J. King, “FinFETs for nanoscale CMOS digital integrated circuits”, IEEE/ACM International Conference on Computer-Aided Design, 2005, pp. 207-210.

[61] A. Bansal, S. Mukhopadhyay, K. Roy, “Device-optimization technique for robust and low-power FinFET SRAM design in nanoscale era”, IEEE Transactions on Electron Devices, Vol. 54, No. 6, 2007, pp. 1409-1419.

[62] A. Carlson, Z. Guo, S. Balasubramanian, R. Zlatanovici, T.-J. K. Liu, B. Nikolić, “SRAM read/write margin enhancements using FinFETs”, IEEE Tranaction on Very Large Scale Integation. Systems, Vol. 18, No. 6, 2010, pp. 887-900.

[63] J. J. Gu, Y. Q. Liu, Y. Q. Wu, R. Colby, R. G. Gordon, P. D. Ye, “First experimental demonstration of gate-all-around III-V MOSFETs by top-down approach”, IEDM Tech. Dig., 2011, pp. 769-772.

[64] Y. Liu, Q. Xu. “On modeling faults in FinFET logic circuits”, IEEE International Test Conference, 2012, pp. 1-9.

[65] C.-W. Lin, M. C.-T. Chao, C.-C. Hsu, “Investigation of gate oxide short in FinFETs and the test methods for FinFET SRAMs”, VLSI Test Symposium, 2013, pp. 1-6.

[66] M.-h. Chi, “Challenges in manufacturing FinFET at 20nm node and beyond”, Globalfoundries, 2012.

[67] M. O. Simsir, A. N. Bhoj, N. K. Jha, “Fault modeling for FinFET circuits”, IEEE/ACM International Symposium on Nanoscale Architectures, 2010, pp. 41-46.

[68] A. N. Bhoj, M. O. Simsir, N. K. Jha, “Fault models for logic circuits in the multigate era”, IEEE Transactions on Nanotechnology, Vol. 11, No. 1, 2012, pp. 182-193.

[69] J. Vazquez, V. Champac, C. Hawkins, J. Segura, “Stuck-open fault leakage and testing in nanometer technologies”, IEEE VLSI Test Symposium, 2009, pp. 315–320.

[70] V. Champac, J. V. Hernandez, S. Barcelo, R. Gomez, C. Hawkins, J. Segura, “Testing of stuck-open faults in nanometer technologies”, IEEE Design & Test of Computers, Vol. 29, No. 4, 2012, pp. 80-91.

[71] G. Harutyunyan, G. Tshagharyan, V. Vardanian, Y. Zorian, “Fault modeling and test algorithm creation strategy for FinFET-based memories”, VLSI Test Symposium, 2014, pp. 1-6.

[72] G. Harutyunyan, G. Tshagharyan, Y. Zorian, “Test & Repair Methodology for FinFET-Based Memories”, IEEE Transactions on Device and Materials Reliability, Vol. 15, No. 1, March 2015, pp. 3-9.

[73] G. Harutyunyan, G. Tshagharyan, Y. Zorian, “Impact of Parameter Variations on FinFET Faults”, IEEE VLSI Test Symposium, 2015, pp. 145-148.

[74] G. Tshagharyan, G. Harutyunyan, S. Shoukourian, Y. Zorian, “Overview Study on Fault Modeling and Test Methodology Development for FinFET-Based Memories”, IEEE East-West Design and Test Symposium, 2015, pp. 19-22.

[75] G. Tshagharyan, G. Harutyunyan, S. Shoukourian, Y. Zorian, “FinFET-Based Memory Testing Using Multiple Read Operations”, No. 15/718,284, filed on September 28, 2017.

[76] L. Jiang, Y. Liu, L. Duan, Y. Xie, Q. Xu, “Modeling TSV Open Defects in 3D-Stacked DRAM”, IEEE International Test Conference, 2010, pp. 174-182.

[77] Y.-Ch. Yu, Ch.-W. Chou, J.-F. Li, Ch.-Y. Lo, et al, "A Built-In Self-Test Scheme for 3D RAMs", IEEE International Test Conference, 2012, pp. 1-9.

[78] S. Deutsch, B. Keller, V. Chickermane, S. Mukherjee, et al, "DfT Architecture and ATPG for Interconnect Tests of JEDEC Wide-I/O Memory-on-Logic Die Stacks", IEEE International Test Conference, 2012, pp. 1-10.

[79] P.-Y. Chen, Ch.-W. Wu, D.-M. Kwai, "On-Chip Testing of Blind and Open-Sleeve TSVs for 3D IC before Bonding", IEEE VLSI Test Symposium, 2010, pp. 263 – 268.

[80] M. Cho, Ch. Liu, D. H. Kim, S. K. Lim, et al, "Design Method and Test Structure to Characterize and Repair TSV Defect Induced Signal Degradation in 3D System", IEEE/ACM International Conference on Computer-Aided Design, 2010, pp. 694-697.

[81] Y.-J. Huang, J.-F. Li, Ch.-W. Chou, "Post-Bond Test Techniques for TSVs with Crosstalk Faults in 3D ICs", IEEE International Symposium on VLSI Design, Automation, and Test, 2012, pp. 1-4.

[82] Y.-H. Lin, S.-Y. Huang, K.-H. Tsai, W.-T. Cheng, et al, "A Unified Method for Parametric Fault Characterization of Post-Bond TSVs", IEEE International Test Conference, 2012, pp. 1-10.

[83] Y.-J. Huang J.-F. Li, J.-J. Chen, D.-M. Kwai, et al, "A Built-In Self-Test Scheme for the Post-Bond Test of TSV s in 3D ICs", IEEE VLSI Test Symposium, 2011, pp. 20-25.

[84] M. Taouil, M. Masadeh, S. Hamdioui, E. J. Marinissen, "Interconnect Test for 3D Stacked Memory-on-Logic", Design, Automation and Test in Europe Conference, 2014, pp. 1-6.

[85] K. Amirkhanyan, A. Davtyan, G. Harutyunyan, T. Melkumyan, S. Shoukourian, V. Vardanian, Y. Zorian, "Application of Defect Injection Flow for Fault Validation in Memories", IEEE East-West Design and Test Symposium, 2012, pp. 19-22.

[86] G. Harutyunyan, S. Shoukourian, V. Vardanian, Y. Zorian, “A new method for march test algorithm generation and its application for fault detection in RAMs”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 31, No. 6, Jun. 2012, pp. 941-949.

[87] G. Harutyunyan, S. Shoukourian, V. Vardanian, Y. Zorian, “Impact of process variations on read failures in SRAMs”, IEEE East-West Design and Test Symposium, 2013, pp. 15-18.

[88] L. Dilillo, P. Girard, S. Pravossoudovitch, A. Virazel, “Efficient march test procedure for dynamic read destructive fault detection in SRAM memories”, Journal of Electronic Testing: Theory and Applications, Vol. 21, No. 5, Oct. 2005, pp. 551–561.

[89] A. Benso, A. Bosio, S. Di Carlo, G. Di Natale, P. Prinetto, “March AB, March AB1: New March tests for unlinked dynamic memory faults”, IEEE International Test Conference, 2005, pp. 834-841.

[90] G. Harutunyan, V.A. Vardanian, Y. Zorian, “Minimal March-Based Fault Location Algorithm with Partial Diagnosis for All Static Faults in Random Access Memories”, IEEE Design and Diagnosis of Electronic Circuits and Systems, 2006, pp. 260-265.

[91] S. Hamdioui, R. Wadsworth, J.D. Reyes, A.J. van de Goor, “Memory Fault Modeling Trends: A Case Study”, Journal Electronic Testing: Theory and Applications, Vol. 20, No. 3, 2004, pp. 245-255.

[92] G. Harutyunyan, A. Hakhumyan, S. Shoukourian, V. Vardanian, Y. Zorian, “Symmetry Measure for Memory Test and Its Application in BIST Optimization”, Journal Electronic Testing: Theory and Applications, Vol. 27, No. 6, Dec. 2011, pp. 753-766.

[93] G. Harutyunyan, S. Shoukourian, V. Vardanian, Y. Zorian, “Extending Fault Periodicity Table for Testing Faults in Memories under 20nm”, IEEE East-West Design and Test Symposium, 2014, pp. 12-15.

[94] G. Harutyunyan, “Extending Fault Periodicity Table for Testing External Memory Faults”, IEEE East-West Design and Test Symposium, 2016, pp. 490-493.

[95] S. Hamdioui, Z. Al-Ars, A.J. van de Goor, “Testing Static and Dynamic Faults in Random Access Memories”, IEEE VLSI Test Symposium, 2002, pp. 395-400.

[96] K. Pagiamtzis, A. Sheikholeslami, “Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey”, IEEE Journal of Solid-State Circuits, March 2006, Vol. 41, No. 3, pp. 712-727.

[97] Y. Zorian, “Built-In-Self-Test Technique for Content-Addressable Memories”, 1992, US Patent 5107501.

[98] W. K. Al-Assadi, A. P. Jayasumana, and Y. K. Malaiya, “On fault modeling and testing of content-addressable Memories”, IEEE International Workshop on Memory Technology, Design and Testing, 1994, pp. 78-81.

[99] J. Zhao, S. Irrinki, M. Puri, and F. Lombardi, “Testing SRAM-Based Content Addressable Memories”, IEEE Transactions on Computers, October 2000, Vol. 49, No. 10, pp. 1054-1063.

[100] Zh. Xuemei, Y. Yizheng, Ch. Chunxu, “Tests for Word Oriented Content Addressable Memories”, Asian Test Symposium, 2002, pp. 151-156.

[101] J.-F. Li, “Testing Ternary Content Addressable Memories With Comparison Faults Using March-Like Tests”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, May 2007, Vol. 26, No. 5, pp. 919-931.

[102] M. Lin, Ch. Yunji, S. Menghao, Q. Zichu, Zh. Heng, H. Weiwu, “Testing Content Addressable Memories Using Instructions and March-Like Algorithms”, IEEE International Conference on Electronics, Circuits and Systems, 2008, pp. 774-777.

[103] P. Manikandan, B. B. Larsen, E. J. Aas, S. M. Reddy, "Test of Embedded Content Addressable Memories", IEEE International Symposium on Electronic System Design, 2010, pp. 113-118.

[104] H. Grigoryan, G. Harutyunyan, S. Shoukourian, V. Vardanian, Y. Zorian, “Generic BIST Architecture for Testing of Content Addressable Memories”, IEEE International On-Line Testing Symposium, 2011, pp. 86-91.

[105] B.-H. Lin, Sh.-H. Shieh, C.-W. Wu, “A Fast Signature Computation Algorithm for LFSR and MISR”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 19, No. 9, September 2000, pp. 1031-1040.

[106] M. C.-T. Chao, H.-Y. Yang, R.-F. Huang, Sh.-C. Lin, Ch.-Y. Chin, “Fault Models for Embedded-DRAM Macros”, Design Automation Conference, 2009, pp. 714-719.

[107] H. Aziza, J-M. Portal, J. Plantier, “Non volatile memory reliability prediction based on oxide defect generation rate during stress and retention tests”, International Semiconductor Device Research Symposium, 2011, pp. 1-2.

[108] K.-L. Cheng, J.-Ch. Yeh, Ch.-W. Wang, Ch.-T. Huang, Ch.-W. Wu, “RAMSES-FT: A Fault Simulator for Flash Memory Testing and Diagnostics”, IEEE VLSI Test Symposium, 2002, pp. 281-286.

[109] O. Ginez, J.-M. Daga, M. Combe, P. Girard, C. Landrault, S. Pravossoudovitch, A. Virazel, “An Overview of Failure Mechanisms in Embedded Flash Memories”, IEEE VLSI Test Symposium, 2006, pp. 108-113.

[110] T.-W. Kuo, P.-Ch. Huang, Y.-H. Chang, Ch.-L. Ko, Ch.-W. Hsueh, “An Efficient Fault Detection Algorithm for NAND Flash Memory”, ACM SIGAPP Applied Computing Review, Vol. 11, No. 2, 2011, pp. 8-16.

[111] S. Martirosyan, G. Harutyunyan, S. Shoukourian, Y. Zorian, “An Efficient Testing Methodology for Embedded Flash Memories”, IEEE East-West Design and Test Symposium, 2017, pp. 422-425.

[112] G. Harutyunyan, Y. Zorian, “An Effective Embedded Test & Diagnosis Solution for External Memories”, IEEE International On-Line Testing Symposium, 2015, pp. 168-170.

[113] K. Shibin, S. Devadze, A. Jutman, "On-line Fault Classification and Handling in IEEE1687 based Fault Management System for Complex SoCs", IEEE Latin-American Test Symposium, 2016, pp. 69-74.

[114] Z. Al-Ars, S. Hamdioui, "Fault Diagnosis Using Test Primitives in Random Access Memories", IEEE Asian Test Symposium, 2009, pp. 403-408.

[115] M. de Carvalho, P. Bernardi, M. Sonza Reorda, N. Campanelli, et al, "Optimized Embedded Memory Diagnosis", IEEE International Symposium on Design and Diagnostics of Electronic Circuits & Systems, 2011, pp. 347–352.

[116] N. Campanelli, T. Kerekes, P. Bernardi, M. de Carvalho, et al, "Cumulative embedded memory failure bitmap display & analysis", IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems, 2010, pp. 255-260.

[117] T.J. Bergfeld, D. Niggemeyer, E.M. Rudnick, "Diagnostic Testing of Embedded Memories Using BIST", Design, Automation and Test in Europe, 2000, pp. 305-309.

[118] Y. Zorian, S. Shoukourian, "Embedded-Memory Test and Repair: Infrastructure IP for SoC Yield", IEEE Design and Test of Computers, Vol. 20, No. 3, June 2003, pp. 58-66.

[119] S. Shoukourian, V. Vardanian, Y. Zorian, "SoC Yield Optimization via an Embedded-Memory Test and Repair Infrastructure", IEEE Design & Test of Computers, Vol. 21, No. 3, May-June 2004, pp. 200-207.

[120] A.J. van de Goor, I. Schanstra, "Address and Data Scrambling: Causes and Impact on Memory Tests", IEEE International Workshop on Electronic Design, Test and Applications, 2002, pp. 128-137.

[121] K. Amirkhanyan, K. Darbinyan, A. Davtyan, G. Harutyunyan, S. Shoukourian, V. Vardanian, Y. Zorian, “Generation of Memory Structural Model Based on Memory Layout”,

Patent No.: US 9,514,258, Date of Patent: December 6, 2016, Appl. No. 13/531,189, filed on June 22, 2012.

[122] G. Harutyunyan, S. Martirosyan, S. Shoukourian, Y. Zorian, “Memory Physical Aware Multi-Level Fault Diagnosis Flow”, IEEE Transactions on Emerging Topics in Computing, 2018.

[123] G. Harutunyan, V. Vardanian, “A March Test for Full Diagnosis of All Simple Static Faults in Random Access Memories”, IEEE East-West Design and Test Workshop, 2006, pp. 68-71.

[124] G. Harutyunyan, D. Melkumyan, “Fault Location and Diagnosis Algorithm for Static and Dynamic Faults in SRAMs”, Proceedings of the National Academy of Sciences of Armenia and the State Engineering University of Armenia. Series of Technical Sciences, 2010, Vol. 63, No. 3, pp. 280-287.

[125] G. Harutunyan, V. A. Vardanian, “Minimal March-Based Fault Location Algorithm with Partial Diagnosis for Random Access Memories”, Computer Science and Information Technologies, 2005, pp. 519-522.

[126] G. Harutunyan, V.A. Vardanian, Y. Zorian, “A March-Based Fault Location Algorithm with Partial and Full Diagnosis for All Simple Static Faults in Random Access Memories”, IEEE Design and Diagnosis of Electronic Circuits and Systems, 2007, pp. 145-148.

[127] G. Harutunyan, H. Kocharyan, V. A. Vardanian, “An Efficient 2-Phase March Algorithm for Full Diagnosis of All Simple Static Faults in Random Access Memories”, IEEE East-West Design and Test Symposium, 2007, pp. 110-113.

[128] G. Harutunyan, V. A. Vardanian, Y. Zorian, “An Efficient March-Based Three-Phase Fault Location and Full Diagnosis Algorithm for Realistic Two-Operation Dynamic Faults in Random Access Memories”, IEEE VLSI Test Symposium, 2008, pp. 95 – 100.

[129] K. Darbinyan, G. Harutyunyan, S. Shoukourian, V. Vardanian, Y. Zorian, “A Robust Solution for Embedded Memory Test and Repair”, IEEE Asian Test Symposium, 2011, pp. 461-462.

[130] T. Melkumyan, G. Harutyunyan, S. Shoukourian, V. Vardanian, Y. Zorian, "An Efficient Fault Diagnosis and Localization Algorithm for Successive-Approximation Analog to Digital Converters", IEEE East-West Design and Test Symposium, 2012, pp. 15-18.

[131] H. G. Kerkhoff, J. Wan, “Dependable Digitally-Assisted Mixed-Signal IPs Based on Integrated Self-Test & Self-Calibration”, IEEE International Mixed-Signals, Sensors and Systems Test Workshop, 2010, pp. 1-6.

[132] G. Li, J. Qian, Q. Yang, “Hybrid Hierarchical and Modular Tests for SoC Designs”, IEEE North Atlantic Test Workshop, 2015, pp. 11-16.

[133] Y. Zorian, S. Shoukourian, “Test Solutions for Nanoscale Systems-on-Chip: Algorithms, Methods and Test Infrastructure”, International Conference on Computer Science and Information Technologies, 2013, pp. 1-3.

[134] B. Keller, K. Chakravadhanula, B. Foutz, V. Chickermane, et al, “Efficient Testing of Hierarchical Core-Based SOCs”, IEEE International Test Conference, 2014, pp. 1-10.

[135] F. DaSilva, Y. Zorian, L. Whetsel, K. Arabi, R. Kapur, “Overview of the IEEE P1500 Standard”, IEEE International Test Conference, 2003, pp. 988-997.

[136] IEEE Std. 1149.1, IEEE Standard for Test Access Port and Boundary-Scan Architecture, 2001.

[137] 1687-2014 - IEEE Standard for Access and Control of Instrumentation Embedded within a Semiconductor Device.

[138] G. Harutyunyan, “An Approach for Scheduling Parallel and Serial Testing of Embedded IP Cores in Nanoscale SoCs”, Reports of National Academy of Sciences of Armenia, 2018, Vol. 118, No. 1, pp. 26-32.

[139] D. Sargsyan, G. Harutyunyan, S. Shoukourian, Y. Zorian, “Automated Flow for Test Pattern Creation for IPs in SoC”, IEEE East-West Design and Test Symposium, 2017, pp. 21-24.

[140] 1685-2014 - IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows.

[141] L. Martirosyan, G. Harutyunyan, S. Shoukourian, Y. Zorian, “A Power Based Memory BIST Grouping Methodology”, IEEE East-West Design and Test Symposium, 2015, pp. 27-30.

[142] V. Iyengary, K. Chakrabarty, E.J. Marinissen, “On Using Rectangle Packing for SOC Wrapper/TAM Co-Optimization”, IEEE VLSI Test Symposium, 2002, pp. 253-258.

[143] S. Martirosyan, G. Harutyunyan, S. Shoukourian, Y. Zorian, “Method and Apparatus for SoC with Optimal RSMA”, No. 15/684,780, filed on August 23, 2017.

[144] International Technology Roadmap for Semiconductors, Update 2012. [Online]. Available: http://www.itrs2.net.

[145] M. Nicolaidis, “Theory of Transparent BIST for RAMs”, IEEE Transactions On Computers, Vol. 45, No. 10, pp. 1141-1156, 1996.

[146] M.G. Karpovsky, V.N. Yarmolik, “Transparent memory BIST”, IEEE International Workshop on Memory Technology, Design and Testing”, 1994, pp. 106-111.

[147] I. Voyiatzis, C. Efstathiou, C. Sgouropoulou, “Symmetric transparent online BIST for arrays of word-organized RAMs”, International Conference on Design & Technology of Integrated Systems in Nanoscale Era, 2013, pp. 122-127.

[148] A. Dutta, S. Alampally, A. Kumar, R. A. Parekhji, "A BIST Implementation Framework for Supporting Field Testability and Configurability in an Automotive SOC", Workshop on Dependaable and Secure Nanocomputing, 2007.

[149] A. Cook, D. Ull, M. Elm, H.-J. Wunderlich, H. Randoll, S. Dohren, “Reuse of Structural Volume Test Methods for In-System Testing of Automotive ASICs”, IEEE Asian Test Symposium, 2012, pp. 214-219.

[150] F. Reimann, M. Glass, A. Cook, L. Rodriguez Gomez, J. Teich, D. Ull, H.-J. Wunderlich, U. Abelein, and P. Engelke, “Advanced diagnosis: SBST and BIST integration in automotive E/E architectures,”, ACM/EDAC/IEEE Design Automation Conference, 2014, pp. 1–6.

[151] P. Bernardi, R. Cantoro, S. D. Luca, E. Sánchez, A. Sansonetti, “Development Flow for On-Line Core Self-Test of Automotive Microcontrollers”, IEEE Transactions on Computers, Vol. 65, No. 3, pp. 744 – 754, March 2016.

[152] A. Jutman, M. S. Reorda, H.-J. Wunderlich, “High Quality System Level Test and Diagnosis”, IEEE Asian Test Symposium, 2014, pp. 298-305.

[153] J. C. Vazquez, V. Champac, A.M. Ziesemer Jr., R. Reis, et al, “Built-in aging monitoring for safety-critical applications”, IEEE European Test Symposium, 2009, pp. 9-14.

[154] D. Appello, P. Bernardi, R. Cagliesi, M. Giancarlini, et al, “Automatic Functional Stress Pattern Generation for SoC Reliability Characterization”, IEEE European Test Symposium, 2009, pp. 93-98.

[155] ISO 26262, https://www.iso.org/standard/43464.html.

[156] G. Tshagharyan, G. Harutyunyan, Y. Zorian, “An Effective Functional Safety Solution for Automotive Systems-on-Chip”, IEEE International Test Conference, 2017, Paper ET 2.2, pp. 1-10.

[157] Tezzaron Semiconductor, “Soft Errors in Electronic Memory – A White Paper”, http://www.tezzaron.com, Jan. 2004.

[158] R. C. Baumann, “Radiation-induced soft errors in advanced semiconductor technologies”, IEEE Transactions on Device and Materials Reliability, Vol. 5, No. 3, pp. 305-316, Sep. 2005.

[159] D. Hely, F. Bancel, M.-L. Flottes, B. Rouzeyre, “Test Control for Secure Scan Designs”, European Test Symposium, 2005, pp.190-195.

[160] F. Novak, A. Biasizzo, “Security extension for IEEE std 1149.1”, Journal of Electronic Testing, Vol. 22, No. 3, pp. 301–303, June 2006.

[161] K. Rosenfeld, R. Karri, "Attacks and Defenses for JTAG", IEEE Design & Test of Computers, Vol. 27, No. 1, pp. 36-47, January-February 2010.

[162] L. Pierce, S. Tragoudas, “Enhanced Secure Architecture for Joint Action Test Group Systems”, IEEE Transactions on Very Large Scale Integration Systems, Vol. 21, No. 7, pp. 1342-1345, July 2013.

[163] G. Sengar, D. Mukhopadhayay, D.R. Chowdhury, “An Efficient Approach to Develop Secure Scan Tree for Crypto-Hardware”, International Conference on Advanced Computing and Communications, 2007, pp. 21-26.

[164] J. Lee., M. Tehranipoor, C. Patel, J. Plusquellic, “Securing Designs against Scan-Based Side-Channel Attacks”, IEEE Transactions on Dependable and Secure Computing, Vol. 4, No. 4, pp. 325-336, October-December 2007.

[165] R. Baranowski, M. A. Kochte, H.-J. Wunderlich, “Fine-Grained Access Management in Reconfigurable Scan Networks”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 34, No. 6, pp. 937-946, June 2015.

[166] J. Dworak, A. Crouch, J. C. Potter, A. Zygmontowicz, M. Thornton, “Don't forget to lock your SIB: hiding instruments using P1687”, IEEE International Test Conference, 2013, page 1-10.

[167] G.-M. Chiu, J.C.-M. Li, "A Secure Test Wrapper Design Against Internal and Boundary Scan Attacks for Embedded Cores", IEEE Transactions on Very Large Scale Integration Systems, Vol. 20, No. 1, pp. 126 – 134, Jan. 2012.

[168] T. Kogan, Y. Abotbol, G. Boschi, G. Harutyunyan, I. Kroul, H. Shaheen, Y. Zorian, "Advanced Functional Safety Mechanisms for Embedded Memories and IPs in Automotive SoCs", IEEE International Test Conference, 2017, Paper 14.2, pp. 1-6.

[169] Ch. Eychenne, G. Harutyunyan, Y. Zorian, "System-on-Chip Online Self-Test Runtime Control for Functional Safety", IEEE International Workshop on Automotive Reliability & Test, 2017, Paper S01-01, pp. 1-4.

[170] R. W. Hamming, "Error Detecting and Error Correcting Codes", Bell System Technical Journal, Vol. 29, No. 2, pp. 147-160, 1950.

[171] M.Y. Hsiao, "A Class of Optimal Minimum Odd-weight-column SEC-DED Codes", IBM Journal of Research and Development, Vol. 14, No. 4, pp. 395-401, July 1970.

[172] G. Tshagharyan, G. Harutyunyan, S. Shoukourian, Y. Zorian, "Experimental study on Hamming and Hsiao Codes in the Context of Embedded Applications", IEEE East-West Design and Test Symposium, 2017, pp. 25-28.

[173] H. Grigoryan, G. Harutyunyan, S. Shoukourian, V. Vardanian, Y. Zorian, "Determining a Desirable Number of Segments for a Multi-Segment Single Error Correcting Coding Scheme", Patent No.: US 9,053,050, Date of Patent: June 9, 2015, Appl. No. 13/310,479, filed on December 2, 2011.

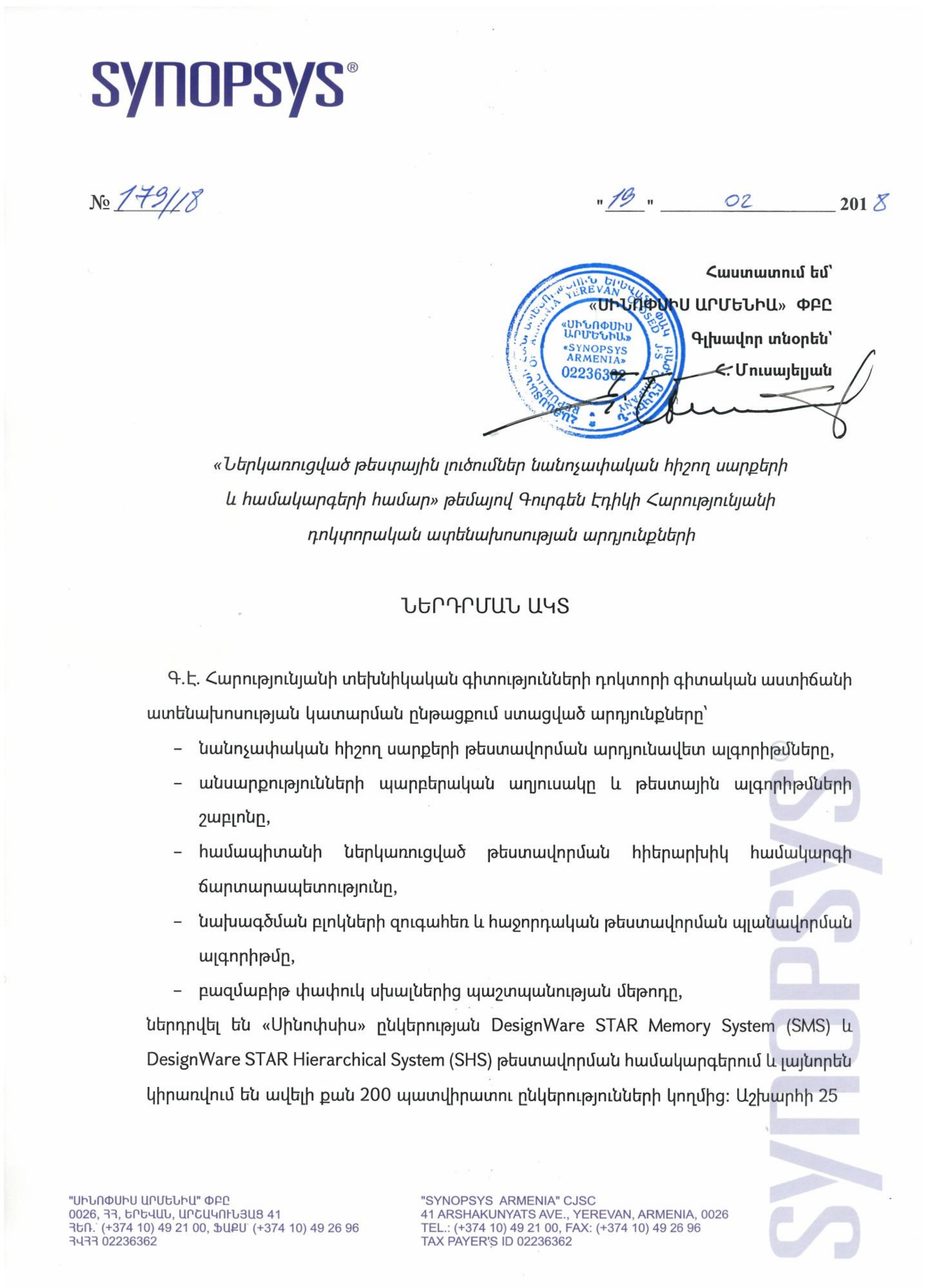
[174] G. Boschi, E. Spano', R. Mariani, G. Harutyunyan, Y. Zorian, "Comprehensive Online Functional Safety Solutions for Embedded Memories", IEEE International Workshop on Automotive Reliability & Test, 2017, Paper S02-01, pp. 1-3.

[175] H. Shaheen, G. Boschi, G. Harutyunyan, Y. Zorian, "Advanced ECC Solution for Automotive SoCs", IEEE International Symposium on On-Line Testing and Robust System Design, 2017, pp. 71-73.

[176] G. Harutyunyan, S. Shoukourian, G. Tshagharyan, "Security Issues in Test and Repair Infrastructure for Systems-On-Chip", International Conference in Information and Communication Technologies, 2017, pp. 114-122.

[177] G. Tshagharyan, G. Harutyunyan, S. Shoukourian, Y. Zorian, "Securing Test Infrastructure of System-on-Chips", IEEE East-West Design and Test Symposium, 2016, pp. 29-32.

APPENDIX A. IMPLEMENTATION ACT



SYNOPSYS®

№ 179/18 "19" 02 2018

Հաստատում եմ՝
«ՍԻՆՈՓՍԻՍ ԱՐՄԵՆԻԱ» ՓԲԸ
Գլխավոր տնօրեն՝
Է. Մուսայելյան

«ՍԻՆՈՓՍԻՍ ԱՐՄԵՆԻԱ» «SYNOPSYS ARMENIA» 02236362 YEREVAN

[illegible]

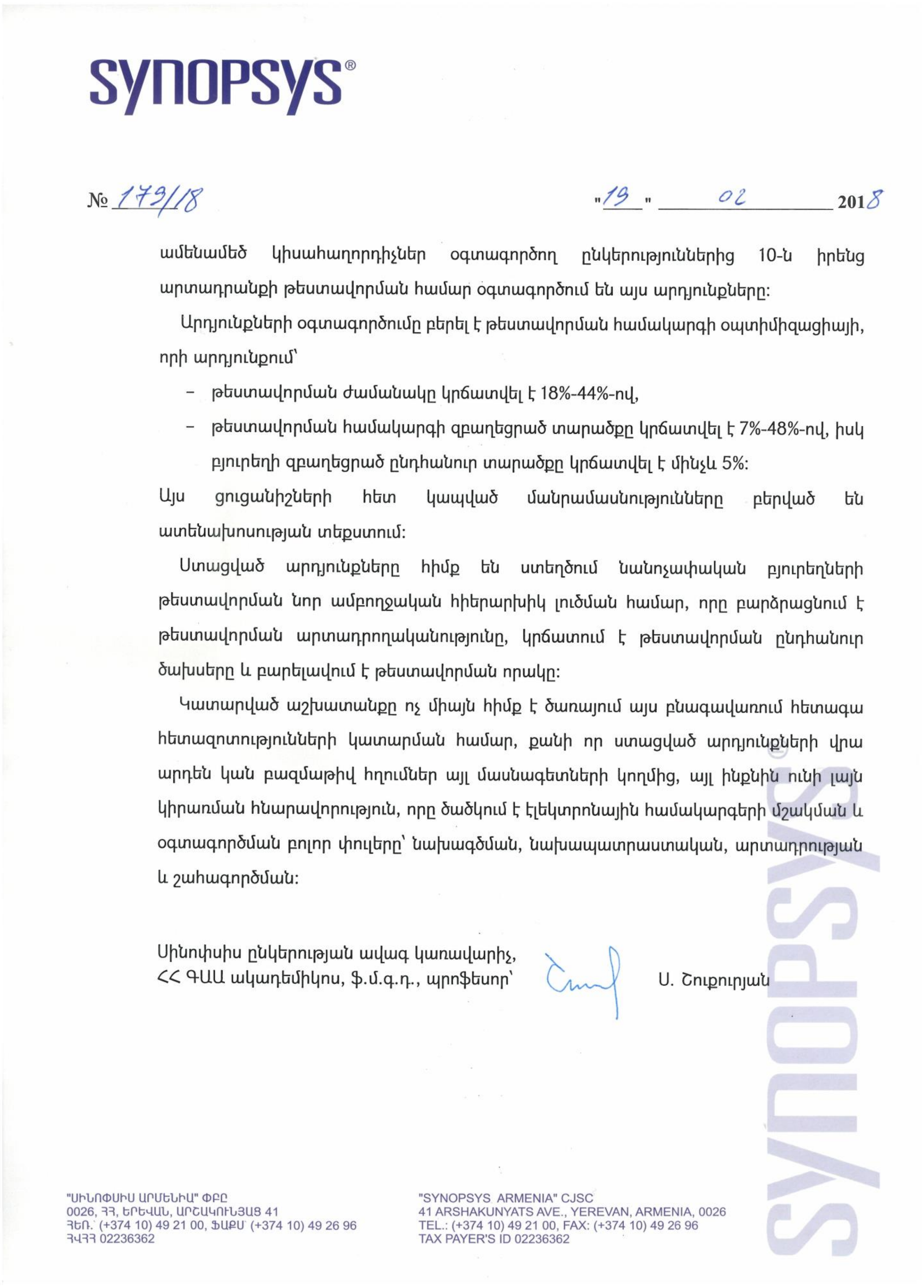
«Ներկառուցված թեստային լուծումներ նանոչափական հիշող սարքերի և համակարգերի համար» թեմայով Գուրգեն Էդիկի Հարությունյանի դոկտորական ատենախոսության արդյունքների

ՆԵՐԴՐՄԱՆ ԱԿՏ

Գ.Է. Հարությունյանի տեխնիկական գիտությունների դոկտորի գիտական աստիճանի ատենախոսության կատարման ընթացքում ստացված արդյունքները՝

- նանոչափական հիշող սարքերի թեստավորման արդյունավետ ալգորիթմները,
- անսարքությունների պարբերական աղյուսակը և թեստային ալգորիթմների շաբլոնը,
- համապիտանի ներկառուցված թեստավորման հիերարխիկ համակարգի ճարտարապետությունը,
- նախագծման բլոկների զուգահեռ և հաջորդական թեստավորման պլանավորման ալգորիթմը,
- բազմաբիթ փափուկ սխալներից պաշտպանության մեթոդը,

ներդրվել են «Սինոփսիս» ընկերության DesignWare STAR Memory System (SMS) և DesignWare STAR Hierarchical System (SHS) թեստավորման համակարգերում և լայնորեն կիրառվում են ավելի քան 200 պատվիրատու ընկերությունների կողմից: Աշխարհի 25



SYNOPSYS®

№ 179/18 "19" 02 2018

ամենամեծ կիսահաղորդիչներ օգտագործող ընկերություններից 10-ն իրենց արտադրանքի թեստավորման համար օգտագործում են այս արդյունքները:

Արդյունքների օգտագործումը բերել է թեստավորման համակարգի օպտիմիզացիայի, որի արդյունքում՝

- թեստավորման ժամանակը կրճատվել է 18%-44%-ով,
- թեստավորման համակարգի զբաղեցրած տարածքը կրճատվել է 7%-48%-ով, իսկ բյուրեղի զբաղեցրած ընդհանուր տարածքը կրճատվել է մինչև 5%:

Այս ցուցանիշների հետ կապված մանրամասնությունները բերված են ատենախոսության տեքստում:

Ստացված արդյունքները հիմք են ստեղծում նանոչափական բյուրեղների թեստավորման նոր ամբողջական հիերարխիկ լուծման համար, որը բարձրացնում է թեստավորման արտադրողականությունը, կրճատում է թեստավորման ընդհանուր ծախսերը և բարելավում է թեստավորման որակը:

Կատարված աշխատանքը ոչ միայն հիմք է ծառայում այս բնագավառում հետագա հետազոտությունների կատարման համար, քանի որ ստացված արդյունքների վրա արդեն կան բազմաթիվ հղումներ այլ մասնագետների կողմից, այլ ինքնին ունի լայն կիրառման հնարավորություն, որը ծածկում է էլեկտրոնային համակարգերի մշակման և օգտագործման բոլոր փուլերը՝ նախագծման, նախապատրաստական, արտադրության և շահագործման:

Սինոփսիս ընկերության ավագ կառավարիչ,
ՀՀ ԳԱԱ ակադեմիկոս, ֆ.մ.գ.դ., պրոֆեսոր՝ [illegible] Ս. Շուքուրյան

APPENDIX B. ADDITIONAL TEST ALGORITHMS

Minimal March-Based Fault Location Algorithm with Partial Diagnosis for Traditional Faults (Flow 1). A March test algorithm of complexity 11N, N is the number of memory words, is defined for fault detection and partial diagnosis. 3N March-based test algorithms are used for location of the aggressor words of inter-word coupling faults. Then another March-based test algorithm of complexity $4\lceil \log B \rceil$ is applied to locate the aggressor bit in the aggressor word. The proposed minimal test algorithm has higher fault location ability and lower time complexity than other known test algorithms developed for fault location in SRAMs. For memories with non-interleaved word structure, an additional phase with a test algorithm of complexity $(6 + 4\lceil \log B \rceil)N$ is to be used to detect and locate intra-word faults.

The following traditional functional fault models in the memory cell array are considered:

- Class of stuck-at faults SAF(0) and SAF(1), the defective cell contains respectively logic value 0 and 1 permanently;
- Class of transition faults TF(↑, 0) and TF(↓, 1), the defective cell fails to undergo respectively 0 to 1 (rising) and 1 to 0 (falling) transition;
- Class of state coupling faults CFst(a, b), the coupled (victim) cell is forced to logic value b, $b \in \{0, 1\}$, only if the coupling (aggressor) cell contains logic value a, $a \in \{0, 1\}$;
- Class of idempotent coupling faults CFid(t, b) – the victim cell is forced to logic value b, $b \in \{0, 1\}$, if the aggressor cell undergoes a transition t, $t \in \{\uparrow, \downarrow\}$;
- Class of inversion coupling faults CFin(t, ↔), the contents of the victim cell is inverted (↔) if the aggressor cell undergoes a transition t, $t \in \{\uparrow, \downarrow\}$;
- Class of write disturb coupling faults CFds(wa, t), the contents of the victim cell undergoes a transition t, $t \in \{\uparrow, \downarrow\}$ if an operation to write a logic value $a \in \{0, 1\}$ is applied to the aggressor cell.

In addition:

- By CF(Ap, As, Vs) it is denoted the inter-word coupling fault between an aggressor and a victim cells with a relative addressing position Ap $\in$\{H, L\}, i.e. the aggressor has a higher (H) or lower (L) address than the victim;
- As $\in$\{0, 1, w0, w1, $\uparrow$, $\downarrow$\} denotes the state of the aggressor cell or operation on the aggressor cell sensitizing the fault;
- Vs $\in$\{0, 1, $\uparrow$, $\downarrow$, $\leftrightarrow$\} denotes the state of the victim cell.
- D is a data background pattern;
- $\overline{D}$ is the inverse of D;
- $\Uparrow$ (respectively, $\Downarrow$) is the ascending (descending) address order;
- $\Leftrightarrow$ is an arbitrary address order;
- $\Uparrow_m^n$ is the ascending address order starting at address m and ending at address n;
- v is the address of the victim cell;
- w_a (respectively, w_v) denotes the write operation performed only to the aggressor (respectively, victim) cell;
- r_v the read operation performed only to the victim.

For Phase 1 the following March test algorithm of complexity 11N is proposed for fault detection and partial diagnosis:

March MC (11N): $\Uparrow$(W0); $\Uparrow$(R0, W1); $\Uparrow$(R1, W0); $\Leftrightarrow$(R0); $\Downarrow$(R0, W1); $\Downarrow$(R1, W0); $\Leftrightarrow$(R0).

Table B1 lists the March syndromes of March MC.

For location of the aggressor word, a March-based test algorithm of complexity 3N is applied in Phase 2. Table B2 lists all March-based test algorithms for all groups of faults corresponding to March syndromes determined after Phase 1. For location of the aggressor bit in the aggressor word, a minimal March-based test algorithm of complexity 4 $\lceil \log B \rceil$ is applied in Phase 3.

Table B1. March syndromes of March MC (Flow 1)

Faults	R0	R1	R2	R3	R4	R5
CFst(L, 0, 0)	0	1	0	0	0	0
CFst(H, 0, 0)	0	0	0	0	1	0
CFst(L, 0, 1)	0	0	1	1	0	1
CFst(H, 0, 1)	1	0	1	0	0	1
CFst(L, 1, 0)	0	0	0	0	1	0
CFst(H, 1, 0)	0	1	0	0	0	0
CFst(L, 1, 1)	1	0	0	0	0	0
CFst(H, 1, 1)	0	0	0	1	0	0
CFid(L, ↑, 1)	1	0	0	0	0	0
CFid(L, ↑, 0)	0	0	0	0	1	0
CFid(L, ↓, 1)	0	0	0	0	0	1
CFid(L, ↓, 0)	0	1	0	0	0	0
CFid(H, ↑, 1)	0	0	0	1	0	0
CFid(H, ↑, 0)	0	1	0	0	0	0
CFid(H, ↓, 1)	0/1	0	1	1	0	0
CFid(H, ↓, 0)	0	0	0	0	1	0
CFin(L, ↑, ↔)	1	0	0	0	1	0
CFin(L, ↓, ↔)	0	1	0	0	0	1
CFin(H, ↑, ↔)	0	1	0	1	0	0
CFin(H,↓, ↔)	0/1	0	1	1	1	0
CFds(L, w0, ↑)	0	0	0	0	0	1
CFds(H, w0, ↑)	1	0	1	1	0	0
CFds(L, w0, ↓)	0	1	0	0	0	0
CFds(H, w0, ↓)	0	0	0	0	1	0
CFds(L, w1, ↑)	1	0	0	0	0	0
CFds(H, w1, ↑)	0	0	0	1	0	0
CFds(L, w1, ↓)	0	0	0	0	1	0
CFds(H, w1, ↓)	0	1	0	0	0	0
SAF(0)	0	1	0	0	1	0
SAF(1)	1	0	1	1	0	1
TF(↑, 0)	0	1	0	0	1	0
TF(↓, 1)	0/1	0	1	1	0	1

Table B2. March-based test algorithms for Phase 2 (Flow 1)

March syndromes	Faults	March-based test algorithms	Length
100010	CFin(L, ↑, ↔)	$\Uparrow_0^{v-1}$(w0); w_v0; $\Uparrow_0^{v-1}$(w1, r_v0)	3N
010001	CFin(L, ↓, ↔)	$\Uparrow_0^{v-1}$(w1); w_v1; $\Uparrow_0^{v-1}$(w0, r_v1)	3N
010100	CFin(H, ↑, ↔)	w_v0; $\Uparrow_{v+1}^{N-1}$ (w0); $\Uparrow_{v+1}^{N-1}$ (w1, r_v0)	3N
001110	CFin(H, ↓, ↔)	w_v1; $\Uparrow_{v+1}^{N-1}$ (w1); $\Uparrow_{v+1}^{N-1}$ (w0, r_v1)	3N
101110	CFin(H, ↓, ↔)	w_v1; $\Uparrow_{v+1}^{N-1}$ (w1); $\Uparrow_{v+1}^{N-1}$ (w0, r_v1)	3N
010000	CFst(L, 0, 0), CFid(L, ↓, 0), CFds(L, w0, ↓), CFst(H, 1, 0), CFid(H, ↑, 0), CFds(H, w1, ↓)	$\Uparrow_0^{v-1}$(w1); w_v1; $\Uparrow_{v+1}^{N-1}$(w0); $\Uparrow_0^{v-1}$(w0, r_v1); $\Uparrow_{v+1}^{N-1}$(w1, r_v1)	3N
000010	CFst(L, 1, 0), CFid(L, ↑, 0), CFds(L, w1, ↓), CFst(H, 0, 0), CFid(H, ↓, 0), CFds(H, w0, ↓)	$\Uparrow_0^{v-1}$(w0); w_v1; $\Uparrow_{v+1}^{N-1}$ (w1); $\Uparrow_0^{v-1}$(w1, r_v1); $\Uparrow_{v+1}^{N-1}$ (w0, r_v1)	3N
000100	CFst(H, 1, 1), CFid(H, ↑, 1), CFds(H, w1, ↑)	w_v0; $\Uparrow_{v+1}^{N-1}$(w0); $\Uparrow_{v+1}^{N-1}$ (w1, r_v0)	3N
001100	CFid(H, ↓, 1)	w_v0; $\Uparrow_{v+1}^{N-1}$ (w1); $\Uparrow_{v+1}^{N-1}$ (w0, r_v0)	3N
100000	CFst(L, 1, 1), CFid(L, ↑, 1), CFds(L, w1, ↑)	$\Uparrow_0^{v-1}$(w0); w_v0; $\Uparrow_0^{v-1}$(w1, r_v0)	3N
001101	CFst(L, 0, 1), TF(↓, 1)	$\Uparrow_0^{v-1}$(w1); w_v1; w_v0; r_v0; $\Uparrow_0^{v-1}$(w0, r_v0)	3N
000001	CFid(L, ↓, 1), CFds(L, w0, ↑)	$\Uparrow_0^{v-1}$(w1); w_v0; $\Uparrow_0^{v-1}$(w0, r_v0)	3N
101001	CFst(H, 0, 1)	$\Uparrow_{v+1}^{N-1}$ (w1); w_v0; $\Uparrow_{v+1}^{N-1}$(w0, r_v0)	3N
101100	CFid(H, ↓, 1), CFds(H, w0, ↑)	w_v0; $\Uparrow_{v+1}^{N-1}$(w1); $\Uparrow_{v+1}^{N-1}$(w0, r_v0)	3N

Let V be the fault-free state of the victim cell and $\alpha \in \{0,1\}$ be the non-sensitizing value of the aggressor cell of a coupling fault determined after Phase 1.

Phase 3 Algorithm: (w_vV, w_aD_1, w_aD_2, r_vV, ..., w_vV, w_aD_{k-1}, w_aD_k, r_vV), $k = 2\lceil \log B \rceil$

The background patterns used in Phase 3 algorithm are listed below, $\alpha \in \{0,1\}$:

D_1	α	α	α	α	α	α	α	α	α	α	α	α	α	α	α	α
D_2	α	$\overline{\alpha}$	α	$\overline{\alpha}$	α	$\overline{\alpha}$	α	$\overline{\alpha}$	α	$\overline{\alpha}$	α	$\overline{\alpha}$	α	$\overline{\alpha}$	α	$\overline{\alpha}$
D_3	α	α	α	α	α	α	α	α	α	α	α	α	α	α	α	α
D_4	α	α	$\overline{\alpha}$	$\overline{\alpha}$	α	α	$\overline{\alpha}$	$\overline{\alpha}$	α	α	$\overline{\alpha}$	$\overline{\alpha}$	α	α	$\overline{\alpha}$	$\overline{\alpha}$
	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
$D_{2\lceil \log B \rceil -1}$	α	α	α	α	α	α	α	α	α	α	α	α	α	α	α	α
$D_{2\lceil \log B \rceil}$	α	α	α	α	α	α	α	α	$\overline{\alpha}$	$\overline{\alpha}$	$\overline{\alpha}$	$\overline{\alpha}$	$\overline{\alpha}$	$\overline{\alpha}$	$\overline{\alpha}$	$\overline{\alpha}$

Minimal March-Based Fault Location Algorithm with Partial Diagnosis for All Static Faults (Flow 2). Phase 1 of the proposed minimal March-based fault location algorithm comprises the following minimal March test algorithm of complexity 18N for detection of all unlinked static faults and partial diagnosis:

March MSS (18N): ⇑(W0); ⇑(R0, W1, W1, R1); ⇑(R1, W0, W0, R0); ⇓(R0, W1, W1, R1); ⇓(R1, W0, W0, R0); ⇓(R0).

After Phase 1 the set of all FPs are partitioned into groups where all FPs in a group have the same March syndrome. After this phase, it is difficult to locate the aggressor bit of a coupling fault since the faults included in the same group require different March-based test algorithms of different lengths for the location process. To make this process easier and develop a test algorithm that in overall has minimum length, for Phase 2 it is proposed to use March-based test algorithms of complexity N+O(1) that partition the groups of FPs into subgroups where all FPs in the same subgroup have the same extended March syndrome. Table B3 shows all necessary March-based test algorithms for Phase 2. Table B4 lists all subgroups of FPs obtained after Phases 1 and 2.

Table B3. Test algorithms for Phase 2 (Flow 2)

$E_1(x)=W_vx;\ R_vx;\ R_vx;\ W_vx;\ \Uparrow_0^{v-1}(W1);\ \Uparrow_{v+1}^{N-1}(W1);\ R_vx;\ R_vx$
$E_2=W_v1;\ W_v0;\ R_v0;\ W_v0;\ R_v0;\ \Uparrow_0^{v-1}(W1);\ \Uparrow_{v+1}^{N-1}(W1);\ W_v0;\ R_v0;\ W_v1;\ W_v0;\ R_v0;\ W_v0;\ R_v0$
$E_3(x)=W_v(\sim x);\ W_vx;\ R_vx;\ W_vx;\ R_vx;\ \Uparrow_0^{v-1}(W1);\ \Uparrow_{v+1}^{N-1}(W1);\ W_v(\sim x);\ W_vx;\ R_vx;\ W_vx;\ R_vx$
$E_4=W_v1;\ W_v0;\ R_v0;\ W_v0;\ R_v0;\ \Uparrow_0^{v-1}(W1);\ \Uparrow_{v+1}^{N-1}(W1);\ R_v0;\ W_v0;\ R_v0$
$E_5(x)=\ W_vx;\ R_vx;\ R_vx$
$E_6=W_v1;\ R_v1$

Table B4. March syndromes and fault groups w.r.t Phases 1 and 2 (Flow 2)

Phase 1		Phase 2		
March syndromes	Groups	Test algorithms	March syndromes	Groups
001000000	CFds <0W0; 1/↓/->$_{a<v}$	$E_1(1)$	0000	CFds < 0W0; 1 / ↓ / - >$_{a<v}$
	CFds <1W0; 1/↓/->$_{a<v}$			CFds < 1W0; 1 / ↓ / - >$_{a<v}$
	CFds <1W1; 1/↓/->$_{a>v}$			CFds < 1W1; 1 / ↓ / - >$_{a>v}$
	CFds <0W1; 1/↓/->$_{a>v}$		0011	CFds < 0W1; 1 / ↓ / - >$_{a>v}$
	CFdrd <0; R1/↓/1>$_{a>v}$		0100	CFdrd < 0; R1 / ↓ / 1 >$_{a>v}$
	CFdrd <1; R1/↓/1>$_{a>v}$		0001	CFdrd < 1; R1 / ↓ / 1 >$_{a<v}$
000000001	CFds <0W0; 0/↑/->$_{a<v}$	$E_1(0)$	0000	CFds < 0W0; 0 / ↑ / - >$_{a<v}$
	CFds <1W0; 0/↑/->$_{a<v}$			CFds < 1W0; 0 / ↑ / - >$_{a<v}$
	CFdrd <0; R0/↑/0>$_{a>v}$		0100	CFdrd < 0; R0 / ↑ / 0 >$_{a>v}$
	CFdrd <1; R0/↑/0>$_{a<v}$		0001	CFdrd < 1; R0 / ↑ / 0 >$_{a<v}$
000000100	CFds <1W1; 1/↓/->$_{a<v}$	$E_1(1)$	0000	CFds < 1W1; 1 / ↓ / - >$_{a<v}$
	CFds <0W0; 1/↓/->$_{a>v}$			CFds < 0W0; 1 / ↓ / - >$_{a>v}$
	CFds <1W0; 1/↓/->$_{a>v}$			CFds < 1W0; 1 / ↓ / - >$_{a>v}$
	CFds <0W1; 1/↓/->$_{a<v}$		0011	CFds < 0W1; 1 / ↓ / - >$_{a<v}$
	CFdrd <0; R1/↓/1>$_{a<v}$		0100	CFdrd < 0; R1 / ↓ / 1 >$_{a<v}$
	CFdrd <1; R1/↓/1>$_{a>v}$		0001	CFdrd < 1; R1 / ↓ / 1 >$_{a>v}$
000010000	CFds <0W0; 0/↑/->$_{a>v}$	$E_1(0)$	0000	CFds < 0W0; 0 / ↑ / - >$_{a>v}$
	CFds <1W1; 0/↑/->$_{a>v}$			CFds < 1W1; 0 / ↑ / - >$_{a>v}$
	CFds <1W0; 0/↑/->$_{a>v}$			CFds < 1W0; 0 / ↑ / - >$_{a>v}$
	CFds <R1; 0/↑/->$_{a>v}$			CFds < R1; 0 / ↑ / - >$_{a>v}$
	CFds <0W1; 0/↑/->$_{a>v}$		0011	CFds < 0W1; 0 / ↑ / - >$_{a>v}$
	CFdrd <0; R0/↑/0>$_{a<v}$		0100	CFdrd < 0; R0 / ↑ / 0 >$_{a<v}$
	CFdrd <1; R0/↑/0>$_{a>v}$		0001	CFdrd < 1; R0 / ↑ / 0 >$_{a>v}$
100110000	CFtr <0; 1W0/1/->$_{a<v}$	$E_3(0)$	1100	CFtr < 0; 1W0 / 1 / - >$_{a<v}$
	CFwd <0; 0W0/↑/->$_{a<v}$		0100	CFwd < 0; 0W0 / ↑ / - >$_{a<v}$
	CFtr <1; 1W0/1/->$_{a>v}$		0011	CFtr < 1; 1W0 / 1 / - >$_{a>v}$
	CFwd <1; 0W0/↑/->$_{a>v}$		0001	CFwd < 1; 0W0 / ↑ / - >$_{a>v}$
000001100	CFtr <0; 0W1/0/->$_{a<v}$	$E_3(1)$	1100	CFtr < 0; 0W1 / 0 / - >$_{a<v}$
	CFwd <0; 1W1/↓/->$_{a<v}$		0100	CFwd < 0; 1W1 / ↓ / - >$_{a<v}$
	CFtr <1; 0W1/0/->$_{a>v}$		0011	CFtr < 1; 0W1 / 0 / - >$_{a>v}$
	CFwd <1; 1W1/↓/->$_{a>v}$		0001	CFwd < 1; 1W1 / ↓ / - >$_{a>v}$
000010001	CFds <R0; 0/↑/->$_{a>v}$	$E_5(0)$	00	CFds < R0; 0 / ↑ / - >$_{a>v}$
	DRDF <R0/↑/0>		01	DRDF < R0 / ↑ / 0 >

Continuation of Table B4

001001000	CFst <0; 1/0/->$_{a<v}$	E_6	1	CFst < 0; 1 / 0 / - >$_{a<v}$
	CFst <1; 1/0/->$_{a>v}$		0	CFst < 1; 1 / 0 / - >$_{a>v}$
010000100	CFst <0; 1/0/->$_{a>v}$	E_6	1	CFst < 0; 1 / 0 / - >$_{a>v}$
	CFst <1; 1/0/->$_{a<v}$		0	CFst < 1; 1 / 0 / - >$_{a<v}$
000110000	CFtr <0; 1W0/1/->$_{a<v}$	E_2	11000	CFtr < 0; 1W0 / 1 / - >$_{a<v}$
	CFwd <0; 0W0/↑/->$_{a<v}$		01000	CFwd < 0; 0W0 / ↑ / - >$_{a<v}$
	CFtr <1; 1W0/1/->$_{a>v}$		00011	CFtr < 1; 1W0 / 1 / - >$_{a>v}$
	CFwd <1; 0W0/↑/->$_{a>v}$		00001	CFwd < 1; 0W0 / ↑ / - >$_{a>v}$
	CFst <1; 0/1/->$_{a>v}$ CFrd <1; R0/↑/1>$_{a>v}$		00111	CFst < 1; 0 / 1 / - >$_{a>v}$ CFrd < 1; R0 / ↑ / 1 >$_{a>v}$
001000100	CFds <R0; 1/↓/->$_{a<v}$ CFds <R1; 1/↓/->$_{a<v}$ CFds <R0; 1/↓/->$_{a>v}$ CFds <R1; 1/↓/->$_{a>v}$	E_5(1)	00	CFds < R0; 1 / ↓ / - >$_{a<v}$ CFds < R1; 1 / ↓ / - >$_{a<v}$ CFds < R0; 1 / ↓ / - >$_{a>v}$ CFds < R1; 1 / ↓ / - >$_{a>v}$
	DRDF <R1/↓/1>		01	DRDF < R1 / ↓ / 1 >
011000000	CFtr <0; 0W1/0/->$_{a>v}$	E_3(1)	1100	CFtr < 0; 0W1 / 0 / - >$_{a>v}$
	CFwd <0; 1W1/↓/->$_{a>v}$		0100	CFwd < 0; 1W1 / ↓ / - >$_{a>v}$
	CFtr <1; 0W1/0/->$_{a<v}$		0011	CFtr < 1; 0W1 / 0 / - >$_{a<v}$
	CFwd <1; 1W1/↓/->$_{a<v}$		0001	CFwd < 1; 1W1 / ↓ / - >$_{a<v}$
000000011	CFtr <0; 1W0/1/->$_{a>v}$	E_3(0)	1100	CFtr < 0; 1W0 / 1 / - >$_{a>v}$
	CFwd <0; 0W0/↑/->$_{a>v}$		0100	CFwd < 0; 0W0 / ↑ / - >$_{a>v}$
	CFtr <1; 1W0/1/->$_{a<v}$		0011	CFtr < 1; 1W0 / 1 / - >$_{a<v}$
	CFwd <1; 0W0/↑/->$_{a<v}$		0001	CFwd < 1; 0W0 / ↑ / - >$_{a<v}$
100000011	CFtr <0; 1W0/1/->$_{a>v}$ CFrd <0; R0/↑/1>$_{a>v}$	E_4	111	CFtr < 0; 1W0 / 1 / - >$_{a>v}$ CFrd < 0; R0 / ↑ / 1 >$_{a>v}$
	CFwd <0; 0W0/↑/->$_{a>v}$		011	CFwd < 0; 0W0 / ↑ / - >$_{a>v}$
	CFst <0; 0/1/->$_{a>v}$		110	CFst < 0; 0 / 1 / - >$_{a>v}$
	CFrd <1; R0/↑/1>$_{a<v}$		001	CFrd < 1; R0 / ↑ / 1 >$_{a<v}$

Table B5 lists all March-based test algorithms for location of the aggressor bit in Phase 3. Table B6 shows which of the test algorithms listed in Table B5 should be applied for a subgroup in Phase 3 to locate the aggressor bit for all FPs from the subgroup with the same March syndrome determined after Phases 1 and 2.

Note that for some March syndromes from Phases 1 and 2 there is "-" in the corresponding 3rd column meaning there is no need to apply a test algorithm in Phase 3 since a single-cell fault has been found which does not need additional operations for its location. Also there are some faults not needing to apply Phase 2 and for them there is "-" in the 2nd column in Table B6. For these faults the 3rd column indicates the test algorithms to be applied in Phase 3.

Table B5. Test algorithms for Phase 3 (Flow 2)

Test algorithms	Length
$A_1(L, H, x) = W_v x$; $\Uparrow_L^H$ (W0, $R_v x$, $R_v x$)	3N
$A_2(L, H, x) = \Uparrow_L^H$ (W0); $W_v x$; $\Uparrow_L^H$ (W1, $R_v x$, $R_v x$)	4N
$A_3(L, H, x, y) = W_v x$; $\Uparrow_L^H$ (Wy, $R_v x$)	2N
$A_4(L, H, x, y) = \Uparrow_L^H$ (W(~y)); $W_v x$; $\Uparrow_L^H$ (Wy, $R_v x$)	3N
$A_5(L, H, x) = W_v(\sim x)$; $\Uparrow_L^H$ (W0, $W_v x$, $R_v x$, $W_v(\sim x)$)	4N
$A_6(L, H, x) = \Uparrow_L^H$ (W0); $W_v(\sim x)$; $\Uparrow_L^H$ (W1, $W_v x$, $R_v x$, $W_v(\sim x)$)	5N
$A_7(L, H, x) = W_v x$; $\Uparrow_L^H$ (W0, $W_v x$ $R_v x$)	3N
$A_8(L, H, x) = \Uparrow_L^H$ (W0); $W_v x$; $\Uparrow_L^H$ (W1, $W_v x$ $R_v x$)	4N
$A_9 = \Uparrow_0^{v-1}$(W1); $W_v 1$; $\Uparrow_0^{v-1}$(W0, $R_v 1$); $\Uparrow_{v+1}^{N-1}$(W1, $R_v 1$)	3N
$A_{10} = \Uparrow_{v+1}^{N-1}$(W1); $W_v 1$; $\Uparrow_0^{v-1}$(W1, $R_v 1$); $\Uparrow_{v+1}^{N-1}$(W0, $R_v 1$)	3N
$A_{11} = W_v 0$; $\Uparrow_{v+1}^{N-1}$(R0, $R_v 0$)	2N
$A_{12}(L, H, x) = \Uparrow_L^H$ (W0); $W_v x$; $\Uparrow_L^H$ (W1, $R_v x$)	3N
$A_{13}(L, H, x) = \Uparrow_L^H$ (W0); $W_v x$; $\Uparrow_L^H$ (W0, W0, $R_v x$)	4N
$A_{14} = W_v 0$; $\Uparrow_0^{v-1}$(W1, W1, $R_v 0$)	3N
$A_{15} = W_v 0$; $\Uparrow_0^{v-1}$(R0, W1, R1, $R_v 0$)	4N
$A_{16} = W_v 1$; $\Uparrow_0^{v-1}$(W0, W0, $R_v 1$); $\Uparrow_{v+1}^{N-1}$(W1, $R_v 1$)	3N
$A_{17} = W_v 1$; $\Uparrow_0^{v-1}$(W1, $R_v 1$); $\Uparrow_{v+1}^{N-1}$(W0, W0, $R_v 1$)	3N
$A_{18} = W_v 1$; $\Uparrow_0^{v-1}$(R0, W1, R1, $R_v 1$); $\Uparrow_{v+1}^{N-1}$(R0, W1, R1, $R_v 1$)	4N
$A_{19} = W_v 0$; $\Uparrow_{v+1}^{N-1}$(R1, W1, W0, W0, $R_v 0$)	5N

Table B6. Aggressor bit location (Flow 2)

March syndromes of Phase 1	March syndromes of Phase 2	Test algorithms
001000000	0000	A_{16}
001000000	0011	A_{12}(v+1, N-1, 1)
001000000	0100	A_{1}(v+1, N-1, 1)
001000000	0001	A_{2}(0, v-1, 1)
000010000	0000	A_{19}
000010000	0011	A_{12}(v+1, N-1, 0)
000010000	0100	A_{1}(0, v-1, 0)
000010000	0001	A_{2}(v+1, N-1, 0)
000000100	0000	A_{17}
000000100	0011	A_{12}(0, v-1, 1)
000000100	0100	A_{1}(0, v-1, 1)
000000100	0001	A_{2}(v+1, N-1, 1)
000000001	0000	A_{13}(0, v-1, 0)
000000001	0100	A_{1}(v+1, N-1, 0)
000000001	0001	A_{2}(0, v-1, 0)
000010001	00	A_{11}
000010001	01	-
001001000	1	A_{4}(0, v-1, 1, 0)
001001000	0	A_{3}(v+1, N-1, 1, 1)
010000100	1	A_{4}(v+1, N-1, 1, 0)
010000100	0	A_{3}(v+1, N-1, 1, 1)
000110000	11000	A_{5}(0, v-1, 0)
000110000	01000	A_{7}(0, v-1, 0)
000110000	00011	A_{6}(v+1, N-1, 0)
000110000	00001	A_{7}(v+1, N-1, 0)
000110000	00111	A_{3}(v+1, N-1, 0, 1)
100110000	1100	A_{5}(0, v-1, 0)
100110000	0100	A_{7}(0, v-1, 0)
100110000	0011	A_{6}(v+1, N-1, 0)
100110000	0001	A_{7}(v+1, N-1, 0)
000001100	1100	A_{5}(0, v-1, 1)

Continuation of Table B6		
000001100	0100	$A_7(0, v-1, 1)$
000001100	0011	$A_6(v+1, N-1, 1)$
000001100	0001	$A_8(v+1, N-1, 1)$
001000100	00	A_{18}
001000100	01	-
011000000	1100	$A_5(v+1, N-1, 1)$
011000000	0100	$A_7(v+1, N-1, 1)$
011000000	0011	$A_6(0, v-1, 1)$
011000000	0001	$A_8(0, v-1, 1)$
000000011	1100	$A_5(v+1, N-1, 0)$
000000011	0100	$A_7(v+1, N-1, 0)$
000000011	0011	$A_6(0, v-1, 0)$
000000011	0001	$A_8(0, v-1, 0)$
100000011	111	$A_5(v+1, N-1, 0)$
100000011	011	$A_7(v+1, N-1, 0)$
100000011	110	$A_3(v+1, N-1, 0, 0)$
100000011	001	$A_4(0, v-1, 0, 1)$
100000000	-	A_{14}
100010000	-	$A_{13}(v+1, N-1, 0)$
100000100	-	$A_{13}(v+1, N-1, 1)$
100000001	-	A_{15}
000110001	-	$A_4(0, v-1, 0, 0)$
001001100	-	A_9
011000100	-	A_{10}
100000010	-	$A_3(0, v-1, 0, 1)$

A March-Based Fault Location Algorithm with Partial and Full Diagnosis for All Static Unlinked Faults (Flow 3).

This test algorithm is for those cases, when the preliminary March test algorithm for detection of the victim cell is supposed to be unknown and there is a need to locate the aggress cell. The other case is when the fault detection March test is known but it stopped (the performance of the test algorithm is done until detection of the first fault) when a fault is detected and therefore the corresponding March syndrome cannot be obtained completely. That is why a test algorithm is proposed which takes as an input only the address of the failed (victim) bit and without a March syndrome, or by means of an incomplete March syndrome it can locate and diagnose the static faults.

In Table B7, March LD1, a March-based test algorithm of complexity 2N+O(1) is proposed for Phase 1 which finds out whether the fault is a single-cell or a two-cell fault. Note that in Tables B7 and B9 by "-" is denoted that the mentioned sequence of operations is applied only to the victim cell and it has no address direction. If the obtained March syndrome is in the list shown in Table B8 then it is a single-cell fault. Besides location, the test algorithm gives a group of faults that can be present in the bit. This information will be used in the fault diagnosis step.

Table B7. Phase 1: March LD1 (2N) (Flow 3)

Address direction	Operations
$\Uparrow$	W0
-	W_v1; W_v1; R_v1; R_v1; W_v0; W_v0; R_v0; R_v0
$\Uparrow$	W1
-	W_v0; W_v0; R_v0; R_v0; W_v1; W_v1; R_v1; R_v1

Table B8. March syndromes of fault groups w.r.t Phase 1 (Flow 3)

March syndrome	Faults
11000011	SF < 0 / 1 / - >, TF < 0W1 / 0 / - >, WDF < 1W1 / 0 / - >, RDF < R1 / 0 / 0 >, IRF < R1 / 1 / 0 >
01000001	DRDF < R1 / 0 / 1 >
00111100	SF < 1 / 0 / - >, TF < 1W0 / 1 / - >, WDF < 0W0 / 1 / - >, RDF < R0 / 1 / 1 >, IRF < R0 / 0 / 1 >
00010100	DRDF < R0 / 1 / 0 >

If the March syndrome obtained from March LD1 is not in the list shown in Table B8 then an additional March-based test algorithm March LD2 of complexity 41N + O(1) must be used in Phase 2 (see Table B9).

Table B9. Phase 2: March LD2 (41N) (Flow 3)

Address direction	Operations
$\Uparrow$	W0
-	W_v0; R_v0
$\Uparrow_{0}^{v-1}$	W1, W1, R1, R_v0
$\Uparrow_{v+1}^{N-1}$	W1, W1, R1, R_v0
$\Uparrow_{0}^{v-1}$	W0, W0, R0, R_v0
$\Uparrow_{v+1}^{N-1}$	W0, W0, R0, R_v0
-	W_v1; R_v1
$\Uparrow_{0}^{v-1}$	W1, W1, R1, R_v1
$\Uparrow_{v+1}^{N-1}$	W1, W1, R1, R_v1
$\Uparrow_{0}^{v-1}$	W0, W0, R0, R_v1
$\Uparrow_{v+1}^{N-1}$	W0, W0, R0, R_v1
-	W_v0; R_v0
$\Uparrow_{0}^{v-1}$	W1, W_v1, W_v1, R_v1, R_v1, W_v0
$\Uparrow_{v+1}^{N-1}$	W1, W_v1, W_v1, R_v1, R_v1, W_v0
$\Uparrow_{0}^{v-1}$	W0, W_v1, W_v1, R_v1, R_v1, W_v0
$\Uparrow_{v+1}^{N-1}$	W0, W_v1, W_v1, R_v1, R_v1, W_v0
-	W_v1; R_v1
$\Uparrow_{0}^{v-1}$	W1, W_v0, W_v0, R_v0, R_v0, W_v1
$\Uparrow_{v+1}^{N-1}$	W1, W_v0, W_v0, R_v0, R_v0, W_v1
$\Uparrow_{0}^{v-1}$	W0, W_v0, W_v0, R_v0, R_v0, W_v1
$\Uparrow_{v+1}^{N-1}$	W0, W_v0, W_v0, R_v0, R_v0, W_v1

An Efficient 2-Phase March Algorithm for Full Diagnosis of All Static Unlinked Faults (Flow 4). A new, two-phase March test algorithm is proposed for diagnosis of all static faults. In the first phase, the known March MSS (18N): ⇑(W0); ⇑(R0, W1, W1, R1); ⇑(R1, W0, W0, R0); ⇓(R0, W1, W1, R1); ⇓(R1, W0, W0, R0); ⇓(R0) test algorithm is used as a minimal March test algorithm for detection of all static faults. Table B10 shows the test algorithms of Phase 2 for each subgroup. The overall length of the two-phase test algorithm for full diagnosis of all static faults is 30 which is shorter from March FD by 5N.

Table B10. Test algorithms of Phase 2 (Flow 4)

March Syndrome	Fault subgroup after Phase 1	Test Algorithms for Phase 2	Length
000110001	< 0; 0 / 1 / - >$_{a<v}$, < 0; R0 / 1 / 1 >$_{a<v}$	⇑(W0); ⇑(R0); ⇑(R0, W1)	4N
010000100	< 1; 1 / 0 / - >$_{a<v}$, < 0; 1 / 0 / - >$_{a>v}$	⇑(W0); ⇑(W1, R1); ⇑(R1)	4N
(0/1)00110000	< 1; 0 / 1 / - >$_{a>v}$, < 0; 1W0 / 1 / - >$_{a<v}$, < 1; 1W0 / 1 / - >$_{a>v}$, < 0; 0W0 / 1 / - >$_{a<v}$, < 1; 0W0 / 1 / - >$_{a>v}$, < 1; R0 / 1 / 1 >$_{a>v}$	⇑(W1); ⇑(W0, R0); ⇑(R0, W0, R0); ⇓(W0, R0, W1).	9N
100000000	< 0W1; 0 / 1 / - >$_{a<v}$, < 1W1; 0 / 1 / - >$_{a<v}$	⇑(W0); ⇑(R0, W1); ⇓(W1, W0); ⇑(R0)	6N
(0/1)00000100	< 1W1; 1 / 0 / - >$_{a<v}$, < 0W0; 1 / 0 / - >$_{a>v}$, < 1W0; 1 / 0 / - >$_{a>v}$, < 0W1; 1 / 0 / - >$_{a<v}$, < 0; R1 / 0 / 1 >$_{a<v}$, < 1; R1 / 0 / 1 >$_{a>v}$	⇓(W1); ⇓(W1); ⇓(R1); ⇓(R1, W0, W0);⇓(W1, R1, R1); ⇓(R1, W0)	11N
001000100	< R0; 1 / 0 / - >$_{a<v}$, < R0; 1 / 0 / - >$_{a>v}$, < R1; 1 / 0 / - >$_{a<v}$, < R1; 1 / 0 / - >$_{a>v}$, < R1 / 0 / 1 >	⇑(W1); ⇑(R1); ⇑(R1, W0, R0, W1); ⇑(R1, R1)	8N
011001100	< 1W1 / 0 / - >, < 1 / 0 / - >	⇑(W0); ⇑(W1, R1, W1, R1)	5N
000010001	< R0; 0 / 1 / - >$_{a>v}$, < R0 / 1 / 0 >	⇑(W0); ⇓(R0, R0)	3N
011000000	< 0; 0W1 / 0 / - >$_{a>v}$, < 1; 0W1 / 0 / - >$_{a<v}$, < 0; 1W1 / 0 / - >$_{a>v}$, < 1; 1W1 / 0 / - >$_{a<v}$	⇑(W0); ⇑(W1, W1, R1, W1, R1); ⇑(W1, R1)	8N
001000000	< 1W1; 1 / 0 / - >$_{a>v}$, < 0W0; 1 / 0 / - >$_{a<v}$, < 1W0; 1 / 0 / - >$_{a<v}$, < 0W1; 1 / 0 / - >$_{a>v}$, < 0; R1 / 0 / 1 >$_{a>v}$, < 1; R1 / 0 / 1 >$_{a<v}$	⇑(W1); ⇑(W1); ⇑(R1); ⇑(R1, W0, W0); ⇑(W1, R1, R1); ⇑(R1, W0)	11N
000000001	< 1; R0 / 1 / 0 >$_{a<v}$, < 0; R0 / 1 / 0 >$_{a>v}$ < 0W0; 0 / 1 / - >$_{a<v}$, < 1W0; 0 / 1 / - >$_{a<v}$	⇑(W0); ⇓(W0); ⇑(R0, R0, W1); ⇓(W0); ⇑(R0, R0)	8N
001001000	< 0; 1 / 0 / - >$_{a<v}$, < 1; 1 / 0 / - >$_{a>v}$	⇑(W0); ⇓(W1, R1); ⇑(R1)	4N
100000001	< R1; 0 / 1 / - >$_{a<v}$, < R0; 0 / 1 / - >$_{a<v}$	⇑(W0); ⇑(R0); ⇑(R0, W1, R1)	5N
(0/1)00110011	< 0W0 / 1 / - >, < 0 / 1 / - >	⇑(W1); ⇑(W0, R0, W0, R0)	5N
001001100	< 0; R1 / 0 / 0 >$_{a<v}$, < 1; R1 / 0 / 0 >$_{a>v}$	⇑(W1); ⇑(R1); ⇑(R1, W0)	4N
011000100	< 1; R1 / 0 / 0 >$_{a<v}$, < 0; R1 / 0 / 0 >$_{a>v}$	⇑(W1); ⇑(R1); ⇓(R1, W0)	4N
000001100	< 0; 1W1 / 0 / - >$_{a<v}$, < 1; 1W1 / 0 / - >$_{a>v}$, < 0; 0W1 / 0 / - >$_{a<v}$, < 1; 0W1 / 0 / - >$_{a>v}$	⇑(W0); ⇓(W1, W1, R1, W1, R1); ⇓(W1, R1)	8N
(0/1)00010000	< 0; R0 / 1 / 0 >$_{a<v}$, < 1; R0 / 1 / 0 >$_{a>v}$, < 0W1; 0 / 1 / - >$_{a>v}$, < 1W0; 0 / 1 / - >$_{a>v}$, < 0W0; 0 / 1 / - >$_{a>v}$, < 1W1; 0 / 1 / - >$_{a>v}$, < R1; 0 / 1 / - >$_{a>v}$	⇑(W0); ⇓(R0, W0); ⇓(R0, R0, W1, W1); ⇑(W0); ⇑(R0); ⇓(R0, W1, R1).	12N
(0/1)00000011	< 0; 1W0 / 1 / - >$_{a>v}$, < 1; 1W0 / 1 / - >$_{a<v}$, < 0; 0W0 / 1 / - >$_{a>v}$, < 1; 0W0 / 1 / - >$_{a<v}$, < 0; R0 / 1 / 1 >$_{a>v}$, < 1; R0 / 1 / 1 >$_{a<v}$, < 0; 0 / 1 / - >$_{a>v}$	⇑(W1); ⇑(W0); ⇑(W0, R0, W1); ⇓(W0, R0); ⇑(W0, R0); ⇓(R0, W1).	11N

An Efficient March-Based Three-Phase Fault Location and Full Diagnosis Algorithm for Realistic Two-Operation Dynamic Faults (Flow 5). In Phase 1, a March test of complexity 75N is proposed for fault detection and partial diagnosis. The structure of test algorithm March DD is described in Table B11.

In Phase 2, March-based test algorithms of complexity 4N to 13N (see Table B12) are proposed for fault location. In Phase 3, March-based test algorithms consisting of 3 to 29 operations are proposed for full diagnosis (see Table B13).

Table B11. Test algorithm March DD (75N) (Flow 5)

Address direction	Operations
⇑	W0
⇑	R0, W1, W1, R1, W1, W1, R1, W0, W0, R0, W0, W0, R0, W0, W1, W0, W1
⇑	R1
⇑	R1, W0, W0, R0, W0, W0, R0, W1, W1, R1, W1, W1, R1, W1, W0, W1, W0, R0
⇑	R0
⇓	R0, W1, R1, W1, R1, R1, R1, W0, R0, W0, R0, R0, R0, W0, W1, W0, W1
⇑	R1
⇓	R1, W0, R0, W0, R0, R0, R0, W1, R1, W1, R1, R1, R1, W1, W0, W1, W0, R0
⇑	R0

Table B12. Test algorithms for Phase 2 (Flow 5)

Test algorithm description	Length
$L_1(x, y, z) = \Uparrow_0^{v-1}(Wy); W_v x; \Uparrow_0^{v-1}(W(\sim y), W_v z, R_v z); \Uparrow_{v+1}^{N-1}(W(\sim y)); W_v x; \Uparrow_{v+1}^{N-1}(Wy, W_v z, R_v z)$	4N
$L_2(x, y) = \Uparrow_0^{v-1}(Wy); W_v x; \Uparrow_0^{v-1}(W(\sim y), W_v 0, R_v 0, R_v 0);$ $\Uparrow_{v+1}^{N-1}(W(\sim y)); W_v x; \Uparrow_{v+1}^{N-1}(Wy, W_v 0, R_v 0, R_v 0)$	5N
$L_3(y) = \Uparrow_0^{v-1}(Wy); W_v 0; \Uparrow_0^{v-1}(W(\sim y), W_v 1, R_v 1, W_v 1, R_v 1, R_v 1); \Uparrow_{v+1}^{N-1}(W(\sim y)); W_v 0;$ $\Uparrow_{v+1}^{N-1}(Wy, W_v 1, R_v 1, W_v 1, R_v 1, R_v 1)$	7N
$L_4(y) = \Uparrow_0^{v-1}(Wy); W_v 0; \Uparrow_0^{v-1}(W(\sim y), R_v 0, W_v 1, R_v 1); \Uparrow_{v+1}^{N-1}(W(\sim y)); W_v 0; \Uparrow_{v+1}^{N-1}(Wy, R_v 0, W_v 1, R_v 1)$	5N
$L_5(x, y) = \Uparrow_0^{v-1}(Wy); W_v x; \Uparrow_0^{v-1}(W(\sim y), R_v x, R_v x, R_v x); \Uparrow_{v+1}^{N-1}(W(\sim y)); W_v x; \Uparrow_{v+1}^{N-1}(Wy, R_v x, R_v x, R_v x)$	5N
$L_6(x, y) = \Uparrow_0^{v-1}(Wy); W_v x; \Uparrow_0^{v-1}(W(\sim y), W_v x, W_v(\sim x), W_v x, W_v(\sim x), R_v(\sim x));$ $\Uparrow_{v+1}^{N-1}(W(\sim y)); W_v x; \Uparrow_{v+1}^{N-1}(Wy, W_v x, W_v(\sim x), W_v x, W_v(\sim x), R_v(\sim x))$	7N
$L_7(x, y, z) = \Uparrow_0^{v-1}(Wy); W_v x; \Uparrow_0^{v-1}(W(\sim y), R_v x, W_v z, R_v z);$ $\Uparrow_{v+1}^{N-1}(W(\sim y)); W_v x; \Uparrow_{v+1}^{N-1}(Wy, R_v x, W_v z, R_v z)$	5N
$L_8(x, y, z) = \Uparrow_0^{v-1}(Wy); W_v x; \Uparrow_0^{v-1}(W(\sim y), W_v z, W_v z, R_v z);$ $\Uparrow_{v+1}^{N-1}(W(\sim y)); W_v x; \Uparrow_{v+1}^{N-1}(Wy, W_v z, W_v z, R_v z)$	5N
$L_9(x, L, H) = W_v x; \Uparrow_L^H(W0, W1, W0, W1, R1, W1, R1, W0, R0, W0, R0, W1, R_v x)$	13N
$L_{10}(x, L, H) = W_v x; \Uparrow_L^H(W1, W1, W0, R0, R0, W1, R1, R1, R_v x)$	9N
$L_{11}(x, L, H) = W_v x; \Uparrow_L^H(W0, R0, R0, W0, W1, R1, R1, R_v x)$	8N
$L_{12}(x, L, H) = W_v x; \Uparrow_L^H(W0, W0, W0, W1, W1, W1, W0, W0, R_v x)$	9N

Table B13. Test algorithms for Phase 3 (Flow 5)

Description of test algorithms	Length
$D_1 = W_v 1; W_v 0; R_v 0; R_v 0; W_{(v+1)\mathrm{mod}N} 0; R_v 0$	6
$D_2 = W_v 1; W_v 1; R_v 1; R_v 1; W_{(v+1)\mathrm{mod}N} 0; R_v 1; W_v 0; W_v 1; R_v 1; R_v 1; W_{(v+1)\mathrm{mod}N} 0; R_v 1$	12
$D_3(x) = W_v x; W_v x; W_v(\sim x); R_v(\sim x)$	4
$D_4(x) = W_v x; W_v(\sim x); R_v(\sim x)$	3
$D_5(x) = W_a x; W_v 1; W_v 0; R_v 0, R_v 0; W_a x; R_v 0;$	7
$D_6(x) = W_a x; W_v 0; W_v 1; R_v 1, R_v 1; W_a x; R_v 1; W_v 1; R_v 1, R_v 1; W_a x; R_v 1$	12
$D_7(x) = W_a x; W_v 0; W_v 1; R_v 1$	4
$D_8(x, y) = W_a x; W_v y; W_v y; W_v(\sim y); R_v(\sim y)$	5
$D_9(x) = W_v x; W_a 0; W_a 0; R_a 0; R_v x; W_a 1; W_a 0; R_v x; R_a 0; W_a 0; R_v x; R_a 0; W_a 1; R_v x; W_a 1; R_a 1; R_v x; W_a 0;$ $W_a 1; R_v x; R_a 1; W_a 1; R_v x; R_a 1; W_a 0; R_v x$	26
$D_{10}(x) = W_v x; W_a 1; W_a 0; R_a 0; R_v x; W_a 0; R_a 0; R_v x; W_a 1; W_a 0; R_v x; R_a 0; W_a 0; R_v x; R_a 0; W_a 1; R_v x;$ $W_a 1; R_a 1; R_v x; W_a 0; W_a 1; R_v x; R_a 1; W_a 1; R_v x; R_a 1; W_a 0; R_v x$	29
$D_{11}(x) = W_v x; W_a 1; W_a 1; W_a 0; R_v x; R_a 0; R_a 0; R_v x; W_a 1; R_a 1; R_v x; R_a 1; R_a 1; R_v x; W_a 0; R_a 0; R_v x$	17
$D_{12}(x) = W_v x; W_a 0; R_a 0; R_a 0; R_v x; W_a 0; W_a 1; R_v x; R_a 1; R_a 1; R_v x; W_a 0; W_a 1; R_a 1; R_v x$	15
$D_{13}(x, y) = W_v x; W_a y; W_a y; W_a y; R_v x; W_a y; W_a(\sim y); R_v x; W_a(\sim y); W_a(\sim y); R_v x; W_a y; W_a y; R_v x; W_a(\sim y);$ $W_a(\sim y); R_v x$	17

APPENDIX C. TEST PATTERNS FOR 7NM PCIE4 IP CORE

Table C1 lists the main tests patterns used in typical 7nm PCIe4 IP core (https://www.synopsys.com/dw/ipdir.php?ds=dwc_pcie4_phy).

Table C1. 7nm PCIe4 test patterns

Test pattern name	Test pattern description
REGISTER	This test reads the default value of all readable registers in the PHY, then writes the inverse pattern and verifies the result. Finally, the test writes the default value back to the registers and verifies the result.
SIMPLE_REGISTERS	Reduced version of REGISTER test. This test reads and writes several registers in the PHY to verify register access interfaces.
INIT_PACKAGE_TESTS	This test does general initialization of the PHY. It is intended that this script be run first before some of other tests to run. It will set the clock frequency, turn on the pattern matchers/generators, enable rx, enable tx and invert the channel data where needed.
BERT	This is functional test to verify that the device is operational. The device is set-up in external loopback.
INTERNAL_LOOPBACK	This is functional test to verify that the device is operational. The device is set-up in internal loopback.
LFPS_LOS	This test sends Low Frequency Periodic Signaling (LFPS) out the transmitter and measures the Loss-of-Signal (LOS) level of the receiver.
TX_VCM	This test enables TX ATB, connects ATB to ADC and checks RTUNE status register.
TX_DC_LEVELS	This test checks the DC TX voltage swing for the un-boosted and boosted settings using a DC test. It uses the on-board ADC to measure these quantities.
TX_RX_DETECT	This test is done issuing a tx rx detect with the rx termination on and off and verify that the transmitter detects and does not detect the receiver.

The CPL description of REGISTER test is adduced below.

REGISTER test (only a small part of the test is presented here since the test itself is too big)

```
## READ SUP_DIG_MPLLA_BW_LOW_OVRD_IN
## READ_CAPTURE MPLLA_BW_LOW=0000000001000011
## READ SUP_DIG_MPLLA_BW_HIGH_OVRD_IN
## READ_CAPTURE MPLLA_BW_HIGH=0000000001000011
## READ SUP_DIG_MPLLB_BW_LOW_OVRD_IN
## READ_CAPTURE MPLLB_BW_LOW=0000000001000011
## READ SUP_DIG_MPLLB_BW_HIGH_OVRD_IN
## READ_CAPTURE MPLLB_BW_HIGH=0000000001000011
## READ SUP_DIG_SUP_OVRD_IN_0
## READ_CAPTURE RTUNE_REQ=0 RTUNE_OVRD_EN=0 RES_REQ_IN=0 RES_ACK_IN=1 RES_OVRD_EN=0
RESERVED_15_5=00000000000
## READ SUP_DIG_SUP_OVRD_IN_1
## READ_CAPTURE TXUP_TERM_OFFSET=000000000 TXUP_TERM_OFFSET_OVRD_EN=0
RX_TERM_OFFSET=00000 RX_TERM_OFFSET_OVRD_EN=0
## WRITE SUP_DIG_MPLLA_BW_LOW_OVRD_IN ffbc
## READ SUP_DIG_MPLLA_BW_LOW_OVRD_IN
## READ_CAPTURE MPLLA_BW_LOW=1111111110111100
## WRITE SUP_DIG_MPLLA_BW_HIGH_OVRD_IN ffbc
## READ SUP_DIG_MPLLA_BW_HIGH_OVRD_IN
## READ_CAPTURE MPLLA_BW_HIGH=1111111110111100
## WRITE SUP_DIG_MPLLB_BW_LOW_OVRD_IN ffbc
## READ SUP_DIG_MPLLB_BW_LOW_OVRD_IN
## READ_CAPTURE MPLLB_BW_LOW=1111111110111100
## WRITE SUP_DIG_MPLLB_BW_HIGH_OVRD_IN ffbc
## READ SUP_DIG_MPLLB_BW_HIGH_OVRD_IN
## READ_CAPTURE MPLLB_BW_HIGH=1111111110111100
## WRITE SUP_DIG_SUP_OVRD_IN_0 0017
## READ SUP_DIG_SUP_OVRD_IN_0
## READ_CAPTURE RTUNE_REQ=1 RTUNE_OVRD_EN=1 RES_REQ_IN=1 RES_ACK_IN=0 RES_OVRD_EN=1
RESERVED_15_5=00000000000
```

```
## WRITE SUP_DIG_SUP_OVRD_IN_1 ffff
## READ SUP_DIG_SUP_OVRD_IN_1
## READ_CAPTURE TXUP_TERM_OFFSET=111111111 TXUP_TERM_OFFSET_OVRD_EN=1
RX_TERM_OFFSET=11111 RX_TERM_OFFSET_OVRD_EN=1
## WRITE SUP_DIG_MPLLA_BW_LOW_OVRD_IN 0043
## READ SUP_DIG_MPLLA_BW_LOW_OVRD_IN
## READ_CAPTURE MPLLA_BW_LOW=0000000001000011
## WRITE SUP_DIG_MPLLA_BW_HIGH_OVRD_IN 0043
## READ SUP_DIG_MPLLA_BW_HIGH_OVRD_IN
## READ_CAPTURE MPLLA_BW_HIGH=0000000001000011
## WRITE SUP_DIG_MPLLB_BW_LOW_OVRD_IN 0043
## READ SUP_DIG_MPLLB_BW_LOW_OVRD_IN
## READ_CAPTURE MPLLB_BW_LOW=0000000001000011
## WRITE SUP_DIG_MPLLB_BW_HIGH_OVRD_IN 0043
## READ SUP_DIG_MPLLB_BW_HIGH_OVRD_IN
## READ_CAPTURE MPLLB_BW_HIGH=0000000001000011
## WRITE SUP_DIG_SUP_OVRD_IN_0 0008
## READ SUP_DIG_SUP_OVRD_IN_0
## READ_CAPTURE RTUNE_REQ=0 RTUNE_OVRD_EN=0 RES_REQ_IN=0 RES_ACK_IN=1 RES_OVRD_EN=0
RESERVED_15_5=00000000000
## WRITE SUP_DIG_SUP_OVRD_IN_1 0000
## READ SUP_DIG_SUP_OVRD_IN_1
## READ_CAPTURE TXUP_TERM_OFFSET=000000000 TXUP_TERM_OFFSET_OVRD_EN=0
RX_TERM_OFFSET=00000 RX_TERM_OFFSET_OVRD_EN=0
```