Patrick Girard
Shawn Blanton
Li-C. Wang *Editors*

Machine Learning Support for Fault Diagnosis of System-on-Chip

Springer

Machine Learning Support for Fault Diagnosis of System-on-Chip

Patrick Girard • Shawn Blanton • Li-C. Wang
Editors

Machine Learning Support for Fault Diagnosis of System-on-Chip

Springer

Editors
Patrick Girard
LIRMM - CNRS
Montpellier, France

Shawn Blanton
Carnegie Mellon University
Pittsburgh, PA, USA

Li-C. Wang
University of California, Santa Barbara
Santa Barbara, CA, USA

ISBN 978-3-031-19638-6 ISBN 978-3-031-19639-3 (eBook)
https://doi.org/10.1007/978-3-031-19639-3

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

General Introduction

Today's electronic systems are composed of complex Systems on a Chip (SoCs) made of heterogeneous blocks that comprise memories, digital circuits, analog and mixed-signal circuits, etc. To fit a critical application standard requirement, SoCs pass through a comprehensive test flow (functional, structural, parametric, etc.) at the end of the manufacturing process. The goal is to achieve near-zero Defective Parts per Million (DPPM) so as to ensure the quality level required by the standard. Unfortunately, imperfections in the manufacturing process may introduce systematic defects, especially when the first devices are produced while the process is not yet mature. Identification of these systematic defects and correction of the related manufacturing process call for efficient diagnosis techniques. Hence, the goal of diagnosis is to extract information from test data in order to identify the nature and the causes of defects that have occurred in a SoC. Note that additional data can also be produced and used to improve the diagnosis process, such as distinguishing test patterns used only during the diagnosis phase.

Failure isolation is critical to identify root-cause of manufacturing issues. The scaling of manufacturing process technology and shrinking of device sizes and interconnects in nano-scale geometries make defect isolation more and more challenging. With the introduction of new transistor devices, lithography, and fabrication technologies, the demand for faster and precise defect isolation will continue to grow. Along with manufacturing process complexity, design complexity has also increased as functionality and computing needs have grown manyfold. Increasing design complexity makes the defect isolation in billon transistor chip ever more challenging. Therefore, fast diagnosis and root-cause identification of failures are essential to maintain high production yields and keep the Moore's Law in meeting the time to market demand.

Chips that pass manufacturing test are then shipped to the customer and integrated in their host system. However, despite the quality level of the manufacturing test procedures, SoCs may fail in the field, either due to the occurrence of a defect not covered during the manufacturing test phase or due to early-life failures or failures caused by various wear-out mechanisms. Early-life failures are caused by latent defects that are not exposed during manufacturing tests, but that are degraded due to

electrical and thermal stress during in-field use, and lead to a failure in functionality. Wear-out or aging manifesting as progressive performance degradation, is induced by various mechanisms such as negative-bias temperature instability or hot-carrier injection. To avoid catastrophic consequences, many systems include in-the-field test techniques, which allow the detection of such problems. After detection, the defective SoC must be diagnosed to identify any possible systematic degradation patterns and avoid their re-occurrence in next-generation products. In this context, the first step during the failure identification process is to reproduce the failure mechanism with any original test and test conditions. Next, a diagnosis program made of several routines is used to identify, step by step, the failing part of the defective SoC and, finally, the suspected defects. Each routine corresponds to the application of a diagnosis algorithm at a given hierarchy level (system, core and cell levels).

Irrespective of the phase during which a SoC fails (after manufacturing, or after online test if the defective SoC is sent back to the manufacturer), defective SoCs undergo logic diagnosis to locate the fault, and then physical failure analysis (PFA) to characterize the fault. Diagnosis is a software-based method that analyzes the applied tests, the tester responses, and the netlist (possibly with layout information) to produce a list of candidates that represent the possible locations and types of defects (or faults) within the defective circuit. The quality of a diagnosis outcome is usually evaluated owing to two metrics: accuracy and resolution. A diagnosis is accurate if the actual defect is included in the reported list of candidates. Resolution refers to the total number of candidates reported for each actual defect. An accurate diagnosis with perfect resolution (i.e., one) is the ideal case. Diagnosis is usually followed by physical failure analysis (PFA), a time-consuming process for exposing the defect physically in order to characterize the failure mechanism. Due to the high cost and destructive nature of PFA, diagnosis resolution is of critical importance. In practice, it is very uncommon to perform PFA on any defect with more than five candidates. This ensures that the likelihood for uncovering the root-cause of failure is maximized when performing PFA.

Historically, conventional approaches based on cause-effect (i.e., fault simulation) and/or effect-cause (i.e., critical path tracing) analysis were used in industry for defect and fault diagnosis. However, with the fast development and vast application of machine learning (ML) in recent years, ML-based techniques have been shown to be highly valuable for diagnosis. They can be used for volume diagnosis after manufacturing to improve production yield or for diagnosis of customer returns to identify any possible systematic degradation patterns. The main advantage of ML-based diagnosis techniques is that they can deal with huge amount of insightful test data that could not be efficiently exploited otherwise in a reasonable amount of time.

A wide range of solutions based on supervised, unsupervised and reinforcement learning have been proposed in the last 10 years. They can be used for failure isolation in logic or analog parts of SoCs, board-level fault diagnosis, or even wafer-level failure cluster identification. A plethora of ML algorithms have been experimented and implemented in new diagnosis tools used today in industry. Benefits can be measured in terms of diagnosis accuracy, resolution and duration.

This book identifies the key challenges in fault diagnosis of system-on-chip and presents the solutions and corresponding results that have emerged from leading-edge research in this domain. In a comprehensive form, it provides necessary background to the reader and proposes a compendium of solutions existing in this field. The book explains and applies optimized techniques from the machine learning domain to solve the fault diagnosis problem in the realm of electronic system design and manufacturing. It demonstrates techniques based on industrial data and feedback from actual PFA analysis. It also discusses practical problems, including test sequence quality, diagnosis resolution, accuracy, and time cost.

The first chapter gives some prerequisites on fault diagnosis. Basic terms, such as defect, fault, and failure, are first enumerated. Then, basic concepts of test and fault simulation are described. After that, the basics of volume diagnosis for yield improvement and fault diagnosis of customer returns are given. Finally, basic information on yield and failure analysis is provided.

Chapter 2 is dedicated to the presentation of conventional methods for fault diagnosis. The chapter focuses on the automated tools and methods along with design features at the architectural, logic, circuit, and layout level that are needed to facilitate silicon debug and defect diagnosis of integrated circuits. These design features are generally referred to as design for debug and diagnosis (DFD). The chapter describes how these DFD features along with automated tools and methods are used effectively in a debug or diagnosis environment for applications ranging from design validation, low yield analysis, and all the way to field failure analysis. The chapter can serve as a steppingstone to understand further how conventional methods for fault diagnosis can be improved by using machine learning–based techniques.

The third chapter provides details of machine learning techniques proposed so far to solve various VLSI testing problems. It focuses on explaining scope of machine learning in VLSI testing. First, it gives a high-level overview of machine learning. After that, it describes the types of machine learning algorithms. Then, it explains some popular and commonly used machine learning algorithms. After that, this chapter discusses some recent machine learning based solutions proposed to solve VLSI testing problems. Finally, it discusses the strength and limitations of these methods.

Chapter 4 is dedicated to machine learning support for logic diagnosis and defect classification. After a preliminary discussion about attempts to distinguish maleficent defects from benign variations, the chapter presents machine learning techniques developed so far for distinguishing variations from reliability threats due to defects. Then, machine learning techniques for identifying different defect types during diagnosis are discussed. A neural network-based fault classifier is presented that can distinguish different fault models at gate level. Finally, the chapter concentrates on distinguishing between transient errors covered by hardening or masking techniques from intermittent faults. A solution based on Bayesian networks is presented for classifying intermittent, transient, and permanent faults.

The fifth chapter is dedicated to machine learning in logic circuit diagnosis. It is organized into three main sections that describe the use of ML for pre-diagnosis,

during-diagnosis, and post-diagnosis, so as to characterize when and how a given methodology enhances the classic outcomes of diagnosis that include localization, failure behavior identification, and root cause of failure. The first section is dedicated to pre-diagnosis, which is concerned with any activities that are performed before diagnosis is deployed. Examples of pre-diagnosis activities include classic work such as diagnostic ATPG, and DFT for increasing testability. In the second section, the use of ML in during-diagnosis activities is described, which generally involves learning while diagnosis executes. For example, a *k*-nearest neighbor model can be created, evolved, and used during on-chip diagnosis to improve diagnosis outcomes. In the third and last section, post-diagnosis is discussed, which includes all activities that occur after diagnosis execution. These approaches usually involve volume diagnosis (i.e., using the outcome results of many diagnoses) to improve diagnostic resolution.

Chapter 6 gives an overview of the various machine learning approaches and techniques proposed to support cell-aware generation, test, and diagnosis. The chapter focuses on the generation of the cell-aware models and their usage for diagnosis. After some backgrounds on conventional approaches to generate and diagnose cell-aware defects, the chapter will present a learning-based solution to generate cell-aware models. Then, it presents a ML-based cell-aware diagnosis technique. Effectiveness of existing techniques will be shown through industrial case studies and corresponding diagnosis results in terms of accuracy and resolution. The chapter will conclude with a discussion on the future directions in this field.

Chapter 7 discusses the state of the art on fault diagnosis for analog circuits with a focus on techniques that leverage machine learning. For a chip that has failed either in post-manufacturing testing or in the field of operation, fault diagnosis is launched to identify the root-cause of failure at subblock level and transistor-level. In this context, machine learning can be used to build a smart system that predicts the fault that has occurred from diagnostic measurements extracted on the chip. The chapter discusses the different elements of a diagnosis flow for analog circuits, including fault modeling, fault simulation, diagnostic measurement extraction and selection, and the machine learning algorithms that compose the prediction system. A machine learning–based diagnosis flow experimented on an industrial case study is finally presented.

The eighth chapter discusses machine learning support for board-level functional fault diagnosis. First, the chapter presents an overview of board-level manufacturing tests and conventional fault-diagnosis models. Next, it discusses the motivation of utilizing machine learning techniques and presents the existing machine learning–based diagnosis models. To address the practical issues that arise in real testing data, the chapter next presents a diagnosis system based on online learning algorithms and incremental updates. In the following, it also presents a diagnosis system that utilizes domain-adaption algorithms to transfer the knowledge learned from mature boards to a new board.

Chapter 9 is dedicated to wafer-level failure pattern analytics. In the first section of the chapter, the application is about early detection of yield excursions with the goal to automatically recognize the existence of a systematic failure cluster

when one occurs. In the second section, analytics is formulated as solving a multi-class classification problem and the discussion focuses on training a high-accuracy neural network classifier. In the next section, techniques to learn an individual recognizer for one pattern class are discussed with the goal to learn with very few training samples. Generative adversarial networks (GANs) and tensor computation–based techniques are used together to implement an unsupervised wafer pattern classification and recognition flow. The last section of the chapter introduces language-driven analytics and explains its use in the analytics context. The authors show how a pretrained language model like GPT-3 can play a role in solving the problem.

Finally, a conclusion summarizes the contribution of the book and some perspectives in the field of fault and defect diagnosis of circuits and systems by using machine learning techniques are given.

Contents

Prerequisites on Fault Diagnosis

Harry H. Chen, Xiaoqing Wen, and Wu-Tung Cheng

1 Defect, Fault, Error, Failure

The life of an *integrated circuit* (IC) consists of three phases, namely the *design* phase, the *manufacturing* phase, and the *operational* phase. Ideally, an IC should perform all required functions correctly at its required speed and within its required power limit. In practice, unavoidable imperfection in the three phases may cause adverse impact on an IC. As illustrated in Fig. 1, an *error* may occur at an IC (i.e., the IC outputs a wrong signal) and may eventually cause a *failure* of a system based on the IC (i.e., the system shows a wrong behavior). System failures may cause anything from inconvenience to catastrophe. Although techniques for robust system design exist that can mask the impact of errors to some extent, they are generally expensive because of larger circuit area and higher power consumption. Therefore, it is imperative to minimize the occurrence of IC errors by all means so as to minimize the possibility of system failures.

An IC error can be caused by incorrectness (i.e., *bugs*) introduced in the design phase. However, discussions on how to reduce design bugs are not the focus of this book; instead, this book assumes that IC design is correctly conducted. As illustrated in Fig. 1, for a correctly-designed IC, an IC error is the effect of either an internal cause (*defect*) or an external cause (*radiation*).

H. H. Chen
MediaTek Inc., Hsinchu, Taiwan

X. Wen
Kyushu Institute of Technology, Iizuka, Japan

W.-T. Cheng (✉)
Siemens Digital Industries Software, Plano, TX, USA
e-mail: wu-tung.cheng@siemens.com

P. Girard et al. (eds.), *Machine Learning Support for Fault Diagnosis of System-on-Chip*, https://doi.org/10.1007/978-3-031-19639-3_1

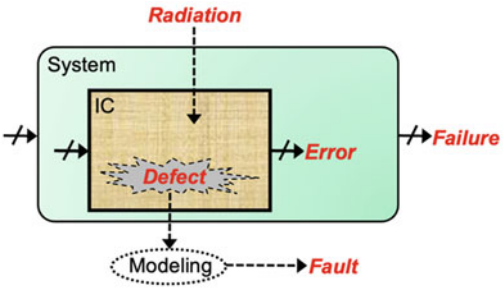

Fig. 1 Various terms

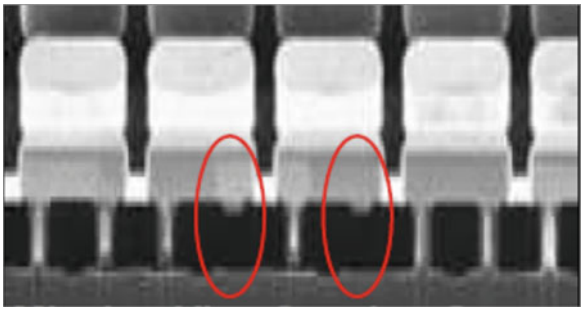

Fig. 2 Example of manufacturing defects (missing vias)

The internal cause of an IC error is a *defect*, which is the unintended difference between the implemented hardware and its intended design [5]. A defect physically and permanently exists since its occurrence. An IC with one or more defects is defective; otherwise, the IC is defect-free.

Defects may occur in the manufacturing phase. An example is shown in Fig. 2. Manufacturing defects are caused by random imperfection (such as particle contamination) or systematic reasons (such as process variations). Therefore, all manufactured ICs need to go through a check process, called *manufacturing test*. *Ideal manufacturing test* passes all defect-free ICs but fails all defective ICs. *Realistic manufacturing test*, however, is imperfect in that it may pass some defective ICs (i.e., *under-test*) and may fail some defect-free ICs (i.e., *over-test*). Passing ICs are shipped to customers for use in electronic systems and failing ICs are discarded. *Yield* is defined as the percentage of passing ICs among all manufactured ICs. Yield loss is *catastrophic* if it is caused by random defects, or *parametric* if it is caused by defects due to process variations. Low yield not only is the nightmare for IC manufacturers but also makes customers worry about IC's quality and reliability. There are two major design-based approaches to improving yield, namely *design for yield enhancement* (DFY) and *design for manufacturability* (DFM). DFY tries to reduce the effect of process variation while DFM tries to avoid

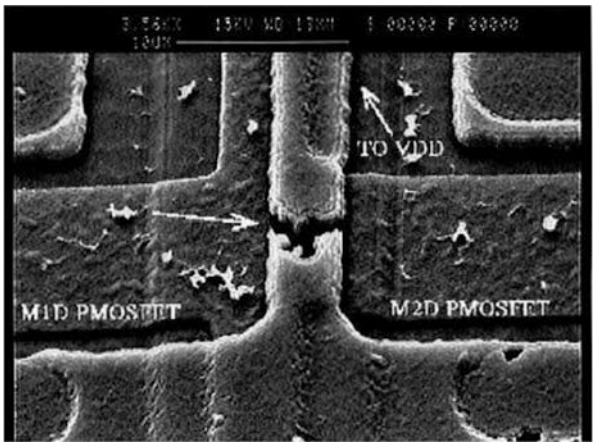

Fig. 3 Example of age defects (stress migration)

random defects. In addition, efforts can be made to reduce over-test, especially for low-power ICs [11].

Defects may also occur in the operational phase, in the forms of *latent defects* and *age defects,* which may cause IC errors and eventually system failures. Latent defects are manufacturing defects that escape manufacturing test. This is especially the case where the quality of manufacturing test is low, causing severe under-test. Age defects are caused by the wear-out of ICs under electrical, thermal, and mechanical stresses in the forms of metal fatigue, hot carriers, electromigration, dielectric breakdown, etc. An example is shown in Fig. 3. In the early operation stage of a system, latent IC defects are the dominant cause of system failures, called *early failures*. In the middle operation stage of a system, latent IC defects become rare and age defects are yet to show up, resulting in only sparse system failures, called *random failures*. In the end operation stage of a system, age defects are the dominant cause of system failures, called *wear-out failures*. Therefore, it is desirable that an IC used in a system can also go through a check process, called *field test*. Defective ICs found by field test can then be replaced before they cause any system failure.

The external cause of an IC error is *radiation*, which comes from the space as protons, electrons, and heavy ions or from IC package materials as alpha particles. Radiation may cause a *single event upset (SEU)*, which is usually a bit flip ($0 \rightarrow 1$ or $1 \rightarrow 0$) at a storage element (a memory cell, a flip-flop, or a latch) in an IC at a certain time [10]. Such a (local) SEU may in turn cause a (global) error at an output of the IC, eventually resulting in a failure of a system based on the IC. An SEU is also called a *soft error* since the affected IC can be free of any permanent defects and its impact is *transient* (i.e., the impact of bit flip at a storage cell disappears after a correct value is loaded). With ever-decreasing feature sizes and power supply voltages, ICs are increasingly becoming susceptible to soft errors.

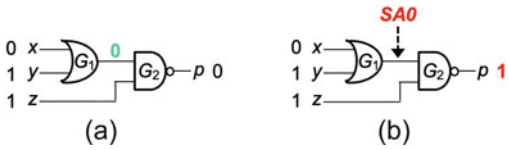

Fig. 4 Stuck-at fault model

However, due to this transient nature, soft errors cannot be tested for as defects that are permanent in nature. The common practice for mitigating soft errors is to adopt radiation-hardening design techniques for storage elements to reduce the chance of radiation to cause soft errors [28]. Such design techniques are referred to as *design for reliability* (DFR).

Generally, defects are the dominant cause of IC errors; thus, defects need to be targeted in manufacturing test as well as field test. However, directly dealing with defects in test-related tasks (e.g., quantifying test quality, generating test data, etc.) is often inefficient and sometimes even computationally infeasible. To solve this problem, one can model the behavior of a defect by a *fault* and only deals with faults explicitly in test-related tasks. Due to the complexity and variety of defects, their behaviors are widely diverse. As a result, no one fault model can be a good representative for all defects. Therefore, it is important to select or build a fault model by taking into consideration of process characteristics, transistor structures, anticipated defects, circuit abstraction levels, etc. Some typical fault models are introduced below.

- *Stuck-At Fault Model*: A gate-level logic circuit is modeled by gates and their interconnections. If the circuit is defect-free, any signal line should be able to take both logic values (0 and 1). The existence of some sort of defects in the circuit may make a signal line to take only one logic value (either 0 or 1). This defect behavior can be modeled by a *stuck-at fault* [9]. If the signal line can only take logic 0 (1), it is said to have a *stuck-at-0 (stuck-at-1) fault* or a *SA0 (SA1) fault*. Fig. 4a shows a fault-free circuit, whose output (p) is 0 for the input value combination $<x = 0, y = 1, z = 1>$. If the output of the OR gate G_1 has a SA0 (i.e., the output of G_1 is fixed at 0 due to the existence of some defect), the circuit output p will be 1 as shown in Fig. 4b. In this case, $<x = 0, y = 1, z = 1>$ is said to be a *test vector* for the SA0 fault.
- *Transistor Stuck Fault Model*: A transistor-level circuit is modeled by transistors and their interconnections. If the circuit is defect-free, any transistor in the circuit should be able to be turned on and off. The existence of some sort of defects in the circuit may make a transistor to be permanently turned on (off). This defect behavior can be modeled by a *transistor stuck-on* (*stuck-off*) *fault*. A transistor stuck-on (stuck-off) fault is also referred to as a *transistor stuck-short* (*stuck-open*) *fault*. As illustrated in Fig. 5, an inverter, whose input and output are x and z, respectively, consists of two transistors, P and N. Thus, this transistor has 4 transistor stuck faults ("*P stuck-on*", "*N stuck-on*", "*P stuck-off*", "*N stuck-off*").

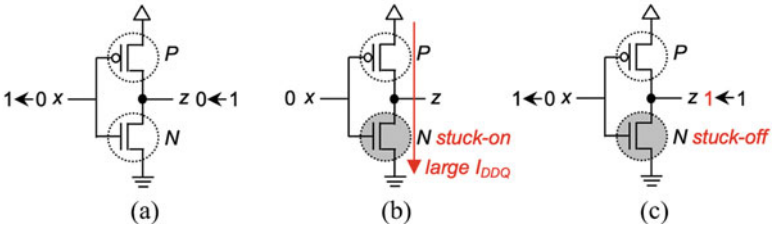

Fig. 5 Transistor stuck fault model

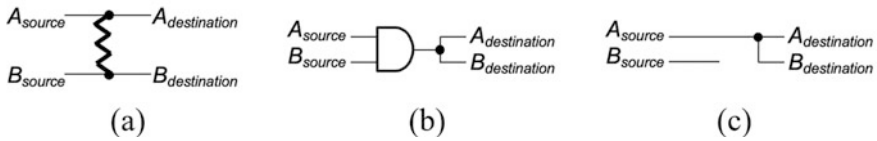

Fig. 6 Bridging fault model

Whether "*N stuck-on*" exists can be determined by applying 0 to x and measuring the quiescent power supply current, I_{DDQ} as shown in Fig. 5b. This is because, if "*N stuck-on*" exists, applying 0 to x will turn P on, resulting in a current path from the power supply to the ground [19]. Whether "*N stuck-off*" exists can be determined by first applying 0 and then 1 to x as shown in Fig. 5c. If the value of z changes from 1 to 0, "*N stuck-off*" does not exist. If the value of z remains at 1, "*N stuck-off*" exists. That is, determining whether "*N stuck-off*" exists needs two test vectors [25].

- *Bridging Fault Model*: Gates in a gate-level circuit and transistors in a transistor-level circuit are all interconnected by wires. Some defects may short two separate wires together, and its effect can be modeled by a *bridging fault* [21] Depending on how to determine the resultant logic value when the two involved wires have opposite logic values, bridging faults can be classified into a few types. For the bipolar technology, the resultant logic value can be assumed to be the AND (OR) result of the logic values of the two wires, resulting in a *wired-AND* (*wired-OR*) *bridging fault*. Fig. 6a shows that wires A and B are shorted together. Its effect can be modeled by a wired-AND bridging fault, whose gate-level representation is shown in Fig. 6b. For the *complementary metal oxide semiconductor* (CMOS) technology, it is more appropriate to model a pair of shorted wires as a *dominant bridging fault*, whose logic value is assumed to be determined by the stronger driver for the two shorted wires. For example, the shorted wires A and B as shown in Fig. 6a can be modeled by a *dominant bridging fault* (A dominates B), whose gate-level representation is shown in Fig. 6c.
- *Delay Fault Model*: Extra delay can be introduced to wires by resistive open and short defects, or to transistors or gates by parameter variations. Such extra delay, either alone or in an accumulative manner, may increase signal propagation delay so much as to break timing requirements, resulting in wrong circuit behaviors.

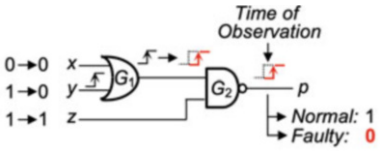

Fig. 7 Transition delay fault model

```
Cell "MUX2" {
    Fault "Z1" {
         test { StaticFault "Z'=1;Condition "D0'=0,"D1'=0,"S'=0;}
         test { StaticFault "Z'=1;Condition "D0'=0,"D1'=1,"S'=0;}
         test { StaticFault "Z'=1;Condition "D0'=0,"D1'=0,"S'=1;}
     }
}
```

Fig. 8 Cell-aware fault model

Extra delay can be modeled by a *delay fault* with several varieties. A *transition delay fault* is used to model an extra-delay-affected transition (*rise* ($0 \rightarrow 1$) or *fall* ($1 \rightarrow 0$)) at a gate [18] and the extra delay is assumed to be large enough to prevent the transition from reaching any output of the circuit at the required time. Thus, each gate is associated with a *slow-to-rise transition delay fault* and a *slow-to-fall transition delay fault*. An example is shown in Fig. 7, where the output of G_1 has a slow-to-rise transition delay fault. Suppose that x remains at 1, z remains at 1, and y is applied with a transition of $1 \rightarrow 0$. In this case, at the time of observation, the output p of G_2 will be 0 instead of the correct value of 1, meaning that the slow-to-rise transition delay fault is detected. A *gate-delay fault* is used to model the extra delay of a gate by explicitly quantifying it [14, 15]. As a result, a gate-delay fault may not cause any wrong circuit behavior if the signal propagation path going through the gate is short enough to prevent the extra delay from breaking timing requirements. A *path-delay fault* is used to model the accumulative extra delay along a path comprising gates and wires [30].

- *Cell-Aware Fault Model*: Complex cells are widely used in ICs and the impact of intra-cell defects is increasingly becoming significant. In order to explicitly deal with intra-cell defects in test-related tasks for higher test quality, a defect-oriented fault model, called the *cell-aware fault model* [12], can be built as follows: First, all possible defects in a standard cell are extracted from its layout. After that, the behavior of each extracted defect is SPICE-simulated under all possible cell input combinations. Finally, all unexpected behaviors are modeled at the IO ports of the cell with a digitalized format for use in various test-related tasks, such as test generation and fault simulation. Fig. 8 shows an example, which shows that the fault Z1 in the cell MUX2 can be detected by any one of the three input value

combinations. That is, when any of the input value combinations is applied to a MUX2 cell with the fault Z1, the output of the cell will show a value different from the one for a fault-free MUX2 cell.

2 Test Basics

Defects are the dominant cause of IC errors and they may lead to system failures. Therefore, *test*, a check process aimed at determining whether an IC is defective, is critically important. Since defects may occur both in the manufacturing phase and in the operational phase of ICs, both *manufacturing test* and *field test* are necessary. On the other hand, today's ICs usually come in the form of Systems on a Chip (SoCs) consisting of heterogeneous blocks that comprise logic circuits, memories, analog and mixed-signal circuits, etc. Therefore, test methods need to be developed for these different blocks and for the entire SoC as a whole. Furthermore, since ICs are often assembled on *printed circuit boards* (PCBs) to make electronic systems, boards also need to be tested. In this section, general information on manufacturing test and field test is provided first. After that, the basics for logic test, memory test, analog test, SoC test, and board test are briefly introduced.

2.1 Manufacturing Test and Field Test

The manufacturing of today's ICs involves highly complicated processes, materials, equipment, and operations. This makes it impossible to perfect every factor in manufacturing, rendering it inevitable that some manufactured ICs are defective. As a result, manufacturing test is indispensable in guaranteeing IC quality.

Manufacturing test usually consists of multiple rounds. The first round of test, namely *wafer test*, is conducted for bare dies on a wafer through direct electrical contact with the bonding pads of the dies. Its purpose is to identify defective dies so that only good dies are packaged. After packaging, the second round of test is conducted through the external pins of packaged ICs. This round of test is necessary because imperfect packaging may introduce new defects and wafer test is often not thorough due to various limitations. The passing ICs of this round of test often goes through a process called *burn-in*, which is conducted by applying electrical, thermal, mechanical and environmental stresses so as to accelerate the occurrence of potential defects to the manufacturing phase, instead of leaving them to occur in the operational phase. Burn-in is especially important for ICs intended for mission-critical applications. After burn-in, the third round of test is conducted to screen out defective ICs whose defects' occurence is accelerated by burn-in. Only ICs passing all three rounds of test are shipped to customers. Note that manufacturing test is usually conducted with powerful *automatic test equipment* (ATE). Each round of manufacturing test usually consists of open-short test, DC parametric test, high-

speed I/O test, memory test, analog test, and logic test under various conditions related to voltages, temperatures, and operating speeds.

Field test is conducted for ICs already placed into a system, especially a mission-critical one. Different from manufacturing test, large and expensive ATE cannot be used for field test. Instead, self-contained test circuitry can be designed into a target IC so that it can test itself without using external ATE. This method is called *built-in self-test* (BIST). An alternative is to design self-contained test circuitry into the PCB board holding a target IC. This method is called *built-out self-test* (BOST).

2.2 Logic Test

The testing of a logic circuit is conducted by applying test stimuli to its inputs and observe test responses at its outputs. If all of the observed test responses match their corresponding expected test responses, the circuit is judged to be defect-free. Obviously, the confidence level of the judgement depends on the quality of the test stimuli used, which is measured by *fault coverage* achieved by the test stimuli. Fault coverage of a given set of test stimuli is obtained by *fault simulation*, while the process of generating a set of test stimuli to achieve high fault coverage is called *test generation*. Furthermore, it is important to note that, in order to improve test quality and test efficiency, the *circuit-under-test* (CUT) itself often needs to be modified or extended with some test-oriented circuitry. This concept is called *design for test* (DFT). In the following, DFT, fault simulation, and test generation are briefly introduced.

2.2.1 Design for Test (DFT)

Most logic circuits are sequential circuits that cannot be efficiently tested. This is because a sequential circuit contains *flip-flops* (FFs), whose inputs are hard to control and whose outputs are hard to observe. As a result, a common DFT methodology, called *scan design*, needs to be applied to sequential circuits [8]. As shown in Fig. 9a, scan design requires that all FFs replaced with *scan FFs*. A scan FF has an original *data input* and an added *scan input*, selectable by the *scan enable* (SE) signal. All scan FFs form one or more scan chains. *Scan test* is conducted as follows: In shift mode ($SE = 1$), a scan chain operates as a shift register, allowing test stimuli to be applied from the outside and previous test responses to be taken to the outside. In capture mode ($SE = 0$), all scan FFs operate as normal FFs to load test responses into FFs. This way, scan design makes all FFs both controllable and observable, thus greatly easing the test generation for sequential circuits. In order to reduce test data volume for scan test, *compressed scan design* can be applied. Fig. 9b illustrates such a technique, called *embedded deterministic test* (EDT), in which a ring-register-based decompressor restores compressed test stimuli on the input side and a compactor reduce test response data volume on the output

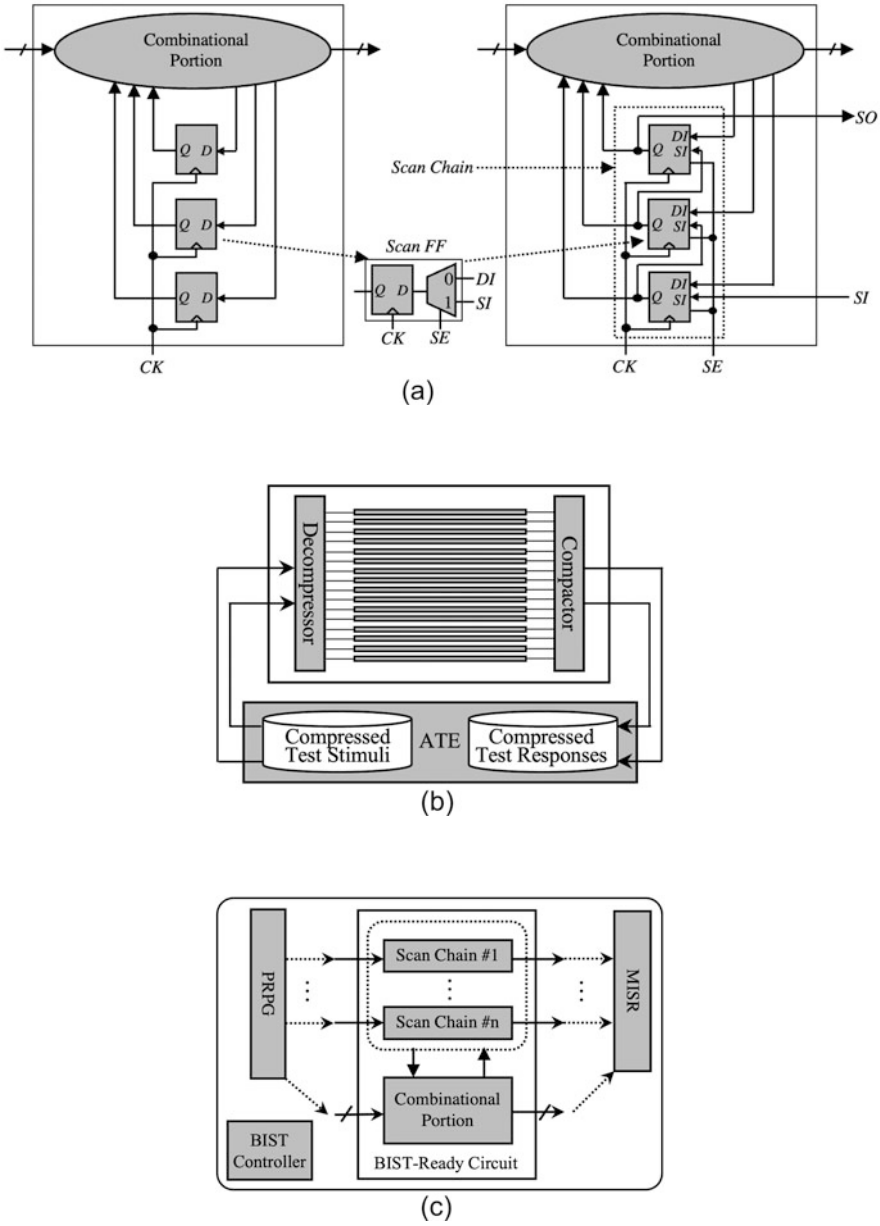

Fig. 9 Various scan-based DFT techniques

side [24]. Scan design is also the base for logic BIST as illustrated in Fig. 9c, in which test stimuli are generated by a *linear feedback shift register* (LFSR)-based *pseudo random pattern generator* (PRPG) and test responses are compressed into a single *signature* by a *multi-input shift register* (MISR) [4]. From the viewpoint of fault diagnosis, normal scan design provides the best test responses without any diagnostic information loss. Compressed scan design suffers from moderate diagnostic information loss due to the compaction of test responses while logic BIST suffers from the worst diagnostic information loss due to the compression of all test response into a single signature.

2.2.2 Fault Simulation

The most basic form of fault simulation for a combinational circuit (or the combinational portion of a scan circuit) c is conducted for a fault f and an input vector v to determine whether f is *defected* by v. Suppose that $r(c, v)$ is the output response to v by the fault-free circuit c and $r(c(f), v)$ is the output response to v by the circuit c with the fault f. If $r(c, v) \neq r(c(f), v)$, f is said to be *detected* by v and v is said to be a test vector for f. $r(c, v)$ can be obtained by logic simulation and $r(c(f), v)$ can be obtained by forcing the behavior of f into c. Note that a test vector v for two faults f_1 and f_2 may lead to identical output responses, i.e., $r(c, v) \neq r(c(f_1), v) = r(c(f_2), v)$. In this case, v can detect both f_1 and f_2 but cannot distinguish between them. A more general form of fault simulation is conducted for a set of faults F and a set of input vectors V. The purpose is to determine which faults in F can be detected by at least one vector in V. The percentage of defected faults is called the *fault coverage* of V. Compared with logic simulation in which each input vector only needs to be processed once, fault simulation is more time-consuming since each input vector needs to be processed once for each fault. To accelerate fault simulation, *parallel fault simulation* makes use of bit-parallelism of logical operations in a digital computer [29]. In the example shown in Fig. 10, a 4-bit word is used to store the signal values of each signal line. One bit is used to represent the fault-free value while the remaining three bits are used to represent the values corresponding to three faults. That is, three faults can be simulated simultaneously in a single pass. From the values obtained at the output q, the test vector $<x = 0, y = 0, z = 1>$ can detect two faults, "y SA1" and "p SA1". Other fault simulation approaches include *deductive fault simulation*, which deduces all signal values in

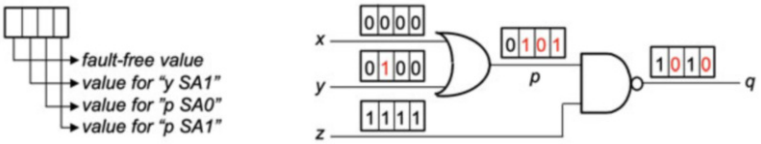

Fig. 10 Fault Simulation

each faulty circuit from the fault-free circuit values and the circuit structure in a single pass [2], and *concurrent fault simulation*, which emulates faults in a circuit in an event-driven manner to avoid unnecessary computation efforts [31].

2.2.3 Test Generation

The basic purpose of test generation for a circuit is to create a small as possible set of test vectors for achieving the highest possible fault coverage. Test generation is usually conducted with an algorithm in the form of *automatic test pattern generation* (ATPG). The target of generation is usually a combinational circuit or the combinational portion of a scan-based sequential circuit. Generally, there are two approaches to test generation, namely *non-fault-oriented* and *fault-oriented.*

Non-fault-oriented test generation for a circuit is conducted by considering the function of the circuit but ignoring its internal structure. Typical test generation based on this approach include *functional test generation*, *exhaustive test generation*, and *random test generation*. Although non-fault-oriented test generation is simple to implement, the number of resultant test vectors is usually large. In addition, although non-fault-oriented test generation itself does not need to consider faults, time-consuming fault simulation usually needs to be conducted to calculate fault coverage.

Fault-oriented test generation for a circuit is conducted by explicitly trying to create a test vector for each target fault under a fault model. Since structural information of the circuit is needed to define a fault model, fault-oriented test generation is also called *structural test generation*, which can be conducted with special algorithms, including D [27], PODEN (Goel 1981), FAN (Fujiwara 1983), and SOCRATES (Schulz 1988). A popular approach to structural ATPG, *path sensitization*, is illustrated in Fig. 11. First, in order to generate a test vector for the target fault as shown in Fig. 11a, "L_7 SA0", *fault activation* is conducted to make 1 to appear on L_7 as shown in Fig. 11b. As a result, the *fault effect D* (1 as the fault-free value and 0 as the value corresponding to "L_7 SA0") appears on L_7 as shown in Fig. 11c. Next, *fault propagation* is conducted to make the fault effect D or $\overline{D}$ to appear on at least on output of the circuit. For this purpose, 0 needs to be put on L_2 in order to make D to L_8 as shown in Fig. 11d and 1 needs to be put on L_3 in order to make $\overline{D}$ to the output x as shown in Fig. 11d and Fig. 11e. Note that fault activation and fault propagation lead to three value assignment requirements: $1 \rightarrow L_7$, $0 \rightarrow L_2$, and $1 \rightarrow L_3$. Finally, *justification* is conducted to determine necessary values for the inputs of the circuit to satisfy the value assignment requirements, and the result is a *test cube* (containing a don't-care value X) as shown in Fig. 11f. Since X cannot be applied by ATE, a logic value needs to be assigned to X by *X-filling*, resulting in a test vector $<a = 0, b = 1, c = 1, d = 1>$.

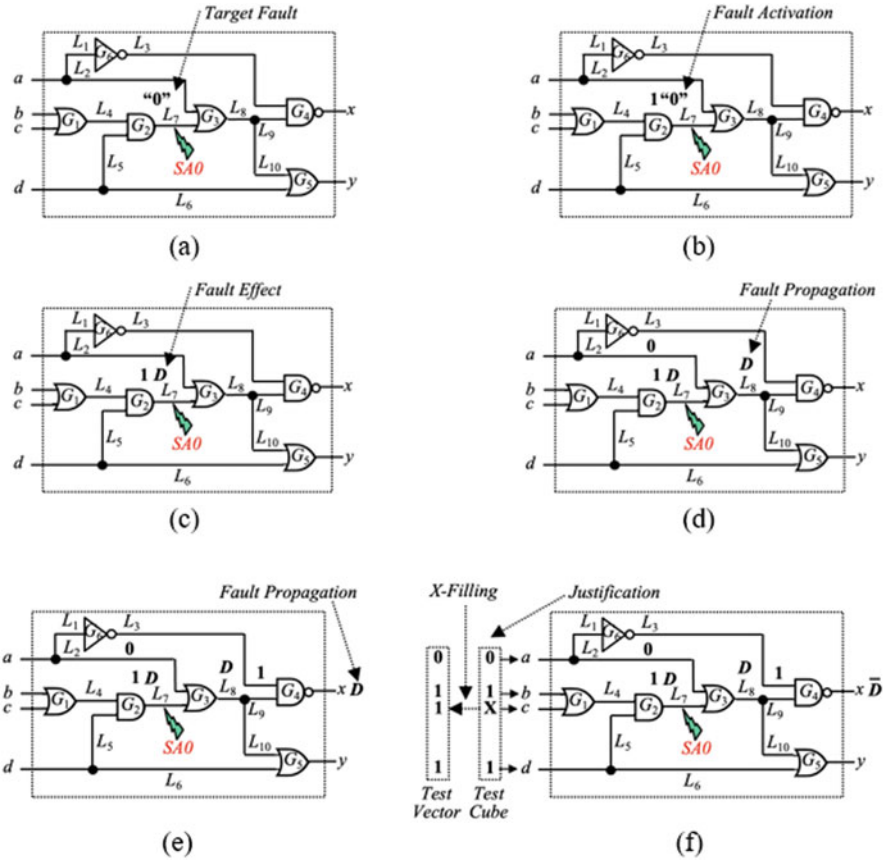

Fig. 11 Test generation basics

2.3 *Memory Test*

Different memories, such as *read-only memories* (ROMs) and *random access memories* (RAMs) require different approaches to testing. A ROM can be tested by reading out all stored values and checking if they are exactly what have been written into it. *Static RAMs* (SRAMs), *dynamic RAMs* (DRAMS), *electrically erasable ROM* (EEPROM) and flash memories need to be tested with test patterns targeted on various memory faults and generated by various memory test algorithms [1, 32].

Classical memory faults include *cell stuck-at faults* (i.e., the value of a memory cell is fixed at 0 or 1), *address decoder faults* (i.e., a memory cell corresponds to multiple addresses or one address corresponds to multiple memory cells), *data line faults* (i.e., defective input and output data registers prevent correct data from being written into or read from a memory cell), *read/write faults* (i.e., defective read/write control lines/logic prevent a read or write operation from being conducted), and

data retention faults (i.e., a memory cell loses its content after a certain period of time). Modern high-density RAMs also suffer from *transition faults* (i.e., a memory cell cannot undergo a 0-to-1 or 1-to-0 transition), *destructive read faults* (i.e., a read operation changes the content of a memory cell), *coupling faults* (i.e., the content of a memory cell is affected by the operations on other memory cells), and *pattern sensitivity faults* (i.e., the content of a memory cell is affected by the contents of other memory cells).

In order to test for the faults mentioned above, many test pattern generation algorithms have been proposed over the years. These algorithms can be classified into different types, such as N, $N^{3/2}$, and N^2, where N is the number of address locations or words. Generally, test patterns generated by an N^2 algorithm can detect more faults than those generated by an N al algorithm but result in longer test time. In addition, test patterns generated by different algorithms detects different faults. Therefore, it is necessary to select a proper set of test generation algorithms for a memory by taking possible faults into consideration. A typical N-class test generation for detecting cell stuck-at faults is the *modified algorithmic test sequence* (MATS), whose length is $4N$. This algorithm has three steps: (1) all memory cells in a RAM are written to logic 0, (2) each address is first read, with logic 0 being the expected value, and then written to logic 1, and (3) each address is read with an expected value of logic 1.

The test result of a memory can be plotted into its *fail bit map*, which shows the identified defective cell locations in the memory. From the shape or distribution of the defective cell locations, the possible cause of the defects may be found. For example, if the defective cell locations are not regular, the possible cause may be cell stuck-at faults. If all memory cells in a row or column fail, the possible cause may be address decoder faults.

2.4 Analog / Mixed-Signal Test

An analog or mixed-signal circuit, either in a standalone form or as part of a *system-on-a-chip* (SoC) device, is usually tested by explicitly checking its functions against specifications through measuring various performance parameters [1, 32]. Typical mixed-signal circuits include A/D converters and D/A converters. The static properties of an A/D or DA converter include linearity, gain error, offset error, monotonicity, miscode, *integral non-linearity* (INL), *differential non-linearity* (DNL). The dynamic properties of an A/D or DA converter include *signal-to-noise ratio* (SNR), *total harmonic distortion* (THD), *spurious-free dynamic range* (SFDR), and *effective number of bits* (ENOB). Obviously, such a parameter has a tolerance range instead of a single expected value. Therefore, it is necessary to determine whether a measured parameter falls within its design specification tolerance range. In the testing of an A/D converter, an *arbitrary waveform generator*

(AWG) is usually used to generate analog input signals and digital output signals are obtained by a tester. In the testing of a D/A converter, digital input signals are applied and analog output signals are obtained by a tester. Therefore, high-precision AWGs and digitizers are required. As a result, analog / mixed-signal test is usually expensive due to long test time and the need for complicated test equipment.

2.5 SoC Test

An SoC usually consists of embedded cores of various types and its test needs to target at all cores as well as the interconnects among them. However, both individual cores and interconnects in an SoC are usually difficult to access. This problem can be addressed with the IEEE 1500 Standard for Embedded Core Test, which is a scalable standard architecture for enabling test reuse and integration for embedded cores and associated circuitry [32, 33]. It uses a scalable wrapper architecture and access mechanism similar to boundary scan to facilitate test access to embedded cores and their interconnects. Note that IEEE 1500 is independent of the functionality of an SoC or its embedded cores. In addition, *built-in self-test* (BIST) also helps in easing the access requirement for embedded cores, especially logic and memory cores.

2.6 Board Test

Generally, various ICs need to be assembled onto a *printed circuit board* (PCB) in order to be used in an electronic device. This assembling procedure is complicated and prune to various defects, making it necessary to conduct *board test* [32, 33] The conventional method, namely *bed-of-nails*, of board test is to directly probe the solder points on the back of a board. This way, test stimuli can be applied to the input pins of component ICs and the test responses can be observed from their output pins. A modern PCB, however, usually consists of surface-mount ICs, whose pins cannot be accessed from the bottom of the board. This makes it impossible to apply the bed-of-nails method. This problem can be addressed with the boundary scan methodology, which has been documented as several IEEE standards. For example, the IEEE Standard 1149.1 for logic ICs inserts additional logic to form a boundary scan chain through all I/O buffers of logic ICs. This chain makes it possible to shift in test stimuli to internal pins and interconnections on a PCB. It also makes it possible to capture output responses at the input buffers on other ICs on the PCB and subsequently shift them out for observation. This way, access to all ICs and interconnections can be established without direct probe contact. The access to the boundary scan chain is provided by the *test access port* (TAP) through a four-wire serial bus interface and instructions applied through the interface. This boundary scan interface can provide access to the DFT features, such as BIST, of individual ICs on a PCB. In addition to IEEE Standard 1149.1, the IEEE Standard 1149.4 is

available for mixed-signal ICs and the IEEE Standard 1149.6 is available for the I/O protocol of high-speed networks.

3 Volume Diagnosis for Yield Improvement

As mentioned earlier, diagnosis is used to identify the location and failure mechanism of the defect. To improve the yield, it is important to identify systematic defects which are caused by the same root cause. To recognize systematic defects, statistics methods are used on volume diagnosis results to identify whether there are common root causes. Yield can be recovered only after the common root causes are removed properly.

In high volume production test environment, there can be many failing dies and lots of failing data to collect and process. Besides ATE, the machines with sufficient CPU and memory to process these volumes of failing data need proper management to get enough throughput.

3.1 Failing Data Limitation in ATE and Its Impact to Diagnosis Accuracy and Resolution

In scan test, each pattern is independent. In other words, the test results are independent on the order of test patterns. The main advantage of this independency is in logic diagnosis. The failing data information is per pattern. The diagnosis operation can be done pattern by pattern. However, diagnosis suspect count per pattern can be quite large. Statistics method is used to find the common defect suspect from all failing patterns. The suspect count can be further reduced by using passing patterns.

3.2 How to Collect Failure Data for Diagnosis?

To achieve high speed and precise test results, ATE is quite expensive. The hardware to store test results is expensive as well. In general, ATE has limited failure data storage space. To do chain diagnosis, to achieve good diagnosis results, at least 100 failing patterns are needed. Each failing pattern has many failing cycles when scan chains are faulty. Some ATE does not have such big failing data storage. To do diagnosis, it has to retest repeatedly to collect enough failing data. For logic diagnosis, each failing pattern has small number of failing cycles. Typically, 1000 to 2000 failing cycles are sufficient to achieve good diagnosis resolution.

3.3 Failing Data Format

To achieve accurate diagnosis, it is important to know which observe data is failing or passing. However, due to limited ATE failure data storage, some information is lost. Since data collected from ATE only identify failing data, it can be mis-leading to assume non-failing observe data are passing. To avoid wrong diagnosis results, it is important for precisely label each observe data of each pattern as failing, passing or unknown. Base on ATE setting, the failing data truncation can be pin-based, cycle-based or even pattern based. Diagnosis accuracy depends on proper failing data truncation information.

3.4 Volume Diagnosis Server Farm Setup to Process Thousands of Failing Data in Production Flow

The resource to do a diagnosis job depends on the memory and the time it needs to generate the diagnosis report. For a fixed amount of volume diagnosis job, the throughput can be improved by either using smaller memory or smaller run time. In a typical server farm to process volume diagnosis, there are much more machines with smaller memory. To improve the throughput of volume diagnosis, it is more important to reduce the memory used for each diagnosis job than to speed up the run time.

3.5 Beside Failing Data Per Die, What Other Data to Collect for Statistics Yield Analysis

It is common that each die may go through same test under several operation corners such as different voltage, different clock frequencies, different temperatures. All these test environment data should be considered when analyzing volume diagnosis results to find common root causes. Also die locations in wafer, and lot information should be considered as well.

4 Fault Diagnosis of Customer Return

Despite best screening efforts during a product's manufacturing process prior to customer delivery, defects do escape and end up as product failures encountered by the customer. Generally speaking, customer expectations of quality vary depending on the market and product segment. When instances of failure exceed an acceptable level or consequences of failure are severe enough, the product is returned to the

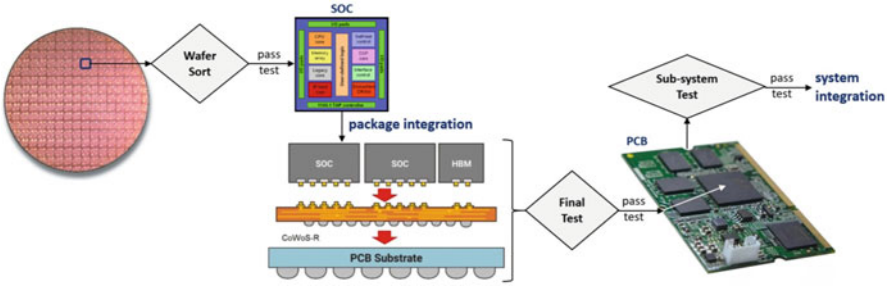

Fig. 12 Manufacturing test flow from wafer to PCB

manufacturer for root cause diagnosis. The goal is to prevent future occurrences of failure and to ensure minimal negative impact on the manufacturer's quality reputation. When the perception of quality takes a hit, it results in direct loss of business; and a great deal of effort and expense are needed to recover customer trust.

In looking across the full range of modern electronic products from hand-held mobile smart-phones to data center servers/routers/switches to artificial intelligence (AI)-based self-driving cars, one finds a complex assembly of electronic and mechanical hardware components with a stack of software layers above ultimately ending in interactions with the human user. The electronic components can be sophisticated system-on-chips (SoC) and multi-die integrated packages in their own right. When the user experiences a failure, it could be caused by defects in the software, in the hardware, or a combination of deleterious interactions among the components. Indeed, the likelihood of failure rises as an emergent property of increasing complexity [7].

Multiple testing steps are executed in the process of integrating and assembling components to create a final product. The purpose is to catch defects as early as possible since allowing escapes to the next step incurs additional cost. For example, consider the manufacturing test steps shown in Fig. 12 as a die component moves from wafer to package to printed circuit board (PCB). Dies on a wafer are tested and only those passing are diced and proceed to the packaging step. A defective die that escapes wafer test will be integrated with others in a multi-die package. If the integrated package fails final test, the defective die is successfully screened; but replacing it in the finished package may be too costly or impossible. Thus, the entire defective package has to be discarded rendering naught the costs of package integration and other companion good dies. If the defective die escapes package final test and causes failure in PCB sub-system test, the un-recuperated cost is even higher whether the PCB is re-worked or discarded. A well-accepted industry rule-of-thumb is that the cost of failure goes up 10X with each successive stage of integration and assembly.

To reduce the cost of failure, diagnosis should be performed at earlier steps in the test flow. Testing is never perfect and its inaccuracy has two aspects – allowing

defects to escape and failing a good part. The latter aspect is commonly called overkill and it could be due to faulty test equipment or over-stringent test conditions. The yield at each test step, defined as the ratio of passing parts to tested parts, is carefully monitored to make sure it stays above an acceptable threshold. Falling below threshold is known as a yield excursion and must be resolved timely since profitability is negatively impacted. Low yield excursions could be due to overkill or a rise in defects at earlier manufacturing steps. Through failure diagnosis, the manufacturer is profit-driven to resolve low yield excursions as well as to enhance yield via defect reduction.

The current semiconductor and electronic product supply chain is a complex ecosystem of vendor-customer relationships. For example, the semiconductor foundry is a supplier to the fabless design house who in turn is a supplier to the product system integrator who sells the system product to the user customer. The same entity can be both a customer and a vendor depending on whether one is facing upstream or downstream in the supply chain respectively. Consider a system integrator who sources components from multiple hardware suppliers. Incoming inspection through some form of system-level test (SLT) is performed on all received components. The system integrator judges each supplier on component quality while driving down costs in order to maximize profitability by selling system products to the user customers. Failure diagnosis of a customer return then becomes a shared responsibility up and down the supply chain.

Just as the cost of failure rises with each stage of assembly and integration, so goes the difficulty of failure diagnosis. For a complex system, diagnosing customer failure is notoriously laborious and time-consuming. A well-known case from the automotive space is the Toyota sudden acceleration issue that resulted in numerous fatalities and vehicle recalls. Investigations spanned several years identifying multiple causes including operator error, ill-fitting floor mat, sticky accelerator pedal, and possible design flaw in the electronic throttle control system [34]. For automotive electronic components where quality level is required to reach below 1 DPPM, root-cause analysis of customer returns may last more than a year and a significant portion of the cases still results in no-trouble-found (NTF).

In diagnosing a customer return, the first step is to reproduce the failure in a repeatable fashion. This may require replication of the customer's operating environment which is not always feasible. Once product defect, and not operator error, can be firmly established, then a series of experiments are carried out to isolate and narrow down possible causes. Defect isolation happens in both time and space. In the time dimension, some failures may take hours or longer to trigger. Long error detection latency (EDL) is a major challenge in complex system failure debug. Thus, reducing EDL is a topic of high research interest [20]. Usually both software and hardware have enhanced capabilities to do logging, take snapshots, and run self-checking and diagnostics to aid defect localization. Time-critical low-level software-hardware interactions in embedded systems is a frequent source of problems when the hardware itself is somewhat marginal. The problem can often be resolved by software changes to accommodate larger hardware variations.

When a system integrator is able to identify the hardware SoC component that is the prime suspect for causing system failure, it is extracted and sent to the SoC supplier for further diagnosis. Out of the system operation context, the supplier's challenge is to confirm that the SoC is indeed defective by running stand-alone tests on the isolated component. If no failure occurs when the original SoC production test patterns are applied, new patterns are added to target areas of the SoC that are suspected to be involved based on system failure symptoms. In the best case, the defect is exposed by high-coverage structural fault model-based test patterns in which scan diagnostic tools can help find the physical root cause. But the divide-and-conquer block-based approach of structural test may miss certain functional interactions in the SoC. As well in low-power designs with reduced operating voltage margins, subtle variation-related defects may only be detected by functional patterns or component-level SLT [23]. Catching defects with functional patterns raises the difficulty of diagnosis significantly. To aid functional failure analysis, embedded debug logic to gain deeper internal visibility [22] and sensors to obtain more granular measurements of internal health [17] have seen increasing adoption by SoC designers. In the worst case, the SoC supplier is unable to confirm that the component is the cause of system failure, so it ends up falling under the NTF category [6].

In summary, fault diagnosis of customer returns is still an unsolved and expensive challenge due to the inherent complexity involving many parties and factors. A customer failure is also a failure of the supply chain to deliver a quality product. Thus every effort should be made to screen defects at earlier steps, perform failure analysis, and improve upstream processes to minimize defects via yield learning. Though defects arise from random process variations, systematic aspects of the design and manufacturing process influence the probability of defect occurrence. Identifying systematic factors requires a minimum volume of failed samples to deploy volume diagnosis. The rarity of customer returns may not meet that minimum volume requirement. Indeed if there is a sufficient number of customer returns, it's a strong indication of serious inadequacies in quality control and yield learning in the supply chain.

5 Yield and Profitable Quality (*Changed from Initial ToC*)

When a product is manufactured in volume, some instances (so-called parts) may turn out to be defective and cause failures during deployment. Fig. 13 shows a simplified flow of design → fabrication → test → customer from which we shall derive the fundamental equation relating yield, quality and cost.

For a product, let N be the number of parts produced by manufacturing. Let's associate the descriptive labels *good* and *bad* with *non-defective* and *defective* parts respectively. Thus $N = G + B$ where G and B are respectively the number of good and bad parts. Intrinsic yield Y_0 of the manufacturing process is defined to be the ratio of good parts to total parts produced, i.e., $Y_0 = G/N$. In practice, Y_0 is not a

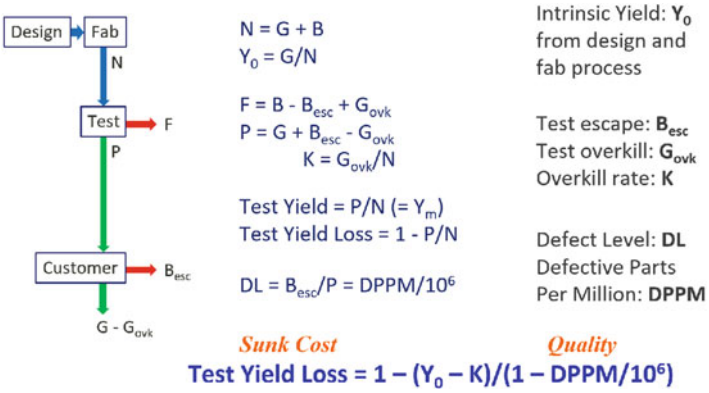

Fig. 13 Simplified flow of design→fabrication→test→customer process

known quantity though it is influenced by factors from the design and manufacturing process. Y_0 is estimated by testing each part to decide whether it is good (by passing the test) or bad (failing the test). Let P and F be the number of parts that passed or failed the test respectively where $N = P + F$. Then an estimate or measured value of Y_0 called test yield Y_m, is defined to be the ratio of P to N, i.e., $Y_m = P/N$.

If testing was perfect in separating good from bad, then Y_m is an exact measure of Y_0. But in reality, testing can make two kinds of mistakes: (1) let a bad part pass which is known as an *escape*; or (2) fail a good part which is called an *overkill*. Let B_{esc} be the number of escapes and G_{ovk} be the number of overkills. We obtain the following equations for F and P: $F = B - B_{esc} + G_{ovk}$ and $P = G + B_{esc} - G_{ovk}$. Let us also define the overkill ratio $K = G_{ovk}/N$. Only parts passing test are shipped to the customer. During product use, assume every bad product shipped results in customer failure. Then the measure of maximum failure rate as experienced by the customer is called *defect level* $DL = B_{esc}/P$ which is a direct reflection of product quality.

By algebraic manipulation of the definitions above, a relationship between Y_m and DL can be derived: $Y_m = (Y_0 - K)/(1 - DL)$. In the electronics industry, it is common to use the term *defective parts per million* (*DPPM*) instead of *DL*. *DL* is related to *DPPM* by $DL = DPPM / 10^6$. For the discussion that follows, it is more convenient to shape key concepts by considering *test yield loss* which is $1 - Y_m$. Test yield loss represents the portion of manufactured parts that fail in the testing stage and thus discarded. It is detrimental to profitability because costs associated with the discarded parts are a direct loss. When test yield loss is significant, the associated cost dwarfs all other expenditures involved in test development and volume production test. The equation for test yield loss is:

$$\text{Test Yield Loss} = 1 - (Y_0 - K) / \left(1 - \text{DPPM}/10^6\right)$$