Machine Learning Support for Logic Diagnosis

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik
der Universität Stuttgart
zur Erlangung der Würde eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigte Abhandlung

Vorgelegt von

Laura Isabel Rodríguez Gómez

aus Madrid, Spanien

Hauptberichter:	Prof. Dr. Hans-Joachim Wunderlich
Mitberichter:	Prof. Dr. Sybille Hellebrand

Tag der mündlichen Prüfung: 13. Juli 2017

Institut für Technische Informatik
der Universität Stuttgart

2017

To my grandpa Jesús

Contents

List of Figures

Chapter 6

List of Tables

Chapter 5

Chapter 6

Acknowledgments

This work would have never been possible without the contribution of many people, and I am happy to have the chance to express my gratitude.

I would like to thank Prof. Hans-Joachim Wunderlich for the opportunity to work at his department, where I found the space to grow and to learn how to approach scientific work, and also for the interesting discussions that greatly contributed to this thesis. I want to thank Prof. Sybille Hellebrand for her always constructive feedback and questions. Special thanks go to Marcus Eggenberger for his help with the German summary of this work.

A big thanks is due to my colleagues at the Institut für Technische Informatik for many insightful discussions. I am particularly grateful to those with whom I had the pleasure of sharing my teaching responsabilities: Rafał Baranowski, Chang Liu, Alejandro Cook, Dominik Ull, Eric Schneider, Claus Braun, Alexander Schöll and Ahmed Atteya. I am also thankful to Mirjam Breitling, Helmut Häfner and Lothar Hellmeier for their administrative and technical assistance.

I am greatly indebted to Willi Kessler, Gert Schley, Marcus Eggenberger, Manuel Strobel, Adrià Sales, Anto Levatino, Edu Ferrera and Max Schwilk, for sharing their experiences as PhD students in various universities and disciplines and providing me with continuous support.

Finally, my biggest debts of gratitude. To Pablo, for his patience and for his encouragement throughout the rockiest parts of the way. And to my parents and sisters, whose love and support in all of my adventures and undertakings are invaluable.

Stuttgart, July 2017

Laura Rodríguez Gómez

Abstract

Tiny feature sizes in deep submicron technologies pose both a yield and reliability threat. Imperfections in the manufacturing process may introduce systematic defects, especially as the first devices are produced when the process is not yet mature. The identification and correction of systematic process problems calls for efficient test and diagnosis techniques. Due to the increasing amount of variations introduced in the process parameters, however, these techniques must also tolerate a certain degree of indeterminism.

Chips that pass manufacturing test are then shipped to the customer and integrated in their target system. During the lifetime of a circuit wearout mechanisms can cause some of the structures to degrade. As a result, defects may appear in originally healthy chips. To avoid catastrophic consequences, many systems include in-the-field test techniques, which allow the detection of such problems. After detection, the defective parts must be diagnosed to identify any possible systematic degradation patterns, which point to weak structures. In addition to the indeterminism introduced by variations, the exact environmental conditions are unknown when in the field. Thus, test and diagnosis in the field must also handle uncertainty.

Whether after manufacturing or after online test, defective parts undergo logic diagnosis to locate the fault, and then physical failure analysis (PFA) to characterize the fault. However, PFA is a costly procedure because it requires physical inspection of the chip area. It benefits from logic diagnosis, which performs fault localization and narrows down the suspect area. PFA also benefits from a fast identification of the problem that would allow a prioritization of costly physical analysis procedures. Such a characterization is especially interesting in the case of faults which disappear when analyzed in the lab, either because they were only affected by noise or because the underlying fault is partially reversible. This thesis presents a fast characterization method which can be integrated in the diagnosis flow. It takes advantage of machine learning techniques, which can handle inaccurate or ambiguous information. The method incurs minimal overhead, and characterizes faults detected in manufacturing test as well as online.

Zusammenfassung

Die fortschreitende Skalierung von Prozesstechnologien gefährdet sowohl die Produktionsausbeute als auch die Zuverlässigkeit. Gerade bei neuen Fertigungsprozessen können Herstellungsfehler zu systematischen Defekten in Schaltungen führen. Um solche Defekte zu finden und den Fertigungsprozess zu überarbeiten, werden effiziente Test- und Diagnoseverfahren benötigt. Eine zunehmende Variabilität in den Prozessparametern führt zu Indeterminismen in den Fertigungsergebnissen, und die eingesetzten Test- und Diagnoseverfahren müssen diese tolerieren können.

Chips, die den Fertigungstest bestehen, werden an Kunden ausgeliefert und ins Zielsystem eingebaut. Über die Lebensdauer des Chips hinweg kommen allerdings Alterungserscheinungen zum Tragen, die zum Verschleiß einzelner Strukturen und damit wiederum zu Defekten führen können. Daher werden in der Praxis häufig sogenannte Onlinetests während des Betriebs eingesetzt, um solche Probleme frühzeitig zu erkennen und größere Schäden zu verhindern.

Nachdem ein Problem erkannt wurde, müssen die fehlerhaften Chips untersucht werden, um die Strukturen zu identifizieren, die für Alterung anfällig sind. Hierbei muss die Diagnose einen weiteren Indeterminismusfaktor berücksichtigen, da die genauen Umgebungsbedingungen, die den Fehler aktiviert haben, zum Diagnosezeitpunkt zumeist nicht bekannt sind.

Um systematische Defekte zukünftig zu vermeiden und den Fertigungsprozess zu überarbeiten, muss nach der Diagnose noch eine physikalische Analyse (PA) durchgeführt werden. Da diese Analyse sehr aufwändig ist, wird die Fehlerregion zunächst durch eine Logikdiagnose eingeschränkt. Die Logikdiagnose kann allerdings nicht zwischen systematischen und unsystematischen Defekten unterscheiden, so dass keine priorisierte physikalische Analyse für Chips mit systematischen Defekten vorgenommen werden kann.

Diese Doktorarbeit stellt ein Verfahren vor, das eine schnelle Fehlerklassifikation und damit eine priorisierte physikalische Analyse ermöglicht. Das Verfahren basiert auf maschinellem Lernen, und kann mit Indeterminismus umgehen. Sowohl Herstellungs- als auch Alterungsdefekte werden sehr schnell charakterisiert. Dabei sind die Kosten des Verfahrens sehr gering und es kann problemlos in bestehende Diagnoseprozesse integriert werden.

CHAPTER 1

Introduction

Short time-to-market requires the semiconductor industry to produce high quality electronic devices as efficiently as possible. The time constraint requires efficient test and diagnosis procedures, as well as fast systematic problem identification to speed up yield learning. Manufacturing and test are complex processes which deal with increasing degrees of uncertainty as technology evolves and moves in the deep submicron regime. Uncertainty affects also the diagnostic process, which can only confirm the nature of the problem after costly physical analysis of the defective parts.

An electronic device is printed on a silicon wafer by doping different regions of the semiconductor with different donors and forming transistors, which are, in turn, connected with polysilicon or metal wires to form cells. Cells are finally connected by metal wires and conform the complete structure of the circuit. The production process is, however, error prone. It tends to introduce excess or voids in the material, causing circuits to deviate from their functional or performance specification. For instance, an accidental connection between two metal wires or a void in an interconnect cause the circuit to implement a different function. A crack in an interconnect, on the other hand, increases the resistance in the wire and may potentially introduce an additional delay. These distortions in the physical structure of the chip are referred to as point defects.

Due to limitations in the manufacturing tools, chips fabricated in recent technologies are affected by variations. Variations cause different devices that implement the same

design to exhibit diverging timing behaviors. Thus, each instance of a design may have a different maximum frequency and minimum supply voltage that ensure its correct operation. If exercised in conditions over its performance capacity, the device will produce incorrect output values.

To screen out defective chips, all devices must undergo manufacturing test. Tests are performed on all produced chips to ensure high product quality. The objective of test is to exercise the structure of the circuit thoroughly in a short time, uncovering a large number of point defects effectively. Test checks the observed responses against the expected ones and splits the chips in two groups: pass and fail.

The pass/fail categories do not correspond exactly to healthy/faulty chips. Devices which passed all tests are deemed healthy and can be shipped to the customer, the faulty ones must be further analyzed, and not just discarded. A careful analysis must be performed to distinguish those circuits affected by point defects from those only affected by variations, and which could work in different environmental conditions. This is so for two reasons: it allows early correction of systematic problems and avoids unnecessary yield loss. Moreover, the tiny structures in deep submicron technologies are sensitive to noise. In addition to distinguishing errors caused by point defects from those caused by performance limitations, a mechanism is needed to identify incorrect behavior caused by transient noise. Correctly identifying all cases allows to ship high-quality devices while avoiding discarding healthy chips.

While a process is not mature, it renders low yield values, that is, a high rate of the produced chips is defective. Identifying the systematic problems and its root cause allows correcting the process and enhances yield ramp-up. An early systematic problem detection is hence crucial to ensure the efficiency of the manufacturing process.

Detecting systematic problems requires identifying the location, nature and size of the point defect. The goal is to correct the process as soon as possible to fabricate larger numbers of fault-free devices. The diagnosis procedure is responsible for the problem identification task. Diagnosis is applied only to failing chips, and often includes logic diagnosis, a second diagnostic pass and physical failure analysis. Logic diagnosis locates the faulty line in the device. The algorithms typically take the results of test as a starting point. However, test detects the problem, but its goal does not include diagnostic resolution. Diagnosis benefits from larger amounts of information than those provided by test. For this reason, a second diagnostic test pass is often performed to extract more information about the possible underlying cause. The diagnostic pass

may include adaptively generating new test patterns which guide the diagnosis more precisely. This process can be performed for every faulty device. Because additional information may help discard or confirm candidate locations, its application is costly but extremely useful to narrow down the defect location.

After diagnosis, failure analysis physically analyzes the chip to confirm the location and identify the nature of the problem. With nowadays size and complexity, the analysis would need to cover a large surface. Also, since the process may be partially destructive, targeting the correct location in the first attempt is a requirement. The analysis of the location reveals the underlying physical distortion. A large number of defective chips affected by the same distortion indicates a systematic problem. Early correction of systematic problems can succeed if systematic problems are detected and correctly identified in a timely manner. By using as an input the outcome of logic diagnosis, the area to be studied can be reduced to a few locations, which eases and speeds up physical analysis. The nature of the problem, however, must be identified by physical failure analysis.

Although effective, the root cause identification procedure presents some problems. The first is its time consuming nature. Chips are analyzed in no particular order. As the chip is detected to malfunction, it undergoes diagnosis and is then queued for physical failure analysis. This is a suboptimal strategy: not only are the checks for different defect types heterogeneous, but also an earlier exam of devices affected by systematic problems would confirm the root cause sooner and hence speed up yield ramp-up. The second is the nature of some of the problems present in devices. High-frequency power droop or crosstalk effects are not as easily visible as additional metal connecting two independent wires. They are only active in specific conditions and may sometimes indicate a problem in the design or in operating conditions rather than in manufacturing. Even worse, if the chip was only affected by noise, it does not present any weak or defective structures, and so physical analysis will not be able to draw any conclusion.

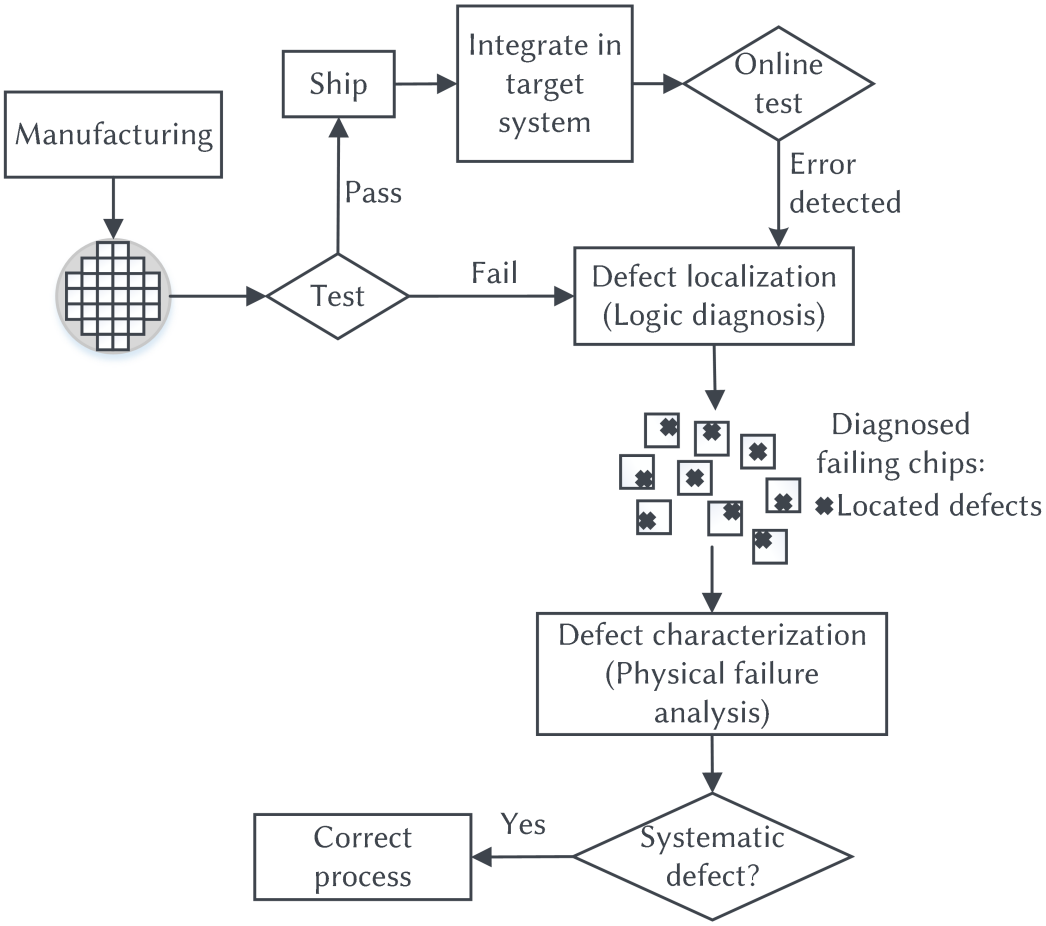
The need for defect detection and diagnosis is not exclusive of manufacturing test. Fault-free devices at the time of shipping may still fail in their intended environment. This can be caused by noise in the environment that affects the nanometer structures, or due to degradation. Wearout mechanisms can cause the transistors and interconnects in a digital device to degrade until permanent malfunction. Defects can hence also appear online after a certain time. For this reason, and very particularly in safety critical applications, online tests are introduced.

Like for manufacturing test, structural tests are highly effective to uncover defects. However, to be applicable online, they need to be carefully integrated in the system in order not to disrupt its function and performance properties. If the integration is successful, online test has the advantage that some faults are only detectable in this scenario, as their activation conditions may only be fulfilled in very specific environments or configurations.

Devices detected to be faulty during operational mode are returned to the manufacturer, who must analyze them to detect any possible generalized degradation profiles. Diagnosis and physical failure analysis are again performed. However, the amount of information gathered and stored in a safety critical system is typically limited due to cost or scheduling constraints. The available test data after online test is hence more limited than after manufacturing test. A second diagnostic pass could solve the problem and provide more results. Still, on top of being costly, some of the problems that may appear online disappear again if the stress conditions are removed. Very often the automotive and aerospace industry face the so-called no-trouble-found problem: field returns of safety-critical systems produced erroneous outputs which were detected by online test but which, upon further analysis in the laboratory, no longer behave erroneously and hence provide no information. The reason for this may be either random noise which caused the problem, partially reversible changes in the circuit caused by degradation, or temporary extreme environmental conditions unknown at the time of diagnosis. The lack of diagnosability leaves physical analysis with very limited information as a starting point, hence slowing the process down.

Figure 1.1 depicts the complete flow. Chips from manufacturing are passed on to test, which divides them into passing and faulty. The first are integrated in the target system and undergo online test, while the latter are diagnosed and analyzed. Field returns detected to be faulty during operation are also returned for diagnosis and analysis. In case of a systematic problem, the process or design are corrected.

Field return analysis faces the same problems as manufacturing defect analysis, and most of them aggravated. The lack of information complicates the analysis, the exact timing behavior is unpredictable, and also the characteristics of the problem (whether noise or defect, the location, size and nature in case of the latter) are unknown. Because environmental conditions play a role in the activation of some defects, this additional degree of uncertainty poses an added challenge to the already complicated defect analysis.

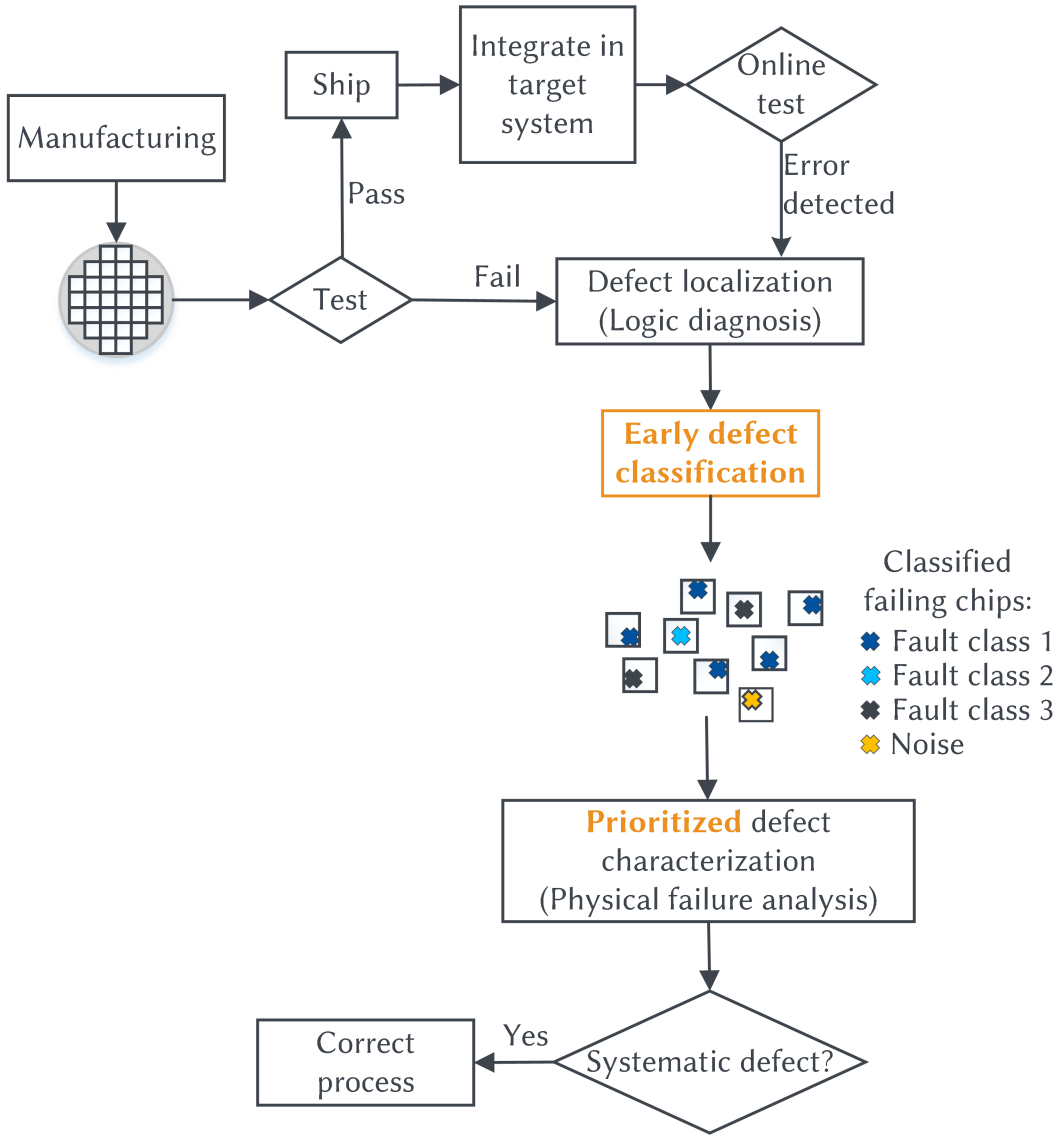


▲ **Figure 1.1** — Test and diagnosis flow

To increase yield and reliability, systematic defects and structures systematically affected by wearout mechanisms must be identified as soon as possible. To succeed, the identification approach must be robust with respect to uncertainty, given the number of factors that cause the device to deviate from its intended characteristics. Time and cost optimization are also mandatory requirements. Hence, prioritization of physical analysis according to the underlying fault type would be beneficial. The prioritization scheme must be integrated in the test and diagnosis flow. A low cost scheme is required, since test and diagnosis already represent a big part of the costs of producing electronics.

This thesis presents a low overhead modification in the traditional flow that helps prioritize the application of costly diagnostic procedures, and guides physical analysis even in the case of reversible faults. Figure 1.2 depicts the resulting flow. The block

highlighted in orange is the proposed modification, which distinguishes noise from critical defects and classifies the latter.



▲ **Figure 1.2** — Test and diagnosis flow after manufacturing: proposed flow

The approach makes use of *machine learning* because of its robustness even in the presence of uncertainty, such as caused in this problem by variations and the physical characteristics of defects. Its overhead in the complete flow is negligible, and it performs a fast classification to prioritize costly analysis resources, hence speeding up the identification of systematic problems, introduced during manufacturing or as a result of wearout.

This work is organized as follows: chapter 2 includes a description of defects and their corresponding fault models, as well as their relation with yield and reliability. Chapter 3 presents the algorithms for manufacturing and online test, diagnosis and preliminary fault characterization state-of-the-art. Chapter 4 presents the challenges introduced by variations and at the same time the potential of machine learning algorithms to be deployed in this context. The following chapters contain the contribution to the field. Chapter 5 presents a Bayesian network-based approach to distinguish transient noise from intermittent or permanent critical faults. Chapter 6 introduces a method for fault classification that can further guide physical analysis without requiring costly second diagnostic passes. Finally, chapter 7 presents the conclusions and directions for further research.

CHAPTER 2

Defects, faults and errors

Malfunctioning of a circuit has a wide range of sources. During the manufacturing process, imperfections in the fabrication introduce physical distortions in the structure. Depending on the size and nature of the distortion, it may cause the circuit to deviate from its intended functional specification, potentially introducing risks for the system. Also, as the circuit degrades, wearout mechanisms may cause defects to appear during the lifetime of the device. Finally, devices fabricated in nanometer technologies are very sensitive to environmental conditions. Some devices only malfunction under certain external conditions.

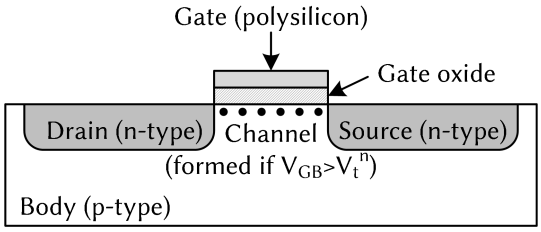
Defects introduced in manufacturing impact yield levels negatively. A test escape or the appearance of lifetime defect mechanisms can compromise the reliability of the system. Both for test and reliability assessment, fault models are used which abstract the defect behavior. Targeting defects adds complexity to the approaches and would make them, for the best part, unfeasible for nowadays circuits.

This chapter presents the most common defect mechanisms that may appear both in manufacturing and during the lifetime of the circuit. It then introduces the fault models with which the defects are represented at higher abstraction levels, and finally concludes with an introduction to the reliability threat defects introduce and their relation with yield.

2.1 Defect mechanisms

In complementary metal oxide semiconductor (CMOS) technology, the basic units, also referred to as gates or cells, are formed by transistors. The physical properties of transistors determine the functional and performance properties of the chip. Moreover, they are responsible for the sensitivity to noise of chips, and explain some of the fault models considered in this work. For this reason, this section explains the basic structure and function of a transistor, its implications on the design with CMOS technology, and the defects that can affect transistors and interconnects.

A transistor [Weste11] is a four terminal device which conducts depending on the voltage differences of its terminals. To form a transistor, some regions of the silicon are doped with an electrons (n) or holes (p) donor. Figure 2.1 sketches an nMOS transistor: a transistor which conducts based on electron mobility. The four terminals are marked on the figure: gate, bulk (body), source and drain. The body of the transistor is p-doped silicon, while the source and drain are of type n. The gate is usually polysilicon, and is separated of the body by a thin oxide layer. When the voltage between the *gate* and *bulk* terminals V_{GB} is greater than the threshold voltage V_t^n, a *channel* is formed along which the transistor can conduct a current between the source and drain terminals.

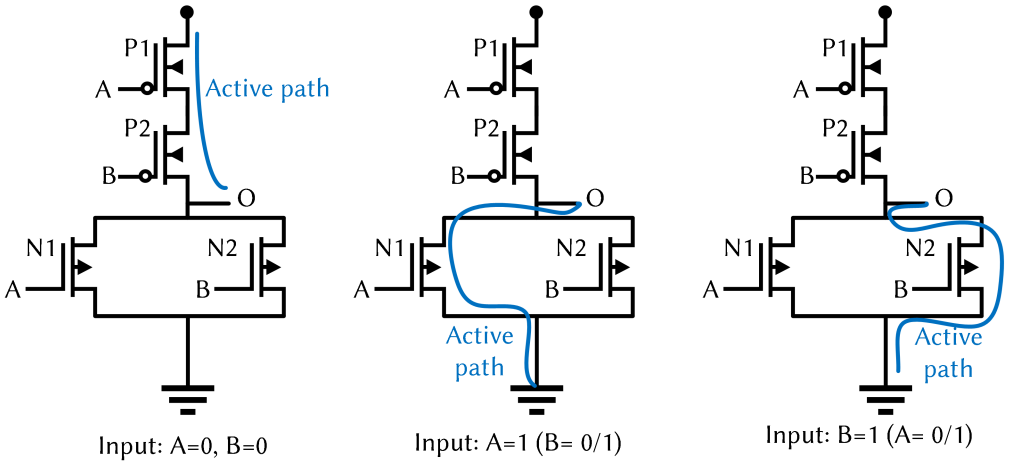


▲ **Figure 2.1** — nMOS transistor

The pMOS transistor is its complementary structure: it sits on an n-doped silicon, and its source and drain are p-doped. Its body is usually connected to power, and for a given threshold voltage V_t^p, it conducts when $V_{GB} < V_t^p$.

Both types of transistors are combined to form CMOS cells [Weste11]. A CMOS cell is formed by complementary p (or pull-up) and n (or pull-down) networks. The disposition of the transistors is such that for any input values, only one of the networks will be active. This ensures that the output terminal of the cell is connected either to power or to ground. Figure 2.2 depicts the CMOS cell which implements a logic NOR function. The

figure shows the active path for each input pattern (indicated below the corresponding illustration). Note that the expected logic output value can be interpreted as 0 or 1 because the networks are complementary and prevent an intermediate value of being placed at the output.



▲ **Figure 2.2** — CMOS NOR cell: transistor structure

To implement complex logic functions, cells are connected by wires. The wires are organized in layers of metal, and with aggressive scaling and increasing transistor density, the amount of interconnect infrastructure, including wires and vias, has also increased [Weste11].

This section describes the defects that can be introduced in cells and interconnects, not only during manufacturing, but also during the lifetime of a digital circuit.

2.1.1 Manufacturing

Imperfections in the manufacturing process complicate the fabrication of nanometer technology devices. In particular, spot defects such as excesses or voids in the material, and parametric variations, such as transistor channel length or doping, cause the fabricated circuit to deviate from the layout specification. This deviation of the physical structure with respect to the intended layout may modify the properties of the system. Depending on the nature, size and shape of the deviation, it may eventually lead to erroneous behavior.

Shorts

One of the most common defects introduced during the fabrication of chips are interconnect bridging defects [Segur04]. A bridge or shorting defect is the unwanted connection of two or more neighboring lines in the integrated circuit (IC). Such a problem can be caused by metal slivers between two lines or by larger blobs of material that short two or more lines. The electrical properties of the metal connecting both lines, as well as the properties of the surrounding circuit, translate into an effect on the circuit behavior.

A bridge will cause the circuit to malfunction if its resistance R_{bridge} value is below a certain critical value $R_{critical}$. In such case, the value of the connected lines depends on the strengths of the pull-up and pull-down networks. A bridge is activated if the pull-up network is active for the driving gate of one of the lines while the other line is pulled down. The resulting voltage of the bridge depends on the properties and strengths of the conducting elements: if the pull-up network is stronger, then a logic 1 will be interpreted, and 0 otherwise. The relative strengths of the driving gates are also patten-dependent. In [Segur04], the authors summarize in a table the $R_{critical}$ values of a defect bridge between a three-input NAND gate and a two-input NAND gate. The value varies depending on the logic state, i.e., on the input values of both gates.

Bridges can introduce non deterministic behavior in the circuit if the two affected lines are dependent. Two lines are dependent if one of them is function of the other, for instance the input and the output of a gate. A feedback bridge is a short between two dependent lines which introduces an undesired loop in the combinational part of the design. Depending on the structure of the circuit and the applied pattern, an oscillating behavior may be introduced [Chess98a].

Opens

An *open* [Wunde10] is a void or crack in the material which alters its physical properties. In interconnects, the resistance introduced by the void determines the impact on the system behavior. If the open defect causes the line to be interrupted, leaving one of the ends floating, it is called a full open. On the other hand, if the current can still flow although the region has a higher resistance value than designed, it is a resistive open. Resistive opens impact the timing behavior and introduce and additional delay. If a full open appears, the line is interrupted, and the value of the *floating* end depends on the capacitance between the line and the adjacent neighbors, and the voltage values of the

latter. The trapped charged in the floating line is Q_0. The total parasitic capacitance to power is denoted as C_{UP}, and those tied to ground, C_{DOWN}. The voltage of the floating line V_{FL} can then be calculated as [Wunde10]:

$$V_{FL} = \frac{C_{UP}}{C_{UP} + C_{DOWN}} V_{DD} + \frac{Q_0}{C_{UP} + C_{DOWN}}$$

so V_{FL} is mainly determined by the trapped charge and the ratio of the parasitic capacitance tied to power.

Finally, an intragate open causes two terminals inside a cell to be disconnected [Wadsa78]. It is also caused by a crack or void in the material, and it restricts the conductivity capacity of the affected path. In the worst case, there is a full open which causes the path not to conduct. This again breaks the CMOS principle that always one path conducts, either to power or to ground. As a result, for those patterns that should activate the affected path the output is left in high impedance. It has been observed [Wadsa78] that the output keeps the previous value, introducing sequential behavior in purely combinational parts of the circuit.

Parametric failures

In nanometer technologies, parameter variation has become a major issue. With aggressive scaling, the factor by which the manufactured devices differ from the designed ones has increased. This leads to a different kind of problem: in contrast to the previously introduced spot defects, parametric variations induce often a timing problem under certain given conditions. The relevant parameters may be divided into intrinsic and extrinsic [Segur04]. The intrinsic parameters include channel width and length, random doping fluctuations, different ratios between the nMOS and pMOS devices, and effective gate oxide thickness variations. Extrinsic parameters include temperature and supply voltage. These parametric variations affect the conducting behavior of transistors, and hence the timing of the logic cells. For instance, the power fed to a transistor gate controls the formation of the channel [Weste11]. Temperature, on the other hand, impacts carrier mobility [Wolpe12]. Intrinsic parameters, especially channel length, also impact the threshold voltage. A shift the threshold affects the formation of the channel, which impacts the timing behaviour [Segur04]. The challenge in parametric failures, however, is that one parameter variation on its own need not be fatal. It is

rather a combination of them that leads to failure, making the prediction, test and diagnosis of such problems extremely hard.

2.1.2 Lifetime

The environmental conditions in which a device works, sensitivity to noise and aging may cause a chip to malfunction, even if it passed all manufacturing tests. The term *noise* refers to a temporal malfunction caused by external conditions. An example is the radiation in aerospace industry [Dodd03], where heavy ions hit the microelectronic designs, causing memory and logic elements for a cycle or small number of cycles.

Problems during lifetime may also appear in the power supply [Chen98]. For instance, erroneous behavior may be caused by several transistors fed by the same power line and switching at the same time, causing spikes. Variations in the power supply may cause timing deviations, which in turn may cause an observable error [Segur04]. Also, if many neighbors switch in the same direction, some may suffer of power starvation and propagate an incorrect value to the outputs. This phenomenon is referred to as high-frequency power droop [Polia06].

A problem of different nature may arise if two interconnects are too close and run parallel for a long distance. In this case, the activity of one line may affect a neighboring line, i.e., interferences may happen. In particular, coupling capacitance or *crosstalk* causes a delay [Chen02] in the victim line transition if the aggressor(s) switch in the opposite direction. The induced delay can be expressed as a function of the distance (in time) between the transition in the aggressor and the transition in the victim. The largest delay is introduced when both transitions are aligned [Kahng00].

Another source of lifetime problems is wearout, also referred to as aging. Due to workload, temperature and voltage conditions along the life of the chip, the structures in it degrade. Wearout mechanisms cause defects to appear during the lifetime of the device. With scaling of CMOS technology, the sensitivity to wearout effects has become more severe [Segur04]. The most important wearout mechanisms include electromigration, hot carrier injection, bias temperature instability and dielectric breakdown.

Electromigration

Electromigration [Lieni05] is an aging mechanism which affects the interconnect infrastructure: wires, vias and contacts. It is caused by high current densities, which often

appear in integrated circuits. Unidirectional high current densities cause the metal atoms to gradually move. Eventually, this shift of the material can cause voids in the metal, or hillocks that can potentially cause a short. The mean time to failure (MTTF) of a wire can be estimated from Black's Equation [Maric13]:

$$MTTF = \frac{A}{J^n} exp\left(\frac{E_a}{kT}\right)$$

where A is a constant that depends on the cross-section of the wire, k is the Boltzmann constant, E_a is the activation energy of the material, J is the current density, n is a scaling factor and T is the temperature. Electromigration requires particular attention to be paid when designing the vias, since they are more prone to suffering this effect [Weste11]. With the introduction of copper, wires became more resistant to electromigration. Tin has also been identified as a good solution to mitigate the problem, but electromigration is still a problem in CMOS circuits [Maric13].

Negative bias temperature instability

Negative bias temperature instability (NBTI) [Weste11] affects pMOS transistors which have a negative bias and work in high temperature. The stress conditions over a long period of time cause some of the charges to be trapped, deriving in a shift of the transistor parameters, such as the threshold voltage. The total degradation induced by NBTI was found to have two components: a permanent and a recoverable one [Maric11]. The permanent component is caused by defects near the silicon/oxide interface, and increases with the time the device is under stress conditions (i.e., negative bias). The recoverable component, on the other hand, disappears in the *relaxation time*, that is, when the stress conditions are removed. The permanent component accumulates over time and is never reduced, which leads to eventual non-recoverable degradation after some time. Although NBTI mainly affects pMOS transistors, an analogous effect is observed in some technologies for nMOS devices under positive bias [Maric13].

Due to the recoverable component, NBTI is partially reversible when the permanent component is not yet significant [Maric13]. For this reason, upon later analysis the faulty behavior of the device cannot always be reproduced. This is known as the no-trouble-found (NTF) problem, a well known issue in automotive or avionic industries [LiVol11].

Hot carrier injection

A hot carrier is a particle (electron or hole) with high energy that allows it to overcome the interface state and deviate from its intended trajectory [Maric13]. In CMOS technology, hot carrier injection (HCI) injects these high energy particles in the gate oxide region near the drain. The effects of HCI translate into a shift of the threshold voltage in the affected transistor. This increase in the threshold voltage is often modeled as a function of time in which the device is under stress conditions. Temperature and the length of the channel also have an impact on the wearout effects: some of the possible causes of HCI include higher than specified voltage supplies, short effective channels or accidental peaks in the power rail that increase the power supply [Segur04]. Although HCI primarily affects nMOS transistors, it can also affect pMOS devices and enhance other wearout effects such as NBTI [Maric13].

Dielectric breakdown

Gate oxide or dielectric breakdown [Weste11] occurs when a large electric field is applied to the dielectric material, causing it to lose its insulating properties. If the electric field is very large, a so-called hard breakdown occurs. A hard breakdown causes severe local damage and may short the transistor, bonding the silicon substrate to the polysilicon of the gate in extreme cases [Segur04]. In CMOS technologies, this situation arises only with high voltage supply values [Maric13]. For weaker electric fields and thin oxides the degradation is partial, and is referred to as soft breakdown. In soft breakdown, the gate current or voltage may increase slightly. However, the clearest indication of breakdown is an increase in the gate current noise. For ultra-thin oxides, soft breakdown appears first. Then, if the chip endures the stress condition for a long time, soft breakdown is followed by progressive breakdown, which induces an increase of the gate current. Although soft breakdown does not always imply a final hard breakdown, it favors it as it weakens the oxide structure.

The time to the appearance of dielectric breakdown is a function of voltage, temperature and oxide thickness. The degradation of the dielectric material is not reversible.

2.2 Fault models

Test pattern generation, test pattern quality assessment and diagnosis, among other tasks, deploy an abstraction of the defects in the physical structure of the circuit. The behavior of defects is represented with *fault models*, which abstract the effect of the physical distortion at a higher level. Logic level is a higher abstraction level that represents the voltages of the lines as binary variables. In other words: it interprets low voltage values as a logic 0 and high voltage values (close to V_{DD}) as a logic 1. With this transformation, faults can be represented using the *conditional line flip* (CLF) fault model [Wunde10]. A CLF has the form $s_v \oplus \left[cond\right]$, where the variable s_v identifies the victim signal to be flipped and *cond* are the *activation conditions*, expressed as a Boolean formula. The CLF calculus allows the description of all fault models.

2.2.1 Stuck-at faults

The *stuck-at* model [Bushn13] has been widely used in many applications. It assigns a fixed value to a victim line s_v of 0 (s_v@0) or 1 (s_v@1). For a long time the stuck-at has been the state-of-the-art representation for test pattern generation or diagnosis. In CLF, a stuck-at-1 fault can be represented as $s_v \oplus \left[\neg s_v\right]$. The condition that triggers a flip in the line is simply that the value of the victim is 0. As a result, the line is permanently fixed to 1. Analogously, a stuck-at-0 can be represented as $s_v \oplus \left[s_v\right]$. The stuck-at model can represent a short to power or ground. However, it is not flexible enough to represent defects with sophisticated activation conditions.

2.2.2 Delay faults

As mentioned in 2.1, defects can cause a circuit to deviate from its intended timing specification. In a fault free circuit each gate has an expected delay, derived from *nominal delay* and variations (cf. section 4.2.2). The timing specification of the complete circuit can be derived from the delay of its components. A *delay fault* is a model that assumes a mismatch between the expected delay and the actual delay of a component due to a defect. There exist different delay faults, the most important of which are path delay fault, transition fault and small delay fault [Wang06].

A *transition fault* [Bushn13] assumes that the time for the rising or falling transition deviates from the expected transition time. Each gate may thus have a slow-to-rise (str) and slow-to-fall (stf) fault. In the transition fault model, the delay is bigger than

the slack of the shortest path, i.e., all outputs to which the fault can be structurally propagated are affected. The activation condition for this kind of fault is hence a transition at the inputs of the gate that generates a transition at the output. Another model is the *path delay fault* model [Wang06]. A path is a set of gates and signals which connect a primary input or a flip-flop to a primary output or a flip-flop. Unlike the transition fault model, the path delay fault model considers the cumulative delay along a complete path instead of that of an individual gate. The number of paths in a circuit may be exponentially large, which makes the path delay model less practical than the transition model [Wunde10]. Finally, *small delay faults* [Tehra11] introduce small additional delays in the circuit. They reflect the behavior induced by small sized defects, and pose a challenge for test, since they can only be detected with a long path that crosses the fault site.

Crosstalk-induced delay can also be modeled in CLF calculus [Wunde10]. The aggressor line is denoted as s_a, and its value in the previous time unit as s_a^{-1}. The same applies for the victim line s_v and its evaluation in the previous time unit s_v^{-1}. The transition in the victim signal is delayed if the condition that both lines flip in opposite direction is fulfilled: $s_v \oplus \left[(s_v \oplus s_v^{-1}) \wedge (s_a \oplus s_a^{-1}) \wedge (s_a \oplus s_v)\right]$. Note that for simplicity the time is considered discrete in the above formula. However, the time points can be easily substituted by a time point in the continuous time for a more exact representation of the actual defect.

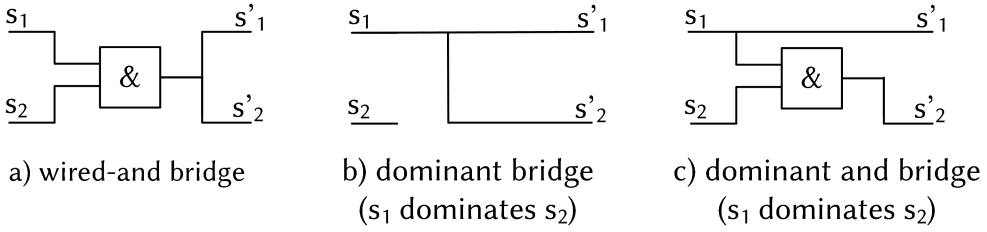
2.2.3 Bridges

Different bridge models exist to reflect the effects of bridge defects [Wang06]. The common activation condition to all of them is that the two affected lines have different logic values. However, the resulting value of the line differs depending on the assumptions made by the fault model.

The affecting line is referred to as *aggressor*, while the affected one receives the name of *victim*. In some fault models, both lines may be at the same time aggressor and victim. So-called wired bridges [Wunde10], for instance, consider that the two lines are affected and get the same value. These models assume a zero-resistance bridge, and the resulting logic value of the signals is calculated by combining the two fault-free values with a logic operation. Wired bridges come in two flavors: the AND-wired bridges, which model n-networks stronger than their p counterparts, and OR-bridges,

which assume the opposite. This model was developed for technologies in which one net clearly dominates over the other.

More realistic for CMOS technologies, the *dominant* bridge models [Wang06] were developed. They assume one of the driving cells dominates over the other: the victim line takes the value of the aggressor always. To reflect the behavior of resistive bridges, the dominant AND- and OR-bridges were developed: the victim takes the value of the aggressor only for a certain value. In a dominant AND-bridge, for instance, the victim line will be driven to 0 if the aggressor is 0. However, if the aggressor is 1, the victim maintains its fault-free value [Wang06]. Figure 2.3 shows the difference between the wired and the dominant fault models (in the AND variant).



▲ **Figure 2.3** — Bridge fault models. Adapted from [Wang06]

The byzantine bridge model [Cheun07] causes either of the two lines, or both, to flip upon fulfilling the activation condition that their values are different. This model accounts for the fact that in a bridge, intermediate voltage levels may be generated which are, in turn, interpreted differently by the gates fed by the affected lines. The more realistic fault models take into account the driving strengths of the involved wires. More sophisticated models, such as the voting model [Acken91], were proposed. Also, depending on the resistance of the connection, the bridge may even present a timing component, introducing a delay when the activation condition is met [Li03].

The CLF calculus is also able to represent all of these fault models. A dominant-and bridge, for instance, would be written as $s_v \oplus \left[\neg s_a \wedge \left(s_v \oplus s_a\right)\right]$. In other words: the fault is activated if the aggressor and victim have opposite values and the fault-free value of the aggressor is a 0.

2.2.4 Transistor faults

Transistor defects are also represented at higher abstraction levels. The most relevant fault models are the stuck-on and stuck-off [Wang06]. The stuck-on model assumes the affected transistor is always conducting. Hence, for some patterns a path will be created from the power supply to ground. In a CMOS inverter, for instance, if the nMOS transistor is stuck-on and a logic 0 is applied at the input, the voltage at the output is an intermediate value. It can be interpreted as a 0 or a 1 by the subsequent gates.

A stuck-off, on the other hand, assumes a permanent disconnection between the drain and source terminals of a transistor. A stuck-off transistor introduces a memory effect in the cell. The value of the output signal is calculated as the logic function implemented by the cell for all patterns except for those which exercise the affected path between a power line and the output. In the example of the inverter, if the nMOS is affected by a stuck-open, it never conducts. Hence, for the patterns that exercise that path (input = 1) the cell retains the previous output value. For the rest of the patterns (in this case only input = 0), the power-up net is active and the cell produces the correct value.

2.2.5 Non-determinism in fault models

While the activation conditions of faults are known, modeling still poses a challenge. Due to variations, the exact physical structure of the circuit is unknown. On top of that, the size and nature of the defect cannot be predicted, either. With unpredictable exact timing behavior due to variations, shrinking feature sizes and more complex fault activation conditions appearing, the degree of uncertainty increases. It is not realistic to expect deterministic behavior at logic level. Physical parameters and the nature of the problem influence the activation rate, causing the activation to look random for those patterns in which the logic activation conditions are met. As an example, the NBTI effects may appear random at logic level since they depend on the temperature, which is not taken into account in logic simulation. Thus, an additional classification [Const03] of the faults into permanent, intermittent or transient is introduced. The category depends on their activation rate and location:

- *Permanent* faults are those which are activated in the same location(s) and have a deterministic behavior.
- *Intermittent* faults are those activated in the same location(s), but whose activation appears random. They are characterized by an activation rate, which determines

for which patterns, among those which satisfy the logic condition, the fault is activated.

- *transient* faults represent random noise. They affect random locations and have a low activation rate.

In new technologies, both intermittent and transient faults are gaining importance.

Permanent, intermittent and transient faults can also be represented using the CLF model by introducing additional conditions. A permanent fault is activated always, and needs no further specification. However, intermittents and transients are activated with a certain rate. Let $s_v \oplus cond$ be the CLF expression for a fault. If the fault is actually intermittent, its activation is not static but depends on a rate λ_a. To account for the indeterminism introduced by the lack of accuracy of the models and the lack of knowledge of the exact physical parameters, it is enough to generate a random number r within a range $[r_{min}, r_{max}]$. Then, the condition $r < \lambda(r_{max} - r_{min})$ is added to the activation. As a result, the CLF model for the intermittent fault is $s_v \oplus \left[cond \wedge (r < \lambda(r_{max} - r_{min}))\right]$. Analogously for a transient fault with activation rate μ_a.

2.3 Reliability and yield

The goal of manufacturing is to efficiently mass-produce high quality circuits [May06]. Many aspects of manufacturing impact that goal, including costs, yield and reliability. Yield is an important indicator of the maturity of the production process. It measures the proportion of fault-free devices among all produced ones. The cost per device and yield are inversely related: for a given global manufacturing cost, the higher the yield, the lower the cost per circuit and viceversa.

Reliability measures the probability that a chip will continue to function correctly over time. Both yield and reliability are threatened by continuous scaling: in nanometer technologies the manufacturing process must come up with solutions to print structures smaller than the used wavelength, and chips become increasingly sensitive to wearout mechanisms and noise [McPhe06].

2.3.1 Yield ramping

When the production of a new design is started, yield levels are generally low. Yield losses come mainly in three different flavors: parametric, systematic and random

[Madge05]. Each yield loss class is caused by different factors and has varying impact on the final total yield. The corresponding corrective actions to tackle the problem are also heterogeneous.

Parametric yield losses are those circuits which are functional, but do not comply with the non-functional specification. For instance, due to process variations, the timing of the circuit and hence its maximum operating frequency are not fixed. Instead, the longest path delay follows a distribution [Kim03]. Depending on the target frequency, the slowest circuits may easily fall out of the non-functional specification. For example, they may deviate from the intended frequency requirements, and require a longer clock cycle to correctly perform the function.

Systematic yield loss is caused by errors in the production flow. A process which is not yet mature may systematically introduce voids or cracks in the interconnects, or dope the silicon incorrectly, or have misaligned masks. The undesired effect will be injected in a high number of devices, drastically reducing yield.

Finally, random yield losses are defects introduced arbitrarily, and can be caused by spurious contamination during production [Wali09].

The process of increasing the yield levels to a significantly higher value, proper of a mature process, is referred to as *yield ramping* or *learning*. Yield learning requires first the identification of the problem, and then the correct countermeasure for each yield loss type.

Parametric yield losses

Parametric yield losses can be avoided with techniques such as binning. Binning separates the ICs in different categories instead of merely labeling them as passed or fail. It is a well-established industrial practice, and industry reported selling microprocessors in up to six different speed grades already at the beginning of the 2000s [Belet02]. The categories are based on the performance of the device with respect to different non-functional aspects, for instance speed [Kim03] or voltage [Shen12]. With shrinking technologies, however, the variation has increased and a careful selection of test patterns, infrastructure and configuration must be made to keep the technique affordable.

The simplest versions of binning applied functional tests to the devices under different configurations to pick out the maximum frequency or minimum voltage they need [Bushn13]. However, testing all possible configurations of parameters is not

feasible, and different approaches have been proposed to minimize test time while being able to predict circuit performance. The authors in [Zeng04] demonstrate the correlation between structural delay tests and functional testing frequencies. In [Belet02], experiments are performed on a microprocessor to compare the correlation of structural and functional tests. They conclude that path-based test correlates better than transition based test, with the limitation that the test generator must be improved to infer a high path-coverage test sequence which meets the test cost requirements in terms of time and memory. In [Paul07], some long paths are chosen at design time. They can be configured as ring-oscillators, and the authors measure the timing in a small subset of voltage and frequency configurations. The differences in the timing measure the sensitivity of the circuit, and this sensitivity is used to extrapolate for other configurations, which allows to predict the maximum frequency at different operating points for the design.

Binning may also be performed based on the supply voltage. In [Shen12], the authors infer an optimal number of bins given a low bound for yield and the corresponding binning algorithm. They take the test measurements of a population, and represent the voltage intervals as horizontal lines on a graph. Then, the optimal number of bins is inferred. The algorithm requires that each fault-free chip falls in at least one bin, hence, the problem is a cover of vertical lines which crosses all voltage intervals of the population. The authors in [Licht13] take advantage of the relation between performance and leakage to propose their binning approach. In particular, and given that systems have a power budget, they propose to reduce the supply voltage to the fastest circuits. That way, leakage is reduced but the chips' performance is not affected. This reduces yield losses since it rescues the fastest chips, which would be discarded by a leakage screen.

Systematic yield losses

To reduce systematic yield losses, the root cause must be identified to correct the imperfection in the process as soon as possible. Often, process monitors are introduced in the wafers in order to control the development of manufacturing. Process monitors are special test structures manufactured in the same technology as the chips, and probed by the test equipment to check the existence of systematic defects. The original snake and comb test structures introduced were able to detect a short or an open. However, they could not provide information about the size of the introduced defects. More

recent structures haven been proposed. An example is the NEST structure presented in [Hess01], which is formed by a set of serpentine-shaped nested wires. Each wire is connected to two pads, and electrical measurements on them allow the detection of systematic problems. For instance, the resistance between two pads may give an indication of opens or shorts. Moreover, depending on the measurements, it is possible to infer which lines are affected and hence the defect size distributions. The NEST structure requires only one mask and can be implemented in one layer. However, other structures can be introduced to monitor the introduction of defects between different layers. The authors of [Khare94] propose to use the double bridge test structure to extract defect densities and sizes between two layers, while in [Hamam04] the structure includes open and short monitors, which allow not only the detection but also the location of the defect through the electrical measurements.

Despite careful design based on knowledge about previous processes, test structures cannot always reflect the diversity of production IC structures [Blant12]. For this reason, defective ICs are also deployed as a source of systematic yield loss information. In order to increase yield levels, two approaches are possible [Bushn13]. The first is known as diagnosis and repair, and it consists of performing a thorough diagnosis for the defective parts, which are then repaired. Although yield values increase, so does manufacturing cost. The economic costs make diagnosis and repair less attractive to correct systematic problems. Process diagnosis and correction, on the other hand, tries to find the source of the problem and correct it. Once the systematic problem is corrected, the defect density decreases, causing yield to increase.

In order for the process diagnosis and correction approach to succeed in rapidly ramping yield, the identification of the root cause of the problem must also be a fast process. Ideally, it should not impose a high overhead on the already costly test flow. Otherwise, handling many chips affected by defects will get prohibitively expensive. A fast and cheap failure analysis diagnostics approach combined with process imperfection correction becomes the crucial step for yield learning [Madge05].

The general definition of yield is the ratio of fault-free dies per wafer. However, some authors go one step further and point out that the fault-free devices discarded by mistake also constitute yield loss [Madge05]. This could apply to parametric yield losses, which have been tackled with binning techniques. However, it also applies to those circuits affected by spurious noise during test. If the noise affects a component in the device and propagates an incorrect value to the outputs, then the chip will be labeled

as faulty and discarded. Also, chips for which the outputs were sampled incorrectly due to the resolution of the test equipment may be identified as faulty. These categories will not only pose a problem for the analysis, since no defect will be found, but will also contribute to the number of unnecessary yield losses. Hence, careful analysis needs to be conducted to avoid this situation. However, due to sensitivity to noise and sophisticated defect mechanisms, in nanometer technologies the distinction between noise and systematic problems is not yet resolved.

2.3.2 Lifetime tests

After manufacturing and passing test, a device can be shipped to the customer. However, test escapes as well as aging mechanisms can lead to system failure. In order to ensure the reliability throughout the complete life cycle of the device, two kinds of strategies are deployed, namely, circuit failure prediction and online tests.

Circuit failure prediction attempts to anticipate the time in which degradation will have affected the system and will lead to failure. The main advantages of this technique are the early failure detection, before the system actually degrades and ends up in an unsafe state, and that it is generally an inexpensive approach [Agarw07]. However, the success of the prediction is determining, and a careful selection of features must be performed for prediction. Prediction methods are based on online collecting information about the workload or timing properties of the system combined with a prediction model, generally implemented in software and which can run online, hence preventing the system of degrading with catastrophic consequences.

In [Baran15], the authors propose to monitor the workload of some representative cells in the circuit. The selection of the set of cells is crucial, as they need to characterize the whole circuit. The choice is based on feature selection with a wrapper method [Hall99], which extracts relevant features well correlated with the corresponding expected output even if they are uncorrelated among them. The resulting set is then monitored online, and a software evaluator which contains a model of the expected aging pattern checks for the consistency of the signals.

The authors from [Agarw07] propose the design of special monitors which are provided with a guardband. When the guardband is violated, it means that one of more paths have aged enough to delay the signal arrival into the guardband. However, the functional flip-flops of the circuit still capture the correct value. Consequently, the degradation of

the system is detected in advance while the system itself is fully functional and within performance specification. The monitors are combined with a self-adjusting model based on the degradation registered after small periods of time.

Other methods try to identify degradation by monitoring the timing performance of the circuit. In [Agarw08], the authors propose a method to calculate a maximum bound for degradation. Based on the results of this prediction, they choose the optimal placement for the set of monitors at the end of long critical paths. The costs are reduced with the approach presented in [Liu15b], in which the authors consider path segments instead of whole paths, hence increasing the path coverage. To avoid the clocking problems of placing the monitors in arbitrary nets, the monitors are controlled by the inverted clock.

Unlike failure prediction, online error detection can only discover errors once they have already happened. The disadvantage is, of course, that some errors can lead to system failures. However, with good fault coverage and latency values, it only needs a fault tolerant design to allow for graceful degradation while detecting any problem. To this end, many architectures have been proposed which involve an observer, who predicts the fault-free outputs of the module and detects any mismatches [Kocht10] [Sharm88] [Drine03]. Some test approaches are developed so that no additional hardware is needed [Psara10], while other techniques [Wunde98] can be integrated as part of a system-level online strategy [Reima14], [Abele14].

Online error detection techniques are crucial not only because they identify fault occurrences, but also because this instant detection allows the system to collect data about the patterns which activated the fault and also about the obtained response. This information is relevant for later analysis, particularly for faults which are partially reversible or only activated under specific environmental conditions. Without online error detection and information registration, the analysis of such faults is extremely complex.

CHAPTER 3

Test, diagnosis and fault classification

Test is the procedure in which input stimuli are applied to a unit, and its responses checked against the expected ones. In case of a mismatch, the circuit is deemed faulty. The goal of test is to detect faulty devices as early as possible. Hence, the first step of test is the generation of input patterns which uncover the defects in the device effectively. In order to test a device, the target fault model is first established. Then, a test set is generated which attempts to activate and propagate to the outputs as many of the considered faults as possible. Finally, the test patterns are applied to the device. If the obtained response deviates from the expected output pattern, the chip is identified as faulty and discarded.

To ensure high product quality and reliability, tests are performed throughout the complete lifetime of a circuit. Manufacturing test sorts out the failing devices, which undergo the fault location phase, and finally physical failure analysis (PFA). Although the last step is particularly costly, it is key to yield ramp up: PFA identifies systematic defects so that they can be corrected as soon as possible. PFA cost is partially reduced by applying logic diagnosis. Narrowing down the suspected defect area reduces the time required for PFA.

After a chip is shipped to the customer and integrated in its target system, in-the-field

test are performed to detect any possible degradation effects. Should any problem be detected, the chip must go through the diagnosis and PFA procedure to identify weak structures systematically affected by degradation. Due to the lack of insight into the defect mechanism, physical analysis is performed without the possibility of prioritizing it and hence saving time and resources.

This chapter gives an overview of the test flow and the involved steps. The work at hand will be integrated in this flow and provide the base for a prioritized physical analysis. First, section 3.1 introduces the basic algorithms involved in test and diagnosis. Section 3.2 explains the basic manufacturing flow, and how test algorithms are deployed to ensure high product quality. Section 3.3 introduces the online test strategies that make it possible to ensure reliability and which gather information about any possible faults in the system for later analysis. In section 3.4 an overview of all available test data sets is presented, and finally section 3.5 presents available fault classification techniques and how PFA benefits from them.

3.1 Test algorithms

Detecting defective chips requires tailored test pattern generation to uncover problems at any possible location. Test pattern generation is performed based on a fault model, i.e., a representation at a higher abstraction level of the physical distortion in the chip. Test generation algorithms proceed to choose input combinations that activate the fault and make it observable at a primary (or pseudo-primary) output. Approaches based on fault models scale with nowadays circuit complexities, since higher abstraction levels require less computational effort.

The quality of a test set can be assessed according to some metrics, with *fault coverage* being one of the most often used. Fault coverage measures the percentage of targeted faults which are detected by a certain test set. The fault detection estimation is performed by means of fault simulation: given a fault model, all possible faults in the circuit are simulated and the manufacturer can assess the quality of the test patterns.

3.1.1 Logic simulation

The *simulation* of a circuit is the evaluation of a model that represents the circuit. Simulation is required for many of the test and diagnosis flow tasks. Fault dropping,

test pattern fault coverage validation and diagnosis are some of the tasks that depend on simulation. The methods presented in this thesis make use of logic simulation results. The accuracy and performance of logic simulation are of great importance for the precision and overhead of the method, and are thus discussed in this section.

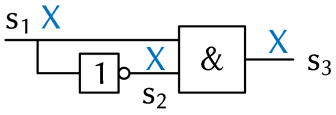
The so-called simulation level indicates the detail level of the model. Some of the most used circuit simulation levels are register-transfer level (RTL), logic, switch and layout [Wang06] [Bushn13]. RTL is the most abstract structural description of a design. Registers in RTL correspond to the sequential elements in the design, while the logic implemented by the system is represented as a Boolean function instead of a netlist. Logic level represents the circuit as a gate-level netlist, that is, as cells with interconnections. Switch level is a step below gate level, and represents the circuit as interconnected transistors, which behave as ideal switches. Finally, at the layout level, the representation includes all geometrical details. Lower abstraction level simulation renders more accurate results. However, precision also increases the complexity of model evaluation, making layout simulation often unaffordable for very large circuits. Gate level is a widely spread representation level, since much has been achieved to optimize simulation. Also, fault models have been developed for logic representation.

Gate level simulation of a circuit requires firstly a representation of the structure: a netlist of interconnected cells. Often, the circuit is *levelized*. Levelization is the process of assigning a level to each cell, so that a cell's level is always higher than the level of its predecessors. A relevant property for simulation of levelized circuits is that all gates in one level are independent of one another. This can be exploited in simulators.

In logic simulation, the voltages in the lines of the circuit are abstracted as logic 0 or 1. A voltage under the threshold voltage is interpreted as 0, and a value over the threshold is interpreted as a logic 1. The simulation in which only these two values are possible is designated two-valued logic simulation. It is, however, not always possible or desirable to give a 0 or 1 value to all signals. Storage elements, such as flip-flops, which have not been initialized have an undefined value. Also input bits which are not specified, or intermediate voltage values. Three valued simulation includes the value X, which represents unknown or undefined values.

Three-valued simulation often introduces a degree of pessimism. Figure 3.1 depicts a circuit in which unknown values introduce pessimism. Signal s_1 feeds both the AND cell and the inverter. As a result, s_2 is the inverted form of s_1, that is $s_2 = \neg s_1$. The output signal, s_3, is the AND logic function of s_1 and s_2, $s_3 = s_2 \wedge s_1$. When an X is

assigned to s_1, the simulator inverts the value. The inverse value of X is typically also undefined, hence assigning a value of X also to s_2. When propagated forward to the AND cell, the same problem arises: the AND function of two undefined values is also undefined. However, reconvergence may cause the value of the signal to be defined. In fact, the circuit calculates the output of the function $s_3 = s_2 \wedge s_1$. Substituting s_2 for $\neg s_1$, the function can be rewritten as $s_3 = \neg s_1 \wedge s_1$, that is, 0. The simulator is pessimistically assigning unknown values, and propagating them in the subsequent logic.

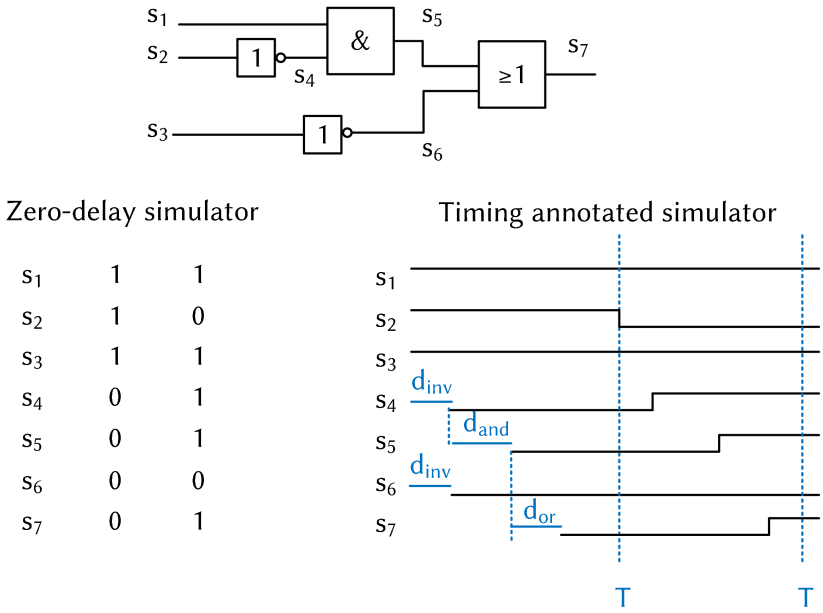


▲ **Figure 3.1** — Pessimism in three-valued simulation

Logic simulation has different granularity possibilities for timing annotation. Zero-delay logic simulation evaluates the circuit values from input to output without timing considerations. It assumes that every cell immediately propagates the new value to the output without a delay. Timing-annotated logic simulators, on the other hand, consider that every cell has a delay. The unit delay model, for instance, considers that all cells have a delay of one time unit. According to this scheme, the longest topological path (in the sense of the path with the highest number of cells) is also the longest in terms of timing. This does not, however, reflect the properties of manufactured designs, in which the delays among cells and even pin-to-pin delays withing a cell vary. Further extensions of timing-annotated simulation allow the annotation of a different delay for each cell and line.

Figure 3.2 illustrates the difference between a zero-delay and a timing-annotated logic simulator. The simulated circuit is represented at the top of the figure. Two input patterns are applied, and the simulation is presented in the lower half of the figure. On the left side, the zero-delay logic simulator result is presented. Since the simulator considers there is no delay in the calculation of a cell output, it is as if the signals' values propagated without delay and stayed stable during the whole clock cycle. The right side, however, shows a different picture. To begin with, signals in deeper levels (i.e. closer to the outputs) have not been initialized. The figure shows them as undefined by not assigning them a value. They are initialized after the corresponding delay, when the values of the previous signals propagate through the gates. For instance, signal s_4,

which is driven by an inverter fed by s_2, only gets assigned a value after a delay d_{inv}, which represents the delay of an inverter gate. In the same way, signal s_5 only gets assigned a value when the preceding signals have been assigned and after the and gate delay, d_{and}. The vertical dashed lines represent the clock cycle. The outputs propagate in time, and so at multiples of T, the values match those of the zero-delay simulator. In the figure, the values of the gate delays have been left as variables. Note that this is representative for any delay model assumed in the simulation, and is easy extensible to different delays for rising and falling transitions.



▲ **Figure 3.2** — Zero-delay vs timing annotated simulator

From the implementation point of view, a simulator can be plain or event-based. Event-based simulators manage a data structure that orders the upcoming events, and only evaluates those cells for which a change at the output has been observed. Plain simulators, on the other hand, evaluate every cell for every pattern. A plain simulator can take advantage of a parallel pattern evaluation: instead of evaluating one bit at a time, the complete word length is used to evaluate more than one pattern in parallel. Although intuitively event-based simulators are more efficient in terms of computational effort, the regularity of plain simulators makes them more suitable to be mapped onto high-throughput architectures such as graphics processing units [Schne17]. Simulation algorithms on parallel architectures parallelize pattern simulation. They also take

advantage of levelization, since all gates in the same level are independent and can be simulated in parallel. This reduces the cost of logic simulation dramatically, hence making it affordable even with nowadays circuit complexities.

Fault simulation

Simulation with fault injection is the base of many efficiency assessment techniques for test and diagnosis. With growing complexities and number of transistors, evaluating precise models is often unaffordable for today's circuits. Logic simulation is faster by several orders of magnitude than simulation at lower abstraction levels, and faults can be injected with any of the fault models introduced in section 2.2.

Like logic fault-free simulation, logic fault simulation can be performed using the values 0 and 1. These values serve as the logic interpretation of the voltage values of the lines in the circuit: over a given threshold voltage the value is interpreted as a 1, and below it as a 0. The inclusion of faults slightly modifies the structure of the circuit, but this resulting new model can also be evaluated with simulation.

The simulation of faults which only involve one component, such as slow gates, requires the simplest strategy. In [Waicu87], the authors describe the slow-to-rise and slow-to-fall gate fault models, generally referred to as transition faults. Thanks to a ranking of the gates, they are able to evaluate the faulty instance of the circuit in just one pass. Although levels are not formally introduced, the proposed ranking is similar to levelization in that a gate can only be evaluated if its predecessors have already been evaluated, too. The authors point out that both patterns, the initialization and transition propagation patterns, are necessary to simulate a transition fault. To optimize simulation time, the authors propose a fault collapsing strategy, which does not allow such a high compression as for stuck-at faults, since transition faults have less equivalences. They also take advantage of parallelism at the pattern level, by using the complete word length to simulate more than one pattern at a time. This extension was proposed to enhance stuck-at simulators for transition faults. When integrated in a zero-delay logic simulator, this technique only allows the representation of faults that cause a large delay.

Faults which involve more than one signal pose a challenge for simulators, because dependencies appear that were not present in the original structure. Bridge fault simulation, for instance the approach presented in [Chess98b], must distinguish between the two involved lines. For a dominant bridge between an aggressor and a victim line,

the situation of both in the levelized graph plays a relevant role in the simulation strategy. The authors refer to the line involved in the bridge that is closer to the inputs as *back wire*, while the line closer to the outputs is the *front wire*. For a dominant bridge, if the aggressor is the back wire, the simulation can be performed in one pass. When the simulation reaches the victim line, it will simply read the value of the aggressor and manipulate the victim line signal accordingly. However, in the opposite case, the simulator must evaluate in the first pass until the level of the aggressor, and then go back to the victim level, manipulate the signal and propagate it to the output. One additional consideration must be made in this case: if the aggressor belongs to the output cone of the victim, a feedback loop is established. Feedbacks in bridges exceed the capability of most simulators, because they may cause oscillating behavior and their evaluation is hence imprecise in high-level simulators. If the bridge model is a byzantine bridge, then the restriction holds for both lines: if either of them belongs to the output cone of the other, the faults may only be inaccurately simulated. The strategy in [Chess98b] can be extrapolated to any fault model involving more than one line if the manipulation of the signal values is adjusted to the desired fault model.

Just like in logic simulation, the timing granularity of the simulator has a big impact on the optimism or pessimism introduced in the simulation. Zero-delay logic simulation allows only a coarse representation of faults related to timing. It can only represent those faults bigger than the delay of the circuit [Schne17], which leads to mismatches between the expected and the observed outputs.

Simulators able to evaluate a timing annotated circuit [Holst15] also allow fault injection by manipulating waveforms instead of one single value for the complete cycle. Timing-annotated fault simulation can identify glitches and hazards, which may potentially activate a fault that would be overlooked by a zero-delay logic simulator. As a key component of quality assessment procedures, timing annotated simulation presents a huge advantage due to its increased accuracy with respect to simpler logic simulators.

The lack of accuracy of zero delay logic simulators, and particularly if faults need to be injected, limits the precision of fault coverage estimations or of logic diagnosis techniques. Another significant advantage of timing annotated simulators is that, with a simple manipulation at the preprocessing stage, factors such as variations can also be represented.

3.1.2 Test generation

The purpose of test generation is twofold. On the one hand, to ensure high product quality, the test set must exercise all parts of the design in order to uncover the highest possible number of defects. On the other, test time increases time to market and production costs. Thus, the generated test set must be as short as possible, i.e., contain few patterns. The properties of the generated test set are determining for the quality of diagnosis and impact also the methods presented in this thesis. Hence, this section describes the different possibilities and highlights their advantages and disadvantages.

Generating tests for all possible defects in a circuit is not feasible for nowadays design complexities. Automated test pattern generation (ATPG) approaches often tackle fault models. The universe of all candidates of the fault model is generated and a test set is derived which detects as many as possible. The ratio of detected faults is referred to as *fault coverage* [Bushn13], and is a quality metric for a given test set. Fault coverage can be validated via simulation: all faults in the set are injected and the test set is provided as stimuli. An ATPG algorithm will try to generate at least one pattern to detect every fault.

Due to increasing circuit complexity, the size of the fault universe can become too large. However, some faults are indistinguishable or *equivalent* [McClu71], i.e., they cause the same behavior in the circuit. A pattern which detects one is guaranteed to detect also the other. An example of structurally equivalent faults are the stuck-at faults at the input and output of an inverter. The stuck-at-0 fault at the input is equivalent to having a stuck-at-1 at the output. An ATPG algorithm need not consider both faults in the initial fault universe, so ATPG algorithms first perform fault collapsing, that is, they remove structurally equivalent faults, leaving only one representative of each equivalent class.

Even after fault collapsing, generating one pattern specifically for each fault would not only take long, but also result in undesired long tests. Test generation algorithms hence usually perform fault dropping: once a pattern is generated, all remaining faults are simulated. If any of them are detected by the pattern, then they are dropped from the list, hence saving test time generation.

The line or cell affected by a fault has an *input* and an *output cone*. The input cone is the set of cells and lines whose value propagates along a path to the affected component. Conversely, the set of cells and interconnects to which the value of the affected

component propagates is the output cone. A pattern can detect a fault if it fulfills two conditions: the fault is a) activated and b) propagated to an observable output. The activation implies assigning the component affected by the fault the opposite value than expected, and needs to be complemented by the justification of the line: setting the correct values at the input so that the desired activation values will get propagated along the input cone. Propagation requires transmitting the value through the output cone which causes the test output to differ from the fault-free response. It is sufficient if one of the outputs differs: the fault is already detected.

The complexity of nowadays circuits calls for automated methods to generate high-quality test patterns. Designs have generally a combinational and a sequential part. The sequential part defines the state of the circuit. Sequential ATPG [Bushn13] approaches exist for test generation which consider the sequential behavior of the circuit. Unfortunately, sequential test generation may be a highly complex process depending on the amount of sequential elements of the design and their sequential depth. In particular, the justification of the line and the propagation to an observable output turn test generation into a non-trivial process. Justification implies generating a sequence of patterns which set the system into the correct state to activate the fault. Propagation requires also a sequence of patterns to make the difference observable at an output of the design. Sequential ATPG often results in long test sequences, contradicting the goal of minimizing test costs. For this reason, often instead of deploying sequential ATPG, special test infrastructure is introduced (see section 3.2.2) and combinational ATPG is used instead.

Test generation for combinational circuits can be tackled following so called structural test techniques. The D-algorithm [Roth66] uses a five-valued representation. Additionally to 0 and 1, the values X (*don't care*), D (fault-free value is 1, faulty is 0) and $\neg D$ (fault-free value is 0, faulty value is 1) are included. The algorithm first activates the fault. For a stuck-at 0, for instance, this means setting the line to D. Then, the algorithm propagates the value to the output. Finally it justifies the lines so that the propagation and activation are possible. The algorithm terminates when all lines have been justified. The D-Algorithm works in one single pass for reconvergence-free circuits. However, in a circuit with reconvergence, however, conflicting assignments may be required and the algorithm needs to backtrack. The path-oriented Decision Making (PODEM) [Goel81] algorithm is an improvement over the D-Algorithm which restricts the decisions to the inputs, hence reducing the search space. It can be further reduced

by introducing the concept of headlines, which mark reconvergence fan-ins. Headlines are an improvement of FANout-oriented test pattern generation (FAN) [Fujiw83] over PODEM. The identification of headlines helps the algorithm take better decisions to justify the lines, optimizing the test generation time.

ATPG may also be formulated as a satisfiability problem [Larra92] [Sauer12]. The first step is to build a model of a Miter structure. A Miter structure consists of an instance of the fault-free (or reference) circuit, a faulty instance and a comparator which detects differences between the responses of both circuits. The Tseitin transformation allows to model the Miter structure with a set of clauses in CNF form. This representation can be then supplied to a Boolean satisfiability solver, which finds an assignment to detect the faults. The method is explained in detail for stuck-at faults in [Larra92], and can be extended for other fault models. Even timing faults, which require a pattern pair for detection (the first for initialization and the second for activation) can be targeted with such as approach. SAT-based ATPG has been deployed for path delay [Egger07] or small delay faults [Sauer12].

SAT-based ATPG can be performed to generate a manufacturing test pattern set if no further constraints are allowed. Despite manufactuing test's efficiency to detect faults, its usefulness when it comes to identifying the underlying fault is limited. For this reason, different kinds of tests with enhanced diagnostic properties must also be generated. SAT-based offers the flexibility to encode different conditions as clauses and can be hence deployed not only for detection, but also for diagnostic test pattern generation (cf. 3.1.3).

The generation of test data is a complex task, since test deals with two major factors responsible for uncertainty. On the one hand, the physical parameters of a defect cannot be foreseen. It is unaffordable to model all possible defects with all possible physical parameter combinations for a circuit. Inductive fault analysis [Shen85] is a technique which takes into account the technology and gathers information from previous production to create a list of faults likely to appear in the circuit. With this information at cell level, it is possible to create test patterns that target intragate defects [Hapke14]. On the other hand, variations include a certain degree of uncertainty with respect to the timing behavior of the circuit. To generate more efficient test patterns in these these circumstances, variation-aware test pattern generators have also been developed [Sauer14].

3.1.3 Logic diagnosis

Diagnosis has the objective of identifying the location and nature of underlying defects in order to improve yield or performance [Jha03]. The first step of diagnosis is the localization of the fault. Usually, this first step is based on a logic representation of the circuit, from which the defective component (cell or interconnect) is identified. Logic diagnosis receives the result from test as input, that is, the passed and failed patterns. It also requires the netlist of the design, which it uses to simulate the patterns. Fault localization or logic diagnosis is performed after test and its result is used to guide the physical failure analysis procedure.

Logic diagnosis can be performed following a cause-effect or an effect-cause approach. Cause-effect approaches [Wunde07] assume a fault model and generate the universe of all possible candidates based on a fault model. For the stuck-at fault model, for instance, the universe would include the stuck-at 0 and at 1 for every line in the design. The expected output for each candidate can be computed with simulation. However, the simulation of the complete test set for all candidates may take too long. To save the computation time for every diagnosis, a *dictionary* can be generated in which the expected faulty responses are stored along with the candidates which can cause them. Due to circuit complexity, however, the number of patterns and faults grows very large and the dictionary may explode. The most significant effort in these techniques has been invested in reducing the size of the dictionary.

One example of cause-effect technique is presented in [Yamaz13], where resistive open faults are diagnosed. The method first performs path tracing, and discards the candidates which do not propagate to the affected output pins. Then, diagnosis is performed using the small delay fault model as reference. Although effective, this approach has the unrealistic prerequisite that the tool knows the defect class beforehand. Additionally, the lack of fault simulation precision due to unpredictable electrical parameters may cause such approaches to lose precision, since they look for candidates whose simulated responses match the observed test output.

Effect-cause approaches, on the other hand, reason about the structure of the circuit. The simplest approaches tend to consider the presence of a single fault, also known as single fix condition [Wunde07]. This translates into a search for a single line which explains all failing patterns. The single fix condition does not hold for some defects, such as bridges, because it is possible to have more than one culprit in the circuit. However, based on a dictionary for stuck-at faults, that is, a set of precomputed responses if a

stuck-at fault is present, it is possible to diagnose more complex fault models. The composite model for bridges is described in [Millm90]. An output pattern computed in the dictionary will not always match the observed response. However, a composition of dictionary entries of the stuck-at faults at the involved signals allows a more accurate prediction and enhances the diagnosis of more complex fault models.

In [Venka01], the authors propose a diagnosis method for sequential circuits. As a first step, they extract the input cone of each output, that is, the set of cells in the device through which values propagate from the inputs to the considered output. Whether the output is a primary output or scan element is irrelevant, they are both considered. This reduces the suspect lines to those contained in the topological input cones of the points where a faulty value was observed. The candidates contained in the input cones of scan elements are eliminated as suspects, since shifting in a scan pattern makes the value contained in the flip flop independent of the logic which feeds it. The cones can be further reduced deploying path tracing. Since the assumption is that one single line caused all failing patterns, only the elements contained in the intersection of all the input cones are considered. Composite faults as presented in [Millm90] are also constructed. The authors perform fault simulation of all candidates, and rank them according to three types of evidence: intersections, mispredictions and nonpredictions. An intersection is an error observed both in the fault simulator and in the tested design. A misprediction is a pattern (or slice thereof) predicted to fail by the fault simulator but for which the tested design renders the correct output. A nonprediction is a pattern (or slice) expected to be fault-free, that is, the fault simulator propagated no fault to the outputs, but for which an incorrect value was observed in the tested circuit. Based on the coincidences between the response obtained from fault simulation and test output, the candidates are ranked. Candidates with full pattern intersection, that is, perfect match with the observed behavior, are ranked first. The second criteria is the number of partial intersections: a larger number of pins matching the observed response is ranked higher. Finally, a low number of mispredictions ranks higher.

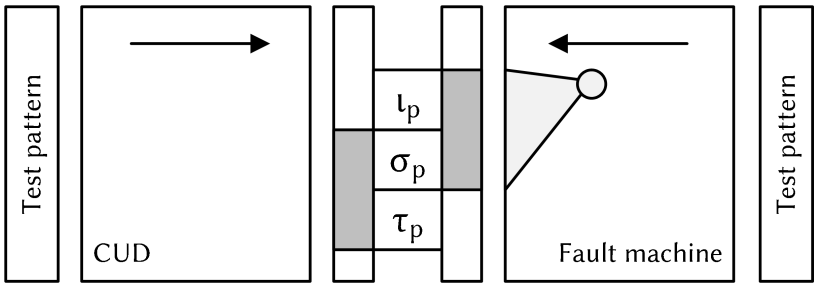
The fix condition assumption must be relaxed for more complex fault models, particularly those for which more than one culprit is possible. The single location at a time (SLAT) assumption determines for each pattern a location that explains the observed behavior. Then, a composition of those faults can be used to derive more complex fault models. For instance, for a bridge in which signals s_1 and s_2 are connected, if s_1 is flipped for a certain pattern p_i and s_2 is flipped for another pattern p_k, a SLAT based

diagnosis will come up with both faults. The limitation is, however, that only patterns for which a perfect match is found are considered. Given the lack of predictability of the defective region and its physical characteristics, assuming a perfect match between the observed output and the one calculated with the stuck-at as culprit limits the approach. In [Chen09], bridge diagnosis is performed based on the SLAT assumption combined with layout information. A list of fault candidates is generated for the SLAT patterns, and a cover which takes physical information into account is generated to diagnose the underlying bridge fault.

In the approach presented in [Desin06], $l_v^{T_k}$ denotes a line l, which, if at faulty value v, explains the subset of patterns T_k. This model receives the name of *TSL fault* (temporary stuck line fault). The method constructs a so-called forest of TSL faults. The forest is formed by trees whose nodes are the TSL faults extracted from the SLAT patterns. Two faults $l1_{v1}^{T_{k1}}$ and $l2_{v2}^{T_{k2}}$ are ordered when one of the sets of explained faulty patterns is contained in the other. For instance, $T_{k2} \subset T_{k1}$ implies that the first fault is better ranked than the second. There may exist disjoint trees if the sets of explained patterns for each fault cannot be ordered. Passing patterns are then used to prune the tree: any TSL fault for which the line and its neighborhood are in the same state as in a faulty pattern is pruned from the tree. The localization is then a covering problem, in which a set of candidates must be found which, together, explain all observed faulty patterns. The authors combine it with additional diagnostic test generation to improve the precision and accuracy of the diagnostic result.

Other approaches, such as [Holst09], further relax the SLAT assumption. Instead of requiring a perfect match, a metric is established to quantify to what degree a location explains the observed faulty behavior. In the approach, the fault simulator or *fault machine* flips the candidate line. The output response is compared to the observed behavior, and the candidate locations are ranked according to the correlation between the expected and the observed responses. The correlation for a certain test pattern p is measured based on a tuple including four different metrics: $(\delta_{\sigma_p}, \delta_{\iota_p}, \delta_{\tau_p}, \delta_{\gamma_p})$, where δ_{σ_p} is the number of pins which were faulty and were expected to exhibit a faulty value according to the fault machine, δ_{ι_p} is the number of pins which were expected to be faulty but rendered a correct value, and δ_{τ_p} those pins which were expected to be correct but had the wrong value during test. Finally, $\delta_{\gamma_p} = min\{\delta_{\sigma_p}, \delta_{\iota_p}\}$, which serves to measure if the fault fully explains the behavior. This value is different than 0 for faults such as delays, when only a subset of the outputs are affected. The metrics are illustrated

in figure 3.3. A tuple $(\sigma, \iota, \tau, \gamma_p) = (\sum_p \delta_{\sigma_p}, \sum_p \delta_{\iota_p}, \sum_p \delta_{\tau_p}, \sum_p \delta_{\gamma_p})$ summarizes the evidence for a certain candidate. The candidate lines are ranked so that the number of outputs which match the predicted output is maximized, while the deviation from the output predicted by the fault machine is minimized.



▲ **Figure 3.3** — Metrics for logic diagnosis [Holst09]

The method in [Holst09] uses both the passing and the failing patterns as information. Moreover, it integrates an adaptive approach which combines diagnosis with test pattern generation to enhance precision and resolution.

In [Wen04], the authors propose to include one symbol per line, and propagate them to the outputs by means of partial symbolic propagation. Depending on the quality of the match, a score is given to each fault for each failing pattern. Then, the *diagnosis table* is constructed with one row per pattern and one column per fault. The outcome of the algorithm is a multiplet: a set of faults which explains all observed failing patterns. The multiplet allows the approach to localize defects which influence more than one line.

The reported approaches make use of a comparison between the uncompressed expected output pattern and the observed one. This applies only in the case in which the ATE or other test equipment has full accessibility to the complete output response. However, many designs are equipped and tested with self-test infrastructure, and in many cases the output is compressed in the form of a signature. This reduces the required bandwidth and still guarantees fault detection. However, as a result, the diagnosis algorithm is provided with less information. This restriction applies also for online test, since non-instrusive execution often implies limited storage space, and test responses are also often compressed. Diagnosis approaches are required, in both contexts, to handle signatures instead of full patterns. To overcome this limitation, approaches such as [Cook11] have been presented. The test set is partitioned in windows, and for each window a signature is extracted. The authors propose building a system of linear equations where the

activation of a fault and its signature are on one side of the equation, and the observed response on the other. The solution of the system gives back the *deviation patterns* (which represent fault activations), which best explain the observed faulty signature. After identifying this, a similar ranking procedure to the one presented in [Holst09] is deployed. This method was even extended to cover more complex fault model in which more than one line are involved. To this end, in [Cook14] a pivot variable is added which allows the identification of more than one fault.

While many of these approaches report high precision diagnosis, they have only been validated with permanent faults. At logic level, and with new technologies introducing more complex activation mechanisms, it is not a realistic assumption. The real size of the devices or the defects in the circuit cannot be predicted beforehand, and introduces indeterminism in the behaviour of the faults. This poses a problem because most diagnosis approaches rank the candidates depending on the number of perfect matches, whose number decreases if an intermittent fault is present. Since most real defects will appear intermittent, diagnosis calls for robust approaches with respect to intermittent problems.

3.2 Manufacturing test

The goal of manufacturing test is to screen out as many defective parts as possible. Test patterns are applied to the device under test, and the responses compared to the reference output vectors. If they do not match, the circuit is deemed faulty.

Additional infrastructure is introduced in the designs to reduce the complexity of test and test-related tasks. The infrastructure comes at the cost of additional area overhead, but succeeds in reducing test generation complexity and increasing observability. As a consequence, fault coverage is also increased.

Faulty chips must be further analyzed to find out the cause of the faulty behavior. Such analysis is divided in two steps: fault localization and fault characterization. In the manufacturing flow, fault localization is performed by means of logic diagnosis. Its quality is, however, limited by the test set. This section presents the basic flow, infrastructure and diagnosis limitations of relevance for this work.

3.2.1 Test flow

Produced chips are affected by defects and variations. The manufacturer tries to avoid delivering defective parts, while highly performant parts can be sold more expensive than those which are defect-free but only perform in a narrower range of temperature or supply voltage conditions. For this reason, test has two main objectives [Segur04]. The first one is the identification of introduced manufacturing defects. The second is the assessment of the range of environmental conditions in which each chip can work.

Tests can be categorized as voltage- or current-based [Segur04]. In voltage-based test the ATE probes the output pins and interprets the output as a logic value. Voltage-based test is the main form of digital device testing, but because of CMOS technology properties, current-based testing is also performed. Defects in CMOS technologies that introduce intermediate values in the signals change the quiescent current of the device, but are often masked by subsequent gates and no out-of-range values are observed at the output. Therefore, so-called I_{DDQ} tests are used as parametric tests for CMOS technologies.

The first step is so-called characterization test [Bushn13]. Characterization is performed before a design is sent to mass production, and extracts the conditions in which the design works. Functional tests are applied to a statistically significant set of devices. The conditions under which test is performed are changed, conducting test under every combination of two or more environment variables. This information is used to correct design errors and check the final specifications of the design.

Manufacturing test is performed after production [Egger12] [Bushn13]. It assumes the correctness of the design, and its objective is to avoid shipping defective parts to the customers. To do so, manufacturing test tries to exercise the circuit in order to uncover the possible introduced defects. The automated test equipment (ATE) applies values to the primary inputs of the design and monitors the outcome of the primary outputs. The number of possible patterns depends exponentially on the number of input pins. Given that all manufactured chips must be tested, exhaustive test is not a feasible strategy for this step. Instead, a subset of input patterns is chosen to detect as many defects as possible. The subset must guarantee a certain fault coverage, and is generated with any of the techniques explained in section 3.1.2, and most often targets stuck-at and delay faults [Segur04]. The chips are sorted as passing or failing. The distinction is made by applying a test set at normal operating conditions and comparing the obtained output to the expected response: if any of the values returned by the chip differs from the

expected one, the chip is deemed faulty.

Production test is not able to distinguish among the different performance profiles of the passing circuits. To do so, circuits are put under stress in order to discard, mainly, those which contain unstable hardware structure which may lead to early life failures. Burn-in [Bushn13] is the most notable of these approaches. Some circuits fail due to an unlucky combination of variations in the device and interconnect parameters and unstable structures. These cases often lead to infant mortality: chips which fail after a short time of actual use. Infant mortality can be prevented by testing chips at higher temperatures, causing them to fail and avoiding their delivery. During burn-in, chips undergo a combination of production test under high temperature or over voltage conditions.

3.2.2 Test infrastructure

The naive approach to test digital devices consists of setting the values of the primary input to the ATPG generated input patterns. The responses are collected at the output of the device and compared with the expected output patterns. This approach achieves poor fault coverage values due to the lack of controllability and observability of the internal nodes [Bushn13]. In particular for circuits which contain sequential elements, that is, flip-flops and registers, limiting the control and observations to the primary inputs and outputs, respectively, severely limits the efficiency of the test set. Test generation for sequential circuits is extremely complex and achieves lower fault coverage levels than test generation for combinational circuits. Design-for-test (DfT) infrastructure is introduced in the designs to overcome these limitations. This section includes a revision of techniques and infrastructure relevant to the work presented in this thesis.

Scan chains [Eiche77] are one of the most popular DfT structures. A scan chain is a large shift register formed by sequential elements of the circuit. To introduce them in a design, its standard flip-flops are substituted with special cells which can work in two different modes: system and test. Scannable cells function as standard storage elements when in system mode. In other words, in system mode no difference is appreciated with respect to the normal behavior. However, if the test mode is activated they are configured as large shift registers, that is, the scan chains. The input of a scan chain is accessible from a primary input of the design, and the values are shifted out through one of the primary outputs.

Scan configurations can be full or partial. Full scan transforms all flip-flops in scannable elements. This results in full observability and controllability, but also in long shifting sequences to set and read out the state of the core. Partial scan only substitutes a subset of the flip-flops in the design, reducing the area overhead. By choosing the elements to be changed for their scannable counterparts so that dependency loops among flip-flops are broken [Kiefe00], the area overhead with respect to full scan is reduced, as is the complexity of the circuit in the context of test.

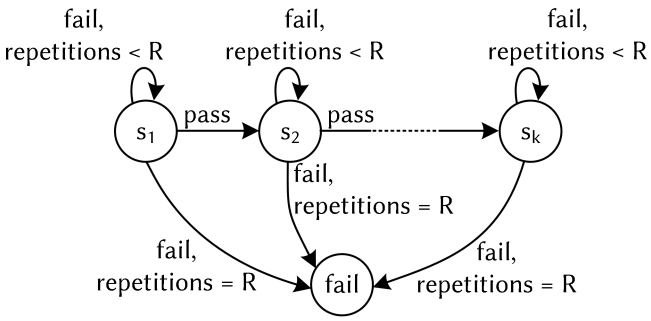
Although scan chains introduce some hardware overhead, the increased observability and controlability reduce test generation and application time. The sequential depth increases the complexity for test pattern generation algorithms [Bushn13], which must find an assignment that activates and propagates the fault to an observable output. Scan chains allow to shift in values and hence set the state of the memory elements. The cells included in the scan chains are considered pseudo-primary inputs/outputs, eliminating the need of long test sequences to justify the state or propagate fault effects to a primary output. The desired state is shifted in when test mode is enabled. Then, in system mode, a functional pattern is applied. Finally, the values of the new state are shifted out in test mode. To reduce time when multiple test patterns must be applied, the shift-in and -out phases of subsequent patterns are superposed. The test time is proportional to the length of the scan chain and to the number of test patterns. Including all scannable elements in one long scan chain would require time proportional to the number of sequential elements in the design to shift in one state. In nowadays designs, this is not practical. Instead of one scan chain, it is possible to consider the sequential elements into multiple shorter scan chains. The test time is so reduced, although at the cost of higher required bandwidth to transfer the data to all scan chains.

To overcome this problem, architectures for structural (BIST) [Wunde98] have been proposed. One of the most popular architectures for BIST is the STUMPS architecture [Barde82], in which the module under test is equipped with a test controller, a test pattern decompressor and a response compactor. The decompressor receives an encoded seed from the ATE and decompresses and distributes the the bits to the scan chains. It is often implemented as a linear-fase shift register (LFSR) which can be characterized by its so-called characteristic polynomial, based on which it can generate pseudo-random sets of patterns. The response compactor, on the other hand, transforms the output vector into a smaller vector which is transmitted back to the ATE.

The patterns provided to the circuit under test can be generated in a pseudo-random

fashion by feeding a seed to the LFSR. However, random patterns do not always render high fault coverage values. Test point insertion [Wang06] places control or observation points in signals with limited testability. Another solution to achieve acceptable fault coverage values is to combine random patterns with deterministic test pattern generation, referred to as *mixed-mode* BIST. To keep the hardware overhead low, it is possible to generate a set of seeds which, when unfolded, will generate a sequence of test patterns that cover the original deterministic test set [Liang02]. Both techniques can be integrated in scan-based BIST [Vrank02]. Provided the core can be set for as long as the test requires in test mode, scan-based BIST techniques can not only be used during manufacturing, but also allow their integration in system-level test [Qian09] [Vo06].

Although manufacturing test is performed under controlled environmental conditions, some noise may still affect the chip, causing it to return an incorrect pattern despite containing no defect. A fail in test caused by noise is a sign of sensitivity to noise, but the chip can still be integrated in robust designs which can tolerate some errors without degradation. Discarding healthy chips which fail test because of noise leads to unnecessary yield losses. Thus, robust circuits can be tested following a rollback strategy. In [Amgal08], the authors propose to divide the test in sessions. Then, a threshold R of repetitions is set. If the device under test delivers a faulty response for a test session, then the session is repeated. This may go on until R repetitions are reached. If this happens, the device is discarded. On the contrary, if the chip delivers the correct response before R repetitions are conducted, then it is considered fault-free and need not be discarded. Figure 3.4 shows the signature rollback test flow for a test of k test sessions.



▲ **Figure 3.4** — Rollback test flow

With increasing complexity of SoCs, the number of scan chains and control registers

for test structures grows, and debug and calibration infrastructure was also integrated in the system. Making all the infrastructure accessible with affordable complexity called for a more flexible approach than merely configuring the segments of interest in one or more large shift registers. Reconfigurable scan networks [Reari05] (RSNs) emerged as the solution to this problem. An RSN is a structure formed by scannable segments, connected in a network-like structure. The input of the network is connected to a primary input, and the output to a primary output. An *active scan path* is formed between the input and the output. The active scan path can be configured dynamically, by writing specific values in some of the scannable elements, known as configuration registers. The rest of the scannable elements are called data registers, and read or write data to the instruments in the system. RSNs allow the reconfiguration of the scan path by shifting in different sequences to the configuration registers. Hence, the complexity of the accesses is held within reasonable limits, enhancing the use of structural test strategies.

3.2.3 Test and diagnosis

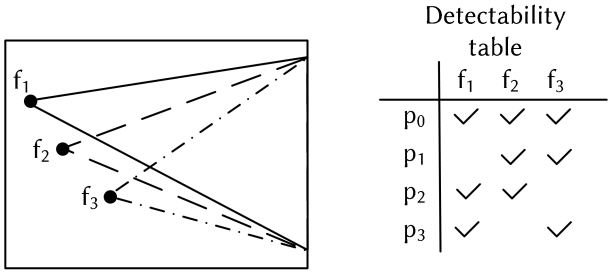
In the first stages diagnosis is deployed to find design problems, while during volume diagnosis the goal is to identify systematic defects as soon as possible. The fault identification goal has very different requirements from those of test, and a whole set of challenges appear to correctly identify the underlying cause of the erroneous behavior.

The results of test are provided as input to the diagnosis algorithm, along with a model of the circuit. Based on it, as explained in section 3.1.3, the diagnosis algorithm tries to find a location in the netlist that explains the faults. The resolution of diagnosis may be strongly limited by the test set from which the information is extracted [Holst09] [Zhang10a].

Since test must be executed for every manufactured chip, one of the goals of test is to limit the costs. Costs include test time, which is directly dependent on the length of the test pattern set. The other goal of test is, naturally, achieving high fault detection values, that is, fault coverage. However, it is not necessary to have different test patterns for each fault, and it is actually beneficial for the objective of test time minimization if the test patterns for one fault also uncover other faults.

This collides with the needs of diagnosis. Localization techniques would actually benefit of disjoint failing behaviors to identify every fault. To illustrate this problem, consider

the circuit depicted in figure 3.5. The circuit contains three fault locations, f_1, f_2, f_3. Their output cones include the same set of primary outputs. On the right of the circuit, the detectability table shows which fault patterns detect what faults.



▲ **Figure 3.5** — Circuit and detectability table

For test, the ideal pattern combination only includes p_0, because it is able to detect all faults. However, this information is insufficient for logic diagnosis, as it does not allow to distinguish between different faults. Without further information, the best result logic diagnosis can give back is a list that contains all three faults. On the other hand, if the pattern set includes any pair of patterns (p_1, p_2), (p_2, p_3) or (p_1, p_3), all faults are still detected and it is now possible to draw a conclusion about the underlying fault location. Because of this difference, the information generated in test may not always suffice for diagnosis. To overcome this limitation, many diagnosis techniques include an adaptive approach [Holst09], which translates into on-the-fly diagnosis pattern generation. In the example, this would mean that after observing the results for p_0, which was the pattern included in the test set, the diagnosis tool generates another pattern, for instance p_1. If the test is passed, then the diagnosis tool can conclude the fault location is f_1, while if it fails it can remove f_1 from the candidate list. Either way, diagnosis resolution is improved so and the tool can keep generating new test patterns until an acceptable resolution is achieved. Other approaches include specific diagnostic test pattern generation [Flenk15] [Zhang10a]. However, diagnostic pattern generation is extremely costly. Diagnostic test pattern generation may not always be applicable for high volume, since it increases test time. On-the-fly generation, on the other hand, targets specific devices under diagnosis, and may also become costly and not applicable for large numbers of devices under diagnosis. Even worse, for equivalent faults there is no pattern that can distinguish them, which poses an unresolved challenge for logic diagnosis. The quality of diagnosis is limited by the topology of the circuit,

the characteristics and outcome of test, and, if applicable, the quality of the diagnostic test patterns.

3.3 Online test and diagnosis

Passing manufacturing test does not ensure quality throughout the complete lifetime of the system. Some faults, particularly those with rare activation conditions, may not be detected during manufacturing test. These test escapes may pose a danger during the lifetime of the chip. On top of this, hardware structures can be affected by noise or aging mechanisms, some of which cause a no-trouble-found problem [LiVol11]. Such problems are not always repeatable in the lab, so their detection and gathering information for later diagnosis must be performed when the fault is activated, in other words, online. The application of online tests is crucial because when the system is functioning in its intended environment, the external conditions are not under control. Some defects can only then be detected, and when the physical structure begins to deteriorate due to aging, online tests can detect the problem. For this reason, the research community has developed a wide range of online test strategies, which allow for the system to be tested in the field and sometimes even without interrupting the functional tasks running.

Online test presents some additional problems to those of manufacturing test. This section highlights the online test properties which are relevant for this work, particularly those which will pose a challenge for the techniques presented in this thesis.

3.3.1 Concurrent and non-concurrent structural test

Online test aims to detect underlying faults before they have catastrophic consequences, i.e., in time to allow for graceful degradation or repair. Functional tests can be deployed to this end, however, they rarely exercise the design sufficiently. As a consequence, structural online test techniques and infrastructure have been developed to detect the faults with a certain *latency*. Latency is the maximum time slot allowed from the occurrence of the fault until its detection.

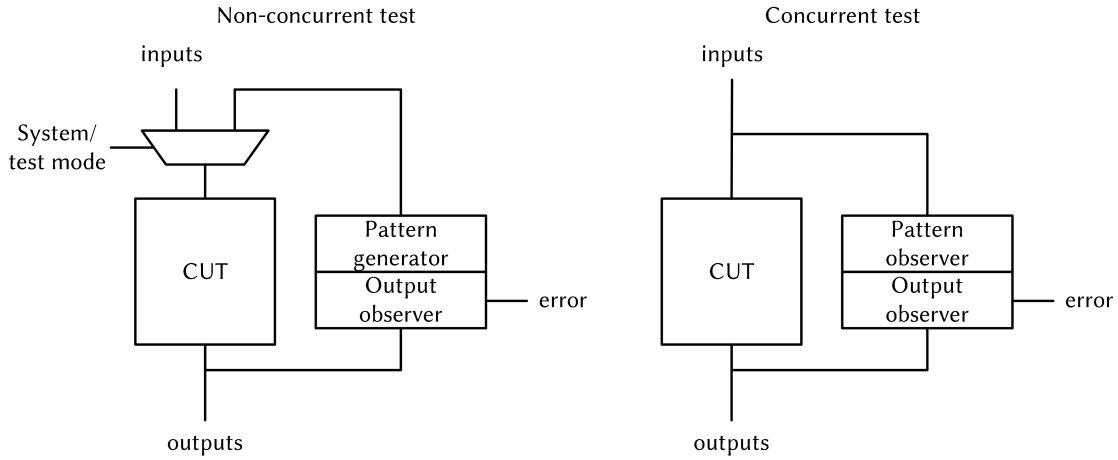
Online test approaches can be categorized as intrusive or not intrusive [Touba97]. Intrusive techniques modify the target module and integrate the testing infrastructure

in it. Non-intrusive techniques, on the other hand, assume the module under test is already optimized and synthesized and may not be modified.

Finally, online test may be performed concurrent or non-concurrently [Kocht10]. The concurrency refers to the test and the system functionality. Non-concurrent test includes those techniques which require an interruption of the functional mode, generally because they provide the testing input patterns and must interrupt the functional mode. BIST is an example of such techniques. Not only must the system enter the special test mode in order to shift the values into the scan chains, but when switched back to the functional mode the state of the circuit is either unknown or must be initialized or restored. Non-concurrent test techniques can be applied online provided the core is idle sufficient time, which is not always feasible in systems with hard real-time constraints. More importantly, the core must be able to resume functional mode.

In real-time systems with tight time constraints, non-concurrent approaches can only be integrated after careful analysis, if possible at all. However, aging and dormant faults still need to be tackled. The term concurrent test englobes those techniques which do not disrupt the functionality of the system to perform fault detection, but can be deployed at the same time as system functionality. The main difference between concurrent and non-concurrent test architectures is the role of the additional hardware. While for non-concurrent test a data source is included, as well as a switch to start the test mode, the hardware in concurrent test architectures acts as a mere observer and checker of pattern-output pairs. Figure 3.6 illustrates the aforementioned main difference. On the left side of the picture, a non-concurrent tester is depicted. The circuit under test (CUT) has a multiplexer at the input which controls what inputs are provided: the system functional inputs of the test patterns generated by the tester. Given that they are exclusive conditions, to activate the test mode the functional mode must necessarily be disabled. On the contrary, for concurrent testers such as depicted on the right side of the figure, there is no activation of test mode. The tester simply observes the patterns and checks that the output values match the expected ones.

The trivial implementation of concurrent test is a duplication of the observed circuit with a comparator. This approach maximizes fault coverage and minimizes latency. However, this is only possible at the cost of hardware overhead. Not only the area of the circuit copy must be accounted for, but also that of the comparator, incurring in an overhead of over 100%. A duplication of the manufactured area, with not only the additional design effort but also validation and test increases the cost to a point in



▲ **Figure 3.6** — Concurrent vs non-concurrent test [Kocht10]

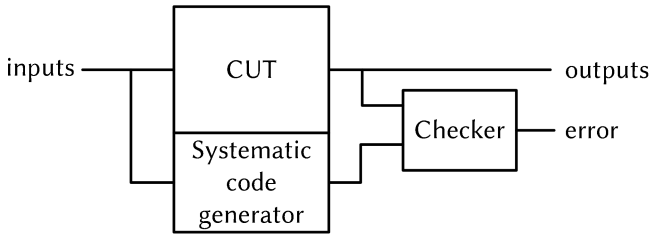
which it is impractical, especially in cost-sensitive markets.

To reduce the overhead, the so-called non-intrusive techniques try to reduce the area overhead of the predictor. Non-intrusive refers to the assumption that the targeted design may not be modified. One such technique is presented in [Sharm88]. The proposed method requires first an ATPG to target the faults in the system. The generated test patterns are then used to infer a logic function for each pin. In other words, the predictor is able to calculate the correct value for every pin whenever one of the test patterns is applied to the circuit. To avoid false alarms, additional logic is implemented to mask the mismatch signal if the applied pattern does not belong to the test set. Although it reduces the additional overhead, the improvement is only about 15% with respect to duplication. Further area reduction is achieved in [Drine03]. The authors relax the assumption of [Sharm88] that the width of the predictor output needs to match the width of the circuit output. Instead of predicting the value of all outputs for all patterns generated by the ATPG, only a subset of pins is observed for each output. This requires an addressing logic which chooses, for each pattern, which are the output pins of the circuit to be compared. Depending on the complexity of this mapping, the overhead gain of the reduced output width predictor can be lost by overly complex addressing logic. Hence, the algorithm selects the output pins so that the required mapping will be simple and the addressing logic is reduced. With this method, the authors claim to be able to reduce the area overhead of duplication by 40% to 50%.

However, there is another important parameter to be considered additionally to the fault

coverage. With the aforementioned methods, latency is not guaranteed and depends on the applied input patterns. Assuming a uniform probability distribution for all input patterns, for a test set T of size $|T|$, the probability that a test pattern will be applied to the input of the CUT of input width i_{width} is $|T|/2^{i_{width}}$. For most patterns, however, not all bits must have a specific value. For instance, if the output of a NAND-gate must be set to 1, it is enough to set one of the inputs to 0, while the other one can be left as an X. This partial specification of test pattern bits increases the probability that the test pattern will be provided, and can be taken advantage of to further optimize the hardware overhead of concurrent testers. In [Kocht10], a method is presented which limits the number of specified bits, extracts an input-output relation and synthesizes it in hardware. This module is the online tester. The authors report lower hardware overhead than other approaches, with the exception of circuits containing large XOR-trees, which require large numbers of specified bits to ensure fault detection.

Intrusive techniques assume the design can still be modified. That is, error detection is implemented as part of the design, and not as an additional observant module. These approaches are suitable for early design stages, where the detecting hardware can still be integrated in the module. The general scheme of an intrusive concurrent tester is depicted in figure 3.7. A *systematic code generator* is built in parallel to the circuit under test (CUT). It receives the same inputs as the CUT. Based on the inputs, the generator appends some bits to the functional output in order to make it fulfill a certain property. For instance, parity checkers [Koren07] append bits to make the parity of the final output always even or always odd. If a fault flips an odd number of bits, then it will be detected. Finally, the checker receives the functional output and the generated code and examines if the property holds.



▲ **Figure 3.7** — Intrusive concurrent testers [Touba97]

In [Touba97], a method is presented for the synthesis of multilevel circuits to provide them with concurrent fault detection. The method considers the overhead of both the

systematic code generator and the checker to choose an optimal configuration. The two extremes of parity checkers are the cases in which there is only one parity group, and when each output is included in a different parity group. The first approach requires less area overhead, but the portions of logic shared by the output bits may cause a fault to flip an even number of bits, hence masking the effect. As a consequence, the fault would go undetected. On the other hand, including one parity group per output, i.e., duplicating the output length, minimizes the probability of fault masking, but increases the hardware cost of the additional logic, that is, the systematic code generator and the checker. The authors in [Touba97] present an optimization of the number of parity groups in order to minimize the area overhead incurred by the concurrent fault detection mechanism. The algorithm tries to optimize a cost function which includes a penalization for added hardware resources: the number of literals saved by merging groups, the complexity reduction of the checker due to parity group merging, and the additional literals needed in the control logic to account for the dependencies between output pins.

The authors in [Vemu08] identify assertions which must always hold if the output is fault-free. These assertions have the form of implications, where some values of the inputs imply certain values at some of the outputs. The assertions are then synthesized and included in the module. The approach aims to keep the overhead low. To achieve this, instead of generating assertions to ensure 100% fault detection, the authors study the distribution of the input patterns from the traces of real programs instead of assuming a uniform distribution. The reason is that some of the bit combinations for the input are not valid functional inputs, and, as a result, some paths are more exercised than others. Also, this skewed distribution causes the faults to be propagated with uneven probability. The proposed method identifies the elements more vulnerable to faults and protects those. An assertion is generated for all patterns. The detected patterns by the assertion are extracted with fault simulation. Finally, the authors employ a greedy approach to extract a minimum number of assertions that cover the maximum possible number of faults.

Online fault detection techniques can identify incorrect outputs. To diagnose the device, however, the chip may either be sent back to the manufacturer and tested, or it can be diagnosed based on the information gathered online by the online tester. Diagnosing on the available information of online test presents the advantage of minimizing the probability of no-trouble-found problems. This is particularly relevant for defects caused

by wearout mechanisms at the beginning of degradation and for noise. Unfortunately, in real systems the storage space is limited, and registering all outputs for all applied patterns is too costly in terms of memory requirements and often also data transfer. Registering the failing patterns, however, is possible, as they are identified by the online checkers presented. Hence, it is sufficient to provide the system with a small memory, in which the writing is enabled by the error/assertion signal and where the input and output patterns for which an error was observed is stored. The information about failing patterns is certainly helpful for diagnosis.

3.3.2 Software-based self-test

Concurrent testing techniques require additional hardware which implements the redundant logic. BIST techniques require area overhead only for the test data source and sink, and the application of scan test may drastically increase power consumption due to the high switching activity caused by the shifting of values. Also, BIST cannot be applied concurrently to system functionality, since its application switches the tested module out of functional mode. To overcome these limitations in those contexts in which the aforementioned methods cannot be applied, the community developed software-based self-test (SBST).

Software-based self-test (SBST) [Psara10] makes use of the instruction set of a processor-like component to perform the test. It is a widely accepted technique for testing processors. An SBST test is a sequence of valid instructions. More specifically, it is a valid program, although carefully generated to uncover as many faults as possible.

SBST eliminates some of the limitations of structural testing techniques [Psara10]. Since the SBST program can be executed like any other task, SBST requires no hardware overhead with the exception of low cost test equipment to load the program onto on-chip memory. Also, some storage space for the program is required. The observable outputs in the case of SBST are memory locations or registers, and the result must also be stored for later analysis. SBST also minimizes overtesting, since it is executed in functional mode and only valid patterns are applied. Moreover, this allows SBST tests to be executed at-speed, which is not always the case with structural scan-based techniques. Finally, it does not require the shift of values along chains of scannable elements, which makes it appropriate to avoid the high power consumption caused by scan test.

The first approaches for SBST were based on a functional approach. In one of the first SBST approaches [Thatt80], the authors represent the functionality of a CISC processor as a graph in which the nodes are registers and the edges are annotated by the instruction which transmits information from one to the other. Then, they generate the test program based on functional fault models, choosing the operands of the instructions arbitrarily. They report a stuck-at coverage of over 90% and a 1K program for the 2000 cell processor used. Other functional approaches such as [Shen98] allow the user to specify which instructions are to be validated. Based on this, the method generates sequences of instructions which load data, execute instructions, and propagate the result to an observable output. The operands can be chosen manually or generated randomly. The authors report a stuck-at coverage of around 90% for the same processor as in [Thatt80]. Some newer approaches include functional test generation with genetic algorithms [Corno04]. Genetic algorithms generate a *population* which survives discrete steps of evolution called *epochs* depending on a *fitness metric*. In [Corno04] the population is a set of test programs, which in every epoch evolve by changing instructions or operands. Their survival probability increases the higher the RTL statement coverage is, which is used as fitness metric.

Purely functional SBST generation approaches have the disadvantage of achieving insufficient fault coverage values or of needing large programs to reach high fault coverage levels. For this reason, methods which target structural faults have also been developed. In [Chen03], the authors propose the use of constrained ATPG. The processor is divided in modules, for each of which ATPG is performed. The preceding and succeeding modules are modeled as virtual constraint circuits. The virtual constraints represent not only the valid instruction codes, but also the propagation constraints. The authors can in this way generate structural tests for each module which are still valid programs. Results report a fault coverage of functionally testable faults of almost 10% higher. The required time to execute the program is reduced to approximately 65% of the functional one.

Another possibility to generate structural SBST is to deploy pre-computed test sets. The approach of such techniques is to extract tests for the different modules. Then, a SAT-based ATPG justifies the inputs, that is, it finds the way to propagate the values to the input of the module. In [Linga07], in addition to this the unsatisfiable instances are extracted to propose improvements for the DfT infrastructure. RTL-based test generation achieves up to 85% fault coverage, reported in [Linga07].

SBST can also be deployed to test the peripherals of an SoC. In [Bolza07], the authors deploy evolutionary algorithms to automatically generate a test for different cores than the processor. The full automation of the process renders suboptimal fault coverage values, but guarantees high efficiency in terms of test generation time. In [Apost07], the authors generate a deterministic test for different cores. Although they report higher fault coverage values than [Bolza07], the manual generation effort is very high. In [Apost09], the authors combine both approaches by providing the evolutionary core with the manually generated constraints. They succeed in generating test sets with higher fault coverage values, close (and even better) than with the deterministic approach. At the same time, the test generation effort is kept low, exploiting the advantages of both techniques.

SBST has the additional advantage that it can be used in more than one test scenario. The advantages with respect to to some structural DfT techniques make it suitable not only for manufacturing testing. SBST test can be executed online if the output is propagated to an observable point (mainly memory). This could lead to significant hardware overhead, but the authors in [Shen98] propose a software method to compress the test results in signatures, reducing the impact of this disadvantage. The online execution of SBST is however subject to certain system requirements and properties. To be executed online, the program must make use of the idle slots of the components and not intrude with the functionality of the core. This generally requires short test programs which can be executed in limited time slots. SBST must also be generated according to the criticality of the tested core. Critical components have harder requirements, which translate into higher fault coverage requirements and shorter latency to uncover the fault. In [Pasch05], the authors propose a method which takes system properties, criticality of the cores and allowed latency into account to generate and schedule SBST accordingly.

3.3.3 Automotive in-system test

Automotive systems integrate an increasing number of electronic control units (ECUs). The failure of one of them can lead to catastrophic consequences if not dealt with in a timely manner. Hence, online tests and diagnostic tasks are crucial to ensure safety. In [Reima14] and [Abele14], a technique is proposed where the author of this thesis contributed to a non-intrusive integration of structural tests at system level. This section presents the approach.

The integration of online test strategies at system level is a non-trivial task. In general, the requirements to integrate test and diagnostic features in a system include the non-disturbance of system functionality. The non-interference of the test and diagnostic features refers not only to the functionality of the semiconductor devices, but also to the communication among them. In safety critical applications, such as the automotive, this is particularly important and involves tight timing constraints to perform test and diagnosis. Moreover, with the automotive domain being such a cost sensitive market, the overhead must be kept to a minimum while the integrated test and diagnostic capabilities are maximized.

In the automotive domain, the availability of the resources and safety requirements varies depending on the scenario. In particular, it is relevant if the component is functionally active in the required scenario. Other aspects such as power or time constraints or additional hardware overhead are equally important. To take advantage of this, four different scenarios have been identified:

- in the field: the most critical of the scenarios. The system is in full functional mode and safety constraints must be met.
- power-up/ power-down: the system is booting or shutting down, but is not safety critical. Timing constraints are relaxed in comparison to in the field, but must still reflect the limit from which the user will perceive long boot or shutdown times.
- workshop: in the workshop, the system is no longer in functional mode and only overhead is a limiting factor.
- partial networking is a technique introduced to save power in automotive systems [Yi15]. It offers the possibility of selectively communicating with one module over the CAN bus, hence allowing selectively turning on and off some of the ECUs.

Unlike functional tests, structural test methods guarantee high fault coverage values. Some structural self testing infrastructure is integrated in the design for manufacturing test. Hence, reusing this infrastructure incurs in little or no additional hardware overhead. Only careful analysis of the timing costs and consequent planning is needed to include the tests of the different modules in the overall system design.

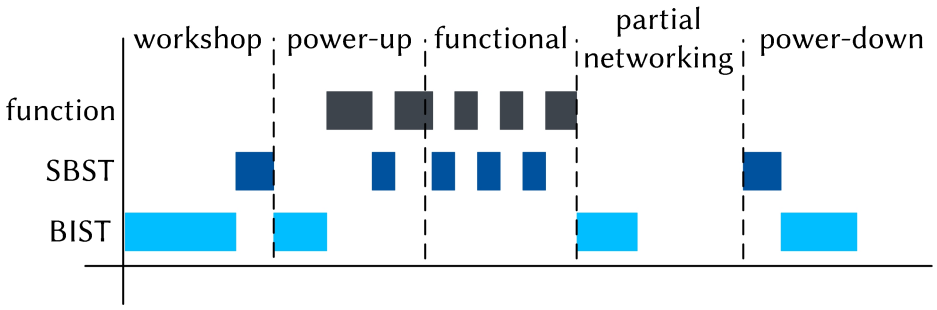
Scan design has become a general approach for all kinds of electronic devices, and very often a logic BIST controller is included in the infrastructure. STUMP-like architectures

are available for a number of electronic devices in a car, and their testing capabilities can be taken into consideration to integrate in the system. This kind of self test is, however, non-concurrent. Consequently, the test must be partitioned in sessions which can then be scheduled when the device is idle. Moreover, since the device is switched to test mode, it cannot be switched back to system mode if a task arrives that is more urgent. That means each test session will run uninterrupted. Also, because BIST erases the state of the device to conduct the test, it cannot be reused online without further consideration. Some scenarios of safety critical applications allow to deploy BIST without endangering the system or the environment. In the automotive scenario, BIST can be reused in the workshop. If test length allows to execute the task in a reasonable time, it can also be used in the startup and shutdown phases. Finally, those electronic components which support partial networking can also be tested while they are off - although this conflicts with the power saving goal.

SBST is also developed and sometimes deployed in manufacturing test. Especially the structural SBST approaches can be useful for online detection of latent or wearout induced defects. Since the test is a valid program, it can be executed as an additional task, provided the system has idle slots for it. The portions of SBST programs that must be executed atomically are kept extremely short. Hence, if a task with higher priority must be executed, the SBST context can be saved like for any other task, the high priority code executed, and then the SBST task resumed. Since SBST is integrated as an additional task or routine, only a change of context is required to restore the system state. Structural SBST requires overhead to store the test program and eventually the responses, and idle time to comply with the latency objectives for fault detection. SBST can be deployed in all scenarios but partial networking, in which the electronic component is off and cannot execute a task.

Such test sets can be integrated at system level. Figure 3.8 presents a scheme of integration of both approaches in all four contexts. During workshop revisions, the functional tasks of the system are not active. Hence, BIST and SBST can be executed in long slots, without need for interrupt. In startup and shutdown phases, however, the perception of delay by the user and the power consumption restrictions limit the application of long test and diagnostic sessions. Both approaches can be used, if their execution time is kept reasonably short.

To integrate diagnostic tasks in the system, a model is developed which represents the system functionality, the diagnostic tasks and the different objectives. The next



▲ **Figure 3.8** — In-system structural test integration [Reima14]

subsections introduce the model and characteristics of each component.

System modeling

An automotive system includes functional tasks in addition to the aforementioned diagnostic tasks. Communication resources complete the functional model of such a system. In addition, the hardware resources must be modeled, and a mapping between the functional tasks and the components must be found. To do so, the system is modeled as a graph based specification based on [Lukas09].

The specification is formed by an application and an architecture. The architecture describes the hardware resources, such as processing units, buses or memories. It is represented by a directed graph $G_R(R, E_R)$, where the vertices R represent the resources and the directed edges E_R represent the available communication interfaces between resources.

The application is the functional model of the system, and is represented by a bipartite directed graph $G_T(T, E_T)$. The functional tasks, represented by the edges T, include both process tasks P and communication tasks C. The directed edges connect one element of the first set with one of the second set, in either direction. This graph represents the data dependencies among tasks. A process task may receive data from more than one communication task. A communication task, however, only has one incoming edge, but may have more than one successor to represent broadcasting. Communication tasks are routed to a communication resource, and process tasks to processing resources.

An allocation $G_A(A, E_A)$ is a directed graph which represents a subset of the resources and dependencies of the resources graph G_R. An allocation represents hence a hardware

configuration of the system, i.e., a possible architecture. Tasks (both process and communication) must be bound to the allocated resources respecting the dependencies among them. For instance, a process task can only be mapped to a processing resource if its predecessors and successors are mapped to communication resources with an interface to the allocated processing resource. More than one task may be mapped to one resource.

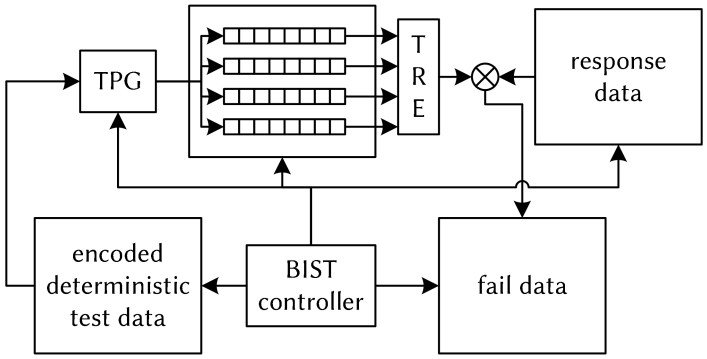
An implementation includes an allocation of the hardware infrastructure and a mapping of the tasks to the resources in the allocation.

Diagnostic tasks modeling

The structural set of diagnostic tasks D to be integrated is formed by SBST and scan-based BIST tasks. They are characterized by different parameters, and modeled as tasks with different properties.

The BIST infrastructure integrated in the system is depicted in figure 3.9. This infrastructure compresses the core under test (CUT), a test pattern generator and the response evaluator. The included BIST follows a mixed-mode scheme, which includes random patterns and some deterministic test data, specifically generated to target harder to detect faults. The encoded test patterns are stored on-chip, and reconstructed when the test is to be applied. The pattern response evaluator compresses the output patterns in a signature. However, in this particular architecture the goal is to exploit the diagnostic capabilities of the proposed techniques. Since it has been shown that it is possible to perform diagnosis from the signatures of faulty test sessions [Cook11] [Cook14], the STUMPS architecture has been adapted. To this end, the BIST controller partitions the test in sessions and one signature per session is generated. The generated signatures are compared with the corresponding expected signature, stored in the *response data* module. If there is a mismatch, the incorrect signature is stored in the *fail data* module.

A BIST execution on a unit is divided in two tasks: the BIST execution task itself, $b_r^T \in B \subset D$, and the data storage task, $b_r^D \in D$. BIST is characterized by three factors: the fault coverage $v(b_r^T)$, the runtime $t(b_r^T)$ and the memory size $m(b_r^D)$. The fault coverage is measured w.r.t the stuck-at fault model, and it is crucial to ensure the diagnostic capabilities of the system. The runtime includes not only the time to apply the test, but also the performance of the test access mechanism and also how long it takes to restore the state of the device to a functional state. Finally, the memory size is calculated based on the number of seeds to be stored for the deterministic patterns



▲ **Figure 3.9** — BIST diagnostic architecture [Abele14]

and the amount of reference signatures to be stored. The values of all characteristics depend on the proportion of deterministic and random test patterns, as well as on the decision to store the seeds and response data locally or transmit them via a functional bus.

An SBST task on a unit r is represented as $s_r \in S \subset D$. To execute an SBST task it is necessary to check if it is possible to schedule the routines during normal operation of the unit. To guarantee a fault detection latency, both the execution time required by the SBST routine and how often it can be executed are crucial. An SBST routine is characterized by the factors fault coverage $v(s_r)$, the execution time $t(s_r)$ and the size $m(s_r)$. The fault coverage is also measured with respect to the stuck-at fault model. The execution time takes into account the setting of the processor to a known state from which the SBST routine will be effective. This phase includes, for instance, register initialization or pipeline flushing. Performance features must typically be controlled to ensure a deterministic execution that guarantees fault detection. Finally, the memory size can easily be measured from the assembly code of the SBST routine.

The set of diagnostic tasks D must be integrated in the system without disrupting its execution of communication tasks.

Diagnostic tasks integration

An implementation is said to be feasible if an allocation and mapping are found so that all dependency conditions are fulfilled. In the original work [Lukas09], the problem is formulated with the constraint that each process task is bound exactly once. Additionally, the mapping of communication tasks and process tasks with dependencies must be performed so that the allocated resources also communicate with each other.

A communication task may only be bound to one allocated resource. The problem of finding a feasible implementation is encoded with binary variables, and solved with a hybrid approach involving integer linear program and evolutionary algorithms.

To integrate the diagnostic tasks, the encoding remains the same but now includes the diagnostic tasks along with the functional ones. The functional tasks, represented by the set F, and the diagnostic tasks, D, form the complete set of all tasks in the system, represented as $T = F \cup D$.

The constraints or the problem are relaxed to find a feasible implementation in this scenario. The priority is the non-disruption of the functional tasks. For this reason, the problem is formulated so that diagnostic tasks are allowed to be bound at most once instead of exactly once. That is, an implementation may leave some of the diagnostic tasks out of the functionality included in the system if that collides with the non-disruption objective, while functional tasks must still be mapped exactly once.

Although the mapping of diagnostic tasks is optional, if a diagnostic task is mapped, then its associated data storage task(s) must also be mapped. For instance, for every b_r^T mapped, the corresponding b_r^D must also be include in the implementation. Conversely, if b_r^T is not mapped, the mapping of b_r^D should be avoided, since it will only imply an unnecessary overhead in the system. The objectives of safety (that is, fault coverage) and costs (in terms of hardware resources and of time) are included in the optimization problem.

The method is able to identify a set of feasible implementations at different costs. The monetary costs are reduced when the test data is stored in the gateway and can be shared among different resources, although this increases the test time due to the transmission overhead. SBST implies in general a small overhead in terms of hardware costs, The hybrid approach succeeds in evaluating 25000 implementations in roughly one hour. Infeasible implementations were discarded. The results allow the designer to choose an implementation considering the safety and cost objectives additionally to the functional ones. The integration of structural test methods for in-system in-the-field test is hence possible at a reasonable cost and increases the safety properties of the system.

3.4 Test data

The available test information after test varies, as discussed in the previous sections, depending on system parameters and test scenario. The amount of available test data depends on the generated test set, the output response format, and the application scenario. Diagnosis may be severely limited by the amount of available information.

Diagnostic test pattern generation [Flenk15] [Zhang10a] enhances diagnosis, but it is costly and generally requires longer generation and application time. Manufacturing and online tests usually only aim at detection for economic reasons. They often provide limited diagnostic capabilities, but this information is available for all chips.

In manufacturing test [Bushn13], test techniques which deploy no compression observe all outputs for all patterns. The passing and failing patterns are identified, and after the test run all responses are available marked as pass or fail. Hence, the complete set of responses marked as correct or incorrect is available.

When compression is deployed, for instance in BIST [Wunde98], only signatures are returned by the device under test instead of the uncompressed version of the responses. The number of signatures may vary: to maximize the diagnostic capabilities of such a set, the complete test set may be divided into windows or sessions [Cook11], and even the number of times one test session is applied can vary [Amgal08]. If BIST is performed during manufacturing, the information about all passing and failing sessions are available. Consequently, after BIST the available test results are the signatures resulting from the test sessions.

If the test is applied online, then storage limitations apply. Typically, a real system cannot register all outputs to all patterns applied during its lifetime. Moreover, the communication slots and runtime limitations discussed in previous sections apply. The space is generally limited and only failing patterns are stored.

In the case of BIST, for instance, this translates into availability only of faulty sessions [Abele14]. If the applied patterns are known, however, such as in the case of deterministic BIST, then the passing sessions are also known, because they can be deducted from the total set and the failing signatures.

The same applies for SBST [Psara10], where the applied test set is known a priori since it is precomputed and loaded on the system in advance. Although only the failing patterns and responses are stored, the passing ones can be easily deducted from the complete test set.

Other online techniques, however, do not include a test pattern generator. In other words, the applied patterns are unknown, and the only information available is the one explicitly stored. Such is the case of the aforementioned discussed concurrent test techniques [Touba97] when the tester merely observes the input patterns but does not act as test pattern generator. Because of the online storage limitation, in such cases only failing pattern information is available. The rest of the applied patterns are unknown, and no information can be deducted about the passing patterns. Any technique which does not include a known test set suffers from this limitation, which in turn complicates the diagnostic analysis.

3.5 Fault classification and physical failure analysis

Physical failure analysis (PFA) benefits from the result of logic diagnosis. The localization of the faulty component in the netlist reduces the suspect area, speeding physical analysis up. However, transient faults and some intermittents with very specific activation conditions may pose challenge for PFA if no defective structure is present in the circuit. For instance, faults activated by noise do not distort the physical structure of the chip. Neither do some intermittent faults, such as crosstalk-induced delays. Ruling out chips affected by noise and guiding PFA to look for specific structures (such as long parallel lines too close to each other) is beneficial to perform a faster defect identification.

The identification of chips affected only by noise allows the deployment of such chips in so-called robust designs, which are developed in such a way that they integrate fault-tolerance mechanisms and can overcome the effects of sporadic transient faults. Separating the chips affected by critical defects from those affected only by transient noise avoids unnecessary yield losses. Although devices which are sensitive to noise may not be suitable for critical applications, they can still work in robust systems where sporadic faults can be tolerated and do not lead the system to a dangerous state. Careful analysis is necessary to distinguish transient faults from intermittent faults that indicate the presence of unstable hardware structures or faults with complicated trigger functions. Chips affected by intermittent faults must be sorted out, as the problem will not disappear and can even worsen due to, for instance, aging mechanisms. For this reason, early classification of chips affected by a critical problem (i.e., a permanent

or intermittent fault) vs. those affected only by noise is essential to ensure yield and safety.

Some methods have been developed for transient fault discrimination. In [Pizza98], the authors propose to distinguish permanent from transient faults by making use of the Bayes' theorem. The conditional probabilities of a failure given the assumption of a permanent, transient or no fault are set analytically. Then, the a posteriori probabilities for each fault type are extracted. This approach distinguishes only permanent from transient faults. Since intermittent faults are becoming more apparent in new technologies [Gil T12], they must be taken into account in transient fault discrimination. Some approaches to tackle this problem considering intermittent faults are threshold-based, that is, they assume under a certain threshold a transient fault, and a critical one otherwise. An example of such techniques is presented in [Bonda00]. The method assumes that each component of the system is equipped with a fault detecting mechanism. The technique is based on the assumption that permanent faults cause an error in every time step, intermittent faults cause an error in each time step with a certain probability, and transient faults cause spurious errors, with very low probability. Each component has an associated score, which is increased when an error is detected in the component. Because of the assumption that intermittent faults are active with a high probability after their first activation, a high number of errors in a time window indicates a permanent or intermittent fault. To account for this, the score is multiplied by a constant lower than 1 after every time step. As a result, older errors are weighed down, and more recent errors gain importance, hence succeeding in implementing a simple system able to identify high frequency errors. A threshold is established for each component which, if surpassed, is considered to indicate the presence of a critical fault. Otherwise, the fault is considered to be transient. The method is limited by the fact that, in newer technologies, intermittent faults are introduced with very rare activation conditions, and hence also low activation rates. On the contrary, the tiny structures of nanometer devices are more sensitive to noise, consequently presenting higher transient error rates. A hard to detect permanent or intermittent fault or an intermittent with a low activation rate may easily be mistaken for a transient if only the number of observed errors is monitored.

The method in [De Kl09] deploys probabilistic reasoning about intermittent faults in a logic circuit. The author proposes a probing strategy to gather more information for diagnosis and distinguish intermittent faults. However, to deploy the method, full

observability of internal nodes and signals of the circuit is required. This diminishes the applicability to realistic cases.

Critical faults, i.e., permanent and intermittent, must undergo physical analysis. After PFA, it is possible to detect which faults are present with a higher frequency and indicate the presence of a systematic defect. However, PFA is a costly process, and its application time can be reduced if the category of the defect can be identified in advance. The benefits of an early identification are twofold: first, chips identified to belong to the same category can be analyzed first, confirming the presence of a systematic fault and thus allowing the correction of the problem earlier, avoiding further production of defective chips. Also, if the nature of the problem is identified in advance, PFA may use this information to change its target defective structure.

Some methods have been proposed that complement logic diagnosis and try to extract the activation function of the fault. The extraction of the activation conditions allows the human expert to infer what kind of problem is present in the circuitry. In [Desin06], the authors extract the neighboring lines of the victim and simulate the test patterns. By monitoring the activity of the neighbors and their logic values, they construct a Karnaugh map for each candidate. This Karnaugh map is deployed to extract the activation function. The considered variables of the function are the neighboring lines. Those combinations observed in passing patterns are marked with a 0, and those observed for failing patterns are marked with a 1. Finally those not induced by any test pattern are marked as X (*don't care*). This method assumes the faults behave in a deterministic way and does not contemplate the case of intermittent faults, which could cause an entry of the Karnaugh map to have 1 and 0 at the same time. This limits the method heavily, since intermittent faults are common in the nanometer regime, particularly due to the unpredictability of the physical characteristics of the circuit and the defect.

In [Maxwe16], a method is presented to diagnose intracell defects. Based on the concept of inductive fault analysis [Shen85] [Hapke14], a technology node and defect size are provided to the tool, which, in turn, simulates defects likely to be introduced and ranks them according to the number of matches. The simulation is performed at physical level to characterize the cells at the beginning of the process, and is the same as deployed for cell-aware test. The identification of the defect is achieved here by deploying a very low abstraction fault machine. The results obtained in test are compared to a fault machine where no fault model is assumed, but rather the physical distortion is simulated. As

a result, the outcome of the tool is a list of locations with an associated defect type. The paper states that some defects may be indistinguishable from test results, and, if necessary, test should be complemented with more specific cell-aware patterns. Both the circuit and the defect have physical characteristics which cannot be foreseen in advance to provide the simulator with precise data. For this reason, this approach incurs significant computational effort to consider all the possible combinations.

CHAPTER 4

Circuit uncertainty and machine learning

Due to imperfections in the manufacturing process, the resulting chip may differ from the specification despite the fact that point defects such as shorts or opens are not introduced. These imperfections cause the performance parameters of the produced chips to vary, limiting the success of deterministic techniques for certain tasks. Different approaches are needed for test and diagnosis which can relax the assumption that the behavior of the circuit is perfectly specified beforehand, that is, deal with a certain degree of uncertainty.

During manufacturing test and diagnosis, a tremendous amount of information is gathered. The research community has identified it thus as a suitable scenario for data mining: the automated extraction of knowledge from this information can vastly accelerate root cause identification and yield ramp-up.

This chapter presents an overview of selected machine learning techniques and the reasons why they are suitable and powerful tools to be deployed in volume test and diagnosis. Section 4.1 introduces briefly some scenarios in which indeterminism is inherent and machine learning techniques have been successful. Section 4.2 presents an overview of the most popular machine learning algorithms depending on the nature and objective of information extraction. Then, section 4.3 introduces the approaches

developed by the community to take advantage of the available information volume.

4.1 Machine learning and indeterminism

The main goal of artificial intelligence is to produce artificial agents that can learn from the environment regardless of the inevitable uncertainty [Sucar15], and eventually make decisions [Schai15]. Any agent, understood as an entity that interacts with its environment, must make decisions based on the information it can extract from the environment. The process is known as *data mining*, and it involves the steps of extraction and inference. The sampling process can be limited by the equipment or human that perform the collection, and as a result, information gathered from real-world data is often partially incorrect or unreliable [Bisho06]. The inaccuracy in the gathered data introduces indeterminism, and so every inference process needs to tolerate a certain degree of uncertainty in order to make decisions.

Machine learning [Bisho06] is the name to design a set of algorithms which approximate solutions to problems that cannot be solved in an if-then-else fashion, but rather a more complex function is necessary. Such algorithms receive data from the real world, which is usually affected by three main issues: missing data, incorrect or imprecise data, and unnecessary information, and perform the inference step. These methods are deployed when there is data uncertainty, the environment introduces indeterminism, or the most suitable model for the data is unknown [Schai15].

Machine learning was typically deployed in problems such as image processing and speech recognition, which involve noisy inputs. Image processing, for instance, needs to recognize certain scenarios regardless of the relative position of the observation with respect to the target and also, to a certain extent, despite the quality of the image. One of the first complex problems tackled with machine learning was the recognition of handwritten digits to automate partially the ordering task of the post service [Gench68] [Pawli88]. The indeterminism in this problem is introduced by the specific traces each of the samples uses to configure a certain number, which differ among them, and also because of the location. The number may be situated in any position in the square. The problem was first tackled by dividing the image into sections and matching the observed traces to previously computed patterns [Gench68]. Given combinations of known traces form a number. Several years later, the authors of [Pawli88] tackled the problem with machine learning. The obtained results were

comparable to the ones obtained with pattern matching. Soon the results were vastly improved with more sophisticated machine learning approaches: in [Botto94], the digits are represented as 28×28 pixels images. The paper shows that machine learning methods are able to successfully solve the problem, and have been proved to be robust against indeterminism. This crucial property has been enhanced [Schai15] to the point that some methods have been developed and deployed as denoisers for images [Vince08].

The same problem was identified early on in the field of speech recognition. A system involving speech recognition rarely operates in isolation conditions: most often, background noise is received along with the target sound. Consequently, the unnecessary information or its impact on the deviation of the target signal have to be identified for correct speech recognition. Also in this scenario, machine learning was taken advantage of and features are enhanced to achieve robust speech recognition [Seltz13] [Hanse96].

Both robustness and generalization are desirable and crucial properties that allow spreading the use of machine learning out of the traditional fields.

4.2 Sources of circuit uncertainty

With decreasing feature sizes of the devices in nanometer technologies, variations account for a big part of the circuit's exact behavior. Adding the nature and size of an a priori unknown defect only worsens matters: the impact of the defect depends on its physical properties and the relation to the physical properties of the circuit. Moreover, the external environmental conditions, which cannot be foreseen, may also disrupt the behavior of the system because nanometer devices are more sensitive. On top of that, the main test and diagnosis tasks rely on models of the circuit, which are generally constructed at a higher abstraction level and hence, per definition, inaccurate. This section presents the main three sources of uncertainty in a circuit: noise and environmental conditions, variations and inaccurate modeling.

4.2.1 Noise and environmental conditions

Testing is performed under controlled environmental conditions. However, when the module is integrated in its target system and environment, the exact setting is unpredictable in most cases. Decreasing feature sizes and variations make the designs more sensitive to noise and environmental factors. The conditions in which an electronic

device works, as well as the random noise in the environment which may affect the state of the components, make it impossible to foresee the resulting performance of the chip.

Heavy particles and radiation may hit the semiconductor material, and, as they lose energy, free some of the electron-hole pairs along their way [Zhao06]. As a result, random memory bits may be flipped (single event upset) or, if the drain of a transistor is hit (single event transient), a delay may be introduced eventually propagating and causing an error [Mochi15]. The occurring rate depends on which environment the system runs in and the technology the chip is fabricated in, as particle strike effects are more frequent at higher altitudes and lower technology nodes [Shiva02]. Initially, memory cells were considered to be more susceptible to particle strike, but logic has also been shown to be increasingly affected as feature sizes and supply voltage decrease [Shiva02], even causing multiple transient faults [Huang16].

Noise can appear also in the power supply, that is, in the power and ground grids. Power supply noise is introduced for two reasons [Segur04]: due to the resistance of the power lines and due to inductive coupling with the logic lines. In the first case, if a large number of devices in the chip switches, the current demand increases. Because of the resistance of the line, it is possible that the ground or power values of two different cells differ. As a consequence, the output values will also no longer correspond to voltages V_{GND} or V_{DD}, but rather to some intermediate value. Depending on the physical characteristics of the devices fed by the resulting signal, a wrong value may be propagated. The second effect arises when the necessary current for the transition of a gate causes an inductive voltage drop at the power and ground values observed by the gate. The performance of the gate can be affected in that its rising and falling transition times increase. If the delay is large enough and gets propagated, an error may be observed.

The target applications also influence the range of temperature in which a system must function. Temperature has a direct impact on the performance of both cells and interconnects. In particular, carrier mobility, threshold voltage and transistor saturation are affected and degrade with higher temperatures [Segur04]. External temperature may cause a system to malfunction if it exceeds the range of minimum and maximum temperatures indicated in the specification. Temperature may also vary within a chip. The switching activity is related to the workload, which is often not exactly predictable for a system. Not all devices across all the chip switch uniformly, since some structures

are more often exercised than others. Power dissipation caused by localized high switching activity can increase the temperature in some parts of the chip, causing so-called hotspots [Mukhe12]. These areas can see their performance affected because of the increased temperature.

Design techniques have been proposed to minimize the impact of noise and environmental conditions [Zhao06] [Das15] [Mukhe12]. However, it is impossible to monitor every environmental parameter continuously for every chip integrated in its target environment. In case of an error, this poses a problem for test and diagnosis techniques. For test of field returns, because the conditions in which the fault was activated are unknown. Hence, the target fault is unknown and the problem may be deactivated by the time the chip gets to the lab or may go undetected because the test set targets the wrong kind of fault. For diagnosis, because an exact model of the circuit cannot be built, since it depends on the unknown environmental conditions.

4.2.2 Variations

As technology evolves and devices are manufactured in deep submicron technology, variations become an apparent problem for the digital device industry [Becke10]. Many of the steps that conform the complete manufacturing process can only be performed with limited accuracy, and small deviations in many of the settings result in heterogeneous performance profile across all produced chips. Predicting the exact behavior of a manufactured device can only be done with limited precision, as uncertainty increases during manufacturing in the nanoscale era.

As opposed to environmental variations, variations caused by changes in the manufacturing process are called *process variations*. This kind of variation is induced by the conditions in the manufacturing environment and the characteristics of the manufacturing equipment, and may affect interconnects as well as devices.

Sources of variations

Manufacturing of digital devices is performed with *photolitography* [Weste11]. The process consists of a set of steps in which a photoresist is placed on the wafer, then the mask which marks the relevant areas for the current step is placed, and finally light is applied. The photoresist then uncovers only the areas of interest. For instance, in the first step, the n- or p-donors are injected in the substrate. To print a well of type n,

for instance, a mask is created which leaves only the well region exposed. Then, the n-donors are introduced in the substrate, only in the desired region.

With decreasing feature sizes, uncertainty in the manufacturing process cannot be further neglected. Variations are introduced in every step of manufacturing, and affect the final performance of the chip.

Ion implantation is usually deployed to dope the material with electron or hole donors. The process actually involves two steps: the implantation itself and an annealing step [Shin16b]. During the introduction of impurities, donors are accelerated and penetrate the silicon, until they lose energy due to collisions and finally stop. The annealing step is meant to repair the possible damage introduced by implantation, for instance the irregularities introduced by donors which form no covalence bonds with the substrate atoms and hence do not work as donors at all. The collisions and the annealing process cannot be totally controlled, and, as a consequence, doping fluctuations are introduced. The resulting doping concentration plays a role in the resulting performance: the doping of silicon may not be heterogeneous or the concentration of donor atoms may deviated from the intended value.

Although the donor concentration has increased in new technologies, the miniaturization of the corresponding features has caused the absolute value of charges in the channel to decrease [Shin16b]. In very small devices, a small fluctuation in the number of donors may cause a large relative deviation in the performance of the design. A deviation of the dopant profile can shift the nominal threshold voltage of the transistor, and even change the effective gate oxide thickness [Sriva05]. The threshold voltage of a transistor may also be shifted due to mobile charge in the gate oxide [Bonin01].

Among the variations which affect the devices in the circuit, line edge roughness (LER) [Shin16a] is one of the main concerns. LER refers to variations between the pattern printed in the mask, that is, the intended pattern, and the actual feature size printed on the die. Often, the sensitivity to light of the photoresist is amplified chemically. Such chemically treated photoresists contain photoacid generators which, upon light exposure, remain as acid anions and diffuse in the resistor layer to deprotect the area. This diffusion, however, cannot be precisely steered, and the acids may diffuse out of the target area. LER accumulates the variations of all photolitography steps: roughness in the masks, fluctuations in time of light exposure, the extent of acid diffusion and the roughness of the photoresist itself. As a result, the critical dimensions of devices on the chip may vary, introducing yet another degree of uncertainty about the performance

of the final chip. Techniques such as optical proximity correction (OPC) are deployed to compensate the effects of diffraction and interference [Segur04]. This is achieved by slightly modifying the mask. Although some variations are reduced, the resulting structures still deviate significantly from the expected shape and size.

Gate oxide thickness fluctuations can also affect the final performance. However, oxide thickness is a well-controlled parameter [Weste11], and in comparison to the previously explained parameters, barely introduces variation.

Interconnects can also be affected by imperfections in the manufacturing process and deviate from their intended specification [Bonin01]. In particular, the aforementioned fluctuations related to the photolitography process affect their geometry, which in turn changes their electrical properties and hence the expected performance. Photolitography may introduce fluctuations in the line width of electrical interconnects and on the spacing among wires. A change in the width of a line causes the resistance to deviate from the intended value, potentially changing the conductivity of the interconnect. A change in the separation between the interconnects poses a risk to signal integrity, because it can violate layout design rules [Weste11]. Depending on the deviation, it may lead to increased interferences between signals.

The height of interconnect structures may also suffer from fluctuations. Variations in the metal an dielectric thickness can be introduced by polishing or deposition profiles [Bonin01]. Similarly to the effects of deviation on the width of a wire, the height affects the size of a conductor element, and, consequently, its resistance. Another cause for resistance fluctuation is the contact and via size, which are affected by process variations [Bonin01].

Types of variations

Not all sources of variations affect all produced chips evenly. Some parameters, such as gate oxide thickness, only vary from wafer to wafer [Bonin01]. The time of exposure to light in the photolitography process affects also all devices in a wafer equally [Shin16a]. Metal thickness is generally well controlled, but can vary between wafers and also across a single wafer, that is, affect different dice unevenly [Bonin01]. Other deviations, such as random dopant fluctuations, affect devices even within a single die heterogeneously [Shin16b].

To model the parameters of a chip, variations are modeled differently depending on the areas they affect. Variations are classified in different types depending on the

distribution of the variation throughout the produced chips. Fluctuations observed from lot to lot, from wafer to wafer and even across a wafer, but which affect all devices on one single die homogeneously are referred to as *inter-die* variations. Inter-die variations capture deviations from lot to to, wafer to wafer and die to die. A parameter P is estimated as the mean, that is, the expected value in the nominal case P_{NOM} plus a deviation which depends on the die, P_{INTER}. The parameter hence has the value $P = P_{NOM} + P_{INTER}$.

Complex manufacturing processes include, however, also *intra-die* variations, that is, fluctuations in some parameters within one single chip with heterogeneous deviations of the nominal profile. Intra-die variations are subdivided in two different classes [Sriva05]: spatial and so-called random. Spatial variations are those which depend on the layout location within the die, such as contact and via size [Bonin01]. Random variations include factors which are hard to control, such as dopant profile fluctuations [Shin16b]. These deviations are also included in the parameter of the model. Spatial deviations are denoted $P_{INTRA}(x_i, y_i)$ because they depend on the layout position, indicated by coordinates (x_i, y_i). Random variations are different from every die i, and are expressed as $P_{RANDOM,i}$. As a result, a parameter is expressed as $P = P_{NOM} + P_{INTER} + P_{INTRA}(x_i, y_i) + P_{RANDOM,i}$ [Bonin01].

To calculate the expected performance of one die, however, not only the expected value (including variations) of the parameters needs to be calculated, but also the interaction among them. How the fluctuation of all parameters comes together will determine the performance of the chip. Generally speaking, the variations in all parameters finally affect the threshold voltage of the devices, which translates into differences in the timing, leakage currents, or electrical properties of the interconnects which affect also timing. The performance characteristics most importantly affected by variations are power and timing behavior [Liou03] [Bonin01] [Weste11]. Process variations were reported to account for 30% leakage variability and 20% frequency deviations already for 90nm technologies [Borka03].

Variations introduce an increasing degree of uncertainty, and cause a more and more important mismatch between the nominal properties and the final produced population. The deviation across the population is not biased, that is, chips are produced with better and worse performances than expected. To ensure high quality products, the uncertainty introduced must be accounted for in test generation, timing analysis and circuit simulation [Becke10], which are determinant for test and diagnosis success.

4.2.3 Inaccurate modeling

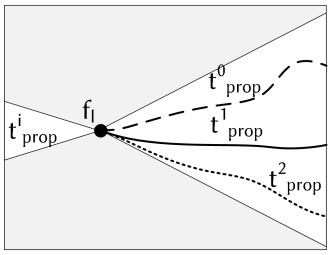
The accumulation of variations accounts for a large degree of uncertainty when modeling a circuit. An instance of a manufactured design cannot be precisely modeled because three factors influence its actual performance: the lack of predictability from design to manufacture, process variations and environmental variations. Moreover, for a defective chip, inaccuracies in fault modeling must also be taken into consideration.

Uncertainty appears as early as with the first post-silicon prototype. After design, steps such as logic synthesis and place and route are mandatory to produce an IC. Logic synthesis models the functions in the design as a set of cells allowed in the manufacturing process, while place and route decides the coordinates on the layout where each component must be placed, as well as fixing the interconnects. To optimize the chip, these functions often rely on cost functions which take diverse aspects into account, for instance, area and timing performance. However, to estimate timing before an actual implementation is available, models of the interconnects and cells are used. The analysis is imprecise not only because of the inherent approximation, which can for instance only estimate parasitics, but also because it may overlook false paths. Hence, from the prediction based on the design to the actual silicon circuit there is a degree of indeterminism introduced [Blaau08].

In addition to this, not all manufactured silicon devices have exactly the same physical properties, nor do they all perform equally in the presence of environmental variations. Given that process variations may account for as much as a 20% frequency deviation [Borka03], an additional uncertainty caused by environmental conditions worsens predictability expectations. The evaluation of the circuit model cannot be performed with traditional techniques. Timing analysis has shifted from static timing analysis to statistical static timing analysis [Blaau08]. With statistical timing analysis it is possible to make an estimation for a population of chips by considering probability distributions for the arrival times of the signals. However, this requires the simulation of a large population. The increasing number of devices and interconnects combined with the number of parameters and all of their variations calls for extremely efficient simulation algorithms [Schne17]. Provided such algorithms are available, Montecarlo techniques can be used to perform statistical analysis of the expected performance. However, although these techniques help draw conclusions about the population, predicting the exact behavior of one instance is still unrealistic. This can impact diagnosis, since the device under diagnosis has specific timing characteristics unknown for diagnosis.

Finally, in defective parts, fault modeling introduces even more uncertainty. The nature and size of the defect impact the physical parameters of the affected chip. Consider a crack in an interconnect at a certain fault location. The fault propagates along different paths for a certain activation pattern pair. The *slack* of a path in the circuit is the difference between the specified circuit delay and the propagation time along that path. The slack is thus equivalent to the time margin the path has to propagate and stabilize the signal. If the defect causes a timing specification deviation smaller than the slack of the path, the fault will not be detected. If the location feeds more than one path, each path with a different slack, then only those paths with a slack smaller than the introduced delay will be affected by the fault.

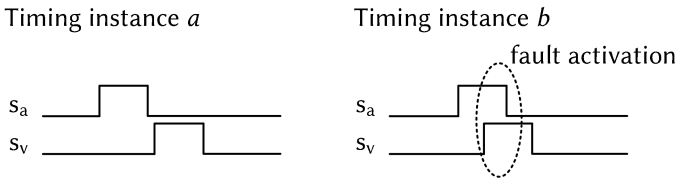
Figure 4.1 illustrates this situation. The transition propagates to the fault location at time point t^i_{prop}. The length (in terms of time) of each of the propagating paths from the location to the outputs is denoted t^o_{prop}, where o marks the identifier of the output. A defect in location f_l with size f_s causes a delay of f_d. The defect will be detected, for this particular instance with a clock period of t, in those output pins for which $f_d > (t - (t^o_{prop} + t^i_{prop}))$. The detection hence depends on the size of the defect, which cannot be predicted. Neither can an order between the propagation times be established, nor can their exact value be predicted, as it depends on variations. It follows that, depending on the physical properties of the chip (and hence depending on t^o_{prop} and t^i_{prop}, fault coverage for the same defect size may vary from one instance to another [Schne17].



▲ **Figure 4.1** — Indeterminism in timing faults

Indeterminism is not exclusive of timing faults. Static faults, i.e., faults with only functional behavior, may also be optimistic or pessimistically modeled without knowledge about the physical parameters of the circuit. Assuming an or-dominant bridge between victim signal s_v and aggressor signal s_a, for instance, the fault is activated whenever the lines have opposing values. In the situation depicted in figure 4.2, the lines of timing

instance a never fulfill the activation condition and hence the fault is not triggered. For another instance b, variations shift the transitions in the lines. In the time interval between t_1 and t_2, the fault is activated. For circuits in which this happens at a critical time and is propagated to the output, a fault is observed. A bridge model in which both signals are affected needs to take the output cones and their overlapping sections, as well as the input cones and their overlapping section. Even for static faults the degree of indeterminism in the behavior of the chip is considerable.



▲ **Figure 4.2** — Indeterminism in static faults

The consequence of all this uncertainty is that deterministic approaches have limited success guarantees. Test generation and diagnosis, in particular, rely on models and techniques that do not consider the probability of inaccurate on ambiguous information. However, with all the aforementioned uncertainty culprits, it no longer makes sense to ignore the increasing need of techniques robust with respect to indeterminism [Becke10].

4.3 Machine learning

The term machine learning includes all algorithms which build a model from a set of inputs to make predictions on new, unseen data. It is closely related to statistics, and many machine learning techniques deploy statistical methods of information. The set of data from which the model is built is generally referred to as *training set*, and its composition also depends on the tackled problem.

For some problems, a model can be automatically constructed on which inference can be performed with probabilistic reasoning. However, often the model is unknown. Machine learning can adjust model parameters or infer a pattern from the data set. We refer to these problems as *supervised* and *unsupervised* learning, respectively. In supervised learning, the algorithm tries to approximate a function from a known data sample. Unsupervised learning, on the other hand, tries to infer a pattern or structure from a data set.

The most suitable technique for each problem varies greatly depending on the goal. The following subsections give an overview into the main methods of inference with graphical models (section 4.3.1), supervised learning (section 4.3.2) and unsupervised learning (4.3.3). The last section in this chapter includes a survey of the state-of-the-art diagnosis techniques which deploy machine learning.

4.3.1 Graphical models

Probabilistic reasoning is a well-established technique to infer probabilities even in the presence of uncertainty. However, naive marginal probability computation may easily become infeasible as the number of variables in a problem grows. For this reason, probabilistic graphical models were introduced. Probabilistic graphical models are a set of strategies which allow probabilistic reasoning in a computationally feasible way [Sucar15]. Based on probabilities, they optimize the calculation of marginal probabilities even in the presence of uncertainty.

There are three aspects according to which graphical models can be classified [Sucar15]. A graph can be directed or undirected if the dependencies among variables are asymmetrical or symmetrical, respectively. These graphic models can also be static or dynamic, depending on if the graph represents the relation among variables at one point of time or for more than one different points of time. Finally, they can be categorized as probabilistic or decisional. The first include only random variables, while the latter include decision variables, i.e., variables for which a value must be chosen.

Bayesian networks [Pearl09] are an example of probabilistic directed graphs. They model the joint distribution of a set of variables, and can come in static or dynamic flavor, which represent a static relation among the variables or a time-dependent one. More sophisticated variants of BNs are able to learn the distributions and even the structure [Bisho06]. Markov chains represent dynamic processes. They are modeled as a state machine, and assume the Markov property: given a state S_i, the next state S_{i+1} is independent of the previous history S_j,where $j < i$ [Sucar15]. Hidden Markov models can reason in presence of unobservable states, and Markov random fields are undirected probabilistic graphs that generalize Markov chains. They are similar to Bayesian networks in that a variable's probability depends on its neighboring variables. However, unlike Bayesian networks, Markov random fields can represent cyclic dependencies.

4.3.2 Supervised learning

In supervised learning the algorithm is provided with a set of tuples (x, y) where x is a point in the input space, $x \in \Psi$, and y is a point in the output space, $y \in \Omega$. The goal of a supervised learning algorithm is to find the mapping function $F : \Psi \to \Omega$ so that $F(x) = y$ for the given input-output pairs. The term supervised refers to the fact that the algorithm learns from a data set in which the *expected* output y is provided a priori.

An input vector x is an f-dimensional vector of real numbers, because $\Psi = \mathbb{R}^f$. Hence, points x have f components: $x = (x^0, ..., x^{f-1})$. Depending on the output universe, supervised learning solves mainly two different types of problems: regression and classification. In regression the predicted value is a real number, so $\Omega = \mathbb{R}$. In a classification problem, on the other hand, a set of c labels $L = \{l_0, l_2, ..., l_{c-1}\}$ conforms the range of the predicted variable, $\Omega = L$.

The function $F : \Psi \to \Omega$ is approximated with a model M with adjustable parameters $(m_0, ..., m_{n-1})$. The model produces an output $M(x)$ for a given input vector. The optimization algorithm adjusts the parameters of the model to minimize a *loss function*. The loss function includes an error metric, for instance the difference between the output value predicted by the model, $M(x)$, and the expected one, y. It may also include other terms to ensure good learning properties. The minimization of the loss function is done for all elements of the so-called training set, $D_{train} = (x_0, y_0), ..., (x_t, y_t)$.

One of the simplest and best known models to approximate a function is linear regression [Bisho06]. Often deployed in statistics, it is a supervised prediction method which approximates the output as a linear combination of the inputs. The inferred variable is a real number, and the model has the form $LR(x) = m_0 x^0 + ... + m_f x^f + m_{f+1}$. In this case, the model parameters m_i are the linear coefficients, which are set to minimize the sum of the squared error between the value $LR(x_i)$ provided by the model and the correct value y_i provided in the training set. The loss function thus has the form $E(D_{train}) = \sum_{(x_i, y_i) \in D_{train}} (LR(x_i) - y_i)^2$. The simplicity of the method and the ease to understand the outcoming model has made it very popular. However, for many problems the output cannot be expressed as a linear combination of the inputs, and thus techniques which consider non-linear functions must be chosen instead.

Decision trees [Wu08] are a supervised learning technique which may be used for classification or regression. The internal nodes in a decision tree contain a test on one of the variables $x^i \in x$, while the leaves are annotated with an output value $y \in \Omega$.

To predict the output value for a vector x, the graph is traversed from the root to the leaves. In each internal node, the corresponding test is performed and a branch is taken accordingly. The structure of a decision tree can also be learned from a training set. To obtain a compact tree, the most important factor is the ordering of the variables. Different techniques may be used for this, such as information gain, which tries to minimize the entropy of the resulting subsets [Wu08]. Decision trees are most often used in contexts in which the result of a test determines the next test to be performed. Medical diagnostics are a good example, where the expert (doctor) orders tests depending on the result of the previously obtained ones.

Artificial neural networks (ANNs) [Lippm87] are able to infer non-linear functions. ANNs are formed by processing elements or *neurons* which transmit impulses along directed edges. Neurons are organized in levels or *layers*. The jth neuron of the ith layer is denoted as $n_{(i,j)}$. Each directed connection between neurons $n_{(i,j)}$ and $n_{(i+1,k)}$ is annotated with a weight $w_{(i,j),(i+1,k)}$. The weights of all connections between layers i and $i+1$ can be represented in a matrix W_{i+1}, where each element $W^{k,j}$ is the weight $w_{(i,j),(i+1,k)}$. The ability to infer non-linear functions is due to the non-linear transformation applied at every layer to the linear combination of the values received from the previous layer. ANNs compute a function of the form:

$$ANN(x) = (\phi_{out}(W_{out}^T \cdots ... \phi_2(W_2^T \cdots \phi_1(W_1^T \cdots x))...)$$

where ϕ is a non-linear function. ANNs iteratively adjust the model parameters, in this case the weight matrices, to minimize the loss function. The loss function includes a measure of the error between the obtained outputs and the labels in the training set, but it may also contain other terms such as weight decay, which penalizes big absolute values in the weights and thus ensures better learning properties. The non-linear function of the output layer may be changed to deploy ANNs for both regression and classification.

Support vector machines (SVMs) [Corte95] are a supervised classification algorithm which generate a hyperplane to divide the data. The hyperplane has the form $m_0 x^0 + ... + m_f x^f + m_{f+1}$ maximizes the margin to the closest sample points, which receive the name of *support vectors*. This hyperplane, however, separates only two linearly separable classes. If the data is not linearly separable, then a *kernel function* is introduced. A kernel function calculates the similarity of two points in a higher dimensional space. The SVM uses the kernel function to infer the linear hyperplane in this new space,

which may be non-linear in the original feature space. For problems in which $|L| > 2$ classes exist, the SVM generates $|L| - 1$ classifiers following a *1 vs all* scheme. The loss function of SVM includes a misprediction metric with an associated cost. By adjusting the misprediction cost, SVMs can have hard or soft margins, i.e., are less or more flexible with respect to outliers. A vector machine adjusts the hyperplane parameters, while the cost parameter and the kernel function must be provided to the algorithm. Although mostly used as supervised method often used for classification problems, deploying SVMs for regression is also possible [Stein08]. Both ANNs and SVMs are deployed in classification problems. Unlike BNs or decision trees, information about all features must be available to perform classification.

4.3.3 Unsupervised learning

Unsupervised learning methods extract information from the training data set. Their training set consists only of points in the parameters space, but does not include an associated output, i.e., $D_{train} = \{x_0, x_1, ..., x_t\}$. Unsupervised learning methods include clustering, in which the data is divided into different subsets or classes. In clustering, a similarity metric is established. According to it, the unsupervised algorithm divides the data points so that points contained in one class are similar. On the contrary, points which belong to different classes must also be different from each other. *k-means* clustering [Wu08] is an example of a clustering algorithm, in which the user gives the number of classes into which the data set should be partitioned, k. Random data points are randomly chosen as the initial *centroids*, and the algorithm then partitions the data set, associating every data point to the closest centroid. The mean of every class is calculated, and the algorithm repeats the two steps iteratively until convergence, i.e., until the centroids do not move.

Another interesting application of unsupervised learning is the identification of relevant structures in the data set to reduce the dimensionality of the problem, extracting sets of relevant features and discarding noisy ones. An example of an unsupervised method for dimensionality reduction is principal component analysis (PCA) [Bisho06]. PCA is an orthogonal linear transformation which maps the original g-dimensional input space into a new d-dimensional space with $d < g$. The linear transformation is optimized so that the information loss is minimal, i.e., the variance in the output space is maximized.

ANNs may also be used for this dimensionality reduction [Hinto06]. Although not strictly an unsupervised method, it is categorized as such because no labels are provided

or needed. The learning problem is formulated as a regression problem, in which a multilayer ANN learns a lossy identity function. Such an ANN is called an *autoencoder* [Goodf16], and it can be seen as a network which implements two functions: an encoding one, $e : \mathbb{R}^f \rightarrow \mathbb{R}^a$, and a decoding one, $d : \mathbb{R}^a \rightarrow \mathbb{R}^f$, so that $d(e(X)) \approx X$. The dimension a of the encoder output may be different than the original vector size f. Thus, the encoder renders an overcomplete or compacted version of the features, which captures important properties extracted by the ANN. This encoded version of the features is then deployed instead of the original vector.

4.4 Root cause identification and yield learning

In early manufacturing stages, yield levels are generally low. Manufacturers must polish the process as they learn about the systematic problems that appear in the produced chips. Diagnosis is thus a key step for yield learning: the sooner the defects are correctly identified, the easier it is to distinguish and correct systematic problems. Very often, however, the results of diagnosis merely give back a location at logic level, and, at best, a fault model. Due to the tremendous amount of information gathered during test and diagnosis, root cause analysis is a particularly well suited scenario to deploy machine learning techniques. In particular, the costly physical analysis procedures can benefit of any approach able to pinpoint in advance to the root cause.

Machine learning is deployed as early as post-silicon validation. Since the amount of data is not so big at this stage, unsupervised learning can be taken advantage of to perform diagnosis. A clustering technique is presented in [DeOri13] to detect bugs in post-silicon. A subset of signals is monitored, and the activity in those lines (more specifically, the ratio of time the signals are at 1) is deployed as feature. The passing tests are labeled as passing, while the label of examples that fail the test is marked as unknown. The passing examples are then included as training set and k-means clustering is applied. The obtained clusters represent the healthy examples. Then, the authors check if the failing examples fall out of the clusters. In that case, an anomaly is detected. When the number of anomalies exceeds a threshold, a bug warning is issued.

More approaches exist to analyze a big population of failing chips to extract the root cause from volume data. In particular, a technique is proposed in [Benwa12] to construct a BN in which the syndromes are represented by observed nodes and the diagnosis candidate and root cause are the hidden nodes. Thus, by observing the test outcome and

inferring the probability of the hidden nodes, the authors extract the root cause. Based on [Desin06], [Xue13] proposes to deploy machine learning in order to improve the resolution of logic diagnosis. The authors propose two classifiers to discard candidates of the list. The first classifier discards any candidate with inconsistent activations, that is, for a configuration of neighboring signals, two tests have a different result (one passes and the other fails, for instance). The candidates remaining on the list undergo a second classifier, which consists of a soft-margin SVM. The authors propose line entropy and unique outputs as features. Line entropy represents the relation between the value of the considered line and its neighbors. The unique output feature accounts for failing outputs which only the considered line can explain. To apply classification, candidates are initially marked as good or bad. A good candidate is, for instance, the diagnosis candidate when the list contains only one element. On the other hand, for chips with diagnosis lists containing over twenty candidates, the authors assume that candidates after position twenty are bad candidates. Typically, such a training set will be unbalanced, since there are many more bad candidates than good ones. To balance the training set, the authors duplicate the points with "good" activation patterns, and then infer with the SVM a classifier into good and bad candidates. The candidates classified as bad can then be discarded.

Since the fault model deployed in fault location may differ from the actual underlying defect, hence not reflecting the behavior accurately, logic diagnosis can be complemented with physical analysis and machine learning to identify systematic defects. In [Blant12], "snapshots" of the layout surrounding the logic diagnosis candidate are taken an processed. The snapshots are transformed into pixel-based images, and clustering is performed to group similar images together. Big clusters (i.e., cluster with a high number of members) are expected to contain systematic defects, and thus machine learning can also be used for yield learning.

At higher abstraction levels, functional board-level diagnosis has also been tackled with machine learning algorithms. The technique in [Zhang10b] is based on Bayesian inference. The conditional probabilities are calculated from previous observations, and they allow inference about the belief that a certain fault is present in the system. In [Ye13], diagnosis is conducted using ANNs and SVMs. Based on information of previous diagnosed chips, a one-layer ANN is built per component which has been replaced in past production. The input features are the syndromes and the output is encoded as a binary variable which expresses the probability of the component actually

being defective. The component whose ANN has a higher output probability is identified as the failing part. The authors also follow a similar approach with SVMs, and model the problem as a classification problem in which the classes are the faulty components. Then, the ANN and SVM are averaged with weights inversely proportional to their training error rates. The outcome of this voter is the final diagnosis. Similarly, board diagnosis is performed with decision trees in [Ye12]. The variables in the problem are the syndromes, and each internal node evaluates if the syndrom was observed or not. The leaves are annotated with the component which is deemed to be the cause.

Finally, some information can also be extracted to predict customer returns. Especially in the automotive industry, where the required DPPM (defective parts per million) is so low, identifying potential customer returns is crucial. In [Sumik12], the authors model the test set as a multivariate space. They extract the principal components and allow for a certain variance. Any point falling out of this margin is considered an outlier, and the authors succeed in predicting customer returns from the results of the manufacturing tests. In [Lin14], the authors predict test escapes from test data. They generate residual vectors of test measurements, which represent the difference between the obtained value and the expected one. The expected value accounts for systematic variations, which makes the approach robust. Then, they use PCA-like analysis to project their vectors onto a new space in which they separate using an SVM.

Given the large volume of information available to the manufacturers from production, test results and diagnosis, machine learning techniques are gaining importance to analyze and predict production results.

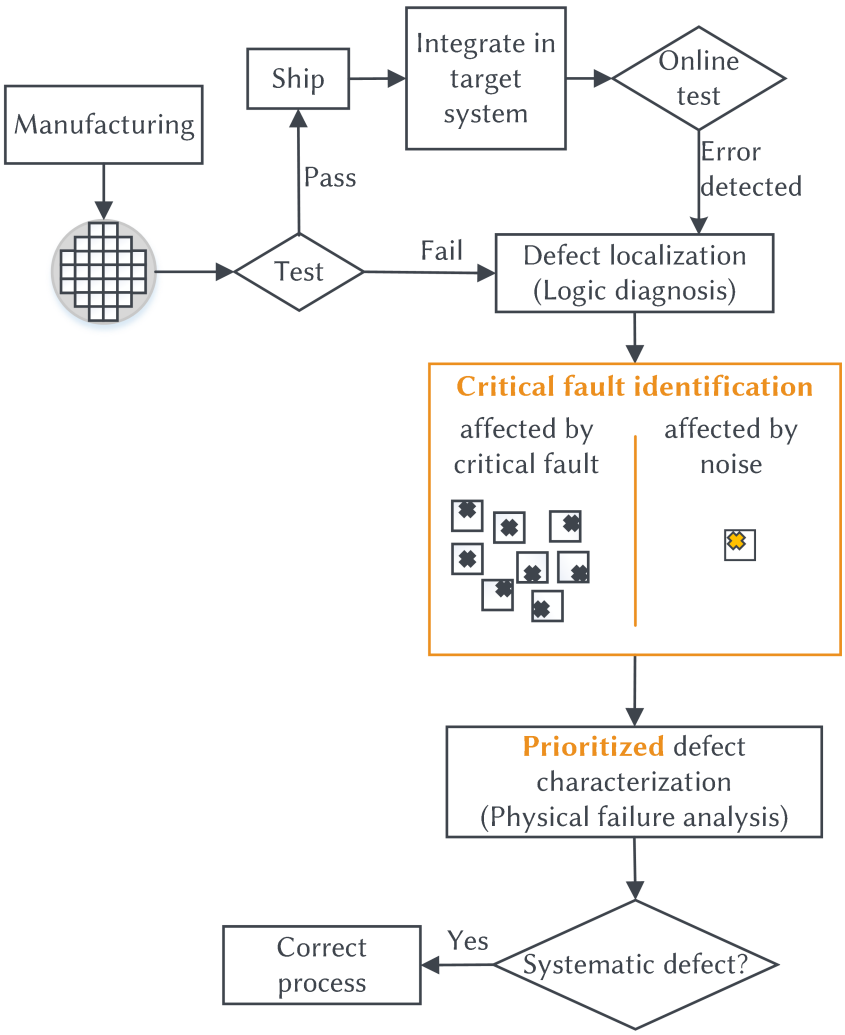
CHAPTER 5

Bayesian networks for identifying critical defects

Circuits may fail not only because of physical distortions in their structure, but also because they are affected by external factors such as noise or particle strikes. *Robust* designs incorporate fault tolerance mechanisms to mitigate the effect of random noise. Thus, a chip exhibiting erroneous behavior or test fail caused by transient errors need not always be discarded. Doing so causes an unnecessary yield loss, since a transient error indicates merely sensitivity to noise and robust designs are equipped to overcome their effects. An intermittent fault, on the other hand, is a sign of unstable hardware structures, and must be identified and discarded to avoid threats to safety.

Chips affected by intermittent faults, particularly if they are the result of a systematic problem, must be analyzed to find the root cause. An early classification to rule out circuits affected by noise helps not only avoid unnecessary yield loss, but also prioritize the deployment of costly physical analysis resources. The critical defect identifier presented in this section is a fast classifier that can be integrated in the traditional test flow, as depicted in figure 5.1

Intermittent faults can be caused by phenomena like high frequency power droop, which induces a fault in the victim line only if there is high switching activity in the neighborhood. The activation rate of such faults is very low, since the conditions

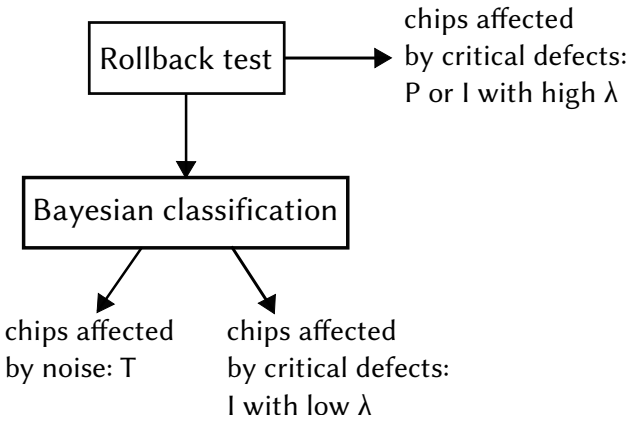


▲ **Figure 5.1** — Diagnosis flow with critical fault identification

for the fault to be triggered are so specific. Transient errors, on the other hand, are caused by noise or particle strikes, and tend to happen in random locations, and with low error rates. Distinguishing between intermittent and transient faults is extremely hard because their observed error rates may be similar. Traditional methods have tried to establish a fixed threshold to discriminate between intermittent and transient faults. If the number of observed faults surpasses said threshold, then the fault is considered intermittent, and if it does not then it is considered transient. However, merely considering the error rate and not reasoning about the activation rate limits the success of such approaches. An error may only be observed if both conditions are met: the activation conditions are fulfilled and the fault can propagate to an observable

output. Some faults may be activated more often, but not propagated to the outputs because some other signal masks them. Hence, it is possible that an intermittent fault has a high activation rate but does not get propagated often to the outputs, resulting in a narrow error rate. Establishing an absolute threshold ignores the impact that observability of the fault may have on the error rate.

The work at hand proposes a hybrid approach to distinguish critical (permanent or intermittent) from non critical (transient) faults. The first filter identifies permanent or intermittent faults with high activation rates based on the results of rollback test. For the rest of the cases, i.e. to distinguish intermittent faults with low activation rates from random noise, Bayesian inference is deployed. Figure 5.2 depicts the steps of the approach.



▲ **Figure 5.2** — Adaptive flow for critical fault identification

The content in this section is organized as follows: section 5.1 presents the adopted test strategy and the diagnosis approach deployed. Section 5.2 illustrates how immediate critical fault classification is performed based on the results of test. Section 5.4 explains the structure and inference of a Bayesian network, and is followed by the explanation on how they can be applied to distinguish between noise and critical defects (section 5.4). Finally, section 5.5 presents the experimental validation of the whole approach.

5.1 Adaptive test and diagnosis

The adaptive method proposed, which was first presented in [Rodri14], takes advantage of the testing technique for robust circuits presented in [Amgal08]. The technique partitions the test set in w different sessions of p^w patterns each. Test sessions are applied with BIST, and the corresponding responses are compacted in a signature. The ith test session is said to have detected a fault if the observed signature s_i is incorrect, that is, differs from the expected signature for the session s_i^{ref}. In such case, rollback test is triggered and the session is repeated. Repetitions go on until a correct signature is observed or the maximum number of repetitionsR_{max} is reached.

If the fault is permanent, then the repeated test sessions are expected to render the same signature. On the other hand, for transient faults the error is expected to disappear. For intermittent faults, the behavior of the fault in the following repetition cannot be predicted: the fault may disappear, or it may repeat the same behavior hence rendering the same faulty signature. For an intermittent fault c_i, the set of patterns that can detect a fault in the candidate location is denoted as $T_d^{c_i}$. If its cardinality is higher than 1, in other words, if more than one pattern in the session can detect the fault, then an intermittent fault can also cause the chip to output a signature which faulty and at the same time different from the first faulty signature.

For a given test window with expected signature s_i^{ref}, the observed signature is s_i. The following applications of the same test window are denoted as primes: s_i' represents the first repetition, s_i'' represents the second, an so on. For a failed test window where $s_i \neq s_i^{ref}$, three different outcomes may be caused by the underlying fault:

- fault-free session ($s' = s_i^{ref}$): c_i is not activated and the chip gives back the expected response.
- repeated faulty signature ($s = s'$): c_i follows the same activation pattern and the same faulty signature is observed.
- different faulty signature ($s \neq s' \wedge s' \neq s_i^{ref}$): c_i followed a different activation pattern and rendered a faulty signature, however different from the first one.

Each test window is marked with a code depending on the result of test. Table 5.1 summarizes the possible codes for a test window for $R_{max} = 2$, with s_i being the obtained signature, s_i' the signature obtained from the second pass, and s_i^{ref} the golden model signature. The first column determines the observed behavior in test: the fault-free

▲ **Table 5.1** — Codes for test sessions depending on the outcome of rollback test for $R_{max} = 2$.

Behavior	Code	Interpretation
$s_i \neq s_i^{ref}$ $s_i' = s_i^{ref}$	10	Transient or intermittent fault
$s_i \neq s_i^{ref}$ $s_i' \neq s_i^{ref}$, $s_i = s_i'$	00	Permanent or intermittent fault
$s_i \neq s_i^{ref}$ $s_i' \neq s_i^{ref}$, $s_i \neq s_i'$	01	a) Intermittent problem activated by different patterns or affecting several nodes b) More than one transient fault in a row

repeated session, the repeated faulty signature, and the case in which a different faulty signature is observed after repeating the test session. The second column, labeled as *code*, is how the rollback tester registers the test session. The last column summarizes the possible faults that could cause the corresponding behavior.

Logic diagnosis is performed after the rollback test is finished. To handle state-of-the-art compression and compaction mechanisms used in the industry, the diagnosis from [Cook11] has been deployed. This approach can handle signatures instead of the uncompressed responses. By constructing and solving a system of linear equations to explain the observed signature, logic diagnosis can be performed and a list of candidate locations is obtained. No information is given, however, about the criticality of the fault. The remainder of the chapter explains how critical fault identification is performed.

5.2 Immediate critical fault discrimination

The proposed adaptive test scheme allows early detection of some critical defects. The first filter, or immediate critical fault discrimination, discards chips which cause test sessions to be coded as 00 or 01. The reason is that permanent faults or intermittent faults with a very high activation rate can cause repeated faulty signatures. For transients, however, the probability of a repeated signature to be observed is extremely low. Thus, observing $s_i = s_i'$ during test indicates the presence of a critical fault, and the circuit can thus be discarded in this first step.

In case of observing $s_i' \neq s_i^{ref}$, $s_i \neq s_i'$, i.e., two different faulty signatures were observed when rerunning the test session, an intermittent fault is assumed. Although this situation can also be caused by transient faults in a row, this conservative approach is considered to ensure high product quality. However, should this assumption be too pessimistic for the target application, the number of repetitions for a test session when

a faulty signature is observed can be increased. With this measure, the probability that several transients in a row may cause a code 01 decreases, and the pessimistic effect is reduced.

Finally, the absolute number of wrong signatures also allows early discrimination. Assuming a transient error rate of μ, the probability that the outcome of a test session with p test patterns is corrupted by noise is $1-(1-\mu)^p$. The probability that some w_f out of w sessions are faulty can be calculated accordingly as

$$\binom{w}{w_f}(1-(1-\mu)^p)^{w_f}((1-\mu)^p)^{w-w_f}$$

Depending on the desired reliability level, the user may define a probability threshold which is acceptable for the application. The maximum number T_{max} of acceptable faulty sessions is adjusted accordingly.

The proposed immediate critical fault discrimination thus discards any chip with behavior that reproduces faulty signatures or causes a high number of them.

5.3 Bayesian networks

The remaining group of devices may still be affected by a critical defect or a transient. Immediate critical fault classification only rules out permanent and intermittent faults with a high activation rate, but the problem of distinguishing low activation intermittent faults from transients remains. At this level classification calls for a more sophisticated approach.

The problem at hand is finding an explanation for the observed faulty signatures. The candidates to explain the test responses are the locations returned by logic diagnosis and random noise. Both the signatures and the causes can be modeled as variables. The relation between a cause and an observed signature can be quantified in the form of a conditional probability, which can be extracted analytically. Bayesian networks (BNs) are a well-suited strategy to reason about the probabilities of the cause variable.

In the following subsections, the general structure and inference strategies for BNs are described. The information in this section has been extracted from [Sucar15] and [Pearl09]. This section only includes the aspects of Bayesian networks with relevance for this work. Readers already familiar with BNs can find the application of BNs to noise rule-out in section 5.4.

5.3.1 Probabilities in Bayesian networks

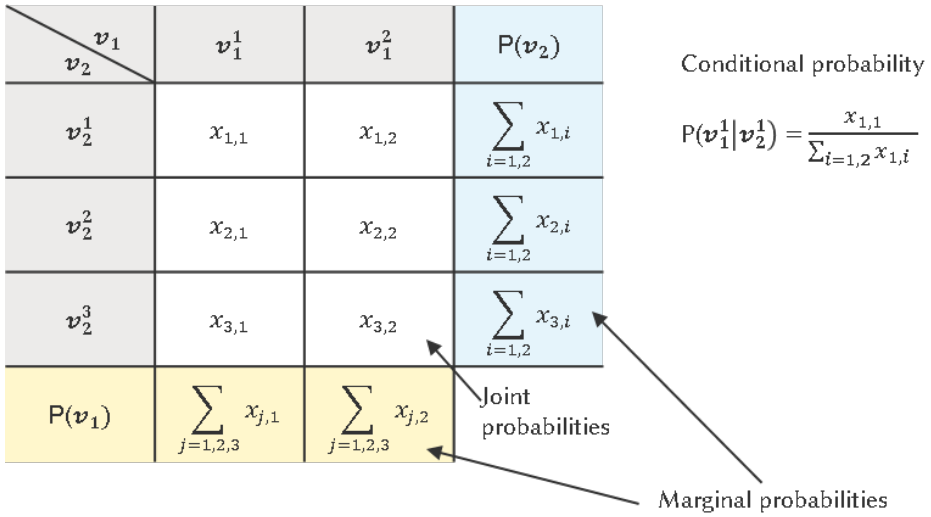
Bayesian networks reason about the probability of events. Events can be represented by random variables, which can have different ranges to represent the occurrences of the event. For instance, to represent the occurrence or not occurrence of an event, a binary variable is enough. If the event has more than two states, the corresponding variable will have a range of discrete values. This would be the case of a variable that represents a traffic light, which can possibly take four values: $\{r, y, g, o\}$ to represent that the red, yellow or green light is on, or that the traffic light is off, respectively. Finally, it is also possible to have variables with continuous ranges. A continuous range makes the representation of variables such as account status or physical measurements possible.

The probability of an event can be expressed with the marginal, conditional and joint probabilities. Given two events represented by variables v_1 and v_2, the marginal probability of their occurrence is expressed as $P(v_1)$ and $P(v_2)$, respectively. The marginal probability of an event quantifies its unconditional probability, that is, the probability for each value irrespective of other events. The conditional probability $P(v_1^i | v_2 = v_2^j)$ quantifies the probability of v_1 taking the value v_1^i given that v_2 takes its value v_2^j. Finally, the joint probability $P(v_2 \cap v_1)$ quantifies how often each combination of values for both events happens. These concepts are illustrated in figure 5.3 for two variables. Variable v_2 can take any of three discrete values v_2^1, v_2^2, v_2^3, while v_1 is a binary variable. The bottom row and rightmost column of the table show the marginal probability for variables v_1 and v_2, respectively. The joint probability of both variables is registered in the cells of the table: for every combination of values, the probability of occurrence is annotated in the corresponding cell. Finally, the conditional probability $P(v_1^1 | v_2 = v_2^1)$ is extracted from the table.

For problems which involve many variables, however, the *chain rule* allows to compute the joint probability of a set of variables based on the conditional probabilities. Given an ordering of the variables indicated by the subindex, the chain rules states that

$$P(v_1, .., v_n) = \prod_{i=1}^{n} P(v_i | v_1, ..., v_{i-1})$$

The chain rule can be simplified if there is independence among the variables. A set of variables V_1 is said to be independent of another set of variables V_2 given a third set V_3 if $P(V_1 | V_2, V_3) = P(V_1 | V_3)$. Consequently, the terms $P(v_i | v_1, ..., v_{i-1})$ can be simplified if v_i is independent of a subset of the variables $v_j, j < i$. The $v_j, j < i$ which



▲ **Figure 5.3** — Joint and marginal probabilities

do not influence the probability of v_i need not be included in the calculation of the conditional probability.

A distinction is made in a Bayesian network between prior and posterior probabilities. The prior probabilities are the marginal probabilities annotated before any knowledge about the rest of the variables is introduced. The posterior probabilities, on the other hand, are the marginal probabilities of the variables when new evidence has been propagated in the network.

5.3.2 Bayesian network structure

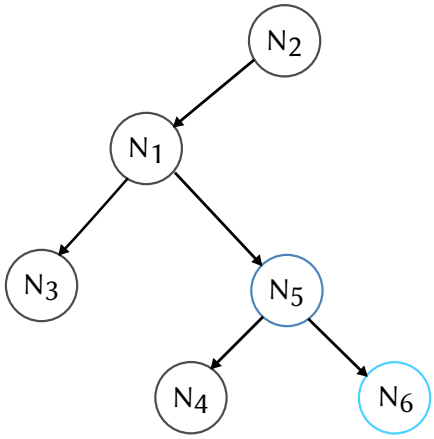
Bayesian networks (BNs) are static directed probabilistic acyclic graphs. BNs model the joint distribution of the variables in the problem, and can also represent causality relations among the variables. BNs allow the computation of marginal probabilities efficiently even for large sized problems. Given the structure and necessary conditional probabilities are provided, they allow the inference of the posterior probability distribution for some variables when some new evidence is presented.

BNs are formed by a set of nodes N, which represent the variables V in the problem. Nodes can be connected by edges. An edge $e = (v_1, v_2)$, where v_1 is the origin and v_2 is the destination node, represents a cause-effect relation in which v_1 is the cause

of v_2. In a Bayesian network, each arc $e = (v_1, v_2)$ is annotated with the conditional probability, $P(v_2|v_1)$. Such probabilities quantify the impact of the values of v_1 on those of v_2. Root variables, i.e., roots represented by nodes which do not have parents, are annotated with an a priori probability distribution. The a priori distribution is an initial estimation of how probable the occurrence of each value of the variable is. These a priori probabilities can be either adjusted from expert knowledge, or assumed to be uniform if no previous observations are available to allow an estimation.

The structure of a BN and the implied independence between variables is what simplifies the calculation of the marginal probabilities. A set of nodes A is independent from a set of nodes B given the set C if $P(A|C, B) = P(A|C)$. In a Bayesian network, this implies the D-separation criteria to be met: there is no trajectory from A to B so that convergent nodes are or have descendants in C and all other nodes are outside C.

In figure 5.4, a small BN with variables $N_1, ..., N_6$ is depicted. Node N_6 is independent from $N_1 ..., N_4$ given N_5. Hence, $P(N_6|N_1, ..., N_5) = P(N_6|N_5)$. This property is taken advantage of to query about the value of a variable.



▲ **Figure 5.4** — Bayesian network structure

5.3.3 Inference in Bayesian networks

BNs take advantage of the rule chain and of the structure of the network. Inference in BNs takes advantage of the chain rule, that is, uses the conditional probabilities to calculate the joint probability of all variables. The calculation can be simplified: the conditional probability of a variable only needs to take into account the variables on

which it depends for. In a BN, the structure of the BN already implies independence among some variables: given the parents of a node, the node is independent from the predecessors of its parents.

The structure of the BN on which inference is to be performed determines the inference algorithm to be used. Queries on trees and polytrees can be easily solved with exact belief propagation. However, more complex structures may require approaches such as variable elimination or conditioning to reduce the complexity of the network. Approximate approaches exist, too, and are often used iteratively for loopy structures. Since the BN used in the work at hand is a singly connected network, Pearl's belief propagation algorithm suffices to query about hidden variables. Hence, this section explains the belief propagation algorithm.

In BNs, the nodes which provide evidence are called *observed*, while variables queried about are referred to as *hidden*. When the values of the observed variables are known, the new evidence E is propagated along the network to update the values for the hidden variables H, $P(H|E)$. The most frequently performed form of inference on a BN is single query inference, i.e., the user inquires about the marginal probability of one single variable.

Message-passing based probability propagation [Pearl09] [Sucar15] is based on the Bayes rule. The belief propagation algorithm was developed for singly connected networks. In a singly connected network, a node B divides the network in two subtrees: the subtree rooted in B, and all other nodes.

The algorithm applies the Bayes rule to find out the probability for each value h_i of the hidden variable H when evidence E is provided, that is, H's posterior distribution:

$$P(h_i|E) = \frac{P(E|h_i)P(h_i)}{P(E)}.$$

However, the evidence can be decomposed according to the topological decomposition of the network by B: evidence obtained from the subtree rooted in B is denoted E^-, and all other evidence is denoted E^+.

$$P(h_i|E) = \frac{P(E^+, E^-|h_i)P(h_i)}{P(E)},$$

but because E^+ and E^- are independent, the queried posterior probability can be reformulated as:

$$P(h_i|E) = \frac{P(E^+|h_i)P(E^-|h_i)P(h_i)}{P(E)}.$$

The evidence collected from the children of B is denoted $\lambda(B) = P(E^-|B)$. Evidence collected from the parents of B is $\pi(B) = P(E^+|B)$. The probability of B is expressed as: $P(B|E) = \alpha\pi(B)\lambda(B)$ where α is a normalization constant. The messages should be interpreted as probabilities, hence, the summation of the belief of all possible values of a variable should add to 1. Normalization can be performed at the end, that is, when the hidden variable is reached. However, it can be performed in every step to avoid overflow when the tree is deep and the number of multiplications of λ-messages is high.

To compute λ, information is transmitted from the leaves to the root. Every node composes a λ-message which it sends to its parents. The vector represents the belief about the parents' value given the information available at the child, and is based on the conditional probabilities and a λ-vector calculated by the node.

Evidence nodes compose a λ-vector which is a one hot encoding, where the only active value is the observed value, and all others are set to 0. For evidence node N_e, $\lambda(N_e) = [0, 0, ..., 0, 1, 0, ..., 0]$. Leaf nodes with no evidence information available simply compose an all-1 vector, $\lambda(N_n e) = [1, 1, .., 1]$. All other nodes compose their λ-vectors with an element-wise multiplication of the λ-messages of their children. For a node N_p with children NC, $\lambda(N_p^i) = \prod_{n \in NC} \lambda_n(N_p^i)$.

A child N_c composes a λ-message for its parent N_p according to

$$\lambda_{N_c}(N_p^i) = \sum_j P(N_c^j|N_p^i)\lambda(N_c^j)$$

that is, for every value N_p^i of the parent, the child computes a belief estimation based on the information it has received, and sends it up. All nodes multiply the λ-vector and their conditional probability table and propagate the result up towards the root of the tree.

From the roots to B, π-messages are propagated. A π-message is computed based on the prior marginal and the conditional probabilities. An evidence node will form a π-vector as a one-hot vector, where the active value is the observed one, $\pi(N_e) = [0, 0, ..., 0, 1, 0, ..., 0]$. If there is no evidence and the node is a root, its prior marginal probability forms its π-vector, $\pi(N_n e) = P(N_n e)$. Otherwise, the π-vector of a node depends on the information received from its parents. It is calculated as the product of

the message received from the parents and the conditional probability table. For a node N_c with parents NP, each component of the $\pi(N_c)$ vector is calculated:

$$\pi(N_c^i) = \sum_{N_p^j} P(N_c^i|N_p^j) \prod_{N_p \in NP} \pi(N_p^j)$$

The π-message from a parent node N_p to one of its children N_c comprises information about all neighbors of N_p except N_c itself. It is calculated by multiplying the π- and λ-vectors and normalizing:

$$\pi_{N_c}(N_p^i) = \alpha \pi(N_p^i) \prod_{n \in (NC - N_c)} \lambda i(N_p^i)$$

When all nodes have been updated, each one contains the posterior probability of the corresponding variable under the given evidence.

5.4 Critical fault discrimination with Bayesian networks

In the problem at hand, the involved variables are the results of test and the underlying cause. The assumption is that either there is one intermittent fault in the circuit (regardless of if there was noise or not), or the circuit is defect-free, i.e., only affected by transient noise. Transient noise may appear randomly, both in fault-free and faulty circuits.

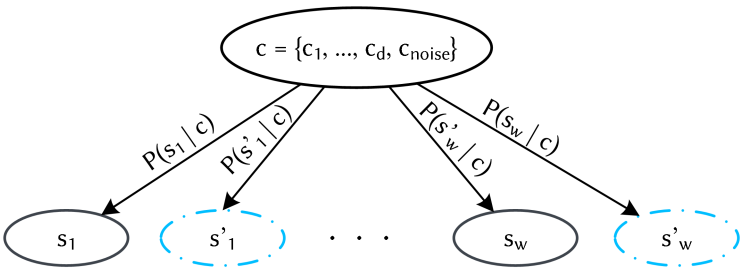
For this reason, the variable representing the cause of test fail is a single multivalued variable, c. The range of c comprises all candidates returned by logic diagnosis, plus one candidate for the fault-free circuit or transient noise case. If logic diagnosis returned candidates $c_1, ..., c_d$, then the range of the fault variable $c = \{c_1, ..., c_d, c_{noise}\}$.

The variables that represent the signatures are denoted s_i, and those representing the signatures of rollback sessions are marked s_i'. Their range includes all possible values that a signature can take. In theory, a signature could get any combination of bits, i.e., for a signature of length m, there are 2^m possible results.

In this particular problem, the automation of the BN construction is possible. The resulting Bayesian network is depicted in figure 5.5. The underlying fault has a causal relation with the observed signatures. In the figure, variable c is represented with the

node in the higher level. The nodes in the bottom represent the signatures obtained from test, S. The number of nodes on the second level may vary due to the number of repeated sessions. For $R = 2$, all sessions are run at least once, and at most twice. In figure 5.5, the repeated test sessions, marked with a dashed contour, are only represented if the first session was faulty. Thus, the number of nodes for the test sessions $|S| \in [w, 2w]$. All 2^m values conform the range of the test session variables.

Since the presence of a fault has an impact on the obtained signature, the fault node is connected with edges to all test session nodes. Variable c is cause for the observed test sessions $s_1, ..., s_w$, where the fault node is the origin and the test session nodes are the destination. Note that the probability distributions of two signatures are independent given an underlying fault. Thus, no relation is included among signatures: the BN structure reflects the independence of their values given a fault.



▲ **Figure 5.5** — Bayesian network for critical fault classification

The conditional probabilities for critical defect identification are calculated analytically based on the expected activation rates and the detectability of the faults. For every intermittent fault $c_1, ..., c_d$ the set of patterns $T_d^{c_i}$ is divided in two different sets according to the observed activation of the fault. The set of patterns $d_s^{c_i}$ comprises those patterns for which the fault was active, i.e., an error was observed. The set $nd_s^{c_i}$, on the other hand, is formed by all patterns that could potentially detect the fault, but no error was observed. In other words, a permanent fault at the same location would have propagated an error, but the intermittent fault was inactive.

Intermittent faults are activated with a certain rate λ. Transient noise, on the other hand, is characterized by an error rate μ. The error rate μ of transient noise is the product of the activation μ_a and propagation rates μ_p, $\mu = \mu_a * \mu_p$.

An intermittent fault at location c_i can explain an observed signature if its effect was not masked by a transient. This means the probability of observing signature $s_i = s$,

where s is a signature that can be explained by c_i, is calculated as a product of three probabilities:

- the intermittent fault was active at location c_i for the detecting patterns,
- it was not active in the non-detecting patterns,
- and there was no random noise.

An intermittent fault is active for one detecting pattern with probability λ. Hence, it is active for all detecting patterns with probability $\lambda^{|d_s^{c_i}|}$. The probability of no activation for non-detecting patterns, on the other hand, is quantified as $(1-\lambda)^{|nd_s^{c_i}|}$. Finally, noise produces an error with probability μ. A signature is free of random noise effect if no transients were active in the p patterns. The probability of a noise-free signature is $(1-\mu)^p$.

As a result, the conditional probability for signature s under the assumption of c_i, with c_i being able to explain the signature, is:

$$P(s_i = s|c = c_i) = \lambda^{d_s^{c_i}} * (1-\lambda)^{nd_s^{c_i}} * (1-\mu)^p$$

If the candidate is not able to explain the signature, then its effect must have been masked by noise. Regardless of the fault being activated or not, the probability for an unexplained signature under the assumption of c_i is:

$$P(s_i = s|c = c_i) = 1 - (1-\mu)^p$$

Under the assumption that no critical fault is present in the circuit, i.e., candidate c_{noise} is considered,

$$P(s_i = s_{ref}|c = c_{noise}) = (1-\mu)^p$$

and

$$P(s_i \neq s_{ref}|c = c_{noise}) = 1 - (1-\mu)^p$$

express the probability that the signature is correct and incorrect, respectively, assuming the presence of background noise.

Once the structure of the Bayesian network is ready and annotated with the necessary conditional probabilities, a query can be performed to inquire about the posterior belief in the values of variable c: $P(c|s_i = s_i^{observed} \forall i)$.

In this work, we assume that no information is available about previous occurrences. Thus, both for the fault node and for the signature nodes, a uniform probability is set as the a priori probability. The a priori probability for each value of the fault variable is $1/(d+1)$. All nodes representing test sessions are evidence nodes, since the result of test is known and the observed value of the signature can be set to 1 in the vector. The nodes representing the test sessions S will only propagate λ-messages up to the fault node. The fault node receives all the messages and calculates the final belief according to the formulas presented previously. The fact that this is such a naive structure allows to use the simple message-passing algorithm, and simplifies the usual complexity of BNs, which is generally NP-hard, to linear for this case.

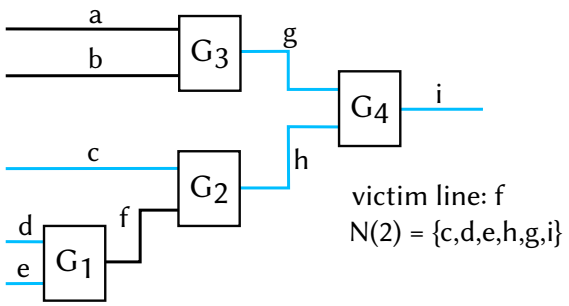
5.5 Experimental validation

For the validation of the method, three different scenarios are considered. In the first, only intermittent faults have been injected. In the second, only transient noise is assumed. For the third case, intermittent faults have been injected along with background noise.

5.5.1 Injected faults

The injected intermittent faults are modeled as high-frequency power droop. Power droop is a phenomenon which affects a node in the circuit depending on the activity of the neighboring lines, and has been aggravated with new technologies. Transistors have been consistently scaled down, and each power segment now feeds an increasing number of devices. If a high number of neighboring lines is switching in the same direction as the victim line, then the victim line may suffer of power starvation and propagate an incorrect value.

For a certain victim node v, we define the neighborhood $N(r)$ topologically, and based on a radius r. The radius defines the maximum topological distance for neighbors, i.e., any neighbor with distance lower than r is considered a neighbor. Figure 5.6 depicts the neighborhood of victim v for $r = 2$.



▲ **Figure 5.6** — Topological neighborhood N(2) of victim line f

The activation of the fault is based on two conditions: an activation rate $\lambda_{activation}$, and that a minimum number of neighbors $switch_{min}$ switches in the same direction as v. In CLF notation, the fault model would be expressed as $v \wedge (switches(N(r)) > switch_{min} \wedge z < \lambda_{activation})$, where z is a random number, $z \in [0, 1]$.

Transient noise is modeled as a random activation at a node chosen randomly. The activation lasts no longer than one cycle. The victim line is flipped for one cycle with a certain probability $\mu_{activation}$. This results in an error rate μ_{error}, which is the product of the activation and propagation rates, i.e., $\mu_{error} = \mu_{activation} * \mu_{propagation}$. The activation rates can be adjusted to represent different scenarios: transient errors caused by radiation happen less frequently than when caused by noise from the power supply or from variations.

5.5.2 Intermittent fault classification

In the first set of experiments, only intermittent faults were injected. To ensure a variety of activation profiles among the intermittent faults, different combinations of neighborhood radius, switching activity threshold and activation rates were considered. Experiments were conducted for radius $r = 1$, $r = 2$, $r = 3$. The number of neighbors which had to switch for the fault to be activated was determined by a percentage. Values 15%, 30% and 50% were considered, and the absolute threshold was calculated accordingly. To introduce non-determinism in the behavior, the intermittent activation rate $\lambda_{activation}$ was set to 0.5. As a result, the final intermittent activation rate λ_{exp} ranges from values higher than 0.5 to rates in the order of 10^{-13}.

▲ **Table 5.2** — Classification results for intermittents with $R_{max} = 2$ and $T_{max} = 10$.

Circuit	E_F	$CF_{immediate}$	CF_{Bayes}	CF	Accuracy
p45k	873	833	11	844	0.967
p100k	947	918	13	931	0.983
p141k	898	870	7	877	0.977
p239k	966	945	15	960	0.994
p259k	912	890	13	903	0.990
p267k	997	970	10	980	0.983
p269k	890	860	12	872	0.980
p279k	817	779	17	796	0.974
p286k	790	762	10	772	0.977
p295k	776	723	17	740	0.954

The number of maximum faulty sessions was set to 10. For a transient error $\mu = 10^{-5}$, this ensures that the possibility of observing more than 10 faulty signatures lies in the order of 10^{-13}.

Table 5.2 shows the results of the adaptive test and diagnosis approach combined with Bayesian reasoning. The first column indicates the circuit to which the experiments correspond. Given that no diagnosis is possible if no error was observed, only experiments with failure are considered. Their number is given in column 2, which is labeled E_F. According to the criteria for immediate classification, some faults could be identified as critical ($CF_{immediate}$) from the test results (column 3). For the rest, Bayesian inference was deployed. The number of faults identified as critical by the Bayesian analysis is indicated in column CF_{Bayes}. As a result, a total number of failures identified as critical is reported in column CF, which gives as result the classification accuracy reported in column 6. The accuracy of the method is over 0.95 for all circuits. The combination of the immediate and Bayesian analysis is hence able to identify the critical faults.

A closer look into the classification of the circuits which undergo Bayesian classification draws some interesting conclusions. Table 5.3 shows the classification of the experiments which needed to undergo Bayesian classification. The first column denotes the circuit, and the second, $E_{Bayesian}$, indicates the number of experiments analyzed with the network. Then, the table distinguishes between those experiments which caused only 1 faulty session, and those in which more than 1 faulty session is observed. For both cases the total number of experiments E, the number of classified as transient T and those classified as intermittent I are stated.

The accuracy of Bayesian reasoning is very high for those cases in which two or more

▲ **Table 5.3** — Bayesian classification for intermittents.

Circuit	$E_{Bayesian}$	1 faulty session			2 to 9 faulty sessions			
		E	T	I	E	T	I	Accuracy
p45k	40	29	29	0	11	0	11	1.000
p100k	29	15	15	0	14	1	13	0.929
p141k	28	21	21	0	7	0	7	1.000
p239k	21	6	6	0	15	0	15	1.000
p259k	22	9	9	0	13	0	13	1.000
p267k	27	17	17	0	10	0	10	1.000
p269k	30	18	18	0	12	0	12	1.000
p279k	38	21	21	0	17	0	17	1.000
p286k	28	18	18	0	10	0	10	1.000
p295k	53	34	34	0	19	2	17	1.000

sessions were faulty. However, for only one session, the network fails to classify the intermittents correctly. Although incorrect, this is a reasonable result because the considered intermittents have an extremely low activation rate, thus behaving de facto as transients.

5.5.3 Transient fault classification

Transient fault experiments were conducted with varying activation rates μ_{exp}. Values $2 * 10^{-3}$, $2 * 10^{-4}$ and $2 * 10^{-5}$ are considered. These probabilities are higher than the transient rates reported for radiation. However, they have been included because they reflect transient errors caused by noise in power supply, or because of parameter variations. Higher rates of transient faults make it harder to differentiate between problems caused by an intermittent fault with low activation rate and a transient problem.

The injected fault rates μ_{exp} are chosen to assess the robustness of the method when μ_{exp} and μ do not match. However, if such high transient activation rates are expected, the maximum number of faulty sessions must be adjusted accordingly. In this case, we assume a μ of 10^{-4}, and so the maximum number of faulty signatures is set to 21.

Table 5.4 summarizes the results of the experiments. The first column indicates the circuit for which the experiments were performed. The second one is the number of experiments with failures, i.e. experiments for which a problem was detected and classification was performed. The number of faults identified as critical is stated in

▲ **Table 5.4** — Classification results for transients with $R_{max} = 2$ and $T_{max} = 21$.

Circuit	E_F	$CF_{immediate}$	CF_{Bayes}	CF	Accuracy
p45k	129	43	1	44	0.659
p100k	125	15	4	19	0.848
p141k	124	22	1	23	0.815
p239k	127	36	1	37	0.709
p259k	126	15	2	17	0.865
p267k	126	22	1	23	0.817
p269k	127	32	2	34	0.732
p279k	124	31	0	31	0.750
p286k	124	14	0	14	0.887
p295k	127	14	0	14	0.890

▲ **Table 5.5** — Bayesian classification for transients.

Circuit	$E_{Bayesian}$	1 faulty session			2 to 9 faulty sessions			
		E	T	I	E	T	I	Accuracy
p45k	86	6	6	0	80	79	1	0.988
p100k	110	6	6	0	104	100	4	0.962
p141k	102	6	5	1	96	96	0	1.000
p239k	91	3	3	0	88	87	1	0.989
p259k	111	4	4	0	107	105	2	0.981
p267k	104	2	1	1	102	102	0	1.000
p269k	95	5	4	1	90	89	1	0.989
p279k	93	1	1	0	92	92	0	1.000
p286k	110	6	6	0	104	104	0	1.000
p295k	113	6	6	0	107	107	0	1.000

column $CF_{immediate}$. The number of faults classified as intermittent by the Bayesian network is specified in the fourth column. Finally, the overall number of critical failures and the final classification precision are indicated in the last two columns.

The overall accuracy is now lower than for the previous experiments, due to the high number of sessions with a faulty signature in the rollback session. A closer analysis of the results of the Bayesian classification is shown in table 5.5. In it, the experiments $E_{Bayesian}$ for which Bayesian classification was deployed are divided into two categories: experiments with only one faulty session on the left, and those with more than one session up to the threshold. For each category, the number of experiments E and those classified as transient T and intermittent I are specified.

The reported results can be explained by the detectability of the considered intermittent

▲ **Table 5.6** — Classification results for intermittents in presence of background noise with $R_{max} = 2$ and $T_{max} = 10$.

Circuit	E_F	$CF_{immediate}$	CF_{Bayes}	CF	Accuracy
p45k	993	878	20	898	0.904
p100k	1744	1643	47	1690	0.969
p141k	978	924	30	954	0.975
p239k	1019	992	12	1004	0.985
p259k	934	901	13	914	0.979
p267k	988	945	27	972	0.984
p269k	987	951	19	970	0.983
p279k	971	865	55	910	0.947
p286k	834	779	19	798	0.957
p295k	967	791	28	819	0.847

candidates: if a fault is hard to detect (very few patterns would propagate an error to the output), then even a few failing patterns will suffice for the Bayesian network to conclude that it was an intermittent fault. The table also shows that the accuracy of the Bayesian classification is much higher than the overall classification reported in table 5.2. The cause for this behavior is the estimation of the transient rate in the network, which is, as already explained, different than the injection rate in some of the experiments. However, this is easy to solve: if the number of repetitions is increased, then fault-free signatures will appear at the end of the run and the pessimistic effect of immediate classification is mitigated.

5.5.4 Intermittent fault with background noise classification

The last set of experiments combines intermittent and transient faults. The aim is to identify the critical faults even if the presence of noise distortions the results of test. To estimate the precision of the adaptive test approach in this case, the neighborhood radius and intermittent activation rates from section 5.5.2 have been injected concurrently to transient faults with the rates specified in 5.5.3.

The results show again a high overall accuracy. However, the Bayesian classification performed worse than before: table 5.7 shows the results of the classification for experiments with less faulty sessions than the threshold. These results are the result of having high-rate noise masking intermittents with a low activation rate or which are hard to detect, thus causing a very low number of failing sessions which cannot be distinguished from those caused by the noise. This problem cannot be solved by any

▲ **Table 5.7** — Bayesian classification for intermittents in presence of background noise.

Circuit	$E_{Bayesian}$	1 faulty session			2 to 9 faulty sessions			
		E	T	I	E	T	I	Accuracy
p45k	115	8	8	0	107	87	20	0.187
p100k	101	7	7	0	94	47	47	0.500
p141k	54	2	2	0	52	22	30	0.577
p239k	27	4	4	0	23	11	12	0.522
p259k	33	4	4	0	29	16	13	0.448
p267k	43	2	2	0	41	14	27	0.659
p269k	36	4	4	0	32	13	19	0.594
p279k	106	3	3	0	103	48	55	0.534
p286k	55	8	8	0	47	28	19	0.404
p295k	176	10	10	0	166	138	28	0.169

method, and remains the corner case of intermittent/transient distinction. Nevertheless, the Bayesian classification is able to distinguish about half of the cases, which, combined with the adaptive test and diagnosis approach renders an accuracy of over 84%.

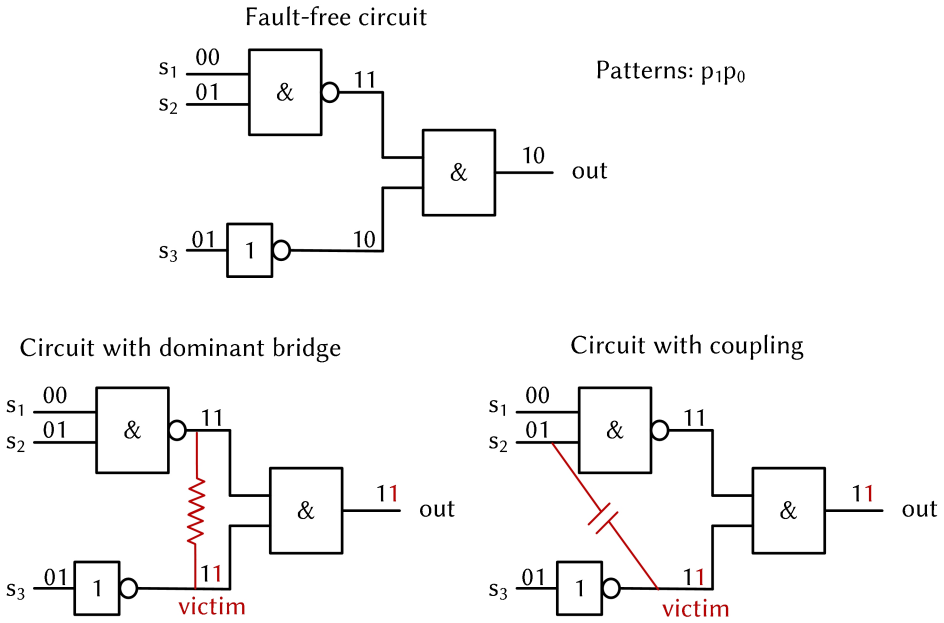
CHAPTER 6

Neural networks for defect classification

Approaches for logic diagnosis are able to extract information from the test results to locate the fault. However, they do not provide sufficient information about the fault mechanism. Chapter 5 explained how chips affected by noise are ruled out. Among the remaining chips, affected by a critical fault, some could be affected by a systematic problem. However, there is not enough information to distinguish fault classes and prioritize for PFA. The test output provides only little insight into the underlying root cause: faults which belong to different classes may cause the same test response [Maly03].

Figure 6.1 shows a small sample circuit, in which this effect can be observed. The upper part of the figure depicts the fault-free instance of the circuit, where the patterns $p_0 = 000$ and $p_1 = 011$ are simulated. The correct output is computed by the fault-free circuit. After applying p_0, a 0 is observed at the output. For p_1, the correct output value is 1. The values are indicated on the picture as 10 at the output value. In the bottom left instance, the circuit is affected by a bridge defect. Due to this unwanted connection (depicted as a resistor in red), the victim line in the bottom is driven to an incorrect value by the aggressor. This causes a faulty output to be sampled in the second test cycle, and so the observed sequence is 11. The same behavior is observed in case of

crosstalk between two lines, where the bottom line is again the victim. If both signals switch in opposite directions, the transition at the victim line gets delayed by a certain amount of time d. When the induced delay is bigger than the slack of the node, then an incorrect value is propagated to the output. In the example, it once again leads to an output sequence of 11.



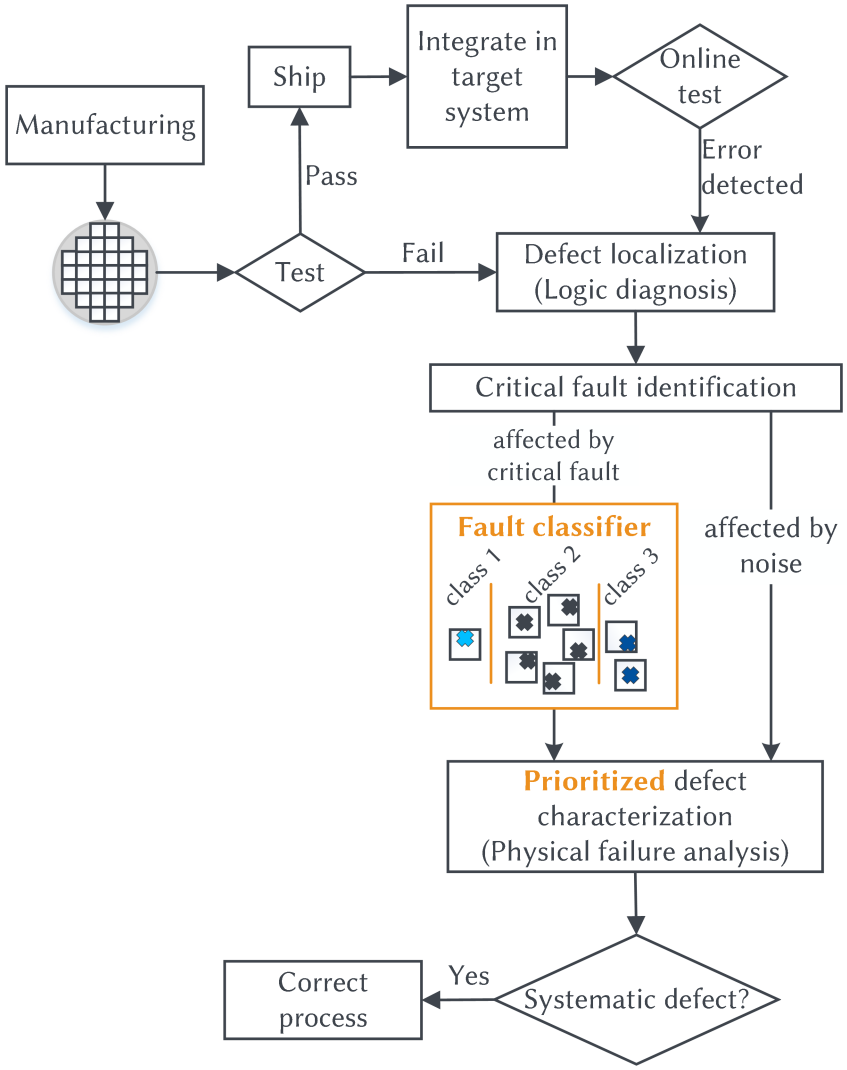
▲ **Figure 6.1** — Different faults cause same output patterns

An output sequence of 11 is good news from the test point of view. Because it differs from the expected fault free response 10, the problem is detected. However, from the output values no deduction can be made about the underlying root cause. This limitation is overcome in manufacturing test by applying a costly second diagnostic test to the affected parts, and then physical failure analysis. This process is both resources and time consuming, and may lead to the no-trouble-found problem if the activation conditions are not fulfilled during the second diagnostic pass.

Some techniques have been developed for fault identification, as described in section 3.5. Their accuracy is limited for intermittent faults. However, intermittent faults appear often, since the reference fault model rarely reflects the behavior of the defect deterministically. This reduces the applicability of these methods for real designs. In this chapter, a technique for fault classification is presented, and can be applied to

circuits affected by critical defects. It was first presented in [Rodri16]. This section expands the scenarios and applicability of the method.

Figure 6.2 depicts the flow now completed with the fault classification approach.

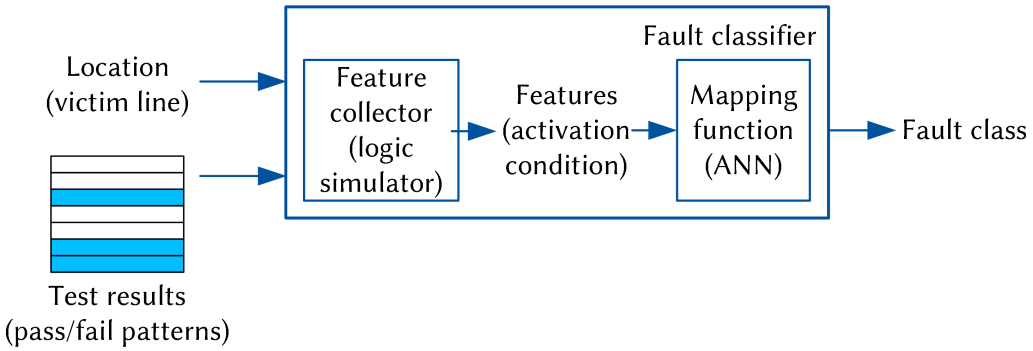


▲ **Figure 6.2** — Complete diagnosis flow with early fault class identification

The benefit is twofold. The proposed modification of the traditional flow uses the defect classifier as an early warning issuer. On the one hand, it is a fast way to point out systematic defects, thus allowing prioritizing the application of later costly analysis techniques. But manufacturing test is not the only scenario where the method can be applied. An analysis of test responses of field returns may point to timing problems, guiding the analysis process. Thus, it helps reduce the no-trouble-found problem, since

it identifies the fault from the online test data. The disappearance of stress conditions when removing the chip from the system does not affect our method, since the test output was collected in presence of stress conditions.

The proposed method receives as input the test result and the result of logic diagnosis, i.e., the fault location or *victim line*. Due to the lack of information to be extracted from the test responses alone, the idea behind the method is to monitor some representative values from the simulation which give a hint about the *activation* conditions. Some carefully chosen values are gathered which help to identify the cause of the observed errors. Figure 6.3 shows the structure of the classifier. It is formed by two main blocks: a so-called *feature collector* and a mapping function. The feature collector receives the inputs and contains a logic simulator and a model of the design. It simulates the test patterns and obtains the set of relevant numbers or *features* to identify the activation condition. The second block, which contains an artificial neural network, maps the features to a fault class. This chapter explains in detail which fault classes were considered, how the relevant values for classification were identified, and the internal structure of both blocks which conform the classifier.



▲ **Figure 6.3** — Structure of the fault classifier

6.1 Faults to be distinguished

The proposed method is conditioned by the fault classes to be considered. In this work, a selection of categories is proposed which covers some of the usual problems in digital manufacturing. However, the method can be easily extended for other fault classes. The considered categories in this work include problems that may appear both during manufacturing or as a consequence of aging. In deep submicron technology nodes,

the interconnect density is increasing. The size of interconnects is decreasing, and the imperfections in the manufacturing process make them prone to errors such as bridges. Moreover, due to phenomena such as electromigration, unwanted connections may appear as a result of degradation after the fault-free chip has been shipped to the customer. In CMOS technology, the dominant bridges are widely used as a fault model. However, the dominant bridge model considers only the case in which the pull-up or -down network is stronger. The situation in which the bridge causes intermediate voltages in the affected lines, which may be interpreted differently by the succeeding gates, is better represented by byzantine bridges. In this work, all three classes have been included: b_{and}, b_{or} and b_{byz}. Recall that the activation and flip of the affected line(s) can be expressed in CLF calculus. For the b_{and} class, given a victim s_v and an aggressor s_a, the model is $s_v \oplus ((s_v \oplus s_a) \wedge s_v)$. Analogously, for b_{or} faults the expression is $s_v \oplus ((s_v \oplus s_a) \wedge \neg s_v)$. The lines s_1 and s_2 affected by byzantine bridges are randomly flipped whenever the condition $s_1 \oplus s_2$ is fulfilled. For the class b_{byz}, the specific value of each line is irrelevant and does not determine if the fault is activated.

Interconnects are subject to faults also caused by the activity in the neighborhood. Capacitive coupling or crosstalk also poses a challenge since its activation happens only in very specific circumstances. The dynamic nature of these faults and the fact that they are not visual, i.e., the layout does not necessarily deviate from the specification, make these problems even harder to identify during physical analysis. The possibility of deploying a fast fault classifier that could accelerate physical analysis even in these particularly hard cases makes the inclusion of crosstalk-induced delay (ct) relevant for this work. Crosstalk-induced delay can be expressed logically for an aggressor s_a and victim s_v. As introduced in 2.2, the CLF expression that considers the values in the previous time unit (s_v^{-1} and s_a^{-1} for victim and aggressor, respectively) is $s_v \oplus ((s_v \oplus s_v^{-1}) \wedge (s_a \oplus s_a^{-1}) \wedge (s_v \oplus s_a))$.

Faults in the cells of the circuit must also be taken into consideration. Cells may deviate from their intended timing specification both due to imperfections and variations in the manufacturing process, or due to wearout induced by NBTI or HCI. The identification of this category is extremely important not only for early identification of systematic defects, but also for those observed behaviours which are not always repeatable in the lab and any additional information to pinpoint the problem is useful. Hence, the gate transition fault model is also included: both the slow to rise s_{rise} and the slow to fall s_{fall} classes. They can be represented in CLF form as $s_v \oplus [(s_v \oplus s_v^{-1}) \wedge s_v]$ for the

rising case and $s_v \oplus [(s_v \oplus s_v^{-1}) \wedge \neg s_v]$ for the falling case.

This work considers the set of classes L as the possible outcome of the tool, where $L = \{s_{rise}, s_{fall}, b_{byz}, b_{and}, b_{or}, ct\}$

6.2 Features

Given that the test outcome provides little information to identify the underlying fault class, the main focus of the approach is the identification of the activation mechanism. In particular, certain values in logic simulation which are related to the fault activation are monitored. These numbers, also referred to as *features*, will form a vector which is the input to our classification function. To ensure the success of the approach, the features must be representative of the labels. Their quality is one of the main factors which impact the accuracy of the method.

Feature collection requires the patterns applied to the circuit to be simulated once. For this purpose, zero-delay logic simulation is used. Logic simulation has been optimized and accelerated and can be deployed even for big circuits with negligible costs. Since logic simulation is the most expensive step for the fault classification of a failing chip, this ensures that the overhead introduced is extremely low.

Manufacturing and even online test know beforehand the input stimuli to be applied. This implies that, based on test results, the test set T can be divided in two sets: the failing patterns T_f and the passing patterns T_p. T_f is the set of test patterns for which the observed response deviates from the fault-free one. T_p is the set of patterns for which the observed response equals the expected one. When an online error detection technique is applied, and it only consists of a detector and not of a test pattern generator, only the faulty patterns can be recorded. In this case, the only known stimuli are the faulty patterns T_f.

6.2.1 Failing pattern information

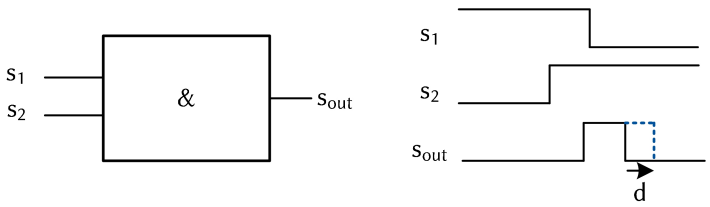
The set T_f is formed by the patterns for which an error was observed. The implication is that the fault was both activated and propagated to an observable output. This is an important characteristic of these patterns: relevant information about the activation mechanism can be extracted here.

The fault is triggered if the activation condition is fulfilled. The activations of the different fault classes for a victim line s_v and an aggressor s_a are summarized in table 6.1. In previous sections, the activation condition of all classes was expressed as a product of terms. Table 6.1 presents the terms aligned in different columns, according to the lines involved in the condition. Terms which depend only on the affected signal belong to the category *victim line*. A further distinction may be made: some conditions only depend on the value of the signal during the considered time step, while others require a transition. The table shows the victim line conditions separated in static and dynamic. This is the case of the activation condition for a slow to rise gate, whose activation is subject to fulfilling $(s_v \oplus s_v^{-1}) \wedge s_v$, that is, the static condition s_v and the dynamic transition required, $s_v \oplus s_v^{-1}$. All variables in the condition represent exclusively the victim line. Other conditions involve the activity of an aggressor line, such as $s_a \oplus s_a^{-1}$ for crosstalk. Finally, some involve the values in the neighborhood with respect to the victim line, such as $s_v \oplus s_a$ in bridges. Again, the conditions which depend on the neighborhood may be static or dynamic. Table 6.1 shows that for four classes, the value of the victim line is relevant for its activation. Three of the four classes require a transition in the victim line, while four depend on transitions in the neighborhood to be activated. Only one class requires dynamic activation conditions in the neighborhood to trigger the fault. Characteristic values of the behavior of the victim line and that of the neighboring lines may serve well as features for this classification problem.

▼ **Table 6.1** — Activation conditions for all fault classes

	victim line conditions		neighborhood conditions	
	static	dynamic	static	dynamic
b_{and}	s_v	-	$s_v \oplus s_a$	-
b_{or}	$\neg s_v$	-	$s_v \oplus s_a$	-
b_{byz}	-	-	$s_v \oplus s_a$	-
s_{rise}	s_v	$s_v \oplus s_v^{-1}$	-	-
s_{fall}	$\neg s_v$	$s_v \oplus s_v^{-1}$	-	-
ct	-	$s_v \oplus s_v^{-1}$	$s_v \oplus s_a$	$s_a \oplus s_a^{-1}$

An additional consideration must be made. The feature extractor is based on a zero-delay logic simulator. As a result, some transitions that can trigger the fault, such as glitches, are not accurately represented in a zero-delay logic simulator. For instance, assume a circuit in which the *and* gate depicted in figure 6.4, fed by signals s_1 and s_2.



▲ **Figure 6.4** — Driving gate inputs

A given failing pattern p_i propagates values $s_1 = 0$ and $s_2 = 1$, while the preceding (not necessarily failing) pattern p_{i-1} drove $s_1 = 1$ and $s_2 = 0$. The expectation is that, since both inputs flipped, the output value remains stable at 0. However, because differences in the paths from the primary inputs to each of the signals, the new value may arrive at different times, causing a glitch. In some fault classes, for instance slow gates which induce a delay d as shown in the figure, this may pose a problem if the signal is delayed to a later time than its slack allows. The inclusion of information about the inputs of the driving gate is related to the identification of transitions in the victim line.

Finally, the effect of the fault must also be considered. Byzantine bridges involve two signals as potential victims. In the case in which both lines are flipped, the victim line provided by logic diagnosis cannot have caused all the erroneous bits observed. Hence, the output of the test must also be observed.

To sum up, four aspects can be considered relevant to identify the root cause: the activity in the victim line, the activity in the inputs of the driving gate, the activity in the neighborhood and the test outcome.

This section presents the information gathered from each of this aspects and how it is transformed into features.

Victim line information

The information about the victim line is the first considered aspect. The activation conditions of the considered classes include static and dynamic variants. If the line is at 0 or 1 during the fault-free simulation characterizes the activation of the fault. The transitions in the victim line are also relevant. The condition can be more fine-grained and distinguish between rising and falling transitions, since $(s_v \oplus s_v^{-1}) \wedge s_v$ can also be expressed as $s_v \wedge \neg s_v^{-1}$. Analogously, $(s_v \oplus s_v^{-1}) \wedge \neg s_v$ equals $\neg s_v \wedge s_v^{-1}$. However, some challenges need to be solved to condense this information in the form of features.

The features vector has a fixed length. All collected information must be represented in a vector of numbers whose length does not vary. The first challenge appears when the activity in the victim line needs to be condensed: the number of failing patterns is a priori unknown and changes with every test set and fault combination. Thus, collecting the value of the victim line for every failing pattern and feeding that vector as input to the ANN is not feasible.

Instead, different sets of patterns are defined based on the activity in the victim line. Let us define $T_f^0 \subset T_f$, where T_f^0 is formed by those failing patterns for which the victim line has a value of 0 in the fault-free simulation, i.e. $T_f^0 = \{t_i \in T_f | vl(t_i) = 0\}$. Analogously for 1: $T_f^1 = \{t_i \in T_f | vl(t_i) = 1\}$.

T_f^{fall} is the set of failing patterns for which the victim line was observed to have a falling transition. Formally, $T_f^{fall} = \{t_i \in T_f | vl(t_i) = 0 \wedge vl(t_{i-1}) = 1\}$. Note that $t_i \in T_f$, but not necessarily $t_{i-1} \in T_f$. Analogously, we define $T_f^{rise}1$ as the set of failing patterns for which the victim line had a rising transition, i.e., $T_f^{rise} = \{t_i \in T_f | vl(t_i) = 1 \wedge vl(t_{i-1}) = 0\}$.

The information relevant to the victim line is compressed in the following numbers:

- victim line at 0: $vl@0 = |T_f^0|/|T_f|$
- victim line at 1: $vl@1 = |T_f^1|/|T_f|$
- rising transitions at victim line: $vl_{rise} : |T_f^{rise}|/|T_f|$
- falling transitions at victim line: $vl_{fall} : |T_f^{fall}|/|T_f|$

The victim line features provide relevant information for some of the fault classes. The relevant features to identify each fault class correspond to the relevant characteristics marked in Table 6.1 for each fault class. If the line had a value of 0 in the fault-free simulation of the failing patterns then the fault mechanism must have driven the line to 1. This hints towards the presence of faults such as a slow-to-fall or an or-bridge problem. Hence, a high ratio of such patterns would strongly indicate that the culprit is one of the two classes. Analogously, should $vl@1$ have a high value, it is an indication for slow-to-rise faults or and-bridges.

If for T_f many patterns are observed to induce a falling transition in the fault-free simulation, this means that the fault activation mechanism possibly has to do with the signal switching. For example, while $vl@1$ was an indication of slow-to-rise faults

or and-bridges, if vl_{rise} is also high then the belief in the presence of a slow-to-rise problem increases. On the other hand, since an and-bridge is activated statically, a high vl_{rise} is not unrealistic for a bridge, but has a low probability of occurring for many of such faults. The feature vl_{fall} would indicate the presence of a slow-to-fall gate.

Driving gate inputs

The inputs of the driving gate also provide important information about the activation of the fault. They allow a more fine-grained identification of possible glitches, hence reducing the noise introduced by the abstraction level of the simulator representation.

The sets of patterns for which at least one input signal changed but the output was expected to remain stable at 0 or 1 are denoted T_{c0} and T_{c1}, respectively. Likewise, the sets of patterns for which the inputs were stable and the output remained at 0 or 1 are T_{s0} and T_{s1}. Finally, T_{tf} and T_{tr} are identified. These are the sets of patterns for which a change at the input caused a falling or rising transition at the output. Formally:

- $T_f^{c0} = \{t_i \in T_f | g(t_i) = g(t_{i-1}) = 0 \wedge I(t_i) \neq I(t_{i-1})\}$
- $T_f^{c1} = \{t_i \in T_f | g(t_i) = g(t_{i-1}) = 1 \wedge I(t_i) \neq I(t_{i-1})\}$
- $T_f^{s0} = \{t_i \in T_f | g(t_i) = g(t_{i-1}) = 0 \wedge I(t_i) = I(t_{i-1})\}$
- $T_f^{s1} = \{t_i \in T_f | g(t_i) = g(t_{i-1}) = 1 \wedge I(t_i) = I(t_{i-1})\}$
- $T_f^{tf} = \{t_i \in T_f | g(t_i) = 1 \wedge g(t_{i-1}) = 0 \wedge I(t_i) \neq I(t_{i-1})\}$
- $T_f^{tr} = \{t_i \in T_f | g(t_i) = 0 \wedge g(t_{i-1}) = 1 \wedge I(t_i) \neq (t_{i-1})\}$,

where $g(t_k)$ is the logic value at the output of gate g given pattern t_k, and $I(t_k)$ is the value at the input of the gate when pattern t_k is applied.

The cardinality of each set is then divided by the number of failing patterns to extract the features, i.e.,

$$\begin{aligned} f_f^{c0} &= |T_f^{c0}|/|T_f| \\ f_f^{c1} &= |T_f^{c1}|/|T_f| \\ f_f^{s0} &= |T_f^{s0}|/|T_f| \\ f_f^{s1} &= |T_f^{s1}|/|T_f| \\ f_f^{tf} &= |T_f^{tf}|/|T_f| \\ f_f^{tr} &= |T_f^{tr}|/|T_f|, \end{aligned}$$

are the values added to the feature vector.

These values identify transitions in the line, even those which are not represented in the zero-delay logic simulator. As a result, they identify the conditions for faults activated by a transition in the victim line. Features f_f^{s0} and f_f^{s1} have low values for faults activated by a transition. A slow-to-rise gate will have higher values for f_f^{tr} and f_f^{c1}. A slow-to-fall gate, on the other hand, is expected to have high f_f^{tf} and f_f^{c0}.

Activity in the neighborhood

Another important source of information is the switching activity in the neighboring lines. As noted in table 6.1, the relevant information comprises both the static and the dynamic values. The static information is defined in relation to the victim line, that is, in the form of $s_v \oplus s_a$. Hence, the distance of the aggressor neighbor is the relevant characteristic. The dynamic part requires transitions in the aggressor line, $s_a \oplus s_a^{-1}$. However, the aggressor line is unknown a priori. The tool cannot sample the values of a certain line and check if they match the activation conditions. The number of neighbors varies with the victim line, so the features that represent the activity in the neighborhood must be reduced from a matrix with dimensions $|n(v_l)| \cdot |T_f|$ to a vector of fixed length.

For the switching activity, this is achieved by calculating for every neighbor the number of opposite transitions with respect to the victim line. First, the set of lines $N(v_l)$ that conform the neighborhood of the victim line is extracted from the layout information. For a neighbor $n \in N(v_l)$, $n(t)$ represents the logic value for pattern t. Then, for every neighbor $n \in N(v_l)$, the set of failing test patterns for which a transition was observed is calculated:

$$trans_f^n = |\{t_i \in T_f | (vl(t_i) \neq n(t_i)) \wedge (vl(t_{i-1}) \neq vl(t_i)) \wedge (n(t_{i-1}) \neq n(t_i))\}|.$$

The feature f_{trans} is the maximum rate of transitions observed in the neighborhood:

$$f_f^{trans} = \frac{max\{trans_f^n | n \in N(v_l)\}}{|T_f|}.$$

A high value for f_{trans} is a strong indicator of the presence of crosstalk effects.

Another feature, f_f^{dist}, monitors the values of the neighbors. f_f^{dist} is a measure of distance between the values of the neighboring lines and the victim line. In other words,

it tries to identify the activation condition of a bridging fault, which requires one of the neighbors to have opposite value. The distance for a certain neighbor n is defined as

$$dist_f^n = \frac{|\{t_i \in T_f | vl(t_i) \neq n(t_i)\}|}{T_f}.$$

The feature is then calculated as the average for the complete neighborhood $N(v_l)$,

$$f_f^{dist} = \frac{\sum_{n \in N(v_l)} dist_f^n}{|N(v_l)|}.$$

Test outcome

Finally, the test outcome may provide some insight into the underlying cause. Logic diagnosis compares the observed results in test with the expected responses of the fault machine. We will define the *unexplained set* T_u as the set of patterns whose observed response cannot be explained by a flip in the victim line. The feature $u_f = |T_u|/|T_f|$ also forms part of our feature vector. It indicates the presence of a fault for which there is more than one culprit, i.e., a byzantine bridge.

6.2.2 Passing pattern information

Passing patterns are those for which the tested circuit did not exhibit faulty behavior, i.e., the fault was either not activated or not propagated. The passing patterns set is denoted as T_p. Extracting values from T_p and comparing them to that of T_f indicates the presence or absence of some fault types. The expectation is that, for each fault class, the columns marked as irrelevant (–) in table 6.1 will keep a random distribution. However, the relevant features for each fault class will present different values for the T_p set.

For example, a byzantine bridge is independent of the value of the signal. Its activation condition is that the bridged signals have opposite values, but whether the victim line is 0 or 1 is irrelevant. If a set of patterns T_p^0 is defined so that $T_p^0 = \{t \in T_p | vl(t) = 0\}$, the feature $vl@0^p = |T_p^0|/|T_p|$ can also be extracted. For the byzantine bridge fault class, the distributions of $vl@0$ and $vl@0^p$ are independent. However, for the or-bridge class there is no independence: $vl@0$ is expected to be substantially higher than $vl@0^p$. On the other hand, for the and-bridge class $vl@0$ is expected to be low, while $vl@0^p$ may take any value.

The mapping function will be fed with analogous features to those described for the failing patterns. Except for the unexplained patterns, which have no correspondence in the fault-free set, the rest of the features are calculated as described in the previous section.

The following sets of patterns are defined:

- $T_p^0 = \{t_i \in T_p | vl(t_i) = 0\}$
- $T_p^1 = \{t_i \in T_p | vl(t_i) = 1\}$
- $T_p^{fall} = \{t \in T_p | vl(t_i) = 0 \wedge vl(t_{i-1}) = 1\}$
- $T_p^{rise} = \{t \in T_p | vl(t_i) = 1 \wedge vl(t_{i-1}) = 0\}$
- $T_p^{c0} = \{t_i \in T_p | g(t_i) = g(t_{i-1}) = 0 \wedge I(t_i) \neq I(t_{i-1})\}$
- $T_p^{c1} = \{t_i \in T_p | g(t_i) = g(t_{i-1}) = 1 \wedge I(t_i) \neq I(t_{i-1})\}$
- $T_p^{s0} = \{t_i \in T_p | g(t_i) = g(t_{i-1}) = 0 \wedge I(t_i) = I(t_{i-1})\}$
- $T_p^{s1} = \{t_i \in T_p | g(t_i) = g(t_{i-1}) = 1 \wedge I(t_i) = I(t_{i-1})\}$
- $T_p^{tf} = \{t_i \in T_p | g(t_i) = 1 \wedge g(t_{i-1}) = 0 \wedge I(t_i) \neq I(t_{i-1})\}$
- $T_p^{tr} = \{t_i \in T_p | g(t_i) = 0 \wedge g(t_{i-1}) = 1 \wedge I(t_i) \neq I(t_{i-1})\}$

to extract the set of features:

- $vl_p^0 = |T_p^0|/|T_p|$
- $vl_p^1 = |T_p^1|/|T_p|$
- $vl_p^{rise} : |T_p^{rise}|/|T_p|$
- $vl_p^{fall} : |T_p^{fall}|/|T_p|$
- $f_p^{c0} = |T_p^{c0}|/|T_f|$
- $f_p^{c1} = |T_p^{c1}|/|T_p|$
- $f_p^{s0} = |T_p^{s0}|/|T_p|$
- $f_p^{s1} = |T_p^{s1}|/|T_p|$
- $f_p^{tf} = |T_p^{tf}|/|T_p|$
- $f_p^{tr} = |T_p^{tr}|/|T_p|$

The features related to the distance and maximum rate of transitions in the neighborhood are also included, i.e.,

$$f_p^{dist} = \frac{\sum_{n \in N(v_l)} dist_p^n}{|N(v_l)|}$$

, where $N(v_l)$ is the physical neighborhood of the line and

$$dist_p^n = \frac{|\{t_i \in T_p | vl(t_i) \neq n(t_i)\}|}{T_p}.$$

Finally, $f_p^{trans} = \frac{max\{trans_p^n | n \in N(v_l)\}}{|T_p|}$, where $N(v_l)$ represents the lines in the neighborhood and

$$trans_p^n = |\{t_i \in T_p | vl(t_i) \neq n(t_i) \wedge vl(t_{i-1}) \neq vl(t_i) \wedge n(t_{i-1}) \neq n(t_i)\}|.$$

6.3 Artificial neural networks for fault classification

The definition of the problem we consider is the mapping of a feature vector to one of the six classes defined, i.e., we want to find a function $f_{class} : \mathbb{R}^f \longrightarrow l \in L$ where f is the number of features from 6.2 and L is the set of classes in 6.1. Fault classification is a supervised classification problem. Given that the classes are not linearly separable, it calls for a non-linear supervised classifier. Artificial neural networks (ANNs) [Bisho06] are well suited for the problem. By observing a correctly labelled data set, they can infer (or *learn*) complex functions which may even be non-linear. This section explains the different topologies (6.3.1) and learning strategies (6.3.2) of ANNs. Finally, subsection 6.3.4 explains the ANN deployed as fault classifier.

6.3.1 Artificial neural networks: structure

An ANN is a structure composed of several processing elements or neurons. A neuron transmits an impulse to the ones in the following layer. This impulse is a real number which receives the name of activation, and is based on the linear combination of the information received from preceding neurons. More specifically, a neuron is associated with a vector of real numbers known as weights. The neuron receives a vector of real numbers, which it multiplies element-wise with the weights vector, and then applies a non-linear function to the sum of the elements in the vector. This result is the activation of the neuron.

Neurons are organized in layers. There are three kinds of layers in an ANN: input, output and *hidden*. Input layers receive the input vector itself, formed by the features. The output layer encodes the prediction of the ANN. Hidden layers perform the non-linear transformation that allows a mapping between the features and the outputs.

The simplest ANN topology is a feedforward ANN such as depicted in figure 6.5. The features are provided to the input layer l_{in}, which propagates the values to the next layer, that is, the first hidden layer. The activation value of the input layer, a_{in}, is hence just the original feature vector X. A hidden layer l_h receives an *activation vector* of numbers, $a_{h-1} \in \mathbb{R}^{|lh-1|}$, from its predecessor layer l_{h-1}. The predecessor of the first hidden layer is the input layer. Each hidden layer l_h has a weights matrix, W_h, of size $|(l_{h-1})| \times |l_h|$, where each element $e_{j,k}$ corresponds to the weight $W_{l_h}(j,k)$ connecting neuron j in layer l_{h-1} and neuron k in layer l_h. The product of activation a_{h-1} and weight matrix W_h gives as result vector $v_l \in \mathbb{R}^{|l_h|}$. A non-linear function ϕ_h is applied to v_h, obtaining activation function a_l. Generally, for hidden layers a non-linear function such as the sigmoid or hyperbolic tangent is chosen. For the output layer, in a classification problem the function is the softmax, which transforms the outputs into probabilities. The final outcome of an ANN is obtained by applying this layer per layer from the input layer to the output:

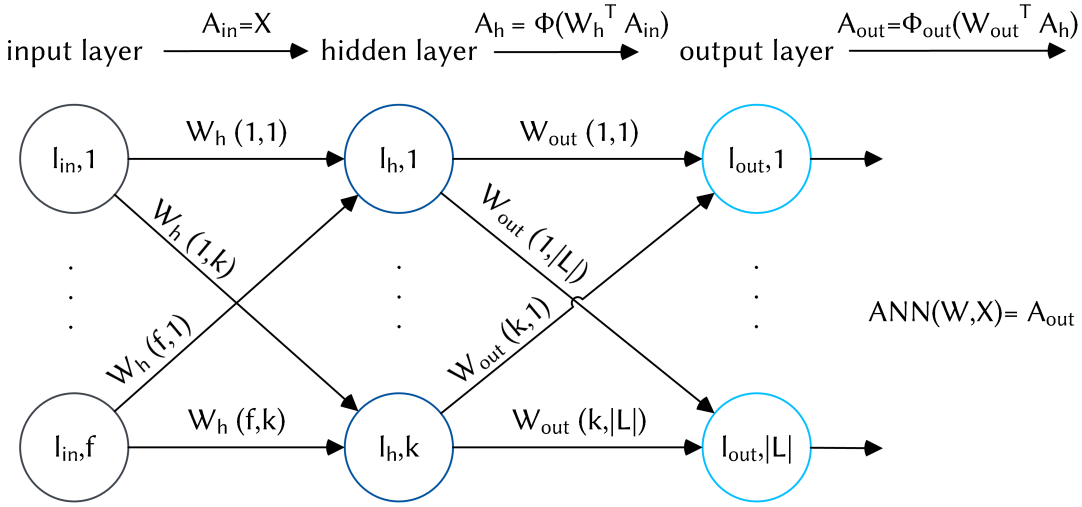
$$ANN(W,X) = (\phi_{out}(W_{out}^T \cdots ... \phi_2(W_2^T \cdots \phi_1(W_1^T \cdots X))...)$$

for a given input vector X and a configuration of weights W.

These basic ANNs can be slightly tuned by introducing a *sparsity* factor [Liu15a]: instead of allowing all connection to be activated, a restriction is imposed so that the activation of a neuron is penalized.

A higher number of hidden layers allows for more complex models. ANNs with a very high number of layers are referred to as *deep* networks, and have been reported to improve pattern recognition results [Krizh12] [Girsh14]. The depth is not only associated with a more complex mapping function between features and output. Deep networks are often taken advantage of to infer structures in the data set. A deep ANN with this goal is called an *autoencoder* [Goodf16]. Autoencoders have been successfully deployed in domains such as image recognition.

Convolutional networks [LeCun98] were designed to exploit the spatial properties of images. Two new types of hidden layers are introduced: feature maps and subsamplers.



▲ **Figure 6.5** — Multilayer feedforward ANN

In a feature map, each unit is fed by a small subset of n neighbors. Thus, each unit has n trainable coefficients. However, convolutional networks also use *weight sharing*, i.e., all the units in a feature map share the same n coefficients. Feature maps feed the second new type of hidden layer: the subsamplers. These layers are introduced to reduce the resolution of feature maps, thus reducing also the sensitivity of the output to particularities of the training set. The dimensionality reduction is achieved by averaging the incoming values of a small subset of neighbors. The underlying idea of convolutional networks is that, if a feature is detected, its exact location is irrelevant. For instance, if an edge is detected, whether the digit was written perfectly centered or not is unimportant. Its relative position to the other detected features is what determines the digit. The the LeNet-5 network, originally introduced in [LeCun98] to recognize the handwritten digits of the MNIST database, vastly improved the performance of previous existing ANNs. The first layers are feature maps and subsampling layers, followed by a regular ANN for classification.

Feedforward networks -both the simple multilayer ANN and the convolutional variant- include connections from a unit in a layer l_i to a unit in another layer l_j, where $j > i$. The functions these nets represent are static, in that they do not consider time dependencies. *Recurrent networks* (RNNs), however, include such dependencies in the representation: the activation of a processing unit can depend on the activation of any unit in the net in a past time. Hence, recurrent networks are different from feedforward networks in that cycles are allowed. RNNs have been successfully deployed for speech recognition

[Grave13].

6.3.2 Artificial neural networks: learning

To learn a function given a data set, the ANN may adjust the weights in order to produce an output as close as possible to the provided label for all elements in the set. In order to do so, the training set D_{train} is partitioned into a number b of *batches*, $D^0_{train}, ..., D^{b-1}_{train}$. The *error* or difference between the expected output and the one provided by the ANN for a data set S is calculated as:

$$E(W,S) = \sum_{(x,y)\in S} (y - ANN(W,x))^2$$

where W is the configuration of the weights in the ANN. The resulting error function only depends on the parameters of the model, i.e., the weights, for a given data set.

A more general function, refered to as *loss* function ($J(W)$), is defined. The loss function is the base of the learning process. In its simplest version, it only includes the error function. However, often additional terms are also integrated to ensure good learning properties.

The ANN adapts the weights W in the system iteratively. For each batch, first a feedforward pass is performed and the associated error is calculated. The error is propagated back to the inputs, and the weights are updated along the network. Then, the next batch is evaluated. This process is refered to as *backpropagation* [Pearl09]. Backpropagation is generally based on gradient descent. The weights are modified by taking steps in the direction in which the error decreases:

$$W = W - \alpha \nabla (J(W))$$

The direction in which the function decreases is the opposite of the gradient $\nabla(J(W))$, and the size of these learning steps is set by a parameter: the *learning rate* α. This process is repeated until a number of repetitions over the complete training set (*epochs*) is reached.

The values of the hyperparameters of an ANN impact its learning power. A low learning rate may cause the ANN to converge very slowly. A high learning rate, on the other hand, can cause the ANN to diverge.

A large number of epochs may cause the ANN to overfit: it infers a too complex function tailored to the training data set, but which extrapolates poorly. On the contrary, a low number of epochs may be the reason for underfitting. In this case, the model is too simple to map the features to the labels.

The size of the batches may also impact the learning process. The original gradient descent algorithm employed only one batch, i.e., $D_{train} = D^0_{train}$. However, depending on the initialization values of the weights, this approach may cause the algorithm to get stuck in a suboptimal local minimum. Varying the value of the batch size and performing an update of the weights after each batch is preferable to avoid local minima. This variant of gradient descent is referred to as *stochastic*.

To ensure better learning properties, the loss function may include penalization terms or modifications. One of the most commonly included is weight decay [Bisho06]. Weight decay consists of penalizing large absolute values in the weights to prevent overfitting. The loss function then includes a term which adds the absolute value of the weights (*l1 norm*) or its squared value (*l2 norm*) to the function. The multiplier γ indicates the weight of the penalization in the loss function. Hence, for a set of data S, the loss function including l2 weight penalization is calculated as:

$$J(W) = E(W, S) + \gamma \sum_{w \in W} w^2.$$

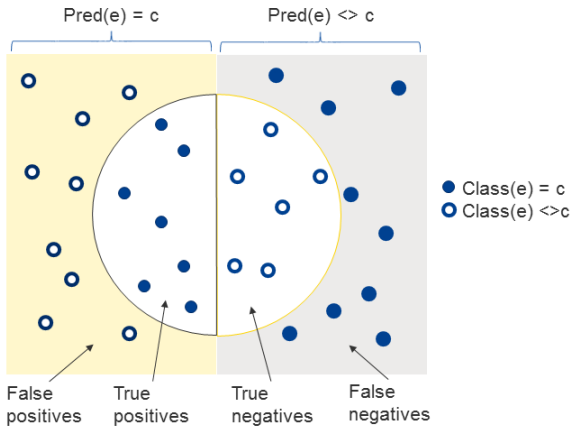
Another technique, which modifies the weight update function is to use *momentum* [Bisho06]. Momentum is a technique which combines the update of the weights with the direction of the gradient in the last steps. In this way, momentum prevents the descent of oscillating due to divergences in the training set (for instance, if in stochastic gradient descent the error of every batch does not decrease in the same direction).

6.3.3 Metrics for neural network based classifiers

The capability of the network to generalize when faced with new data is evaluated with a *test set* D_{test}. The test set is independent of the training set. If the performance of the network differs greatly from the training to the test set, then the ANN has a problem of overfitting. If the performance of the ANN for the training data set is low, then the problem faced is underfitting. The hyperparameters can be adjusted to mitigate both problems, from adapting the complexity of the model by increasing or reducing the number of layers to adding data to the training set. If the problem persists, the ANN must be discarded.

The evaluation of an ANN may be performed according to different criteria. The most popular metrics for multiclass classification problems are accuracy, error rate, precision and recall [Sokol09]. Their values reflect the ability of the ANN to identify the members of a class, and also the probability that the class was correctly predicted.

The outcome of the classification tool can be considered a true positive, true negative (t_n), false positive (f_p) or false negative (f_n). The category depends on the class of the experiment ($class(e)$) and the one predicted by the tool ($pred(e)$). For each class l, the count of true positives t_p^l reflects the number of experiments for which the tool predicts the correct class of the element, i.e., $t_p^l = |\{e|class(e) = pred(e) = l\}|$. True negatives of a class t_n^l are those experiments for which the tool predicts a different class than l for an experiment that does not belong to class l. Formally, $t_n^l = |\{e|class(e) \neq l \wedge pred(e) \neq l\}|$. The false categories correspond to the experiments for which the tool output was incorrect: false positives are those cases for which the tool incorrectly predicted class l ($f_p^l = |\{e|class(e) \neq l \wedge pred(e) = l\}|$), while false negatives are those for which the tool predicts a different class than c ($f_n^l = |\{e|class(e) = l \wedge pred(e) \neq l\}|$). The four categories are illustrated in figure 6.6 for class l.



▲ **Figure 6.6** — Classification outcome: prediction vs actual category (adapted from the matrix in [Sokol09])

The quality of the multiclass classification can be quantified based on these values. Accuracy measures the overall effectiveness of the classifier. The accuracy for class l is calculated as

$$A^l = \frac{t_p^l + t_n^l}{t_p^l + t_n^l + f_p^l + f_n^l}$$

and the overall accuracy is obtained by averaging the per-class accuracy:

$$A^L = \frac{\sum_{l \epsilon L} A^l}{|L|}.$$

Accuracy is a metric that remains invariant if the positives are changed with the negatives. It does not distinguish if the accuracy of the method is guaranteed because the tool correctly identifies the elements that belong to the class or rather because the number of true negatives is high. Hence, the metric precision is included too. The precision of the tool with respect to a class estimates the probability that an element belongs to a class l if the tool classified it as such. The formula for precision for a given class l is thus

$$P^l = \frac{t_p^l}{t_p^l + f_p^l}$$

and the overall precision of the classifier is calculated as

$$P^L = \frac{\sum_{l \epsilon L} t_p^l}{\sum_{l \epsilon L} t_p^l + f_p^l}.$$

Finally, the metric recall is introduced. Recall measures the effectiveness of the classifier to correctly extract as positives the elements which indeed belong to the class. The recall of the classifier for class l is

$$R^l = \frac{t_p^l}{t_p^l + f_n^l}$$

and it can be generalized for all classes as

$$R^L = \frac{\sum_{l \epsilon L} t_p^l}{\sum_{l \epsilon L} t_p^l + f_n^l}.$$

The combination of all three metrics is sensitive to changes in the number of positives and negatives, and also to the relation between false and true results. It thus provides a comprehensive view of the effectiveness of the classifier [Sokol09].

6.3.4 Artificial neural networks for fault classification

The aim of the approach presented in this chapter is to issue a warning if a fault class appears very often. Thus, the main goal of the critical fault identifier presented is to classify the underlying fault in one of the expected fault classes. The availability of data from past production in the manufacturing flow makes it possible to train the algorithm from previous observations. Hence, the problem at hand is a supervised classification problem.

Although more sophisticated flavours of ANNs exist, a simple feedforward ANN suffices for this purpose. The output is encoded with one neuron per class $c_i \in C$, described in section 6.1. Output neurons are activated by a softmax function, and thus the output of the network can be read as a vector of probabilities. The final output of each neuron in the output layer is the probability that the fault class is the one corresponding to the neuron. The label in the training set is encoded as a one-hot vector of length C.

Two different sets of features are considered: only failing information and both failing and passing. Given that the features, previously introduced in subsection 6.2.1 and subsection 6.2.2 are already normalized and belong in the range $[0, 1]$, no further manipulation is performed on them.

The optimizer algorithm is Adadelta [Zeile12], which combines stochastic gradient descent with a dynamically changing learning rate. The learning rates changes faster with decreasing slopes of the function, and slow in steep regions. The algorithm optimizes the error based on the categorical cross-entropy function.

6.4 Experimental validation

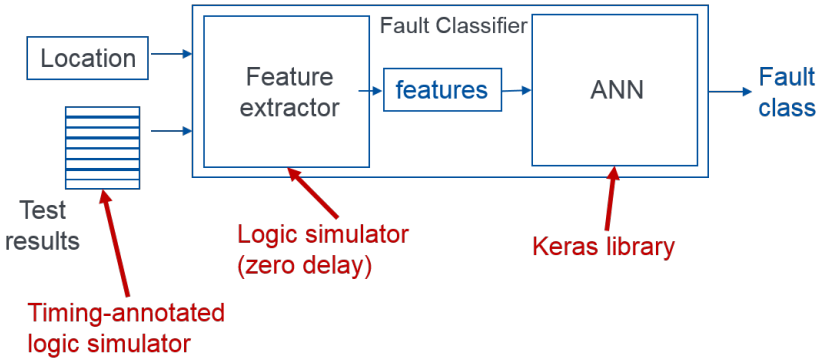
The approach has been validated on ten circuits, five of which belong to the ITC99 benchmark. The rest are industrial circuits kindly provided by NXP. They have all been synthesized using the NanGate 45nm library, and the neighborhood of each line has been extracted from the layout information.

The test set data was generated with a commercial tool targeting transition faults. Because only manufacturing test sets have been considered, and hence the cost constraint is relevant, the ATPG will try to produce as few patterns as necessary. The test set has the typically low diagnostic resolution of manufacturing tests, and was in no way manipulated to include patterns with diagnosis enhancement.

The tested devices were simulated using fault injection in a timing annotated logic simulator. Over 250 faults of each class conform the labelled set of each circuit. The location was chosen randomly. The feature collector only contains a zero-delay logic simulator, without timing information. Thus, the diagnosis tool does not have an exact representation of the faulty behavior, and must be tolerant to some deviation. This is intentional, since the method is expected to be robust despite this typical diagnosis limitation.

The data set was divided into balanced training and test sets, i.e., both training and test sets have roughly the same amount of faults of each class. In this way, learning will not be biased and the global results are meaningful.

The classifier has been written using Keras [Choll15], based on Theano [Bergs10]. Figure 6.7 sketches the experimental setup.



▲ **Figure 6.7** — Experimental setup

Only fully connected ANNs were considered, and the number of hidden layers and cells per layer is considered part of the parameter space. Different configurations have been investigated. This chapter presents one network per circuit, chosen for the best generalization, i.e., the configuration which resulted in the smallest difference between the accuracies for the training and the test set. In case of a draw, the configurations were ranked with the ANNs with a smaller difference in the loss functions. If two or more configurations are still in the same position, the network with the simplest model (less layers, then less cells per layer) is chosen.

6.4.1 Fault classification after test

In a manufacturing test environment, the complete set of patterns applied to the design is known. Moreover, the responses to all patterns can be recorded. Thus, detailed information is available for the fault classifier: the complete test set can be simulated and hence information about passing and failing patterns can be gathered. Consequently, both sets of features presented in 6.2.1 and 6.2.2 can be extracted. The method takes advantage of all the available information in this scenario, and so a vector of length 25 is built which serves as feature vector.

The ANN topology parameter space considered ranges from 1 to 3 hidden layers and 6 to 14 cells per layer. For the sake of clarity, only one configuration is presented per circuit: the one with the smallest difference between training set and test set results. In this scenario, the assumption was made that data of the same design was available from previous production. As a consequence, the ANN was trained with data from the same design, in other words, with product knowledge.

The classification accuracy is presented in table 6.2. The first column indicates the circuit for which the experiments were performed. Columns *ls* and *cs* indicate the number of layers and cells, respectively, of the configuration with the best generalization properties. The following six columns show the accuracy for each of the fault classes, and the last one shows the overall accuracy of the classifier.

▲ **Table 6.2** — Accuracy results for classification after test, with product knowledge

circuit	ls	cs	$A^{b_{or}}$	$A^{b_{and}}$	$A^{b_{byz}}$	$A^{s_{rise}}$	$A^{s_{fall}}$	A^{ct}	A^{L}
b18	2	8	0.9549	0.9495	0.9737	0.9623	0.9501	0.9899	0.9634
b19	1	10	0.9592	0.9627	0.9866	0.9683	0.9613	0.9972	0.9725
b20	2	7	0.9527	0.9414	0.9694	0.9500	0.9520	0.9933	0.9598
b21	2	11	0.9626	0.9525	0.9726	0.9586	0.9525	0.9940	0.9655
b22	1	6	0.9542	0.9603	0.9758	0.9670	0.9414	0.9872	0.9643
p35k	1	13	0.9568	0.9641	0.9827	0.9674	0.9502	0.9874	0.9681
p45k	1	12	0.9778	0.9798	0.9876	0.9856	0.9739	0.9935	0.9830
p78k	1	9	0.9876	0.9974	0.9980	0.9974	0.9869	0.9987	0.9943
p141k	1	9	0.9730	0.9829	0.9835	0.9888	0.9651	0.9895	0.9805
p330k	2	7	0.9789	0.9717	0.9862	0.9809	0.9690	0.9868	0.9789

The accuracy of the method is over 0.94 for all cases. However, as previously discussed, the accuracy metric is insensitive to a change between positives and negatives. A high accuracy may be caused by a large proportion of true positives, but also by a large

number of true negatives. For this reason, recall and precision results are presented also here. Although the accuracy results are similar for all configurations, the results following are presented for the optimal configuration indicated in table 6.2.

Table 6.3 summarizes the recall results, that is, the rate of elements of one fault class which were predicted to belong to the class. The first column indicates the circuit on which the experiments were performed. The following six columns indicate the recall for each fault class. Finally, the last column, labeled as L, shows the overall recall of the classifier.

▲ **Table 6.3** — Classification recall results after test, with product knowledge

circuit	$R^{b_{or}}$	$R^{b_{and}}$	$R^{b_{byz}}$	$R^{s_{rise}}$	$R^{s_{fall}}$	R^{ct}	R^{L}
b18	0.8940	0.8834	0.8697	0.8736	0.8429	0.9770	0.8902
b19	0.9280	0.9119	0.9175	0.8821	0.8659	0.9962	0.9176
b20	0.9149	0.8438	0.8423	0.8544	0.8269	0.9923	0.8794
b21	0.9193	0.8922	0.8726	0.8697	0.8346	0.9923	0.8964
b22	0.9083	0.9052	0.8889	0.8880	0.7854	0.9847	0.8929
p35k	0.8725	0.9048	0.9073	0.8959	0.8659	0.9770	0.9043
p45k	0.9798	0.9710	0.9349	0.9464	0.8769	0.9885	0.9491
p78k	0.9801	0.9918	0.9880	0.9924	0.9464	0.10000	0.9830
p141k	0.9177	0.9658	0.9272	0.9655	0.8880	0.9847	0.9414
p330k	0.9661	0.9370	0.9195	0.9349	0.8846	0.9808	0.9367

With an overall recall of over 0.87 for all circuits, the method overcomes the typical diagnosis limitation of inaccurate circuit and fault model. Some of this inaccuracy is due to the fact that the feature collector, unlike the simulated circuit under test, deploys zero-delay logic simulation. The mismatch between both introduces noise in the features. However, this is a realistic scenario since the diagnosis tools cannot predict the exact behavior of the defective chip.

Given that there is misclassification, it is important to find out the precision in the classification. For instance, a biased network could predict always a fault class l_f. The recall for this fault class would be 1. However, the diagnosis would be useless since when the ANN returns f_c as answer the user cannot trust that the present fault indeed belongs to said category. To evaluate the ANN, the precision in a fault class l_f, P^{l_f}, is defined as the ratio of faults which were identified by the tool to belong to l_f and indeed the injected fault is of type l_f. Table 6.4 summarizes the precision for each fault class in the corresponding six columns. The precision in the output of the tool is over

0.8 for almost all cases. Hence, a prioritization of PFA according to this classification would imply that out of every 5 defective chips, PFA would confirm the systematic problem for 4 of them.

▲ **Table 6.4** — Precision results for classification after test, with product knowledge

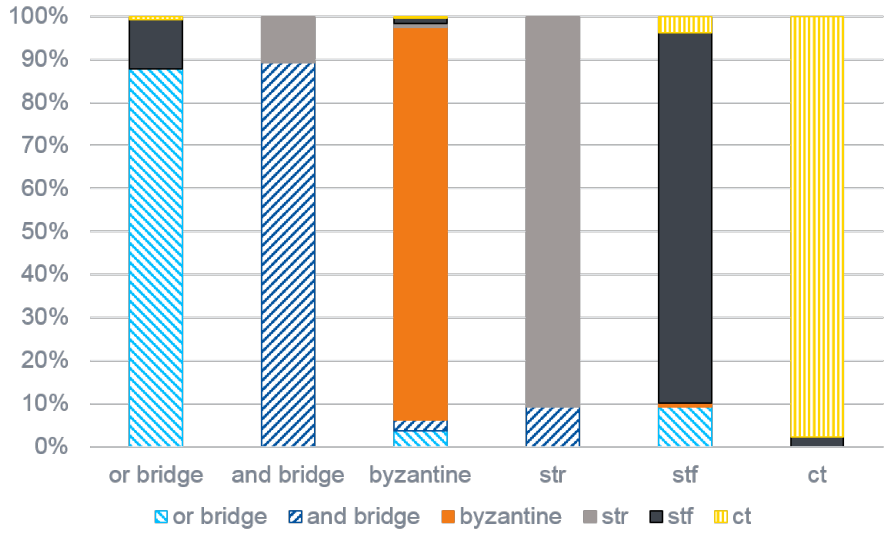
circuit	$P^{b_{or}}$	$P^{b_{and}}$	$P^{b_{byz}}$	$P^{s_{rise}}$	$P^{s_{fall}}$	P^{ct}	P^{L}
b18	0.8151	0.8008	0.9784	0.9084	0.8696	0.9659	0.8902
b19	0.8423	0.8625	0.9895	0.9182	0.9187	0.9886	0.9176
b20	0.8083	0.7810	0.9777	0.8577	0.8884	0.9700	0.8794
b21	0.8436	0.8182	0.9658	0.8902	0.8857	0.9737	0.8964
b22	0.8157	0.8502	0.9707	0.9068	0.8686	0.9449	0.8929
p35k	0.8690	0.8837	0.9916	0.8839	0.8496	0.9515	0.9043
p45k	0.8930	0.9070	0.9919	0.9686	0.9661	0.9736	0.9491
p78k	0.9462	0.9918	1.0000	0.9924	0.9763	0.9924	0.9830
p141k	0.9139	0.9262	0.9758	0.9692	0.9055	0.9554	0.9414
p330k	0.9048	0.8884	1.0000	0.9531	0.9312	0.9446	0.9367

The question that raises next is if there is a pattern of misclassification. In particular, it is interesting to identify which classes cannot be easily differentiated, or if there is any relation between classes. To analyze this, figure 6.8 shows the classification distribution for each fault class. The figure only presents the results for circuit p35k. This is, however, representative, since all circuits tend to mistake the same fault classes and follow the same trend. The horizontal axis has six labels, and represents the injected fault class. Each of the bars shows the percentage of faults that were classified as each of the six fault classes. The leftmost bar, for instance, represents the injected b_{or} faults. Almost up to 90% are correctly classified. An additional set, slightly over 10%, is misclassified as s_{fall}. Finally, not observable on the graph because the percentage is so small, a few experiments were classified as byzantine bridges (b_{byz}).

The chart shows the trends of misclassification. Dominant and bridges, for instance, are most often mistaken for slow to rise gates. The misclassifications are related to the structure of table 6.1. Classes which are often mistaken have overlapping activation conditions. On the other hand, classes with disjoint characteristic feature sets are harder to misclassify. For instance, the slow gates and the dominant bridges overlap in the victim line static condition. Hence, the misclassification $b_{or} \leftrightarrow s_{fall}$ and $b_{and} \leftrightarrow s_{rise}$ is a classification pattern. Crosstalk, on the other hand, is not only more often better classified, but there is also no observable bias for the misclassified examples.

It is important to note that a diagnostic test generator may include test patterns which minimize the coincidence. However, these results have been obtained from circuits tested with a manufacturing test set, hence not fine-tuned to enhance diagnosis resolution. For the example case with a slow-to-rise gate and a dominant-and bridge, the common condition is s_v. The bridge requires also distance to the neighbors, and the slow gate requires a transition. In the best case (for test), one pattern will satisfy all conditions. However, as discussed in section 3.2.3, this conflicts with diagnostic properties, because it does not allow the distinction of fault classes with partially overlapping activation conditions. The limited diagnostic properties of the test patterns limit the distinction of some faults. Yet the results show high accuracy, precision and recall values, providing the manufacturer with a useful guide for PFA application.

From the analysis of all circuits, roughly the same trends are observed. There are, however, some spurious misclassifications, suggesting that the circuit topology and/or the randomly generated training sets have a noticeable impact on the distribution of the features.



▲ **Figure 6.8** — Classification distribution per fault class

The results still allow one more conclusion. For the faults which were correctly classified, the average observed error rate is about 0.1. That means that for around 10% of the test patterns, an erroneous value was observed at the output. However, for faults which were misclassified, this average drops down to a value around 0.02. In average, only about 2% of the test patterns were able to activate and propagate the fault. A low

number of detecting patterns helps increase the probability that *don't care* fields in the activation conditions of one fault class may match the pattern of another fault class. Consequently, the tool may misclassify. However, this seems to confirm that, the better the fault detection quality and the more information available, the better diagnosis quality can be achieved.

6.4.2 Fault classification based exclusively on failing information

For online error detection, if any online test strategy is applied, the patterns are also known and the approach of the previous section can be reused. However, for some components in a system, no test strategy can be integrated in the system without interfering with the system functionality. In those cases, the module can be equipped with online error detection, for instance error detecting codes. Such techniques do not include a test pattern source. If the applied patterns are a priori unknown and only the failing ones have been registered, the set of features deployed in 6.2.2 cannot be reused. However, early classification is still highly advantageous for field returns: troubles which are not repeatable in the lab do not allow a second diagnostic pass, and physical failure analysis can still benefit from additional information that pinpoints the fault class. To apply the method in this scenario, a set of features is needed where the passing information is not considered. For this reason, this section presents the results of classification when only failing information is available. The feature vector is hence formed only by the features presented in 6.2.1, and is thereby formed by thirteen real numbers.

Table 6.5 reports the optimal configuration and the accuracy results for each circuit. The first column indicates the circuit, the next two the layers and cells in the optimal configuration. The accuracy per class follows, and the rightmost column is the overall accuracy of the classifier. The accuracy levels are lower than in the previous case, although it is still over 0.9 for all cases.

The recall and precision results give an insight into what caused the mispredictions. Recall results are presented in table 6.6. For each circuit, the recall for each class is indicated in the corresponding column. The last column indicates the global recall value.

▲ **Table 6.5** — Classification accuracy results after online error detection, with product knowledge

circuit	ls	cs	$A^{b_{or}}$	$A^{b_{and}}$	$A^{b_{byz}}$	$A^{s_{rise}}$	$A^{s_{fall}}$	A^{ct}	A^{L}
b18	1	9	0.9245	0.9232	0.9697	0.9360	0.9292	0.9710	0.9423
b19	2	9	0.9282	0.9380	0.9859	0.9415	0.9324	0.9824	0.9514
b20	2	7	0.9154	0.9247	0.9734	0.9280	0.9034	0.9860	0.9385
b21	1	9	0.9278	0.9258	0.9746	0.9318	0.9225	0.9860	0.9447
b22	2	14	0.9320	0.9232	0.9737	0.9320	0.9246	0.9805	0.9443
p35k	3	14	0.9349	0.9515	0.9854	0.9515	0.9262	0.9807	0.9550
p45k	1	6	0.9138	0.9366	0.9850	0.9491	0.8994	0.9726	0.9427
p78k	3	14	0.9758	0.9699	0.9987	0.9699	0.9725	0.9980	0.9808
p141k	1	12	0.9473	0.9487	0.9822	0.9526	0.9335	0.9789	0.9572
p330k	2	6	0.9354	0.9407	0.9848	0.9565	0.9222	0.9756	0.9525

▲ **Table 6.6** — Classification recall results after online error detection, with product knowledge

circuit	$R^{b_{or}}$	$R^{b_{and}}$	$R^{b_{byz}}$	$R^{s_{rise}}$	$R^{s_{fall}}$	R^{ct}	R^{L}
b18	0.7143	0.7623	0.8467	0.8314	0.7893	0.9885	0.8268
b19	0.8220	0.8634	0.9078	0.7729	0.7778	0.9808	0.8542
b20	0.6936	0.7946	0.8692	0.7778	0.7923	0.9502	0.8155
b21	0.7713	0.7974	0.8726	0.8008	0.7962	0.9540	0.8342
b22	0.8515	0.8017	0.8736	0.7718	0.7625	0.9310	0.8330
p35k	0.8207	0.8413	0.9228	0.8597	0.8084	0.9349	0.8651
p45k	0.7935	0.8216	0.9234	0.8506	0.6885	0.8889	0.8282
p78k	0.9363	0.9098	0.9920	0.9084	0.9195	0.9885	0.9424
p141k	0.8848	0.8333	0.9080	0.8851	0.7915	0.9234	0.8716
p330k	0.8517	0.8613	0.9195	0.8621	0.7423	0.9080	0.8576

The misclassification pattern is consistent with the classification based on test: lower recall values correspond to the dominant bridges and slow gate classes. Crosstalk-induced delay and byzantine bridges, on the other hand, are identified in all cases. The misprediction problems are now aggravated because only half of the information is available. Not only is the test set a non-diagnostic set, but also the information of the passing patterns is unavailable in this scenario. Nevertheless, with this limitation and reduced amount of information compared to the results for the manufacturing fault classification, the method correctly identifies over 82% of the faults for every circuit. For our target, i.e., systematic problem identification, this is statistically significant.

Table 6.7 summarizes the levels of precision for all fault classes and circuits. The precision per fault class, as well as the overall precision, are presented for every circuit.

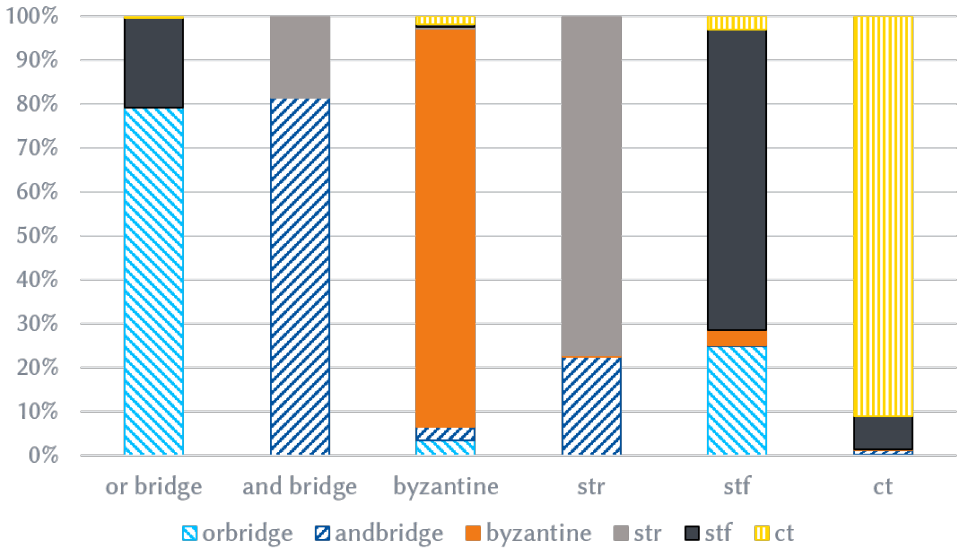
▲ **Table 6.7** — Precision results for classification with failing information

circuit	$P^{b_{or}}$	$P^{b_{and}}$	$P^{b_{byz}}$	$P^{s_{rise}}$	$P^{s_{fall}}$	P^{ct}	P^{L}
b18	0.7561	0.7359	0.9779	0.8097	0.8047	0.8658	0.8268
b19	0.7638	0.7747	0.9947	0.8510	0.8423	0.9275	0.8542
b20	0.7477	0.7265	0.9741	0.8024	0.6936	0.9688	0.8155
b21	0.7511	0.7430	0.9784	0.8069	0.7667	0.9651	0.8342
b22	0.7443	0.7323	0.9744	0.8017	0.7992	0.9567	0.8330
p35k	0.7954	0.8653	0.9917	0.8190	0.7757	0.9531	0.8651
p45k	0.7076	0.7857	0.9877	0.8506	0.7103	0.9469	0.8282
p78k	0.9180	0.9024	1.0000	0.9154	0.9195	1.0000	0.9424
p141k	0.8052	0.8333	0.9875	0.8462	0.8135	0.9526	0.8716
p330k	0.7614	0.7824	0.9917	0.8824	0.7910	0.9480	0.8576

Here, too, byzantine bridges and crosstalk-induced delay problems are identified with higher precision levels. This is related to their very specific activation conditions, which are rarely met when a fault of a different class is active. For the classes with lower recall levels, the precision is also lower, as they get mistaken more often for other fault classes. The most characteristic feature for a slow-to-rise gate, for instance, is a large number of failing patterns which cause a rising transition. However, this can easily be the case too for a dominant or bridge, particularly if it is hard to detect and many of the detecting patterns also happen to cause a rising transition.

Although the recall and precision levels are lower than in the previous section due to the lower amount of information available, the method would still allow a fast classification that prioritizes and guides the physical analysis procedure.

Figure 6.9 shows the classification rates. These rates describe the distribution of classification for an injected fault of class l_f. Interestingly, the results confirm the need for the driving gate input features: the classes most often mistaken are dominant or and slow to fall gates, and *dominant and* and *slow to rise* gates. The introduction of f_f^{c1} and f_f^{c0} mitigates the problem, and these results point to glitches as the main issue for misprediction. This suggests that collecting the features with a zero delay logic simulator limits the recall of the method.



▲ **Figure 6.9** — Classification distribution for classification based on failing information.

6.4.3 Fault identification without product knowledge

The previous sections require a trained ANN. Unfortunately, it is not always the case that information about the handled product is available. For this reason, this subsection presents the results of ANN reuse, i.e., an ANN originally trained with data that belongs to a certain design d is deployed to classify faults observed in chips of type d'.

At the beginning of production, when there is no knowledge about the product, no information is available about previous faults and the method is not directly applicable. To train the ANN, large amounts of data need to be available. However, in a technology node the defects and their activation conditions are the same regardless of the functional design of the circuit. Thus, training the ANN with data from one design to predict the class for faults present in another design is reasonable.

Table 6.8 reports the classification recall of a network trained with data from the design *circuit*. The recall is measured with the test sets of all other nine circuits. The network configuration is the same as reported in subsection 6.4.1. Columns b_{or} through ct report the recall for each fault class. The last column indicates the obtained overall recall.

The results follow a pattern similar to that observed for fault classification with product knowledge: the slow to rise gates are still the class most prone to misclassification, while crosstalk and byzantine bridges have high recall values. Interestingly, circuit

▲ **Table 6.8** — Classification recall results after test, without product knowledge

circuit	$R^{b_{or}}$	$R^{b_{and}}$	$R^{b_{byz}}$	$R^{s_{rise}}$	$R^{s_{fall}}$	R^{ct}	R^{L}
b18	0.8708	0.8964	0.9034	0.8884	0.8088	0.9774	0.8911
b19	0.8898	0.8642	0.8499	0.9004	0.8348	0.9766	0.8863
b20	0.8861	0.8479	0.9004	0.8831	0.8161	0.9881	0.8876
b21	0.8788	0.8917	0.9008	0.9043	0.8404	0.9625	0.8967
b22	0.8527	0.8870	0.9043	0.9021	0.7781	0.9817	0.8846
p35k	0.9239	0.9422	0.8973	0.8290	0.7909	0.9723	0.8914
p45k	0.9514	0.9274	0.8986	0.8419	0.7440	0.9600	0.8857
p78k	0.8101	0.8364	0.8702	0.7483	0.6675	0.9408	0.8121
p141k	0.8734	0.9006	0.9113	0.8822	0.7676	0.9349	0.8778
p330k	0.9076	0.9085	0.8704	0.8596	0.8242	0.9804	0.8916

$p78k$, which rendered the best results in the scenario with product knowledge now performs worse than the other circuits, with an overall recall value 0.07 lower than the other circuits. Regardless of this, although there was no information available to train the ANN and the training was performed with data belonging to the circuit indicated in the first column of table 6.9, the precision results reported show that there is potential for ANN reuse. The method yields an overall accuracy of over 0.8 for all circuits, and is applicable from the beginning of production with still very high precision in the classification of the tool. This classification allows for very early critical defect identification and rapid yield ramp up.

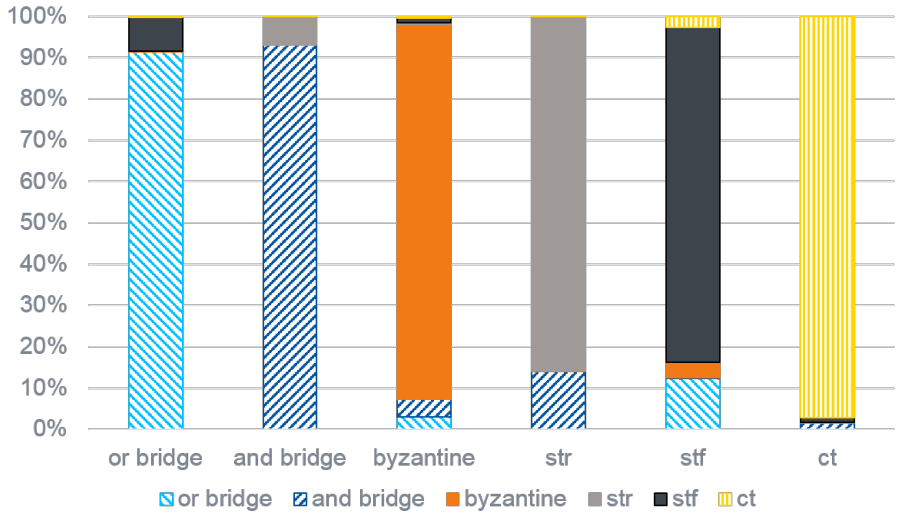
The precision results, reported in table 6.9, also confirm the potential for reuse. Although slightly lower than in the previous case, the lowest precision value in the table is 0.7275, with this value being almost an outlier since most of the results are over 0.8. Here, too, the lower performance of p78k is visible. The confidence in the outcome of the ANN trained with its data is lower than that of all other circuits.

Figure 6.10 shows the distribution of classification for each class of injected fault for the ANN trained with data of circuit p35k. The test data of all other circuits has been deployed as test set. Like in previous results, the misclassifications seem to follow the same trend, with mistakes being more numerous for those classes with related characteristic features. Here again slow gates are often confused with the dominant bridge classes, and crosstalk is the easiest class to spot. However, more spurious misclassification can be observed in this chart, where a yellow fringe, although small, is observable in all columns. Also, more faults are classified as byzantine. This spurious effects confirm the influence of the topology and the random sampling of the faults.

▲ **Table 6.9** — Precision results for classification after test, with no product knowledge

circuit	$P^{b_{or}}$	$P^{b_{and}}$	$P^{b_{byz}}$	$P^{s_{rise}}$	$P^{s_{fall}}$	P^{ct}
b18	0.8244	0.8511	0.9753	0.9061	0.8378	0.9535
b19	0.8313	0.8327	0.9895	0.8628	0.8427	0.9679
b20	0.8686	0.8257	0.9491	0.8902	0.8751	0.9131
b21	0.8743	0.8507	0.9643	0.8976	0.8484	0.9456
b22	0.8276	0.8586	0.9502	0.9017	0.8313	0.9325
p35k	0.8299	0.7852	0.9543	0.9380	0.9004	0.9605
p45k	0.7992	0.8216	0.9530	0.9192	0.8804	0.9567
p78k	0.7646	0.7275	0.8799	0.8279	0.7959	0.8742
p141k	0.8007	0.8365	0.9368	0.9059	0.8261	0.9627
p330k	0.8255	0.8223	0.9807	0.9151	0.8652	0.9517

Nevertheless, their impact on the recall and precision is negligible, and the approach is hence robust to a product change.



▲ **Figure 6.10** — Classification distribution per fault class (after test, without product knowledge)

The experiments with lack of knowledge have also been performed based solely on online error detection results, that is, only based on failing patterns. Table 6.10 summarizes the recall values for this scenario without previous product knowledge. The first column, *circuit*, determines to what design the training data belongs. The follow-

ing columns indicate the recall for each fault class and the global one. The recall is calculated for the test sets of all other nine circuits.

▲ **Table 6.10** — Recall results after online error detection, without product knowledge

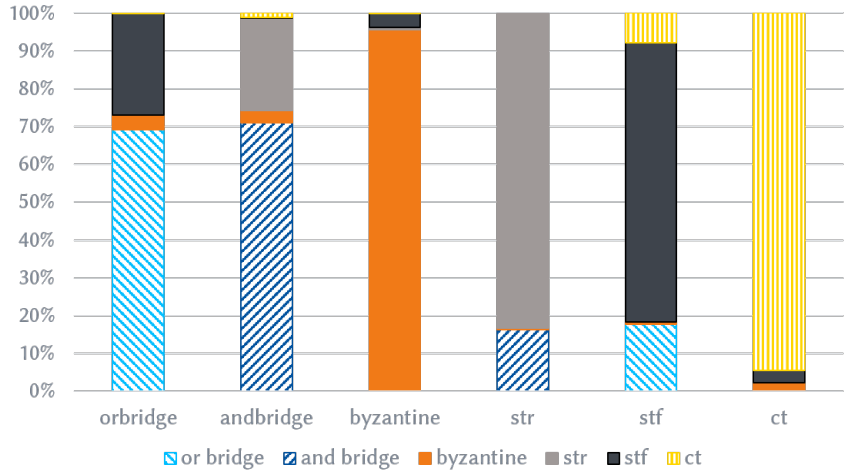
circuit	$R^{b_{or}}$	$R^{b_{and}}$	$R^{b_{byz}}$	$R^{s_{rise}}$	$R^{s_{fall}}$	R^{ct}	R^{L}
b18	0.7187	0.7491	0.8911	0.8291	0.7601	0.9957	0.8264
b19	0.8311	0.8354	0.8679	0.7913	0.7951	0.9711	0.8494
b20	0.6826	0.7843	0.9122	0.8363	0.7867	0.9259	0.8236
b21	0.7329	0.7603	0.8912	0.8547	0.8178	0.9425	0.8356
b22	0.7302	0.7210	0.8972	0.8608	0.7785	0.9000	0.8169
p35k	0.8238	0.8434	0.8947	0.7315	0.6419	0.9170	0.8077
p45k	0.8392	0.7773	0.8911	0.8432	0.6088	0.8995	0.8094
p78k	0.7874	0.8440	0.8506	0.7031	0.7883	0.7327	0.7835
p141k	0.8979	0.8154	0.8999	0.8446	0.7058	0.8897	0.8415
p330k	0.8204	0.8127	0.8652	0.7768	0.6583	0.8799	0.8016

Due to the limited information the recall results are, as expected, below the recall obtained with product knowledge. The precision results are reported in table 6.11, and confirm the trend observed: the less information available (less features and no product knowledge) complicates the analysis with ANNs. However, although the results are not so accurate, the method shows tolerance towards noisy data.

The precision results are, accordingly to the recall, lower than for those scenarios with more knowledge. However, as observed in figure 6.11, the classification trend is still maintained. Only spurious effects of byzantine and crosstalk classification appear, but the classification patterns stay otherwise robust.

▲ **Table 6.11** — Precision results for classification with failing information

circuit	$P^{b_{or}}$	$P^{b_{and}}$	$P^{b_{byz}}$	$P^{s_{rise}}$	$P^{s_{fall}}$	P^{ct}
b18	0.7916	0.7679	0.9859	0.7777	0.7846	0.8530
b19	0.7989	0.7578	0.9936	0.8207	0.8093	0.9257
b20	0.8111	0.7854	0.9497	0.8182	0.6858	0.9119
b21	0.8095	0.7757	0.9907	0.7894	0.7373	0.9303
b22	0.7794	0.7733	0.9569	0.7748	0.7073	0.9272
p35k	0.6918	0.7099	0.9559	0.8330	0.7373	0.9447
p45k	0.6627	0.7966	0.9544	0.8116	0.7233	0.9312
p78k	0.7700	0.6346	0.9924	0.8375	0.6490	0.9598
p141k	0.7344	0.7907	0.9570	0.8277	0.7938	0.9676
p330k	0.6714	0.7291	0.9944	0.8181	0.7059	0.9387



▲ **Figure 6.11** — Distribution for classification based on failing information and without product knowledge

The scenarios with a more accute lack of knowledge highlight interesting effects about the method. The training and data sets were generated randomly, making the distribution of the location of victims also random. Signals located closer to the outputs are generally more prone to experience glitches, which cause mismatches between logic and timing annotated simulation. The features are noisier the closer the victim line is to the outputs. As a result, circuits which had a wider range of values for the features generalized from a more heterogeneous training set. Consequently have less difficulty to predict the class of data from other circuits. On the other hand, circuits which predicted very well for their own data, such as p78k, are now less capable of generalizing for data from other circuits.

The results suggest that ANN reuse is possible, at least for the first phases when there is no data available. If necessary, an ANN can be trained as production goes by and data can be sampled. In the worst case, however, at least a coarse prediction can be made using existing ANNs which already identifies systematic problems.

6.4.4 Identification of intermittent faults

In the previous sections, only permanent faults were considered. However, as discussed in 3.5, supporting the diagnosis of intermittent faults is extremely important. This section takes the results presented previously as the baseline, and compares the recall of the method when intermittent faults are injected instead of permanent. It is important

to note that the ANN has only been trained with permanent faults. The motivation behind this is twofold. In the first place, the goal is to assess the robustness of the method with respect to intermittent faults, i.e., imperfections in the fault modeling. In the second, it is interesting to mitigate the impact of information gathering. In order to train the ANN, a balanced training set must be available. However, it would be extremely rare that in production all six classes of faults appeared evenly. Thus, the results in this section are a first attempt to explore the possibility of generating synthetic data for the training, and hence saving this first obstacle for machine learning techniques.

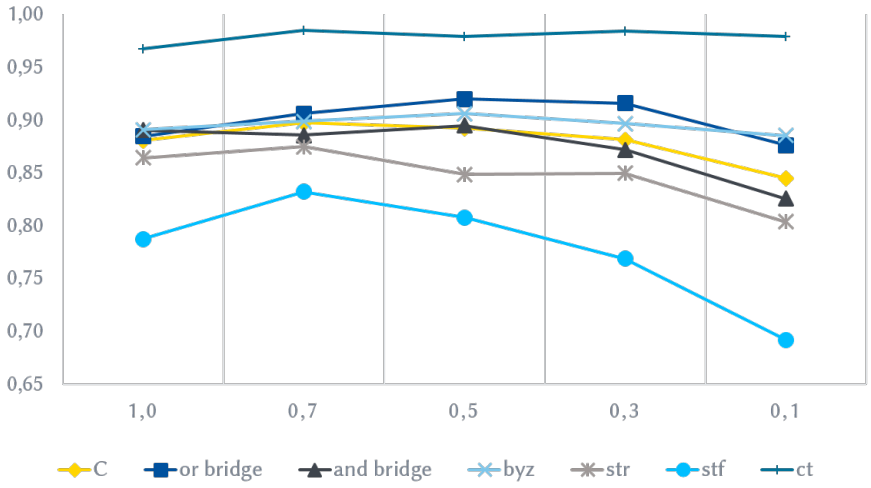
The experiments performed consider four different values for the activation rate: $\lambda_a \in 0.7, 0.5, 0.3, 0.1$. The reference experiments are permanent faults, which can be expressed as $\lambda_a = 1.0$.

The first set of experiments is performed for the test scenario, that is, both passing and failing information are available. The results show that recall and precision decrease as the activation rate deviates from that of the training set. Since the training set is formed by features collected from permanent faults, the less often a fault gets activated, the more its behavior differs from that of the training data set elements. Figure 6.12 shows the evolution of the recall as the activation decreases. The baseline is the test set formed by permanent faults. The X-axis indicates the intermittent activation rate. In average, the recall value for intermittents decreases by 0.1, rendering an overall recall R^L of around 0.85 in the worst scenario with a high behavior deviation.

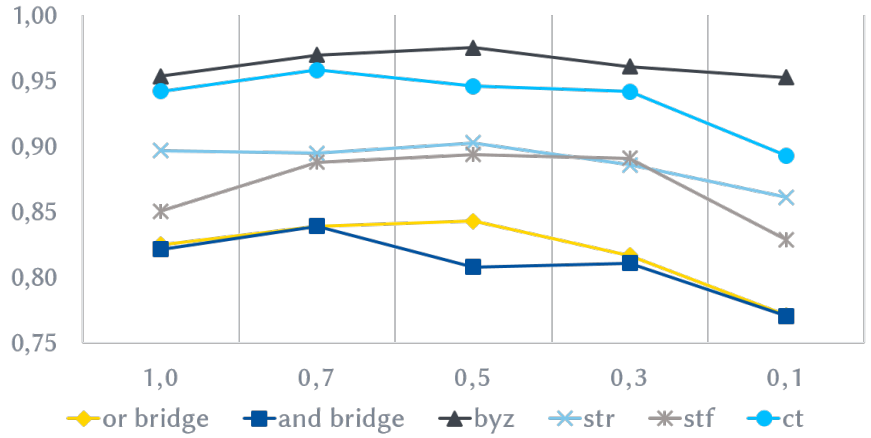
The precision decreases also as the activation rate decreases. Its evolution is presented in figure 6.13 for each of the six fault classes. The value of the overall recall and the overall precision are the same, $R^L = P^L$. Despite the descending curves, the absolute precision value remains over 0.85 for an activation rate of 0.1, and it is over 0.75 for all classes.

Figure 6.14 shows the classification distribution for an ANN trained with data of permanent faults in circuit $p35k$ and tested with intermittent faults activated with a 0.5 probability. The horizontal axis represents the injected fault class. The ratio of colored areas of each class represents the output of the ANN, i.e., how the injected faults of each class were classified. Despite the difference of the activation rate, the trend is still consistent with the previously observed scenarios: permanent faults both with and without product knowledge.

This result suggests that the method can tolerate a high degree of uncertainty also



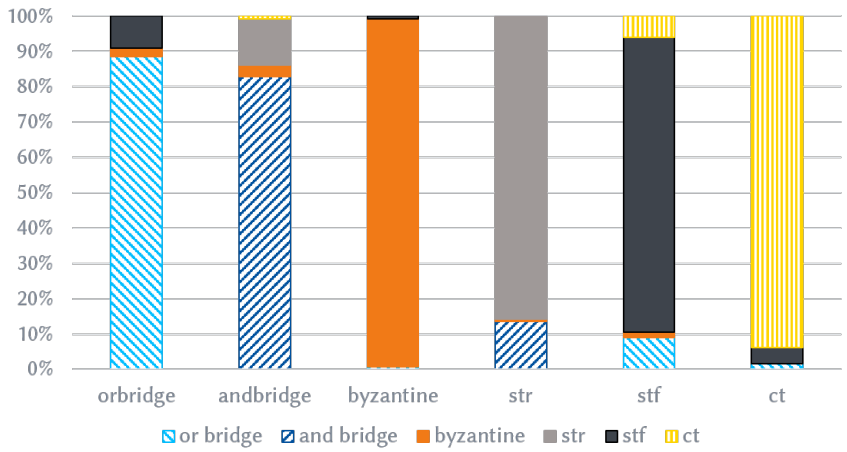
▲ **Figure 6.12** — Recall evolution for decreasing activation rates of intermittent faults in test



▲ **Figure 6.13** — Precision evolution for decreasing activation rates of intermittent faults in test

in the fault modeling, which is promising with the modeling uncertainty expected in nanometer regimes.

A similar tendency is observed for the ANNs trained only with failing pattern information. However, for the online detection scenario the average recall reduction is half of the average recall reduction in the former case. For classification after test, the value of recall for the intermittents with rarest activation $\lambda = 0.1$ was in average 0.371 lower than the recall value from the reference case with $\lambda = 1.0$. Figure 6.15 shows the impact

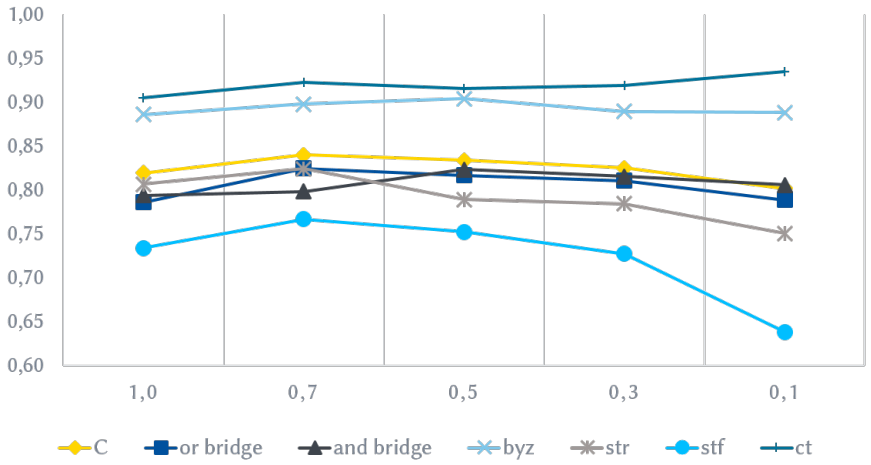


▲ **Figure 6.14** — Classification distribution for intermittent faults after test

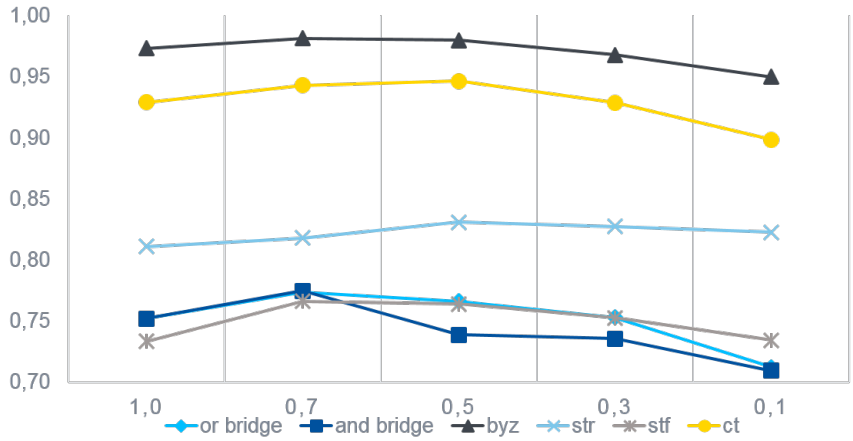
of the activation rate on the recall. Although the recall loss can also be observed for the online error detection scenario, this ANNs seem more robust against these variations: the average recall difference between the reference case and the intermittents with the lowest activation rate is only 0.176. The difference is hence less than half of the difference for the experiments with passing information. The reason is that the passing patterns, which are now noisier because the fault was not always activated, are not considered for the online detection scenario. The failing patterns, on the other hand, always require the activation of the fault, and for this reason the results do not deteriorate like when both passing and failing patterns are considered. Less detecting patterns, however, also lead to decreased diagnostic ability, because the probability that a large rate of the failing patterns satisfies the activation conditions of more than one fault class also increases.

The impact of decreasing activation rates on the precision results is illustrated in figure 6.16. The effect is the same as on recall: the method loses accuracy, but the average loss of precision in this case is 0.0009, while for the scenario with passing and failing pattern information it is 0.0353.

The classification patterns also remain unchanged. Figure 6.17 shows the classification distribution for each class of injected fault. The results belong to circuit $p35k$, and represent the injected faults with a rate of 0.5. The classes ct and b_{byz} are now more often the outcome of the tool, because with lower number of failing patterns the probability that the activation patterns will fulfill the activation conditions of these faults is higher. For example, in the extreme case of a bridge activated by just one



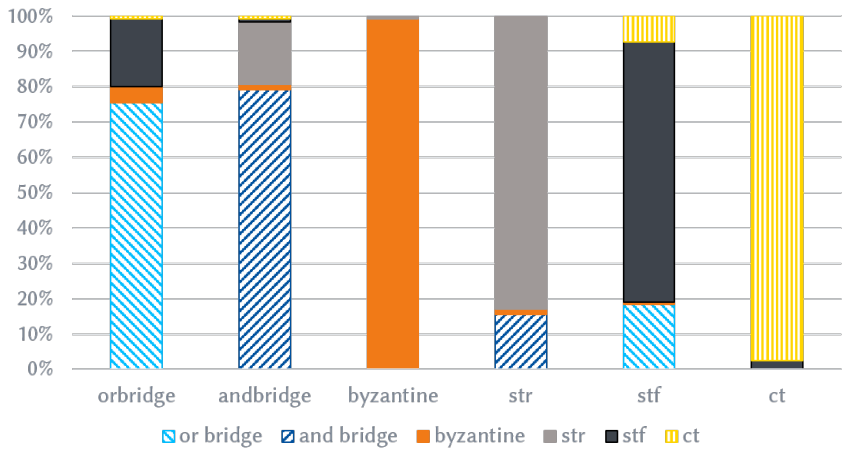
▲ **Figure 6.15** — Recall evolution for decreasing activation rates of intermittent faults in test



▲ **Figure 6.16** — Precision evolution for decreasing activation rates of intermittent faults after online error detection

pattern, it is likely that one of the neighbors will switch and render a high value of the feature f_{trans}, which is characteristic of crosstalk. As a result, the fault may be classified as crosstalk. The more restricted the information is, the less quality the diagnostic process has.

In the online error detection scenario, the diminished recall is mainly caused by a lower number of failing patterns. As discussed in previous sections, a lower error rate leads to less recall in the method. In the test scenario, this is aggravated because a lower activation rate changes the distribution of the values for the features related



▲ **Figure 6.17** — Classification pattern for $p35k$ intermittents with $\lambda_{act} = 0.5$

to the passing patterns. For this reason, classification after test is less robust than classification after online error detection. Nevertheless, the recall and precision values for both scenarios show that there is potential for machine learning application in this field. The presented method is applicable in all presented scenarios for volume prediction. It is robust even when the production of a new design is initiated, and also in presence of intermittent faults. Moreover, it requires no special test set, as even with a manufacturing test without diagnostic properties it allows for a fast prior fault classification. Neural networks and their robustness against uncertainty are thus promising in the context of CMOS nanometer technologies, where uncertainty dominates the modeling on which most test and diagnosis techniques rely.

CHAPTER 7

Conclusions and future work

The manufacturing process of nanometer technologies introduces variations, which translate into deviations from the expected performance specification. On top of that, chips manufactured in these technologies are sensitive to environmental variations and noise. If a defect is introduced, it may impact the behavior of the device in different ways depending on their size and nature. Growing uncertainty in the deep submicron regime calls for test and diagnosis techniques which are robust against indeterminism.

While test detects the defects, diagnosis must locate and characterize the defect. Variation-aware test pattern generation techniques exist, but diagnosis must still make progress to handle indeterminism. Although logic diagnosis location techniques have been broadly investigated, only a few techniques exist to characterize the problem. Unfortunately, they are either too costly, such as physical analysis, or do not handle intermittent faults.

This work has presented two machine learning based techniques for fast fault characterization. The approaches incur in minimal overhead in the manufacturing flow, but succeed in classifying the faults correctly with a precision sufficient to identify systematic problems during manufacturing. As a consequence, costly physical analysis procedures can be prioritized, speeding up yield ramp-up.

The first of the approaches is based on Bayesian networks, and identifies the chips affected by critical defects from those affected only from noise. Not only does this result

help in prioritization, but it also avoids unnecessary yield losses, since chips affected by noise are not really faulty and can be deployed in robust circuits.

The second method is based on artificial neural networks. It monitors some values from logic simulation which represent the activation conditions, and maps them to one of the fault classes. The information obtained in this second step of the classification can be deployed to issue a warning when one of the fault classes appears with higher frequency. The approach can be applied to the chips after manufacturing test, but also succeeds in identifying the fault classes in online test, despite the restricted amount of information available. It can also be deployed from the beginning of production by reusing the neural network of another product.

7.1 Future work

The main limitation of machine learning techniques is often the lack of a suitable training set from which to extrapolate patterns. Possible extensions of this work include exploring the possibility of deploying a completely synthetic training set, that is, based on simulation, and evaluate the effectiveness of the method on chips affected by variations and with different fault sizes. Another open problem is differentiating between slow circuits, i.e., affected by variations and which belong to the slower classes, from devices which are affected by a defect and consequently produce incorrect outputs. Distinguishing between both cases is a far from trivial task, and will generally require additional test and physical analysis. Speeding up this process with an extension to this work would accelerate a correct identification of design problems or defect prone processes.

Bibliography

[Abele14] U. Abelein, A. Cook, P. Engelke, M. Glaß, F. Reimann, L. Rodríguez Gómez, T. Russ, J. Teich, D. Ull, and H.-J. Wunderlich. Non-Intrusive Integration of Advanced Diagnosis Features in Automotive E/E-Architectures. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'14)*. 2014. [pages 26, 55, 60, and 62]

[Acken91] J. A. Acken and S. D. Millman. Accurate modeling and simulation of bridging faults. In *Proceedings of the IEEE 1991 Custom Integrated Circuits Conference*, pages 17.4/1–17.4/4. May 1991. [page 19]

[Agarw07] M. Agarwal, B. C. Paul, M. Zhang, and S. Mitra. Circuit failure prediction and its application to transistor aging. In *Proceedings of the 25th IEEE VLSI Test Symposium (VTS'07)*, pages 277–286. May 2007. [page 25]

[Agarw08] M. Agarwal, V. Balakrishnan, A. Bhuyan, K. Kim, B. C. Paul, W. Wang, B. Yang, Y. Cao, and S. Mitra. Optimized circuit failure prediction for aging: Practicality and promise. In *Proceedings of the IEEE International Test Conference (ITC'08)*, pages 1–10. Oct 2008. [page 26]

[Amgal08] U. Amgalan, C. Hachmann, S. Hellebrand, and H. J. Wunderlich. Signature rollback - a technique for testing robust circuits. In *Proceedings of the 26th IEEE VLSI Test Symposium (VTS'08)*, pages 125–130. April 2008. [pages 45, 62, and 88]

[Apost07] A. Apostolakis, M. Psarakis, D. Gizopoulos, and A. Paschalis. Functional processor-based testing of communication peripherals in systems-on-chip. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 15(8):971–975, Aug 2007. [page 55]

[Apost09] A. Apostolakis, D. Gizopoulos, M. Psarakis, D. Ravotto, and M. S. Reorda. Test program generation for communication peripherals in processor-based soc devices. *IEEE Design & Test of Computers*, 26(2):52–63, March 2009. [page 55]

[Baran15] R. Baranowski, F. Firouzi, S. Kiamehr, C. Liu, M. Tahoori, and H.-J. Wunderlich. On-Line Prediction of NBTI-induced Aging Rates. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'15)*, pages 589–592. 2015. [page 25]

[Barde82] P. H. Bardell and W. H. McAnney. Self-testing of multichip logic modules. In *Proceedings of the IEEE International Test Conference (ITC'82)*, pages 200–204. Nov 1982. [page 44]

[Becke10] B. Becker, S. Hellebrand, I. Polian, B. Straube, W. Vermeiren, and H. J. Wunderlich. Massive statistical process variations: A grand challenge for testing nanoelectronic circuits. In *2010 International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 95–100. June 2010. [pages 71, 74, and 77]

[Belet02] D. Belete, A. Razdan, W. Schwarz, R. Raina, C. Hawkins, and J. Morehead. Use of dft techniques in speed grading a 1 ghz+ microprocessor. In *Proceedings of the International Test Conference (ITC'02)*, pages 1111–1119. 2002. [pages 22 and 23]

[Benwa12] B. Benware, C. Schuermyer, M. Sharma, and T. Herrmann. Determining a failure root cause distribution from a population of layout-aware scan diagnosis results. *IEEE Design & Test of Computers*, 29(1):8–18, 2012. [page 82]

[Bergs10] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*. June 2010. Oral Presentation. [page 128]

[Bisho06] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387310738. [pages 68, 78, 79, 81, 120, and 124]

[Blaau08] D. Blaauw, K. Chopra, A. Srivastava, and L. Scheffer. Statistical timing analysis: From basic principles to state of the art. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 27(4):589–607, April 2008. [page 75]

[Blant12] R. D. Blanton, W. C. Tam, X. Yu, J. E. Nelson, and O. Poku. Yield learning through physically aware diagnosis of ic-failure populations. *IEEE Design & Test of Computers*, 29(1):36–47, Feb 2012. [pages 24 and 83]

[Bolza07] L. Bolzani, E. Sanchez, M. Schillaci, M. S. Reorda, and G. Squillero. An automated methodology for cogeneration of test blocks for peripheral cores. In *Proceedings of the 13th IEEE International On-Line Testing Symposium (IOLTS 2007)*, pages 265–270. July 2007. [page 55]

[Bonda00] A. Bondavalli, S. Chiaradonna, F. di Giandomenico, and F. Grandoni. Threshold-based mechanisms to discriminate transient from intermittent faults. *IEEE Transactions on Computers (TC)*, 49(3):230–245, 2000. [page 64]

[Bonin01] D. Boning and S. Nassif. *Models of Process Variations in Device and Interconnect*, pages 98–115. Wiley-IEEE Press, 2001. ISBN 9780470544365. [pages 72, 73, and 74]

[Borka03] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De. Parameter variations and impact on circuits and microarchitecture. In *Proceedings of the 40th annual ACM/IEEE Design Automation Conference (DAC'03)*, pages 338–342. 2003. [pages 74 and 75]

[Botto94] L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, L. D. Jackel, Y. LeCun, U. A. Muller, E. Sackinger, P. Simard, and V. Vapnik. Comparison of classifier methods: a case study in handwritten digit recognition. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 3 - Conference C: Signal Processing (Cat. No.94CH3440-5)*, volume 2, pages 77–82 vol.2. Oct 1994. [page 69]

[Bushn13] M. Bushnell and V. Agrawal. *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Springer Publishing Company,

Incorporated, 2013. ISBN 1475781423, 9781475781427. [pages 17, 22, 24, 29, 34, 35, 42, 43, 44, and 62]

[Chen98] H. H. Chen and J. S. Neely. Interconnect and circuit modeling techniques for full-chip power supply noise analysis. *IEEE Transactions on Components, Packaging, and Manufacturing Technology*, 21(3):209–215, Aug 1998. [page 14]

[Chen02] L. H. Chen and M. Marek-Sadowska. Incremental delay change due to crosstalk noise. In *Proceedings of the 2002 International Symposium on Physical Design*, ISPD '02, pages 120–125. ACM, New York, NY, USA, 2002. ISBN 1-58113-460-6. [page 14]

[Chen03] L. Chen, S. Ravi, A. Raghunathan, and S. Dey. A scalable software-based self-test methodology for programmable processors. In *Proceedings of the 40th annual ACM/IEEE Design Automation Conference (DAC'03)*, pages 548–553. June 2003. [page 54]

[Chen09] P. J. Chen, J. C. M. Li, and H. J. Chao. Bridging fault diagnosis to identify the layer of systematic defects. In *Proceedings of the IEEE 18th Asian Test Symposium (ATS'09)*, pages 349–354. Nov 2009. [page 39]

[Chess98a] B. Chess and T. Larrabee. Logic testing of bridging faults in cmos integrated circuits. *IEEE Transactions on Computers*, 47(3):338–345, Mar 1998. [page 12]

[Chess98b] B. Chess and T. Larrabee. Logic testing of bridging faults in cmos integrated circuits. *IEEE Transactions on Computers*, 47(3):338–345, Mar 1998. [pages 32 and 33]

[Cheun07] H. Cheung and S. K. Gupta. Accurate modeling and fault simulation of byzantine resistive bridges. In *Proceedings of the 25th International Conference on Computer Design*, pages 347–353. Oct 2007. [page 19]

[Choll15] F. Chollet. Keras. `https://github.com/fchollet/keras`, 2015. [page 128]

[Const03] C. Constantinescu. Trends and challenges in vlsi circuit reliability. *IEEE Micro*, 23(4):14–19, 2003. [page 20]

[Cook11] A. Cook, M. Elm, H.-J. Wunderlich, and U. Abelein. Structural In-Field Diagnosis for Random Logic Circuits. In *Proceedings of the 16th IEEE European Test Symposium (ETS'11)*, pages 111–116. IEEE Computer Society, 2011. [pages 40, 59, 62, and 89]

[Cook14] A. Cook and H.-J. Wunderlich. Diagnosis of Multiple Faults with Highly Compacted Test Responses. In *Proceedings of the 19th IEEE European Test Symposium (ETS'14)*, pages 27–30. 2014. [pages 41 and 59]

[Corno04] F. Corno, E. Sánchez, M. S. Reorda, and G. Squillero. Automatic test program generation: A case study. *IEEE Design & Test of Computers*, 21(2):102–109, 2004. [page 54]

[Corte95] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995. [page 80]

[Das15] S. Das, D. M. Bull, and P. N. Whatmough. Error-resilient design techniques for reliable and dependable computing. *IEEE Transactions on Device and Materials Reliability*, 15(1):24–34, March 2015. [page 71]

[De Kl09] J. De Kleer. Diagnosing multiple persistent and intermittent faults. In *Proceedings of the 21st International Joint Conference on Artifical intelligence (IJCAI'09)*, pages 733–738. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2009. [page 64]

[DeOri13] A. DeOrio, Q. Li, M. Burgess, and V. Bertacco. Machine learning-based anomaly detection for post-silicon bug diagnosis. In *Proceedings of the ACM/IEEE Conference on Design, Automation Test in Europe (DATE'13)*, pages 491–496. 2013. [page 82]

[Desin06] R. Desineni, O. Poku, and R. D. Blanton. A logic diagnosis methodology for improved localization and extraction of accurate defect behavior. In *Proceedings of the IEEE International Test Conference (ITC'06)*, pages 1–10. Oct 2006. [pages 39, 65, and 83]

[Dodd03] P. E. Dodd and L. W. Massengill. Basic mechanisms and modeling of single-event upset in digital microelectronics. *IEEE Transactions on Nuclear Science*, 50(3):583–602, June 2003. [page 14]

[Drine03] P. Drineas and Y. Makris. Concurrent fault detection in random combinational logic. In *Fourth International Symposium on Quality Electronic Design, 2003. Proceedings.*, pages 425–430. March 2003. [pages 26 and 50]

[Egger07] S. Eggergluss, G. Fey, and R. Drechsler. Sat-based atpg for path delay faults in sequential circuits. In *2007 IEEE International Symposium on Circuits and Systems*, pages 3671–3674. May 2007. [page 36]

[Egger12] S. Eggersglüß and R. Drechsler. *Circuits and Testing*. Springer-Verlag New York, 2012. ISBN 978-1-4419-9976-4. [page 42]

[Eiche77] E. B. Eichelberger and T. W. Williams. A logic design structure for lsi testability. In *Proceedings of the 14th Design Automation Conference*, DAC '77, pages 462–468. IEEE Press, Piscataway, NJ, USA, 1977. [page 43]

[Flenk15] T. Flenker, A. Sülflow, and G. Fey. Diagnostic tests and diagnosis for delay faults using path segmentation. In *Proceedings of the 24th IEEE Asian Test Symposium (ATS'15)*, pages 145–150. Nov 2015. [pages 47 and 62]

[Fujiw83] H. Fujiwara and T. Shimono. On the acceleration of test generation algorithms. *IEEE Transactions on Computers*, 32(12):1137–1144, December 1983. [page 36]

[Gench68] H. Genchi, K. Mori, S. Watanabe, and S. Katsuragi. Recognition of handwritten numerical characters for automatic letter sorting. *Proceedings of the IEEE*, 56(8):1292–1301, Aug 1968. [page 68]

[Gil T12] D. Gil-Tomas, J. Gracia-Moran, J.-C. Baraza-Calvo, L.-J. Saiz-Adalid, and P.-J. Gil-Vicente. Analyzing the impact of intermittent faults on microprocessors applying fault injection. *IEEE Design & Test of Computers*, 29(6):66–73, 2012. [page 64]

[Girsh14] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587. June 2014. [page 121]

[Goel81] P. Goel. An implicit enumeration algorithm to generate tests for combinational logic circuits. *IEEE Transactions on Computers*, 30(3):215–222, March 1981. [page 35]

[Goodf16] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. [pages 82 and 121]

[Grave13] A. Graves, A. r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649. May 2013. [page 123]

[Hall99] M. A. Hall. *Correlation-based Feature Selection for Machine Learning*. Ph.D. thesis, 1999. [page 25]

[Hamam04] Y. Hamamura, T. Kumazawa, K. Tsunokuni, A. Sugimoto, and H. Asakura. An advanced defect-monitoring test structure for electrical screening and defect localization. *IEEE Transactions on Semiconductor Manufacturing*, 17(2):104–110, May 2004. [page 24]

[Hanse96] J. H. Hansen. Analysis and compensation of speech under stress and noise for environmental robustness in speech recognition. *Speech Communication*, 20(1):151 – 173, 1996. [page 69]

[Hapke14] F. Hapke, W. Redemund, A. Glowatz, J. Rajski, M. Reese, M. Hustava, M. Keim, J. Schloeffel, and A. Fast. Cell-aware test. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 33(9):1396–1409, Sept 2014. [pages 36 and 65]

[Hess01] C. Hess, D. Stashower, B. E. Stine, L. H. Weiland, G. Verma, K. Miyamoto, and K. Inoue. Fast extraction of defect size distribution using a single layer short flow nest structure. *IEEE Transactions on Semiconductor Manufacturing*, 14(4):330–337, Nov 2001. [page 24]

[Hinto06] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. [page 81]

[Holst09] S. Holst and H.-J. Wunderlich. Adaptive Debug and Diagnosis Without Fault Dictionaries. *Journal of Electronic Testing: Theory and Applications (JETTA)*, 25(4-5):259–268, 2009. [pages 39, 40, 41, 46, and 47]

[Holst15] S. Holst, M. E. Imhof, and H.-J. Wunderlich. High-Throughput Logic Timing Simulation on GPGPUs. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 20(3):1–22, 2015. [page 33]

[Huang16] H. M. Huang and C. H. P. Wen. Layout-based soft error rate estimation framework considering multiple transient faults from device to circuit level. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 35(4):586–597, April 2016. [page 70]

[Jha03] N. K. Jha and S. Gupta. *Testing of Digital Systems*. Cambridge University Press, Cambridge, UK, 2003. ISBN 0521773563. [page 37]

[Kahng00] A. B. Kahng, S. Muddu, and E. Sarto. On switch factor based analysis of coupled rc interconnects. In *Proceedings 37th Design Automation Conference (DAC'00)*, pages 79–84. June 2000. [page 14]

[Khare94] J. B. Khare, W. Maly, and M. E. Thomas. Extraction of defect size distributions in an ic layer using test structure data. *IEEE Transactions on Semiconductor Manufacturing*, 7(3):354–368, Aug 1994. [page 24]

[Kiefe00] G. Kiefer and H.-J. Wunderlich. Deterministic bist with partial scan. *Journal of Electronic Testing*, 16(3):169–177, 2000. [page 44]

[Kim03] K. S. Kim, S. Mitra, and P. G. Ryan. Delay defect characteristics and testing strategies. *IEEE Design & Test of Computers*, 20(5):8–16, Sept 2003. [page 22]

[Kocht10] M. A. Kochte, C. G. Zoellin, and H.-J. Wunderlich. Efficient concurrent self-test with partially specified patterns. *Journal of Electronic Testing: Theory and Applications (JETTA)*, 26(5):581–594, 2010. [pages 26, 49, 50, and 51]

[Koren07] I. Koren and C. Krishna. Chapter 3 - information redundancy. In I. Koren and C. M. Krishna, editors, *Fault-Tolerant Systems*, pages 55 – 108. Morgan Kaufmann, Burlington, 2007. ISBN 978-0-12-088525-1. [page 51]

[Krizh12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems*, NIPS'12, pages 1097–1105. Curran Associates Inc., USA, 2012. [page 121]

[Larra92] T. Larrabee. Test pattern generation using boolean satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(1):4–15, Jan 1992. [page 36]

[LeCun98] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998. [pages 121 and 122]

[Li03] Z. Li, X. Lu, W. Qiu, W. Shi, and D. M. H. Walker. A circuit level fault model for resistive opens and bridges. In *Proceedings of the 21st VLSI Test Symposium (VTS'03)*, pages 379–384. April 2003. [page 19]

[Liang02] H.-G. Liang, S. Hellebrand, and H.-J. Wunderlich. Two-Dimensional Test Data Compression for Scan-Based Deterministic BIST. *Journal of Electronic Testing: Theory and Applications (JETTA)*, 18(2):159–170, 2002. [page 45]

[Licht13] S. Lichtensteiger and J. P. Bickford. Using selective voltage binning to maximize yield. *IEEE Transactions on Semiconductor Manufacturing*, 26(4):436–441, Nov 2013. [page 23]

[Lieni05] J. Lienig and G. Jerke. Electromigration-aware physical design of integrated circuits. In *18th International Conference on VLSI Design held jointly with 4th International Conference on Embedded Systems Design*, pages 77–82. Jan 2005. [page 14]

[Lin14] F. Lin, C. K. Hsu, and K. T. Cheng. Feature engineering with canonical analysis for effective statistical tests screening test escapes. In *Proceedings of the IEEE International Test Conference (ITC'14)*, pages 1–10. Oct 2014. [page 84]

[Linga07] L. Lingappan and N. K. Jha. Satisfiability-based automatic test program generation and design for testability for microprocessors. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 15(5):518–530, May 2007. [page 54]

[Liou03] J.-J. Liou, A. Krstic, Y.-M. Jiang, and K.-T. Cheng. Modeling, testing, and analysis for delay defects and noise effects in deep submicron devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 22(6):756–769, June 2003. [page 74]

[Lippm87] R. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, 4(2):4–22, 1987. [page 80]

[Liu15a] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Penksy. Sparse convolutional neural networks. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 806–814. June 2015. [page 121]

[Liu15b] C. Liu, M. A. Kochte, and H.-J. Wunderlich. Efficient Observation Point Selection for Aging Monitoring. In *Proceedings of the 21st IEEE International On-Line Testing Symposium (IOLTS'15)*, pages 176–181. 2015. [page 26]

[LiVol11] R. LiVolsi, K. McCormick, M. Torres, J. Velamala, R. Zheng, and Y. Cao. Correlation of no trouble found errors to negative bias temperature instability. In *Proceedings of the IEEE Aerospace Conference (AeroConf'11)*, pages 1–8. March 2011. [pages 15 and 48]

[Lukas09] M. Lukasiewycz, M. Streubühr, M. Glaß, C. Haubelt, and J. Teich. Combined system synthesis and communication architecture exploration for mpsocs. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'09)*, DATE '09, pages 472–477. European Design and Automation Association, 3001 Leuven, Belgium, Belgium, 2009. ISBN 978-3-9810801-5-5. [pages 58 and 60]

[Madge05] R. Madge. New test paradigms for yield and manufacturability. *IEEE Design Test of Computers*, 22(3):240–246, May 2005. [pages 22 and 24]

[Maly03] W. Maly, A. Gattiker, T. Zanon, T. Vogels, R. Blanton, and T. Storey. Deformations of ic structure in test and yield learning. In *Proceedings of the IEEE International Test Conference (ITC'03)*, volume 1, pages 856–865. Sept 2003. [page 107]

[Maric11] E. Maricau, L. Zhang, J. Franco, P. Roussel, G. Groeseneken, and G. Gielen. A compact nbti model for accurate analog integrated circuit reliability simulation. In *2011 Proceedings of the European Solid-State Device Research Conference (ESSDERC)*, pages 147–150. Sept 2011. [page 15]

[Maric13] E. Maricau and G. Gielen. *CMOS Reliability Overview*, pages 15–35. Springer New York, New York, NY, 2013. ISBN 978-1-4614-6163-0. [pages 15 and 16]

[Maxwe16] P. Maxwell, F. Hapke, and H. Tang. Cell-aware diagnosis: Defective inmates exposed in their cells. In *Proceedings of the 21st IEEE European Test Symposium (ETS'16)*, pages 1–6. May 2016. [page 65]

[May06] G. S. May and C. J. Spanos. *Introduction to Semiconductor Manufacturing*, pages 1–24. John Wiley & Sons, Inc., 2006. ISBN 9780471790280. [page 21]

[McClu71] E. J. McCluskey and F. W. Clegg. Fault equivalence in combinational logic networks. *IEEE Transactions on Computers*, C-20(11):1286–1293, Nov 1971. [page 34]

[McPhe06] J. W. McPherson. Reliability challenges for 45nm and beyond. In *Proceedings of the 43rd Annual Design Automation Conference*, DAC '06, pages 176–181. ACM, New York, NY, USA, 2006. ISBN 1-59593-381-6. [page 21]

[Millm90] S. D. Millman, E. J. McCluskey, and J. M. Acken. Diagnosing cmos bridging faults with stuck-at fault dictionaries. In *Proceedings of the IEEE International Test Conference (ITC'90)*, pages 860–870. Sep 1990. [page 38]

[Mochi15] A. Mochizuki, N. Onizawa, A. Tamakoshi, and T. Hanyu. Multiple-event-transient soft-error gate-level simulator for harsh radiation environments. In *TENCON 2015 - 2015 IEEE Region 10 Conference*, pages 1–6. Nov 2015. [page 70]

[Mukhe12] R. Mukherjee, P. Ghosh, and A. Pal. Hotspot minimization using fine-grained dvs architecture at 90 nm technology. In *2012 Asia Pacific Conference on Postgraduate Research in Microelectronics and Electronics*, pages 13–18. Dec 2012. [page 71]

[Pasch05] A. Paschalis and D. Gizopoulos. Effective software-based self-test strategies for on-line periodic testing of embedded processors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 24(1):88–99, Jan 2005. [page 55]

[Paul07] S. Paul, S. Krishnamurthy, H. Mahmoodi, and S. Bhunia. Low-overhead design technique for calibration of maximum frequency at multiple operating points. In *2007 IEEE/ACM International Conference on Computer-Aided Design*, pages 401–404. Nov 2007. [page 23]

[Pawli88] T. F. Pawlicki, D.-S. Lee, J. J. Hull, and S. N. Srihari. Neural network models and their application to handwritten digit recognition. In *IEEE 1988 International Conference on Neural Networks*, pages 63–70 vol.2. July 1988. [page 68]

[Pearl09] J. Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, New York, NY, USA, 2009. ISBN 052189560X, 9780521895606. [pages 78, 90, 94, and 123]

[Pizza98] M. Pizza, L. Strigini, A. Bondavalli, and F. di Giandomenico. Optimal discrimination between transient and permanent faults. In *Proc. 3rd IEEE Int'l High-Assurance Systems Engineering Symposium*, pages 214–223. 1998. [page 64]

[Polia06] I. Polian, A. Czutro, S. Kundu, and B. Becker. Power droop testing. In *2006 International Conference on Computer Design*, pages 243–250. Oct 2006. [page 14]

[Psara10] M. Psarakis, D. Gizopoulos, E. Sanchez, and M. S. Reorda. Microprocessor software-based self-testing. *IEEE Design Test of Computers*, 27(3):4–19, May 2010. [pages 26, 53, and 62]

[Qian09] J. Qian, X. Wang, Q. Yang, F. Zhuang, J. Jia, X. Li, Y. Zuo, J. Mekkoth, J. Liu, H. J. Chao, S. Wu, H. Yang, L. Yu, F. Zhao, and L. T. Wang. Logic bist architecture for system-level test and diagnosis. In *Proceedings of the 18th IEEE Asian Test Symposium (ATS'09)*, pages 21–26. Nov 2009. [page 45]

[Reari05] J. Rearick, B. Eklow, K. Posse, A. Crouch, and B. Bennetts. Ijtag (internal jtag): a step toward a dft standard. In *Proceedings of the IEEE International Test Conference (ITC'05)*, pages 8 pp.–815. Nov 2005. [page 46]

[Reima14] F. Reimann, M. Glaß, J. Teich, A. Cook, L. Rodríguez Gómez, D. Ull, H.-J. Wunderlich, U. Abelein, and P. Engelke. Advanced Diagnosis: SBST and BIST Integration in Automotive E/E Architectures. In *Proceedings of the 51st ACM/IEEE Design Automation Conference (DAC'14)*, pages 1–9. 2014. [pages 26, 55, and 58]

[Rodri14] L. Rodríguez Gómez, A. Cook, T. Indlekofer, S. Hellebrand, and H.-J. Wunderlich. Adaptive Bayesian Diagnosis of Intermittent Faults. *Journal of*

Electronic Testing: Theory and Applications (JETTA), 30(5):527–540, 2014. [page 88]

[Rodri16] L. Rodríguez Gómez and H.-J. Wunderlich. A Neural-Network-Based Fault Classifier. In *Proceedings of the 25th IEEE Asian Test Symposium (ATS'16)*, pages 144–149. 2016. [page 109]

[Roth66] J. P. Roth. Diagnosis of automata failures: A calculus and a method. *IBM J. Res. Dev.*, 10(4):278–291, July 1966. [page 35]

[Sauer12] M. Sauer, A. Czutro, I. Polian, and B. Becker. Small-delay-fault atpg with waveform accuracy. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'12)*, pages 30–36. Nov 2012. [page 36]

[Sauer14] M. Sauer, I. Polian, M. E. Imhof, A. Mumtaz, E. Schneider, A. Czutro, H. J. Wunderlich, and B. Becker. Variation-aware deterministic atpg. In *Proceedings of the 19th IEEE European Test Symposium (ETS'14)*, pages 1–6. May 2014. [page 36]

[Schai15] M. Schain. *Machine Learning Algorithms and Robustness.* Ph.D. thesis, Tel Aviv University, 2015. [pages 68 and 69]

[Schne17] E. Schneider, M. A. Kochte, S. Holst, X. Wen, and H.-J. Wunderlich. GPU-Accelerated Simulation of Small Delay Faults. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 36(5):829–841, 2017. [pages 31, 33, 75, and 76]

[Segur04] J. Segura and C. H. Hawkins. *CMOS electronics. How it works, how it fails.* John Wiley and Sons, 2004. [pages 12, 13, 14, 16, 42, 70, and 73]

[Seltz13] M. L. Seltzer, D. Yu, and Y. Wang. An investigation of deep neural networks for noise robust speech recognition. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 7398–7402. May 2013. [page 69]

[Sharm88] R. Sharma and K. K. Saluja. An implementation and analysis of a concurrent built-in self-test technique. In *[1988] The Eighteenth International Symposium on Fault-Tolerant Computing. Digest of Papers*, pages 164–169. June 1988. [pages 26 and 50]

[Shen85] J. P. Shen, W. Maly, and F. J. Ferguson. Inductive fault analysis of mos integrated circuits. *IEEE Design Test of Computers*, 2(6):13–26, Dec 1985. [pages 36 and 65]

[Shen98] J. Shen and J. A. Abraham. Native mode functional test generation for processors with applications to self test and design validation. In *Proceedings of the International Test Conference 1998*, pages 990–999. Oct 1998. [pages 54 and 55]

[Shen12] R. Shen, S. X. D. Tan, and X. X. Liu. A new voltage binning technique for yield improvement based on graph theory. In *Thirteenth International Symposium on Quality Electronic Design (ISQED)*, pages 243–248. March 2012. [pages 22 and 23]

[Shin16a] C. Shin. *Line edge roughness (LER)*, pages 19–35. Springer Netherlands, 2016. ISBN 978-94-017-7597-7. [pages 72 and 73]

[Shin16b] C. Shin. *Random Dopant Fluctuation (RDF)*, pages 37–52. Springer Netherlands, 2016. ISBN 978-94-017-7597-7. [pages 72, 73, and 74]

[Shiva02] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvisi. Modeling the effect of technology trends on the soft error rate of combinational logic. In *Proceedings International Conference on Dependable Systems and Networks*, pages 389–398. 2002. [page 70]

[Sokol09] M. Sokolova and G. Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427 – 437, 2009. [pages 125 and 126]

[Sriva05] A. Srivastava, D. Sylvester, and D. Blaauw. *Statistical Analysis and Optimization for VLSI: Timing and Power*. Integrated Circuits and Systems. Springer, 2005. ISBN 9780387257389. [pages 72 and 74]

[Stein08] I. Steinwart and A. Christmann. *Support Vector Machines*. Springer Publishing Company, Incorporated, 1st edition, 2008. ISBN 0387772413. [page 81]

[Sucar15] L. E. Sucar. *Probabilistic Graphical Models: Principles and Applications*. Springer, London, 2015. ISBN 978-144-716-699-3. [pages 68, 78, 90, and 94]

[Sumik12] N. Sumikawa, J. Tikkanen, L. C. Wang, L. Winemberg, and M. S. Abadir. Screening customer returns with multivariate test analysis. In *2012 IEEE International Test Conference*, pages 1–10. Nov 2012. [page 84]

[Tehra11] M. Tehranipoor, K. Peng, and K. Chakrabarty. *Test and Diagnosis for Small-Delay Defects*. Springer New York, 2011. ISBN 9781441982971. [page 18]

[Thatt80] S. M. Thatte and J. A. Abraham. Test generation for microprocessors. *IEEE Transactions on Computers*, C-29(6):429–441, June 1980. [page 54]

[Touba97] N. A. Touba and E. J. McCluskey. Logic synthesis of multilevel circuits with concurrent error detection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 16(7):783–789, Jul 1997. [pages 48, 51, 52, and 63]

[Vemu08] R. Vemu, A. Jas, J. A. Abraham, S. Patil, and R. Galivanche. A low-cost concurrent error detection technique for processor control logic. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'08)*, pages 897–902. March 2008. [page 52]

[Venka01] S. Venkataraman and S. Drummonds. Poirot: Applications of a logic fault diagnosis tool. *IEEE Design & Test*, 18(1):19–30, January 2001. [page 38]

[Vince08] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 1096–1103. ACM, New York, NY, USA, 2008. ISBN 978-1-60558-205-4. [page 69]

[Vo06] T. Vo, Z. Wang, T. Eaton, P. Ghosh, H. Li, Y. Lee, W. Wang, H. Jun, R. Fang, D. Singletary, and X. Gu. Design for board and system level structural test and diagnosis. In *Proceedings of the IEEE International Test Conference (ITC'06)*, pages 1–10. Oct 2006. [page 45]

[Vrank02] H. Vranken, F. Meister, and H.-J. Wunderlich. Combining Deterministic Logic BIST with Test Point Insertion. In *Proceedings of the 7th European Test Workshop (ETW'02)*, pages 105–110. IEEE Computer Society, 2002. [page 45]

[Wadsa78] R. L. Wadsack. Fault modeling and logic simulation of cmos and mos integrated circuits. *Bell System Technical Journal*, 57(5):1449–1474, May 1978. [page 13]

[Waicu87] J. A. Waicukauski, E. Lindbloom, B. K. Rosen, and V. S. Iyengar. Transition fault simulation. *IEEE Design Test of Computers*, 4(2):32–38, April 1987. [page 32]

[Wali09] F. Wali, M. Knotter, A. Mud, and F. Kuper. Impact of particles in ultra pure water on random yield loss in {IC} production. *Microelectronic Engineering*, 86(2):140 – 144, 2009. The 10th annual {SEMATECH} Surface Preparation and Cleaning Conference (SPCC). [page 22]

[Wang06] L.-T. Wang, C.-W. Wu, and X. Wen. *VLSI Test Principles and Architectures: Design for Testability (Systems on Silicon)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006. ISBN 0123705975. [pages 17, 18, 19, 20, 29, and 45]

[Wen04] X. Wen, T. Miyoshi, S. Kajihara, L.-T. Wang, K. K. Saluja, and K. Kinoshita. On per-test fault diagnosis using the x-fault model. In *Proceedings of the 2004 IEEE/ACM International Conference on Computer Aided Design (ICCAD'04)*, pages 633–640. IEEE Computer Society, Washington, DC, USA, 2004. ISBN 0-7803-8702-3. [page 40]

[Weste11] N. Weste and D. Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison Wesley, 2011. ISBN 9780321547743. [pages 10, 11, 13, 15, 16, 71, 73, and 74]

[Wolpe12] D. Wolpert and P. Ampadu. *Temperature Effects in Semiconductors*, pages 15–33. Springer New York, New York, NY, 2012. ISBN 978-1-4614-0748-5. [page 13]

[Wu08] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, 2008. [pages 79, 80, and 81]

[Wunde98] H.-J. Wunderlich. Bist for systems-on-a-chip. *Integr. VLSI J.*, 26(1-2):55–78, December 1998. [pages 26, 44, and 62]

[Wunde07] H.-J. Wunderlich, M. Elm, and S. Holst. Debug and diagnosis: Mastering the life cycle of nano-scale systems on chip (invited paper). In *Proceedings of 43rd International Conference on Microelectronics, Devices and Material with the Workshop on Electronic Testing (MIDEM'07)*, pages 27–36. MIDEM, 2007. [page 37]

[Wunde10] H.-J. Wunderlich, editor. *Models in Hardware Testing*, volume 43. Springer-Verlag Heidelberg, 2010. [pages 12, 13, 17, and 18]

[Xue13] Y. Xue, O. Poku, X. Li, and R. D. Blanton. Padre: Physically-aware diagnostic resolution enhancement. In *Proceedings of the IEEE International Test Conference (ITC'13)*, pages 1–10. Sept 2013. [page 83]

[Yamaz13] K. Yamazaki, T. Tsutsumi, H. Takahashi, Y. Higami, H. Yotsuyanagi, M. Hashizume, and K. K. Saluja. Diagnosing resistive open faults using small delay fault simulation. In *Proceedings of the 22nd IEEE Asian Test Symposium (ATS'13)*, pages 79–84. Nov 2013. [page 37]

[Ye12] F. Ye, Z. Zhang, K. Chakrabarty, and X. Gu. Adaptive board-level functional fault diagnosis using decision trees. In *Proceedings of the 21st IEEE Asian Test Symposium (ATS'12)*, pages 202–207. Nov 2012. [page 84]

[Ye13] F. Ye, Z. Zhang, K. Chakrabarty, and X. Gu. Board-level functional fault diagnosis using artificial neural networks, support-vector machines, and weighted-majority voting. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 32(5):723–736, 2013. [page 83]

[Yi15] C. H. Yi and J. W. Jeon. Power saving using partial networking in automotive system. In *2015 IEEE International Conference on Information and Automation*, pages 148–152. Aug 2015. [page 56]

[Zeile12] M. D. Zeiler. Adadelta: An adaptive learning rate method. *CoRR*, abs/1212.5701, 2012. [page 127]

[Zeng04] J. Zeng, M. S. Abadir, G. Vandling, L. C. Wang, S. Karako, and J. A. Abraham. On correlating structural tests with functional tests for speed binning of high performance design. In *Microprocessor Test and Verification (MTV'04), Fifth International Workshop on*, pages 103–109. Sept 2004. [page 23]

[Zhang10a] Y. Zhang and V. D. Agrawal. A diagnostic test generation system. In *2010 IEEE International Test Conference*, pages 1–9. Nov 2010. [pages 46, 47, and 62]

[Zhang10b] Z. Zhang, Z. Wang, X. Gu, and K. Chakrabarty. Board-level fault diagnosis using bayesian inference. In *2010 28th VLSI Test Symposium (VTS)*, pages 244–249. April 2010. [page 83]

[Zhao06] C. Zhao and S. Dey. Evaluating and improving transient error tolerance of cmos digital vlsi circuits. In *2006 IEEE International Test Conference*, pages 1–10. Oct 2006. [pages 70 and 71]

List of Abbreviations

Abbreviation	Meaning
Ψ	input space of a machine learning function
λ_a	activation rate of an intermittent fault
Ω	output space of a machine learning function
μ_a	activation rate of a transient fault
ANN	artificial neural network
b_{or}	dominant or bridge fault class
b_{and}	dominant and bridge fault class
b_{byz}	byzantine bridge fault class
BIST	built-in self-test
ct	crosstalk induce delay fault class
$d_s^{c_i}$	set of detecting patterns for fault c_i
IC	integrated circuit
CLF	Conditional line flip
CMOS	complementary metal oxide semiconductor
L	set of labels in a classification problem
$N(v_l)$	set of lines which conform the neighborhood of v_l
$nd_s^{c_i}$	set of non-detecting patterns for fault c_i
R_{bridge}	Resistance of an interconnect bridge defect
$R_{critical}$	Critical resistance value of a bridge
SBST	software-based self-test
s_a	aggressor signal in a fault
s_{fall}	slow-to-fall gate fault class
s_{rise}	slow-to-rise gate fault class
s_v	victim signal in a fault
V_{DD}	Supply voltage

Abbreviation	Meaning
V_{GB}	Voltage between the gate and body terminals of a transistor
V_{GND}	Ground voltage
V_t^p	Threshold voltage of a pMOS transistor
V_t^n	Threshold voltage of an nMOS transistor

Index

Publications of the Author

- **Specification and Verification of Security in Reconfigurable Scan Networks** Kochte, M.A., Sauer, M., Rodríguez Gómez, L., Raiola, P., Becker, B. and Wunderlich, H.-J. Proceedings of the 22nd IEEE European Test Symposium (ETS'17), Limassol, Cyprus, 22-26 May 2017

- **A Neural-Network-Based Fault Classifier** Rodríguez Gómez, L. and Wunderlich, H.-J. Proceedings of the 25th IEEE Asian Test Symposium (ATS'16), Hiroshima, Japan, 21-24 November 2016

- **Adaptive Bayesian Diagnosis of Intermittent Faults**, Rodríguez Gómez, L., Cook, A., Indlekofer, T., Hellebrand, S. and Wunderlich, H.-J. Journal of Electronic Testing: Theory and Applications (JETTA) Vol. 30(5), 30 September 2014, pp. 527-540

- **Advanced Diagnosis: SBST and BIST Integration in Automotive E/E Architectures**, Reimann, F., Glaß, M., Teich, J., Cook, A., Rodríguez Gómez, L., Ull, D., Wunderlich, H.-J., Abelein, U. and Engelke, P. Proceedings of the 51st ACM/IEEE Design Automation Conference (DAC'14), San Francisco, California, USA, 1-5 June 2014, pp. 1-9 HiPEAC Paper Award

- **Non-Intrusive Integration of Advanced Diagnosis Features in Automotive E/E-Architectures**, Abelein, U., Cook, A., Engelke, P., Glaß, M., Reimann, F., Rodríguez Gómez, L., Russ, T., Teich, J., Ull, D. and Wunderlich, H.-J. Proceedings of the Design, Automation and Test in Europe (DATE'14), Dresden, Germany, 24-28 March 2014

Declaration

All the work contained within this thesis, except where otherwise acknowledged, was solely the effort of the author. At no stage was any collaboration entered into with any other party.

Laura Rodríguez Gómez