Efficient Location–Based Logic Diagnosis of Digital Circuits

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

Stefan Holst

aus Cuxhaven

Hauptberichter:	Prof. Dr. rer. nat. H.–J. Wunderlich
Mitberichter:	Prof. Dr. Paolo Prinetto
Tag der mündlichen Prüfung:	28.09.2012

Institut für Technische Informatik der Universität Stuttgart

2012

To my mom.

Acknowledgements

It is a pleasure to thank all those who made this work possible.

I'm very grateful to my parents, Marianne and Erhard Holst, who constantly supported me during my education and studies. Without them, I would not have been able to even start with a work like this.

I thank Prof. Wunderlich, who allowed me to work at the *Institut für Technische Informatik* (ITI) and let me explore the complete spectrum of academic activities from research over teaching to administration. He constantly provided insightful sparks, which nurtured my personal and professional growth towards an interdependent researcher. Furthermore, I thank my second reviewer, Prof. Prinetto, for monitoring my work over the years on conferences and workshops and for providing the finishing touches for this thesis.

The whole experience would not have been so fruitful and so much fun without all my advisors, colleagues and students I worked with at the ITI. I'll follow a loose chronological order in the hope that I miss nobody—if I missed you, it is my fault alone. At the very beginning, Günter Bartsch and Rainer Dorsch served as advisors during my undergraduate studies and helped me getting started at the ITI. Among the many former colleagues, Wolfgang Moser, Abdul-Wahid Hakmi, Christian Zöllin, and Melanie Elm occupy the biggest chunks of my memory. Them included, I also thank all my current research colleagues Michael E. Imhof, Michael Kochte, Abdullah Mumtaz, Rafał Baranowski, Claus Braun, Anusha Kakarala, Marcus Wagner, Alejandro Cook, Atefe Dalirsani, Nadareh Hatami, Dominik Ull, Laura Rodríguez Gómez, Chang Liu, and Eric Schneider for insightful discussions, inspiration, joint research, honest feedback, fun, and much more.

No institute would function so smoothly without excellent secretaries and administrative staff. Thank you Mirjam Breitling, Helmut Häfner, and Lothar Hellmeier for providing the necessary grease and helping all of us reaching our goals.

Last, but not least, I would like to thank all the students from lectures, seminars, student projects for asking critical questions and providing different perspectives. I'm especially grateful to Ozan Kasimoglu, Jan-Peter Ostberg, Alejandro Cook, Tamer Dallou, Stefan Bayha, Hussam El Atali, Maha Badreldein, Marie Delmas, Eric Schneider, Alexander Schöll, and Sebastian Halder who chose me as their thesis advisor and enriched my research with their thoughts and ideas.

Stuttgart, October 2012 *Stefan Holst*

Contents

Abbreviations and Notations

$\oplus$	Exclusive–OR operator
$\equiv$	Equivalence operator
$\vee$	OR operator
$\wedge$, $\cdot$	AND operators
$\rightarrow$	Implication operator
$\cup$	Union operator
$\cap$	Intersection operator
$\emptyset$	Empty set
$\|\vec{v}\|$	Number of ones in the vector $\vec{v}$ (1–norm)
ATE	Automatic Test Equipment
ATPG	Automatic Test Pattern Generator
C	Combinational circuit
C_f	Circuit with a fault f injected
C_{ud}	Circuit under diagnosis, with unknown defect
CLF	Conditional Line Flip
CMOS	Complementary Metal Oxide Semiconductor
CMP	Chemical–Mechanical Polishing
DfD	Design–for–Diagnosability
DfT	Design–for–Test
F	Fault set
GFM	Generalized Fault Model
GPGPU	General Purpose Graphics Processing Unit
I_{ddq}	Supply current in the quiescent state of a circuit
LBIST	Logic Built–In Self–Test
MEPS	Million Evidences Per Second
MISR	Multiple–Input Shift Register
NBTI	Negative Bias Temperature Instability
OPC	Optical Proximity Correction
PFA	Physical Failure Analysis
POINTER	Partially Overlapping Impact couNTER
PPSFP	Parallel–Pattern Single–Fault Propagation
RET	Resolution Enhancement Technique
$\vec{r}$	Response vector, response pattern

SAT	Boolean SATisfiability
SLAT	Single Location At a Time
STUMPS	Self–Test Using a MISR and Parallel Shift register sequence generator
$\vec{t}$	Test vector, test pattern
T	Test vector set
TFSF	Tester–Fail Simulation–Fail
TFSP	Tester–Fail Simulation–Pass
TPSF	Tester–Pass Simulation–Fail

Summary

Logic diagnosis is the task of finding defects within a random logic circuit based on its faulty behavior. Fast and accurate algorithms for logic diagnosis are an integral part of modern chip development. Classic diagnosis algorithms were often based on fault models which contain a priori *assumptions* on the behavior of defects. In recent technologies, fault model based approaches become ineffective because defect mechanisms get more and more complex. So research has started on *location–based* diagnosis algorithms, which use more general fault models or no model at all and report defective substructures directly.

The generality however may also have a negative effect on the accuracy of the diagnosis results. With the lack of a fault model, a diagnosis algorithm has less *knowledge* on possible or likely malfunctions of a circuit. This increases the search space dramatically and may even lead to defect candidates which are physically impossible. Reducing a priori assumptions while retaining sufficient knowledge on likely defect mechanisms is the key to effective logic diagnosis.

This work introduces the *Conditional Line Flip* (CLF) calculus as a way to describe arbitrary defects in logic circuits. This generalized fault modeling approach is used to investigate the assumptions made by diagnostic fault models and diagnosis algorithms found in the literature.

The second main contribution of this work is a location–based logic diagnosis algorithm called *Partially Overlapping Impact couNTER* (POINTER). It builds directly upon the CLF calculus, works independently of any specialized fault model and offers powerful heuristics for sorting defect candidates according to their likelihood in physical chips. The POINTER approach is extended and modified to account for the particular challenges of high precision diagnostics in a lab, during production, and in autonomous online diagnosis in the field.

Experimental results on industrial designs confirm that, despite its generality and lack of application specific knowledge, POINTER performs much better than previous diagnosis approaches. In cases where very high response compaction ratios are used, POINTER even enables fault model independent diagnosis for the first time.

Zusammenfassung

Mit *Logik–Diagnose* bezeichnet man die Aufgabe, Fehler in logischen Schaltungen anhand der beobachteten Syndrome der Gesamtschaltung zu finden. Schnelle und genaue Algorithmen dafür sind essentieller Bestandteil der modernen Chipentwicklung. Klassische Algorithmen basierten oft auf Fehlermodellen, welche a–priori *Annahmen* über mögliche Defekte treffen. Das Verhalten von Defekten wird mit fortschreitender Technologie jedoch immer vielfältiger, so dass fehlermodell–basierte Algorithmen immer ineffektiver werden. Damit begann die Erforschung von *ortsbasierten* Diagnosealgorithmen, die unter weitaus schwächeren Fehlerannahmen defekte Unterstrukturen in Schaltungen direkt auffinden können.

Diese Allgemeingültigkeit kann jedoch auch einen negativen Effekt auf die Präzision der Ergebnisse haben. Mit dem Fehlen eines exakten Fehlermodells hat ein Algorithmus auch weniger *Wissen* über mögliche und wahrscheinliche Fehlfunktionen eines Chips. Dies erweitert den Suchraum dramatisch und kann sogar zu Diagnoseergebnissen führen, die physikalisch unmöglich sind. Der Schlüssel zu einer effektiven Diagnose ist daher die Reduktion von Annahmen unter Beibehaltung von hinreichendem Wissen über mögliche Defektmechanismen.

Die vorliegende Arbeit führt mit dem *Conditional Line Flip* (CLF) Kalkül eine Methode ein, um beliebige Defekte in Logik–Schaltungen zu beschreiben. Diese verallgemeinerte Fehlermodellierung wird dann dazu genutzt, die Annahmen und Eigenschaften vorgeschlagener diagnostischer Fehlermodelle und Diagnosealgorithmen zu beleuchten.

Der zweite Hauptbeitrag dieser Arbeit ist ein ortsbasierter Logik–Diagnosealgorithmus namens *Partially Overlapping Impact couNTER* (POINTER). Er basiert direkt auf dem CLF–Kalkül, arbeitet unabhängig von spezialisierten Fehlermodellen und bietet leistungsfähige Heuristiken zur Sortierung von Defekt–Kandidaten anhand ihrer Wahrscheinlichkeit, tatsächlich im Chip vorzukommen. Der POINTER–Ansatz wird daraufhin noch erweitert, um den speziellen Herausforderungen der Präzisionsdiagnose im Labor, der Diagnose während der Produktion, und der autonomen Diagnose im Feld Rechnung zu tragen.

Die experimentellen Ergebnisse bestätigen, dass POINTER trotz seiner Allgemeingültigkeit und ohne tiefes Wissen über die physikalischen Vorgänge bei Defekten, wesentlich bessere Ergebnisse liefert als bisherige Diagnose–Ansätze. Wenn extreme Test–Antwort–Kompaktierung eingesetzt wird, ermöglicht POINTER sogar erstmals die fehlermodell–unabhängige Diagnose der Syndrome.

1. Introduction

The development and production of modern chips is an error–prone, expensive and highly competitive task. Every semiconductor company competing in the field of leading edge technologies has to face the challenge of building more powerful and more complex silicon chips in large quantities and high quality in a very short time. The faster the development and the production ramp–up, the higher is the potential profit by being the market leader with a new generation of processors or memory chips. To offset the high investment necessary to gain market leadership with a new product, a company must be an efficient learner to reduce the production cost of this product as fast as possible. As competitors enter the market with comparable products and begin their learning process, the competitive advantage of the former market leader is now defined over the achieved production efficiency and margins [Hutch2008].

Today, only very few firms can afford this kind of competition. One indicator are the enormous investments needed to build a high–class production facility. In the recent years, the average cost of a single production site was about 1.5 Billion dollars [Hutch2008], Intel recently announced in [Intel2011] an investment of over 5 Billion dollars to build a new fab. Some even argue, that Moore's Law [Moore1965, Moore1975] will eventually end because of economic considerations rather than technological or physical limits [RuppS2011].

The advancements in manufacturing technology made it possible to exponentially increase the complexity of chips [ITRS2011] fulfilling Moore's Law for more than 40 years. With growing circuit complexity and shrinking geometries, the actual behavior of the silicon is hard to model and cannot always be predicted or simulated [ITRS2011]. In order to improve production efficiency and margins, one has to learn from the flawed chips directly through *failure analysis*. Essential tools for failure analysis are *logic diagnosis algorithms*, which identify the defective structures within a chip by analyzing its faulty test responses [WangWW2006, WundeEH2007, WundeEH*2007a].

This chapter will first discuss the most common sources of malfunctions in logic circuits to identify the targets for all the investigations. Then, common practices in logic test, design–for–test, and design–for–diagnosis are presented in order to show the environments logic diagnosis algorithms have to work in. Then, after highlighting the main applications and challenges for logic diagnosis algorithms, this chapter finishes with an overview of this thesis.

1.1. Sources of Malfunctions in Random Logic Circuits

Due to *process variations*, the behavior of a manufactured chip may differ from expectations or simulation models. In many cases, the chips can be used despite the varied behavior and simulation models are refined to match the observed behavior to improve predictability for future designs. In some cases however, some functionality is rendered unusable due to an abnormality in the behavior, which is then called a *malfunction* and causes *yield loss*.

Figure 1.1 shows one way to categorize the main factors contributing to malfunctions and their possible causes in modern chip manufacturing. The goal of *yield learning* is to identify the causes of yield loss (*yield limiters*) and to avoid them in the future. One of the first steps is the identification of defects within the circuit using logic diagnosis algorithms. Although the diagnosis algorithms developed here work without any knowledge about specific defect mechanisms, the most common ones are discussed here in order to relate them to the causes in the production process.

First, an overview over the most common causes of *random defects* is given. Afterwards, the section on *systematic defects* will focus on the interaction between the design (the layout) and the production process. The physical causes of *aging* are discussed last.

1.1.1. Random Defects

Chip manufacturing consists of hundreds of physical processing steps applied to a silicon wafer. The single steps involve deposition of materials using liquids, gases and galvanic processes, defining structures using photolithography,

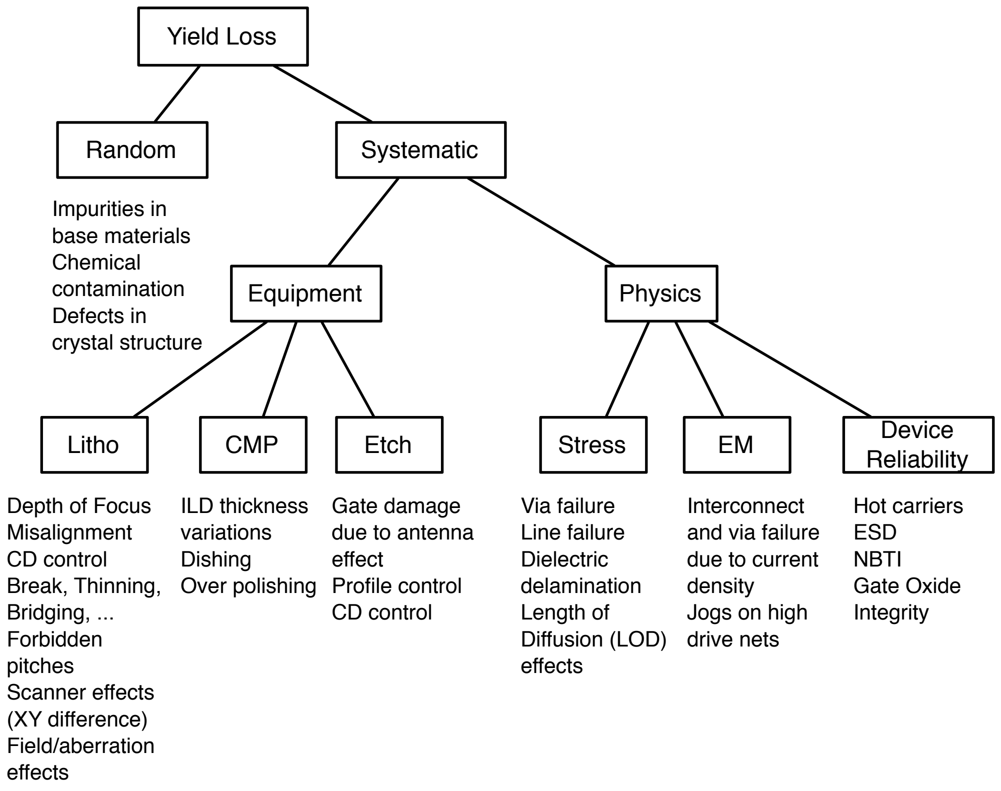


Figure 1.1.: Sources of yield loss; based on [ChianK2007].

implantation of ions using plasmas, removal of materials by etching and mechanical polishing. Although the base materials and processing environments are very pure and clean, impurities and trace amounts of moisture or oxygen are inevitable [BreauC2008]. The majority of these contaminants originate from the source materials, the equipment and the processes themselves rather than from the cleanroom or people [BreauC2008]. Depending on the process step, *contaminations* can cause various kinds of random defects. Some typical examples are [BreauC2008]:

- Early gate oxide breakdown (malfunctioning transistors) due to impurities in the silicon wafer, residual oxygen or other gases entering the reaction chambers in the front–end process.

- Excess or missing material in transistors or interconnect can be caused by improper surface cleaning. Residual contaminants may act as nucleation sites during film growth and the following processing steps are disturbed.

- Micro–bridging and pattern erosion are typical results of a contaminated lithography step. In photolithography, chemical contamination from the light source can lead to lens clouding, particles on the backside of the wafer may cause defocus and the mask material is known to degrade over continued exposure to the ultraviolet light.
- Interconnect defects (like opens or bridges) are often caused by etch residues, incomplete etches, scratches and resulting debris from chemical–mechanical polishing.

The strive for cleaner base materials and the reduction of contaminations adds considerable cost to the production process. To guide and focus these efforts towards maximum gain in yield, it is important to locate defects stemming from random sources and attribute them to the specific process steps [SharmBL*2008].

1.1.2. Systematic Defects

In a common definition, *systematic defects* are the results of causes which are avoidable by tuning of process parameters or changes in the design [ChianK2007]. Of this broad area of research, this section will focus on the dependencies between the layout of random logic and the fabrication process. The two major contributors to the so–called *design–process–interactions* for logic circuits are *photolithography* and *chemical–mechanical polishing* during fabrication of the interconnect layers of the chip (back–end process).

Photolithography

Photolithography is the technique to define the structures to be processed in a chip. A wafer is coated with a light sensitive *photo resist* material and is illuminated by a light source shining through a *mask*. The illuminated part of the photo resist changes its chemical properties and is etched away in the following processing steps. The best known light source used in chip manufacturing, the Argon Floride excimer laser, produces ultraviolet light with a wavelength of $\lambda = 193\mathrm{nm}$. The most advanced technology node for logic circuits currently entering mass production uses a half–pitch size of $22\mathrm{nm}$ [KuhnLK2010, Intel2011a]. Since many years now, the chip structures are much smaller than the wavelength of the used light and complex *resolution enhancement techniques* (RET) are used to print these structures [Liebm2003]. The most prominent among these techniques are *optical proximity correction* (OPC) to reduce corner rounding and line shortening in the silicon, and *phase shifting masks* to exploit optical interference to print structures smaller than λ. RET poses restrictions on the layout of the metal interconnect. Many configurations of neighboring features are impossible to manufacture as they cannot be translated into a RET–enabled mask. These conflicts are mostly taken care of by *design rules* and lithography–aware routing [Liebm2003]. Others can be manufactured, but affect yield due to marginalities in the lithography process [ChianK2007]. Using these configurations is often a trade–off between yield and performance of the layout. Typical results are bridge and open defects in the interconnect which have to be diagnosed in order to discover problematic structures and quantify their impact on the yield.

Chemical–Mechanical Polishing

The interconnect layers of modern chips are fabricated using the Damascene process [ChianK2007]. To process a single metal layer, first, an insulator is added to the surface. Then, the insulator is partly removed with photolithography and etching steps to form trenches and holes for metal lines and vias. Using a step called *electroplating*, metal (usually copper) is deposited onto the surface filling the trenches and holes. Finally, the excess metal is removed by *chemical–mechanical polishing* (CMP).

The challenge is to polish the complete surface as evenly as possible despite the irregularity of the layout features beneath. If the irregularities are not properly accounted for, the polishing process may remove too much or too little material in some regions which in turn increases the probability for open defects or bridging defects [ChianK2007]. Logic diagnosis of each failing chip and statistical processing of the layout locations of the defect candidates is typically used to locate the regions polished improperly.

1.1.3. Aging

Modern circuits may change their behavior under continued use and eventually fail in the field long after production. Many aging–related defects are caused by a design or a production problem which accelerates the physical processes described below. Such defects are called *latent defects* and can be discovered to a certain extend by burn–in testing, which accelerates the aging processes by factors of 1000 to 10000. A more complete discussion on burn–in testing can be found in [McPhe2006].

Electromigration

Under certain conditions, electrons in metal wires carry enough momentum in order to dislocate metal atoms and create voids and bulges in the metal [Segura2004]. An empirical study on aluminum interconnects lead to the so–called Black's Law [Black1967] which states that the median time to failure t_F depends on the temperature T in Kelvin, the electron current density j_e (A/cm^2), the activation energy E_a (eV), the Boltzmann constant k, and a technology constant A_0:

$$t_F = \frac{A_0}{j_e^2} e^{E_a/kT}$$

With shrinking structures, the electron current density j_e usually increases and reduces the mean time to failure considerably. Even more importantly, as there are more and more structures in the same chip area, the power density might increase leading to higher operation temperatures. This temperature has an exponential effect on t_F. There are design rules in place in order to keep electromigration under control. However, t_F is just a statistical quantity and the effects within a signal line strongly depend on the actual metal grain structure [Segura2004]. Some chips may fail much earlier due to this effect than predicted.

Electromigration commonly causes two types of failures. The first type is an open defect caused by a void left by migrated metal atoms which eventually spans the width of the signal line and disconnects it. The second type of defects are bridges caused by bulges created by the dislocated metal atoms and eventually connect to neighboring metal lines [Segura2004].

Oxide Wearout and Breakdown

There is evidence that in so–called *ultrathin* gate oxides (below 30 Å in thickness), electrons tunneling through the oxide can cause *traps* [Segura2004]. Traps are missing oxygen (O) atoms in the amorphous SiO_2 structure causing the resistance of the gate oxide to decrease. The *percolation model* [DegraKD*2001] states that, in continuously stressed transistors, more and more punctual traps are formed within the oxide leading to wearout. Critically aligned traps may finally create a path through the oxide barrier and the increased current thermally damages the transistor leading to a *hard breakdown*, a conducting path through the gate oxide leading to a catastrophic failure of the transistor.

Hot Carrier Injection

Transistors designed for high switching frequencies usually feature relatively short channel lengths. The strong electric field at the drain of such transistors in saturation state can cause *hot electrons* entering the depletion region causing damage to nMOS transistors [Segura2004]. This effect increases the threshold voltage of nMOS transistors leading to a reduction of operating frequency of the circuit over time. Proper guardbanding in the operating frequency can be of help, however unexpected stress on transistors may lead to larger parameter shifts and to timing failures of the circuit [Segura2004].

Negative Bias Temperature Instability

This effect mainly affects pMOS transistors causing an increase in the threshold voltage like hot carrier injection does for nMOS transistors. The parameter shift depends on the stress applied to the device. In contrast to hot carrier injection, transistors are known to recover significantly after the stress is reduced leading to a relaxation of the threshold voltage shift [Segura2004, GieleWM*2008]. The exact physical cause of NBTI is still not fully understood [Segura2004], among the most accepted factors for NBTI are a hydrogen release from the oxide/substrate border and hole trapping in the oxide [GieleWM*2008].

1.2. Cost Efficient Logic Test Infrastructure

During mass production, each chip goes through a series of tests to ensure an acceptable quality of the shipped products [BushnA2000]. The tests that target logic circuits (the networks of state–elements and logic gates which implement the functionality of the chip) are called *logic tests*.

Logic test is performed by *automatic test equipment* (ATE), which provides power and input stimuli, and records electrical characteristics as well as logic data produced by the chip. The chip contains *test infrastructure* to provide the ATE access to logic blocks or to perform self–tests.

Adding the test infrastructure during design (*design–for–test*, DfT), the additional chip area spend and the investments in and operation of ATE create major additional costs which may even exceed the cost for fabrication [BushnA2000]. Moreover, the methods used to test logic circuits and other parts of the chip are already designed to provide the best trade–off between desired product quality and test cost. Any additional feature in the test infrastructure or ATE and any increase in test time per chip needed for logic diagnosis approaches have to be carefully examined regarding increase in test cost and benefit of diagnosis results.

The common industrial practices in design–for–test, design–for–diagnosability (DfD) and their implications for logic diagnosis algorithms are discussed below.

1.2.1. Scan Test

Scan test [EicheW1977] is the most widely used method to test logic circuits. The added test infrastructure connects the state elements like flip–flops or latches to shift registers called *scan chains*. Typical industrial designs use multiple scan chains in parallel to reduce the number of shift cycles (see figure 1.2). During the normal operation of the logic circuit (*functional mode*), these scan chains are deactivated. In *test mode*, data can be shifted via the scan chains and independently of the surrounding logic through the circuit. This enables direct control and direct observation of the state elements within the circuit [WangWW2006].

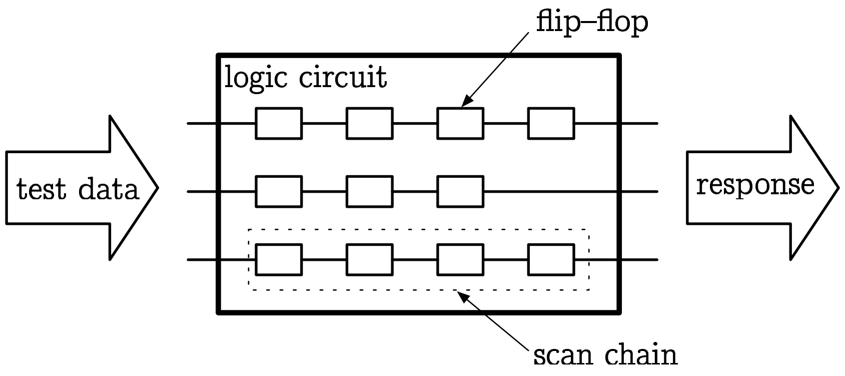


Figure 1.2.: Logic test using scan chains.

If every state element within the tested circuit is accessible via a scan chain, it is called a *full scan design*. In this case, test pattern generation as well as logic diagnosis algorithms are only concerned with the combinational logic structures (gates and signals) between the state elements. Figure 1.3 illustrates the conversion of a full scan design into a purely combinational simulation model called *combinational equivalent*. On the left hand side, a design is sketched, which has two primary inputs i_1, i_2, two primary outputs q_1, q_2 and three flip–flops. All flip–flops are accessible via a scan chain. As all flip–flops can be set directly in test mode using the scan chain, it is equivalent to regarding the output signals of the flip–flops z_1, z_2, z_3 as inputs of the circuit. These inputs are commonly called *pseudo–primary inputs*. All flip–flops can be directly observed in test mode via the scan chain, thus the input signals of the flip–flops coming from the logic circuit a_1, a_2, a_3 can be regarded as *pseudo–primary outputs* [BushnA2000].

The combinational equivalent is independent of the number of scan chains or the ordering of the flip–flops in the chains. Test generation algorithms as well as logic diagnosis algorithms can operate directly on the combinational equivalent under the assumption that the test infrastructure itself is fault–free. For testing and diagnosing the test infrastructure itself, specialized methods are used [WangWW2006] which are not the concern of this thesis. The remainder of this work will always assume a full scan test and use the combinational equivalents.

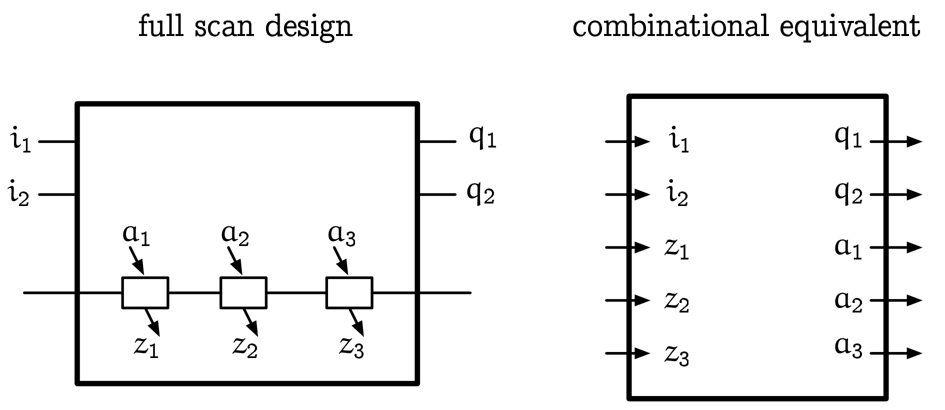


Figure 1.3.: Combinational equivalent of a full scan design.

1.2.2. Scan Compression

The amount of test data to be transfered between the ATE and the chip is reduced by the use of on–chip test data decompressors and test response compactors [WangST2008, HakmiHW*2009] illustrated in figure 1.4.

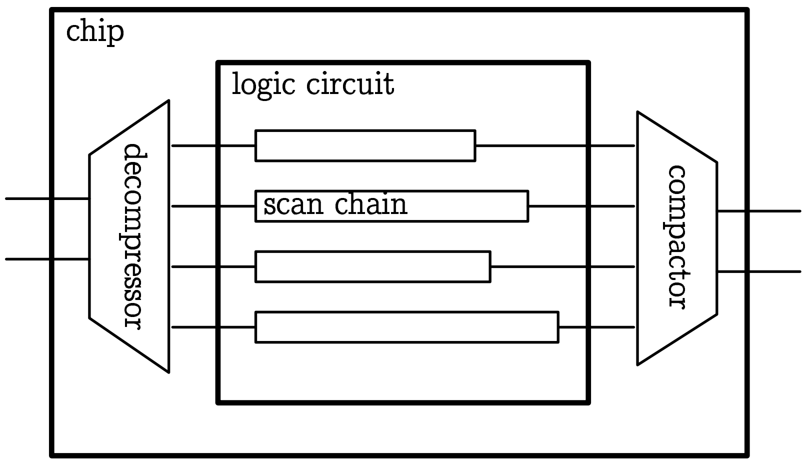


Figure 1.4.: Principle of scan compression.

Test data decompression can be supported rather easily by logic diagnosis algorithms, because the inputs to the combinational equivalent can be obtained by simulating the decompressor structure with the known compressed test data. However, there is no direct access anymore to the full test response of the combinational equivalent. Three basic approaches can be distinguished for enabling diagnosis with test response compaction [Cheng2004].

Bypassing

Failing chips are re–tested with reduced or completely disabled response compaction to obtain more response data for diagnosis [WangWW2006]. The re–testing introduces a major test time overhead, which may be acceptable during prototyping or precision diagnosis but not in a production test environment or for in–field diagnosis. All applications but in–depth failure analysis in a lab require other diagnosis approaches described later.

The major benefit of bypass approaches is that they work with any test compaction structure and logic diagnosis algorithm. One typical use–case of bypassing is the application to *logic built–in self–test* (LBIST) structures. During normal test, a LBIST like STUMPS [BardeM1982] can operate autonomously without the need for an external tester [AbramBF1990]. The *multiple–input shift register* (MISR) in a STUMPS structure is used to calculate a single signature of a complete test of a logic block. The final MISR signature provides only very little diagnostic information which is typically insufficient for reaching the required diagnostic resolution.

Besides completely bypassing the MISR, more advanced methods for enabling diagnosis in a BIST environment include obtaining more information by separating the BIST session into multiple test intervals [Savir1998, WohlWPM2002, LiuC2003, CookHW2012] or running the BIST multiple times while generating each time a signature over a different subset of responses [WuA1999, RajskT1999, GhoshT2000a]. In these approaches, diagnosis is performed *indirectly* or *directly* as described below.

Indirect Diagnosis

In indirect diagnosis, the failing scan cells are calculated out of the compacted data and this reconstructed fail data is used for diagnosis. Indirect diagnosis requires sufficient information for reconstructing fail data. Many techniques have been proposed which are based on error correcting codes [MitraK2004, SalujK1983, PatelLR2003, Das2000], convolutional compactors

[RajskTW*2005, WangCHW2003] or special signature registers [LeiniGM2004, Touba2007, LiuC2003]. To identify a single failing scan cell out of n, at least $\log(n)$ bits must be transferred. Reconstructing failure information out of compacted data is also error–prone. If the number of failing scan cells exceeds the capabilities of the used code, wrong fail information is reconstructed and diagnosis may be mislead.

Direct Diagnosis

Logic diagnosis algorithms may work directly with compacted data by considering the compaction hardware as part of the design itself (see figure 1.5). The direct diagnosis approach does not use an error–prone reconstruction of fail data and works also with compaction circuits with much a higher compaction ratios.

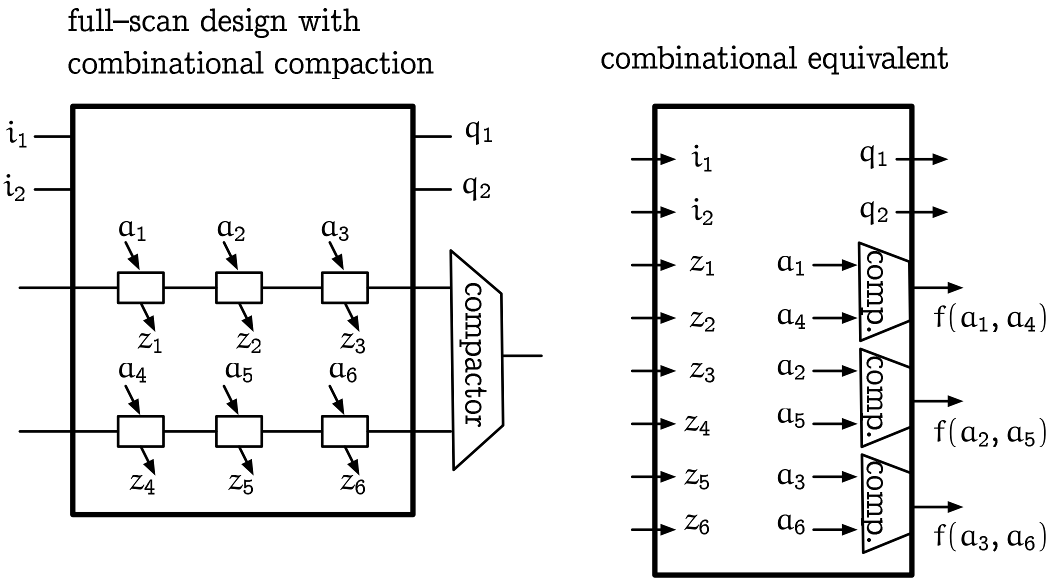


Figure 1.5.: Combinational equivalent for direct diagnosis on a compacted response.

The combinational equivalent grows linearly with the number of test responses compacted into a single signature. Because of this, direct diagnosis is usually only effective with space compactors (one test response per signature) or compactors, which compute each signature out of very few test responses [Elm2011]. Direct diagnosis applied to time compactors is often less effective than indirect diagnosis or bypassing approaches. The high fan–in of compactors pose a challenge to some classes of diagnosis algorithms. Especially back–tracing–based and intersection–based approaches, which will be explained in chapter 3, may loose effectiveness if confronted with compactors and high compaction ratios.

1.3. Volume Diagnosis for Yield Improvement

Some chips will fail the production tests for reasons explained before. The goal of yield improvement, *yield ramp*, or *yield learning* is the optimization of the manufacturing process or the design to achieve higher yield. A higher yield both reduces the production cost per functional chip and improves the product quality of the chips delivered to the customers [WilliB1981].

Volume diagnosis combines diagnosis results from many failing chips in a production line in order to find design and production problems. The results of logic diagnosis of the individual chips are correlated to layout data to identify features or regions on the wafer which are not well manufactured and fail in many chips [KeimTTS*2006]. Statistical learning approaches are applied on aggregated diagnosis data to find features failing more often than usual [BastaWA2008, TamPB2010]. Based on this information, layout and process parameters are optimized [HoraSEL2002].

While there is a huge amount of data to be analyzed for each production line, the fail data for a single device is rather limited. The limitation in response data has multiple reasons. The most obvious one is the test response compaction used to reduce the bandwidth needed between tester and device, and diagnosis has to be performed on compacted response data. Moreover, as tester memory is limited and the test time used on failing devices is usually kept as short as possible, only the first erroneous signatures are recorded. Another limitation stems from the used production test patterns. These pattern sets are kept small to reduce test time. Hence, each pattern excites many faults at the same time making it hard for diagnosis algorithms to distinguish between possible candidates [ChenRPR2006].

Despite all these limitations of the input data for diagnosis, the algorithms must provide good predictions with a limited amount of computing power. With thousands of fail data sets coming in each minute during production, any increase in the analysis time per data set has a great impact on the overall computing power requirement.

1.4. Precision Diagnosis for Failure Analysis

Failure analysis is the investigation of the root cause of a malfunction in a single chip. To understand the underlying defect mechanisms within a failing chip, *physical failure analysis* (PFA) with its complex and time intensive de–processing and imaging techniques is often unavoidable. As finding defects directly through physical inspection of a chip with Billions of transistors is way too expensive, logic diagnosis algorithms are used to identify the structures affected by the defect [Wagne2008].

Precision diagnosis is performed in a lab environment on a small selected set of chips like prototypes or representatives for systematic defects determined by volume diagnosis to guide PFA for the individual chips. Compared to volume diagnosis, the constraints on computing time are reduced but high diagnostic resolution has to be provided to guide the physical inspection accurately. The performance of a diagnosis algorithm in this application is measured by the area that has to be examined physically until the defect is found. This area is reduced by an efficient ranking of possible defect locations and by investigating the functional behavior of the defect itself.

The available time permits the bypassing of any on–chip test compression logic and the application of large diagnostic pattern sets. Logic diagnosis, diagnostic pattern generation and pattern application can even be coupled to diagnose a defect adaptively [GongC1995]. Based on the suspicious region determined by diagnosing responses to a standard pattern set, special diagnostic patterns are generated and applied to improve diagnostic resolution.

1.5. In–Field Diagnosis of Reliability Problems

Also the fault free chips delivered to customers may start to malfunction at a later time during operation due to aging effects or an unforeseen impact of the physical environment. One way to investigate such problems is to collect diagnostic data of structural tests in the field under the environment, where the malfunction is active [CookEWA2011]. For this purpose, the design–for–test infrastructure built in for production test is adapted and extended for in–field use. Diagnosis capabilities built into the chip and in–depth analysis of customer returns help to pinpoint the origin of the malfunctions. With this information, the design can be changed to improve the reliability of future products.

As the test results must be stored somewhere in the system itself, the memory constraints are even more severe than during production test. Thus, the test responses have to be extremely compacted prior to storage, and the biggest challenge for a diagnosis algorithm is to still extract as much information as possible from just a few bits of fail information.

1.6. Overview and Contributions

This chapter has highlighted the two principal challenges logic diagnosis algorithms have to face. The first challenge is the reduction of a–priori fault assumptions to be able to diagnose a wide range of different defect types stemming from all possible sources. The second challenge is test data economy. Constraints on response data storage and test time force diagnosis algorithms to use the available data most effectively. Diagnostic fail data for a single chip may originate from multiple different sources like production test, lab test, or in–field self–test. The effective use of all these sources calls for versatile diagnosis algorithms, which are able to use and combine fail data of various sources and properties to compute the best possible diagnosis result. If the constraints are reduced, diagnosis algorithms should make effective use of the provided flexibilities and provide the best possible diagnosis result despite the lack of specific fault models.

The remainder of this thesis addresses these challenges and is organized as follows:

Chapter 2 *Formal Foundation* develops the formal basis and notations used in this thesis. It includes a new notation for generalized fault modeling, the *Conditional Line Flip* (CLF) calculus, defines the requirements in circuit modeling and the logic diagnosis problem itself.

Chapter 3 *State–of–the–Art* presents the state of the art in diagnostic fault modeling and logic diagnosis. The CLF calculus is used here as a tool to analyze the assumptions made in common fault models.

Chapter 4 *POINTER* presents a new location–based logic diagnosis algorithm called *Partially Overlapping Impact couNTER*. Compared to previous approaches, this algorithm is capable of extracting more diagnostic information out of the available fail data using a sophisticated, yet efficiently computable ranking technique. By using the CLF calculus, it is able to point directly to the faulty structures within the circuit without relying on any particular fault model.

The remaining chapters describe extensions to POINTER in order to apply the algorithm in the main application fields of logic diagnosis.

Chapter 5 *Adaptive Precision Diagnosis* discusses POINTER in a prototyping domain, in which fully interactive communication with the device is possible. The goal is to pinpoint defects as precisely as possible by adaptively generating additional diagnostic tests on demand. By focusing diagnostic test generation in this way, diagnosis time is reduced as well as the dependency on specific fault models.

Chapter 6 *Production Diagnosis* focuses on a production environment, where the tests are predetermined and only the first few failing responses are known. Fast and predictable pattern analysis times are very important in such a setting. POINTER is extended in a way, that analysis time increases linearly with the number of failing tests to analyze, but considers still all passing tests in between to obtain maximum diagnostic resolution.

Chapter 7 *Diagnosis and Extreme Space Compaction* takes the response compaction to an extreme so that it is possible to store test results directly on chip for later analysis. It is shown how POINTER can handle even extremely compacted fail data without compromising diagnostic resolution.

Chapter 8 *Experimental Evaluation* discusses a series of experiments and observations that shows the efficiency and efficacy of the new diagnosis method. The experiments are conducted on known benchmark circuits as well as on industrial designs.

Chapter 9 *Conclusions* summarizes the contributions of this work and discusses possible research directions this work may enable in the future.

2. Formal Foundation

This chapter develops the formal basis for the investigation of fault model independent diagnosis algorithms. First, a generalized fault modeling is developed based on some fundamental considerations on defect mechanisms and defects. Then, the used circuit model is introduced and the logic diagnosis problem is defined. This chapter also establishes the notations for circuits, faults and test data used in the remainder of this work.

2.1. Defect Mechanisms, Defects and Faults

A *defect mechanism* is the physical process that increases the probability of defects. Understanding these mechanisms qualitatively and quantitatively is the ultimate goal of any failure analysis. Relating defects to their underlying mechanisms is the task of humans interpreting diagnosis results, performing measurements and physical failure analysis. The most common defect mechanisms and typical defects caused by these physical processes have been covered in chapter 1.

A *defect* is an unwanted structure or electrical property in the silicon chip. In this thesis, a defect is always assumed to have a specific location within the chip, called *defect site*. Sometimes, the term *spot defects* is used in the literature to contrast the defects handled in this thesis from non–local phenomena leading to failing chips. A defect disturbs the voltages and currents at the defect site.

In the diagnosis literature, faults are often closely related to defects. Therefore, a *fault* is defined here as a representation of a defect with respect to a given circuit model. The circuit model used in this thesis will be defined later in this chapter. If a circuit C was altered according to a fault f, the fault is said to be *injected* into the circuit and is noted as C_f. Sets of faults are typically denoted as F. A circuit with an *unknown defect* is noted with the symbol C_{ud}, which can also be read as *circuit under diagnosis*.

As shown earlier, diagnosis algorithms should be designed with less restrictive assumptions on possible faults to be able to point to the defective structures regardless of the exact nature of the defects. To describe these defects, a generalized fault modeling calculus is needed, which describe the structural and functional aspects of the defect candidates formally.

2.2. Defect Behavior and Failure Observation

Any detectable defect in a C_{ud} disturbs the electrical behavior of the circuitry close to the defect site. These disturbances include degenerated voltage levels of signal lines, increased current drain at gate inputs, increased power consumption, or faulty timing behavior. The nature of the alterations by a defect may exceed the capabilities of the circuit model. For instance, it may introduce indeterministic behavior in an otherwise deterministic circuit model.

Partitioning the defective circuit into two parts helps to abstract from the electrical specifics. One part is the *defect site*. It contains all the gates and signals whose behavior cannot be explained or modeled with the circuit model used. For instance, if the circuit is modeled on the logic level, then all signals with degraded electrical characteristics, all influenced, and all inoperable gates are contained in this defect site. The other part is the rest of the logic circuit, which implements the specified logic functionality and operates normally. The signals that connect these two parts are considered to be ordinary logic signals.

Figure 2.1 shows the defect site and its surrounding circuit. The *input cone* is controlled by the inputs of the circuit and provides the defect site with values according to the circuit model. The defect site may be influenced by the values from the input cone and provides possibly faulty values to the *output cone*. The output cone again performs the specified functionality according to the circuit model.

During test, patterns are applied at the inputs of the circuit and the values at the outputs are observed. As a consequence, logic diagnosis cannot observe the electrical conditions at the defect sites directly, but has to rely on the signals leaving the defect sites towards the outputs. From the perspective of logic

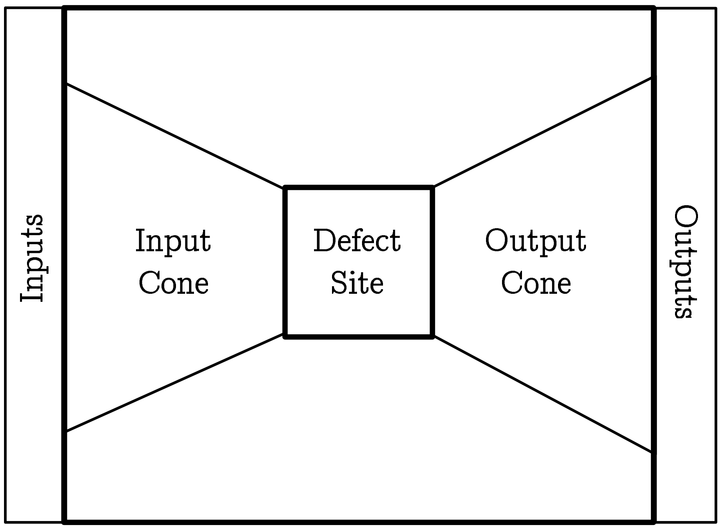


Figure 2.1.: Defect site and surrounding logic

diagnosis, certain internal signals have values, which are inconsistent with the fault–free operation of the circuit according to the circuit model. By finding for each test a small set of signal lines, which have to be altered in order to explain the erroneous outputs, the defect site itself can be located. This set of altered lines form the *structural aspect* of the diagnosis result.

With each new pattern, the electrical conditions at the defect sites change. The output cone receives different inputs and may provide different responses. Even the same pattern can lead to different results each time, if the defect site behaves in a nondeterministic way or depend also on other parameters like temperature or supply voltage level and the alterations are not always the same. The alterations at the defect site are correlated to the electrical conditions within the defect site. They form the *functional aspect* of the diagnosis result.

2.3. Circuit Models

As circuit models, *combinational circuits* are used. The structure of a combinational circuit C is modeled here as a directed, acyclic graph. An example is shown in figure 2.2. The vertices without any incoming edges are called sources or *inputs* to the circuit C. The vertices without any outgoing edges are called sinks or *outputs* of the circuit C. An output vertex may have at most one incoming edge. The vertices with incoming as well as outgoing edges are called *gates* of the circuit C. The edges are commonly called *signals* and their direction denotes the flow of information from a sending vertex called *driver* to a receiving vertex called *receiver*. The receivers of a specific driver are also called its *successors*, and the drivers of a specific receiver are also called its *predecessors*.

A single driver passes the same information to all its receivers. A *line* or *signal line* is either one of the inputs of a vertex, or the output of a vertex. Analogous to a single metal line connected to the output of a CMOS cell, at most one output line is defined per vertex. If a CMOS cell has multiple outputs and different information has to be passed to different receivers, the driver can be split into multiple vertices without loss of generality.

If a driver has more than one direct successor, its fanout contains multiple receivers. In a layout of a chip, a fanout starts typically in a single line of metal and then branches possibly multiple times forming a tree structure to distribute the logic signal to all individual receivers. The tree structure can be represented in the circuit model by a tree of *branch–points*. A branch–point is a vertex with two or more direct successors and models a single branch in the fanout–tree. If necessary, buffer–gates (one–input vertices, which simply pass on the information they receive) can be added to the circuit to include additional branch–points without changing the behavior of the circuit (see figure 2.3). The lines at the inputs of vertices (*input lines*) identify specific fanout branches of a preceding branch–point, the lines at the outputs of branch–points (*output lines*) identify fanout stems.

Each gate implements a function over values of the direct predecessors of the gate. The value of this function is passed to all direct successors of the gate. The input vertices of the circuit do not implement any function. Their values are defined by an *input vector* $\vec{t} = (t_1, t_2, \ldots, t_n)$ in a freely chosen but fixed order and with n being the number of input vertices in the circuit. The values of the output vertices are written as an *output vector* $\vec{r} = (r_1, r_2, \ldots, r_m)$ in a freely chosen but fixed order and with m being the number of output vertices in the circuit. The inputs applied to a combinational circuit and the output values generated by the circuit are called *test data*. An input vector is also called *test vector* or *test pattern*, an output vector is also called a *response vector* or simply *response*. A set of test vectors is typically noted as T.

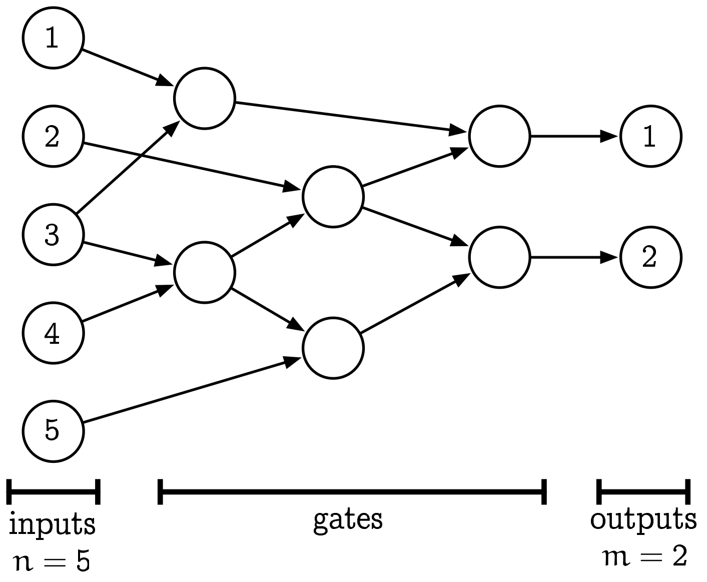


Figure 2.2.: A graph representation of the combinational circuit c17.

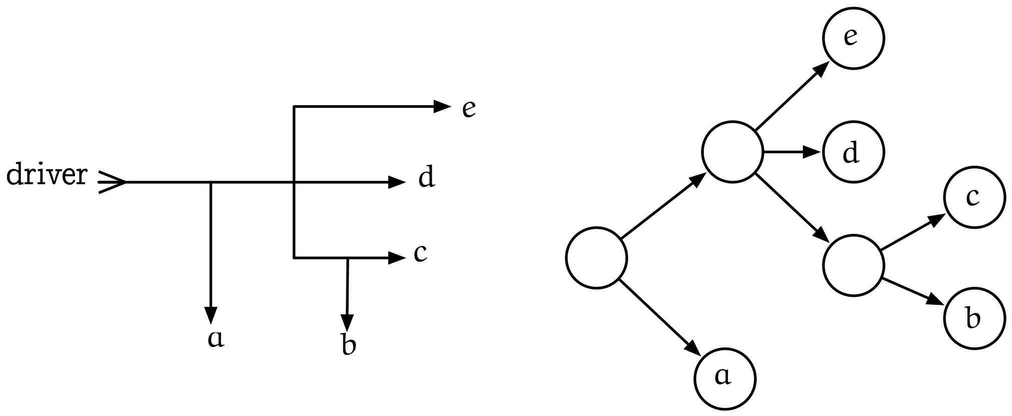


Figure 2.3.: Modeling a complex fanout tree.

As each individual gate implements a function and the circuit structure is defined to be free of any cycles, the output vector of the complete circuit depends only on the applied input vector. This observation is easy to verify as in any acyclic graph there exist an ordering of vertices such that each vertex is either an input or has all its predecessors at an earlier position in this ordering. Therefore, the functions of all gates can be rewritten as functions over the input vector itself by repeated substitution of the functions from the respective predecessors. Especially all elements of the output vector can be expressed as functions over the input vector and leads to the so–called *circuit function* $C(\vec{t}) = \vec{r}$.

The remaining aspect to define is the nature of the values passed through the circuit. Depending on the accuracy desired, they may range from simple Boolean values over temporal information even to electrical characteristics.

With the values being Boolean values $\mathbb{B} = \{0, 1\}$ and all functions of the vertices being Boolean functions, a circuit model for two–valued logic simulation is obtained. In this case, the circuit function $C(\vec{t}) = \vec{r}$ is a Boolean function as well and $\vec{t}$, $\vec{r}$ are vectors of Boolean values. This simple model for the circuit surrounding the potential defect sites is already quite effective for locating a wide range of permanent defects and it will be used in this thesis to validate the new diagnosis approaches.

For locating defects, that influence the timing of internal signals, the same methods can be applied to a circuit model, which also include timing information. As long as the used circuit model is able to properly reflect the behavior of the circuit around potential defect sites, the sites themselves can be located. A discussion on different timing models can be found in [Sapat1994]. The only requirement for applying the conditional line flip calculus defined below to time–aware circuit models is that, for each signal line, a Boolean value is defined at any point in time.

2.4. A Definition of Logic Diagnosis

Given a combinational circuit C and a list of input stimuli $T = (\vec{t}_1, \ldots, \vec{t}_n)$. In the fault–free circuit, the response is completely determined by the current input: $\vec{r}_i = C(\vec{t}_i)$ for $1 \leqslant i \leqslant n$. By applying all n input stimuli, a list of n

responses is obtained $R = (\vec{r}_1, \ldots, \vec{r}_n)$. The combination of the two lists T and R specifies the fault–free functional behavior of the circuit C.

Given a circuit C_{ud} which is similar to C but contains unknown defects. The responses $R_{ud} = C_{ud}(T)$ obtained from this circuit with the same input stimuli are different from R. The problem of *logic diagnosis* is to locate and describe the differences between the circuit structures C and C_{ud} based on the observed functional behavior T, R, R_{ud}. In addition to the functional behavior, one of the circuit structures (usually C) is also known.

The description of the differences between C and C_{ud} usually contains the set of lines and gates involved in the defect site (structural or spatial information) and information on the behavior of those lines (functional information).

2.5. Conditional Line Flips

A *conditional line flip* (CLF) describes a faulty behavior at a single line in the circuit. A CLF at line l describes the fact that the receiving gates sometimes read a faulty value from l. In Boolean logic, a faulty value is always the opposite of the correct one, so this situation can be viewed as a *line flip*, which is sometimes active.

The lines, which are influenced directly by a CLF, are called *victims* or *victim lines*. A CLF is noted by the name of the victim line and an XOR–symbol followed by a condition clause:

$$line \oplus [condition]$$

The condition describes the activity of the CLF. If the condition is satisfied, the affected line is flipped to the opposite value. If the condition is not satisfied, the line is fault free.

The CLF calculus does not pose any restrictions on the structure of the conditions. The conditions may be arbitrary functions over signal values, time, and may include also external parameters supply voltage levels or temperature. Even nondeterministic behavior can be captured by using random variables in the condition. If signal lines are specified in the condition, the general convention is that their fault–free values according to the circuit model are used in the evaluation.

Each line in the circuit can bear a single CLF. Multiple CLFs at the same line can be merged into a single one without loss of generality by combining their conditions appropriately. Figure 2.4 shows all possible locations of CLFs in a combinational circuit. A CLF can be located at all input lines of gates and output ports like f_2, f_3, f_4 or at an output line of branch–points like f_1. Nodes with a single fanout like d do not have a CLF at the output line, because this CLF would be logically equivalent to f_4 with the same condition. A CLF at the output line of a branch–point affects the logic values for all successors of the node. For example, f_1 at gate b will flip the inputs of both successors c and d, whereas $f_2(f_3)$ only influences $c(d)$.

A *static fault* can be described using a set of CLFs in which every CLF condition is expressed as a Boolean formula over the signals of the circuit. It has the form $f = \{v^1 \oplus [f^1], v^2 \oplus [f^2], \ldots\}$ with $v^1, v^2, \ldots$ being the victim signals and $f^1, f^2, \ldots$ being Boolean functions. Again, the condition is evaluated using the fault free value of each signal. The activity of static faults are completely determined by the current fault free signal values in the static state of the circuit. Neither the time domain nor indeterministic behavior is taken into account.

The behavior of *dynamic faults* not only depend on the steady state of a circuit, but also on time. Such behavior can be modeled with CLFs to a large extend by allowing the Boolean functions in the conditions to also depend on previous signal values. This work uses subscripts to the signal names to specify a time difference to the current point in time. Previous clock cycles are specified by using integer values: v_{-1} is the value of v in the previous clock cycle, v_{-2} the value before that. If time differences smaller than one clock cycle need to be specified, real values between 0 and -1 are used. For instance, $v_{-0.5}$ evaluates to the value of v half a clock cycle in the past.

More modeling examples and the relation to known fault models can be found in the next chapter.

2.5.1. Defect Site

The *defect site* can now be formally defined based on CLFs. Given a set D of CLFs that describe together a defective behavior. A defect site is the set of all signal lines that are either victims in D or are used in the conditions of the CLFs in D. The size of a defect site is the cardinality of its set of signal lines.

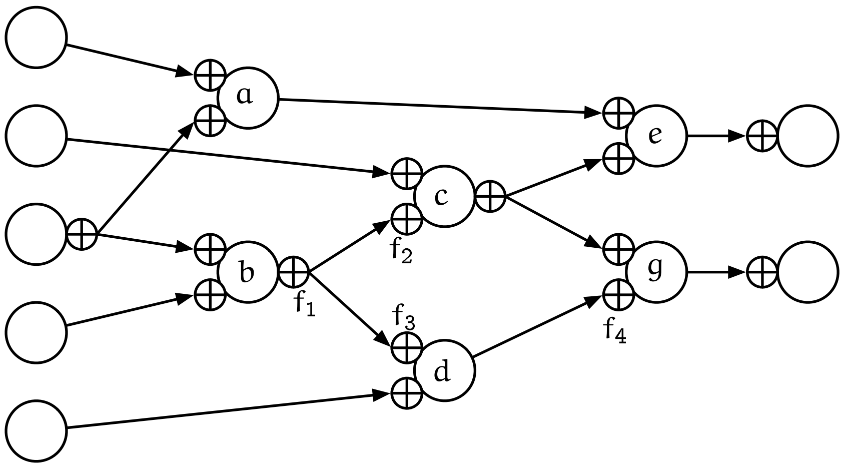


Figure 2.4.: Combinational circuit with conditional line flips.

3. State of the Art

3.1. Diagnostic Fault Modeling

Traditionally, fault models have been introduced to reduce the complexity of pattern generation and pattern analysis algorithms. This reduction has mainly been achieved due to both the limited amount of faults which have to be considered and their simple behavior. These fault models are still used in many diagnosis systems, so they are discussed in the section *Basic Fault Models*.

With increasingly complex defect mechanisms in modern process technologies, two alternatives in fault modeling can be observed in research. On the one hand, more and more specialized fault models have been developed to capture specific defective behaviors in the circuit [Aitke1995]. These efforts are discussed in *Defect–Oriented Fault Models*. On the other hand, there is a trend towards more and more generalized fault models to encompass many different defective behaviors. These are discussed here in the section *Generalized Fault Models*.

3.1.1. Basic Fault Models

Stuck–at Faults

A stuck–at fault [Eldre1959] ties a victim line to a fixed logic value. This model is used today in a broad array of applications from test generation over test coverage analysis to logic diagnosis.

The activity of a stuck–at fault depends on the original value of the victim signal. A stuck–at 0 at signal v is active only, when v carries a logic 1, and a stuck–at

1 on the signal is only active, when v carries a 0. Thus, the canonical CLF representations of stuck–at faults are:

$$v \oplus [v] : \text{stuck–at } 0$$

$$v \oplus [\overline{v}] : \text{stuck–at } 1$$

This notation reflects the basic properties of stuck–at faults. A fault only affects one line and can be expressed with a single CLF. The fault is deterministic and static as the conditions are Boolean functions. The Boolean conditions of the two CLFs depend on and only on the victim line v itself. Any more complex Boolean functions with this property can be simplified into one of these two functions using the known Boolean identities.

Although stuck–at faults do not model defective behavior of circuits accurately, it was shown that stuck–at fault pattern sets also test for many other defects [McCluT2000]. Many diagnosis algorithms follow the idea in [WaicuL1989], where stuck–at faults are not assumed to explain all, but only some failing responses in order to diagnose more complex defects. This generalization of the stuck–at fault model is discussed in detail in the section 3.1.3.

Gross Delay Faults

A *gross delay fault* [KrstiC1998, BushnA2000] assumes that a signal transition gets delayed at a gate by such a large amount that it won't reach any output before the observation time. If a gross delay fault is active for a particular test, it produces at that time the same response as a stuck–at fault at the same signal. In the literature, transition faults are often used synonymously with gross delay faults [BushnA2000]. Some works link transition faults more closely to transistor open defects and define additional conditions for fault activation [Wunde1991]. These conditions are further detailed in section 3.1.2.

The activity of time–related faults like gross delay faults depends not only on the current signal values but also on previous values of signals. In CLFs, these past values of a line are noted with a subscript giving the difference to the current time. For gross delay faults, it is sufficient to use discrete time values to identify the appropriate clock cycles.

Just like the derivation of the basic stuck–at fault in the previous section, CLF can be used now to derive all possible gross delay faults on a signal, given that there are no external aggressors. Two conditions must hold for a gross delay fault to be active. The first condition is that the current signal value must be different from the previous one. With v being a signal, this first condition is $v \oplus v_{-1}$. The second condition may depend on the current value of signal v itself, since no external aggressors are allowed and v_{-1} is determined by v and the first condition. All three possible gross delay faults are therefore:

$$v \oplus [(v \oplus v_{-1}) \cdot f(v)] \text{ with } f(v) \in \{v, \overline{v}, 1\}$$

This modeling accounts for slow–to–rise and slow–to–fall faults as well as a gross delay fault affecting both rising and falling transitions.

Small Delay Faults

If an additional delay at a gate or a signal is too small to fail every possible propagation path, then the syndrome cannot be accurately explained by the gross delay fault model. To distinct such faults from the gross delay model, they are usually called *small delay faults* [ParkMW1988, PramaR1988, CarteIR1987, TehraPC2011]. Small delay faults are of growing concern especially in high–performance circuits with narrow timing margins and variations.

One simple way to express timing behavior within a single clock cycle is to extend the integer subscripts used for denoting time frames towards real numbers. The following CLF delays a signal line v by 10% of the duration of a single clock cycle (T):

$$v \oplus [v \oplus v_{-0.1}]$$

Whenever the current (fault free) value of v is different from the fault free value of v $0.1T$ in the past, the signal line is flipped. Therefore, the faulty value of v equals at any time the *past* fault free value ($v_{-0.1}$).

While stuck–at faults and gross delay faults can be correctly evaluated using only logic simulation, time simulation is necessary to correctly evaluate CLFs for small delay faults. This involves also the selection of a timing model to sufficiently reflect the fault–free circuit behavior. An extensive body of research is available on this topic [Sapat1994, BushnA2000].

3.1.2. Defect–Oriented Fault Models

Bridging Fault Models

Bridges are described by various fault models with different properties in the literature [Wunde2010, Engel2009]. This subsection presents the most common fault models with the help of the CLF calculus focusing on the direct consequences of a short between signal lines for driving and receiving gates. Bridges are also possible between internal features of gates, or they may lead to indirect consequences like combinational loops with additional states or oscillating behavior [Engel2009], which are not discussed here.

In CMOS technology, shorts between signal lines may show quite complex behavior [RenovHB1994], because both signals may be driven with roughly the same strength and the defect causes intermediate voltages on the victim lines. Let v be a victim line involved in a bridge and showing a degenerated voltage level in some situations. If the fault free value of v is $1(0)$ and the degenerated voltage is below(above) the threshold voltage u_{th}, the line is faulty. A possible CLF formulation is shown in figure 3.1a. The function $u(\dots)$ calculates a voltage for the victim line. This voltage can depend on the bridging resistance, the logic values and relative strengths of the bridged signals. The values and strengths in turn depend on the logic inputs of the driving gates. Determining the precise voltage $u(\dots)$ every time through SPICE simulation at the electrical level is usually not practical. Therefore SPICE simulation is used to characterize the cells in a technology once and the generated data is fed into advanced bridging models like the *Voting Model* [AckenM1991], the *Biased Voting Model* [MaxweA1993] or the *Direct Voting Model* [RenovHB*1994a] in order to determine the relative signal strengths and the behavior of the bridge.

Multiple receiving gates may come to different interpretations of the same degenerated input voltage due to variations in the threshold voltages of the transistors in these gates. While one receiver considers a certain voltage on a victim line being logic 1, another receiver of the same victim line reads logic 0 [WilliA1973]. This is known as the *Byzantine effect* or Byzantine Generals Problem [AckenM1992]. In this case, individual CLFs at each receiver are necessary in order to evaluate the voltage on the victim line against the distinct threshold voltages $u_{\text{th}i}$, $i \in \{1, \dots, n\}$ (figure 3.1b). The requirement for a larger

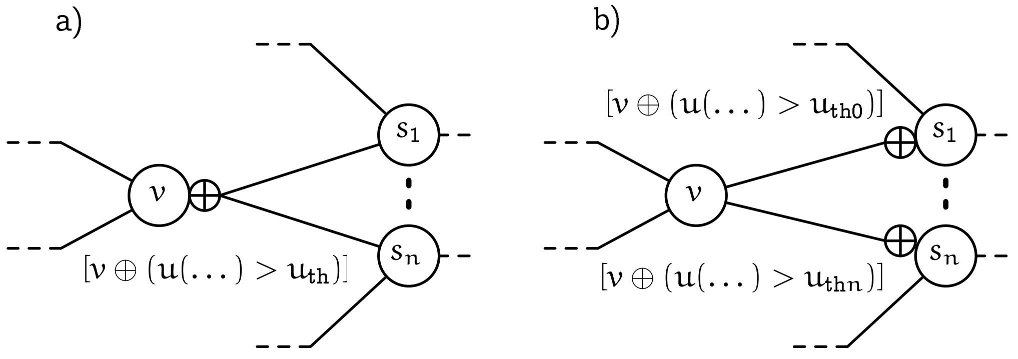

a)
b)
v
s_1
s_n
$[v \oplus (u(\ldots) > u_{th})]$
$[v \oplus (u(\ldots) > u_{th0})]$
$[v \oplus (u(\ldots) > u_{thn})]$

Figure 3.1.: CLF modelling for lines with degenerated voltage. a) all receivers agree on the voltage interpretation, b) receivers disagree due to distinct threshold voltages (Byzantine effect).

number of CLFs (the increase of the defect site) is a consequence of the higher complexity in the behavior of such bridges.

The bridging resistance is often high enough to influence the behavior of the bridge and models were developed to describe such bridges [RenovHB1994, Engel2009]. The bridging resistances of defects are not known in advance and their estimation may be part of diagnosis results [GhoshT2000, KhursRA*2008]. The two main effects to consider in addition to non–resistive bridging fault models are different voltages on the two bridged signal lines and possible time–dependent behavior. The first effect is easily incorporated into CLF formulations presented so far by considering the bridge resistance in the calculation of $u(\ldots)$. CLF formulations for time–dependent, dynamic behavior of bridges [FavalDO*1993] are obtained by combining bridge conditions with conditions for gross delay faults or small delay faults.

Many bridge fault models abstract from electrical specifics to obtain faults, which are easy to evaluate. Instead of taking voltages and signal strengths explicitly into account, the behavior of the such bridges is just determined by the fault–free logic values of the involved signals. Although, these models may not reflect the behavior of CMOS bridges exactly, they are widely used in the literature [AbramBF1990] and shall get special attention here. Such bridges can be regarded as a special case of the more general CLF formulations above. The following discussion derives a general CLF–formulation of all possible static bridge models involving two signal lines. Similar considerations are possible for bridges involving more than two lines as well.

Fault activation is determined only by considering the logic values of the two bridged signals. Depending on the type of bridge and the current values of the signal lines, one or both signals may change their logic value. Such faulty behavior is described by two CLFs at most with Boolean conditions depending on the current values of the involved signals. A necessary precondition for a static bridge to be active is that the two involved signal lines must carry different logic values. If this precondition is true, the behaviors of the two signals v and w are determined by two Boolean functions f_v and f_w. The function f_v depends only on signal w, because the value of signal v is already determined by the precondition. Similarly, function f_w depends only on signal v. This leads to the following generalized CLF formulation of an arbitrary static bridge between two signal lines v and w:

$$v \oplus [f_v(w) \cdot (v \oplus w)],$$
$$w \oplus [f_w(v) \cdot (v \oplus w)].$$

There are exactly four basic expressions for f_v and f_w, respectively. An expression may be constant 0, constant 1 or may use the positive or the inverted value of the other signal in the bridge:

$$f_v(w) \in \{0, 1, \overline{w}, w\},$$
$$f_w(v) \in \{0, 1, \overline{v}, v\}.$$

Any more complex Boolean formula can be simplified by using the precondition and Boolean identities. The formulas given above therefore model every possible static bridge configuration. There are $4^2 = 16$ possible configurations that are derived by choosing one of the four possible expressions for f_v and f_w. From these 16 configurations, there are six that are actually derived from other bridges by interchanging the roles of the signals v and w. This leads to ten unique bridge types including the fault free case (Table 3.1).

Many of the bridge types in this table were first proposed by considering the electrical behavior of shorted signal lines. If v *dominates* w, the value of a strong signal always prevails over a weaker signal [WilliA1973]. More recently, the *dominant–AND* and *dominant–OR* models were proposed [EmmerSB2000] to more accurately capture some resistive bridges in CMOS. In the *wired–AND* (*wired–OR*) model, the strong value 0 (1) always overwrites the weaker value

$f_v(w)$	$f_w(v)$	Bridge Type
0	0	Fault free
0	1	v dominates w [WilliA1973]
0	$\overline{v}$	v AND–dominates w [EmmerSB2000]
0	v	v OR–dominates w [EmmerSB2000]
1	1	v and w swap values
1	$\overline{v}$	w dominates v & v AND–dominates w
1	v	w dominates v & v OR–dominates w
$\overline{w}$	$\overline{v}$	wired–AND [Roth1966]
$\overline{w}$	v	w AND–dominates v & v OR–dominates w
w	v	wired–OR [Roth1966]

Table 3.1.: The ten possible static bridge types

1 (0), which was common in resistor–transistor logic, diode–transistor logic or emitter–coupled logic [Mei1974, Roth1966]. Physical defect mechanisms for the remaining four more exotic bridge behaviors were not reported in the literature.

Open Fault Models

Stuck–open defects on transistors [Case1976, Wadsa1978] or interconnect open defects [XueDJ1994, ArumiRF2008] cause signal lines to not being actively driven to a clear logic value in some situations.

In case of a *full open defect*, the victim line is said to be *floating*. The value of a floating signal may depend on the value it was previously driven to. In case of a *resistive open defect*, the victim line is still connected to a driver, but showing an additional resistance. This resistance lead to an additional delay caused by the defective signal. If the delay is large enough, it can be described using the gross delay fault model, otherwise, it may behave similar to a small delay fault.

Transistor stuck–open defects are typically modelled by variants of transition faults [ReddyRA1984, JainA1985a, SodenTT*1989]. In contrast to gross delay faults, transition faults modelling stuck–open defects require additional robustness conditions for proper fault activation as shown by the following simple example.

If the p–transistor of the input x in a CMOS NAND gate is stuck–open, the output $v = x$ NAND y will be disconnected and hold its charge for the input pattern $(x, y) = (0, 1)$. The expected fault–free value for this input pattern is $v = 1$, so the gate output must have been discharged ($v_{-1} = 0$) before applying this pattern for the faulty gate to show an erroneous $v = 0$. The only pattern sequence that satisfies these conditions is $(1, 1); (0, 1)$. These patterns have to be applied in direct succession, which is not trivial if the inputs of the NAND gate are computed by some other circuitry. Even if two circuit input patterns generate the appropriate logic values at the inputs of the gate, the transition from the first to the second pattern might generate a hazard at the gate input y. If y is 0 even for a short time, the second p–transistor may charge the gate output enough to be interpreted as logic 1 and the open defect is not observed with the second pattern. The gate input y must therefore stay 1 at all times during the application of the test sequence. Hazards at x do not invalidate the test. A CLF that is only active under the proper activation condition of the transistor open defect discussed above can be intuitively noted as:

$$v \oplus [\forall 0 \leqslant t \leqslant 1 \;\; x_{-1} \cdot \overline{x} \cdot y_{-t}].$$

More elaborate notations are possible by using temporal logic [Prior1957] together with appropriate definitions.

While many transistor stuck–open faults allow still some direct control over the victim line, interconnect open faults usually lead to permanently floating lines or additional delay [RodriAF*2008]. Efforts to model the behavior of such interconnect opens usually include considerations concerning the open resistance, capacitive coupling to neighboring signal lines, tapped charges and leakage [XueDJ1994, LiTM2001, SpinnJP*2007, RodriAF*2007, HillePE*2008]. Such considerations can enable diagnosis algorithms identify the exact via or line segment with an open defect on a signal line [VenkaD2000a, RodriAF*2007, RodriAF*2010].

Crosstalk Faults

With continued downscaling, signal lines are placed closer and closer together. This trend increases the probability that capacitive coupling between signal lines causes additional delay or unexpected glitches on signal lines [ChenGB1997].

A very basic model for crosstalk faults can be obtained on the basis of a gross delay fault as follows. In a crosstalk fault, transitions on an aggressor line w cause glitches on an influenced (victim) line v. A glitch is only produced if line w changes from the value equal to w towards the opposite value (see fig. 3.2). The CLF notation of such a fault may be noted as:

$$v \oplus [(w \oplus w_{-1}) \cdot (v \oplus w)]$$

The first part of the condition is true, if there is an event on line w, and the second part is true, if the final value of w is different from the current value of line v. Similar and more realistic CLF formulations can be found based on the small delay fault model.

More recent works consider multiple aggressors [ZachaCK*2003] on a single victim, for instance to generate tests with maximum impact on possibly weak signal lines [GanesK2010]. Some recently proposed diagnosis algorithms use such specialized models to identify crosstalk related problems [TakahPH*2001, MehtaMT*2006].

3.1.3. Generalized Fault Models

Generalized fault models seek to express a large class of possible defective behaviors. Logic diagnosis algorithms based on these fault models have reduced assumptions on the defective behavior of the C_{ud} and allow defect mechanisms to be target of the investigation rather than given by a defect–oriented fault model.

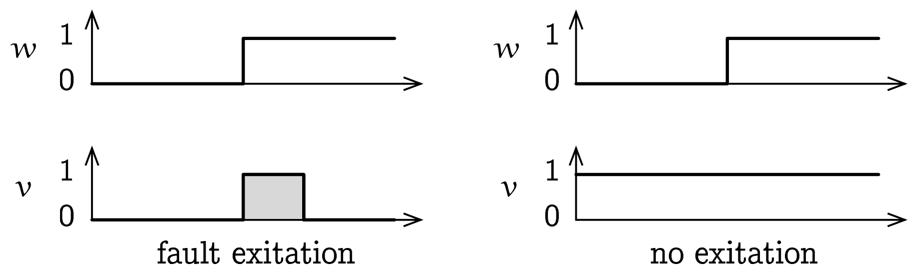


Figure 3.2.: An example of a crosstalk fault.

The most popular notations are *pattern faults*, the X–*fault model*, and *fault tuples*. All of them are described below by showing a way to transform the notations into CLF–formulations. This underlines that the CLF calculus is powerful enough to encompass the general fault models described in the literature and highlights at the same time the assumptions made by the various notations.

Pattern Faults

Pattern faults [Kelle1996] are developed to aid test generation for complex defects. A pattern fault specifies a set of requirements for fault activation and a set of signal line changes to describe the fault effect.

The fault activation requirements are noted as a set of logic values for internal signal lines. Let $l^1, \dots, l^n$ being signal lines which are required to be set to the values $x^1, \dots, x^n$ respectively, with $x^i \in \{0, 1\}$ for $1 \leqslant i \leqslant n$. If the signals need to be set in the current clock cycle, a *requirement block* is noted as:

$$\text{req-block ::= REQ \{ NET } l^1\ x^1 \text{ NET } l^2\ x^2 \text{ ... NET } l^n\ x^n \text{ \}}$$

This requirement block is satisfied if the following Boolean formula evaluates to true:

$$\mathrm{req}(l^1, \dots, l^n) = \bigwedge_{i=1}^{n} (l^i = x^i)$$

If the signals need to be set in the previous clock cycle, a *initialization block* is noted as:

$$\text{init-block ::= INIT \{ NET } l^1\ x^1 \text{ NET } l^2\ x^2 \text{ ... NET } l^m\ x^m \text{ \}}$$

which corresponds to the Boolean formula:

$$\mathrm{init}(l^1, \dots, l^m) = \bigwedge_{i=1}^{m} (l^i_{-1} = x^i)$$

The fault effect is noted as a set of signal changes in a *propagation block*. This block specifies that if a victim signal line v^i has a correct value c^i, it will change to an erroneous value $e^i \neq c^i$ ($c^i, e^i \in \{0, 1\}$ for all $1 \leqslant i \leqslant p$):

$$\text{prop-block ::= PROP \{ NET } v^1\ c^1/e^1 \text{ NET } v^2\ c^2/e^2 \text{ ... NET } v^p\ c^p/e^p \text{ \}}$$

This fault behavior corresponds to the following set of CLFs:

$$\{v^1 \oplus [v^1 \oplus e^1], \dots, v^p \oplus [v^p \oplus e^p]\}$$

Static pattern faults combine a requirement block and a propagation block:

STATIC { req-block prop-block }

and the CLF representation of this fault is

$$\{v^1 \oplus [(v^1 \oplus e^1) \wedge req(l^1, \dots, l^n)], \dots, v^p \oplus [(v^p \oplus e^p) \wedge req(l^1, \dots, l^n)]\}.$$

Dynamic pattern faults include in addition an initialization block to express the necessary transitions for activation of time–dependent faults:

DYNAMIC { init-block req-block prop-block }

and the CLF representation of this fault is

$$\{v^1 \oplus [(v^1 \oplus e^1) \wedge init(l^1, \dots, l^m) \wedge req(l^1, \dots, l^n)], \dots,$$

$$v^p \oplus [(v^p \oplus e^p) \wedge init(l^1, \dots, l^m) \wedge req(l^1, \dots, l^n)]\}.$$

Every pattern fault can be represented as a set of CLFs as shown above. However, the reverse is not true even if all the conditions of the CLFs contain only Boolean formulas. Pattern faults do not allow for disjunctions of requirements. For instance, a CLF like $v \oplus [x \vee y]$ cannot be noted as a single pattern fault because multiple different assignments to signals x and y activate the fault. In general, a set of CLFs can be formulated as a pattern fault if their conditions can be rewritten using Boolean identities to match the forms presented above.

A similar notation is used in the *generalized fault model* (GFM) [KunduZC*2005, KunduSG2006] which also targets test generation. The fault effect can be described as slow–to–rise or slow–to–fall signal with a certain delay. This way, a pattern generation algorithm can be used to sensitize a path of sufficient length from the fault site to an observation point to observe the fault effect.

X–Fault Model

The X–fault model was first used in [BoppaF1998] for describing defects. The refined model described in [WenTSK2003, WenKMYS*2006] assigns each victim line of a defect a distinct X symbol, an unknown value. If $l_1, \ldots, l_n$ are victim lines (possibly fanout branches of the same driver to capture the Byzantine effect), this assignment is equivalent to using unknown Boolean variables $X_1, \ldots, X_n$ in the CLFs:

$$\{v_1 \oplus [v_1 \oplus X_1], \ldots, v_n \oplus [v_n \oplus X_n]\}$$

This way, no assumptions are made on the actual logic behavior of the victim lines. This is an example of an incomplete model of the functional behavior of the defect. Appropriate diagnosis algorithms identify the functional behavior by providing assignments to the unknown values. Such faults cannot be handled by two–valued logic simulation, as the conditions cannot be evaluated right away. Instead, multi–valued simulation is used in order to propagate the unknown values through the circuit to the outputs, then, an assignment to the X–values can be found by implication [WenTSK2003, WenKMYS*2006]. More details are given in section 3.2.2.

Fault Tuples

Another very general fault modeling technique with a wide application field uses *fault tuples* [BlantDD2006]. Among the fault tuple types covered in the work of Blanton, the most relevant for diagnostic fault modeling are tuples that model an activation condition and a fault impact on a signal line. Given a signal line l, a Boolean value x and a time constraint T, then a *condition fault tuple*:

$$\{l, x, T\}^c$$

evaluates to true if there is a time t that satisfies time constraint T and, at that time, l has the value x:

$$\mathrm{cond}(l, x, T) = \exists t \text{ with } T(t) \wedge (l = x)$$

The time constraint may specify a fixed clock cycle N, or relations to an unknown but fixed clock cycle i: $\{N, i, i+N, > i, < i, \geqslant i\}$. *Error fault tuples* are used to describe the impact of a fault on a signal line.

$$\{l, x, T\}^e$$

drives victim signal line l to value x whenever the time constraint T holds. In CLF notation:

$$l \oplus [T \wedge (l \oplus x)]$$

A *product* is a conjunction of multiple fault tuples and is used to model complex defective behavior:

$$\{l_1, x_1, T_1\}^c \cdots \{l_j, x_j, T_j\}^c \cdot \{l_{j+1}, x_{j+1}, T_{j+1}\}^e \cdots \{l_n, x_n, T_n\}^e$$

which is equivalent to $n - j$ CLFs

$$\{l_{j+1} \oplus [T \wedge (l_{j+1} \oplus x_{j+1}) \wedge ccond], \ldots, l_n \oplus [T \wedge (l_n \oplus x_n) \wedge ccond]\}$$

with the common condition

$$ccond = \bigwedge_{i=1}^{j} cond(l_i, x_i, T_i).$$

3.2. Logic Diagnosis

In the literature, logic diagnosis algorithms are often classified into the paradigms *cause–effect* and *effect–cause* [AbramBF1990].

- *Effect–cause* analysis looks at the failing outputs and starts reasoning using the logic structure of the circuits [AbramB1980].
- *Cause–effect* analysis is based on a fault model. For each fault of the model, fault simulation is performed, and the behavior is matched with the outcome of the C_{ud} [RichmB1985].

These paradigms are explored in the next subsection. Then, the following basic concepts of logic diagnosis algorithms are discussed along with the diagnosis systems they are implemented in:

- *Inject–and–validate* simulates a large number of faulty circuits and matches their responses to the syndromes of the C_{ud} [WaicuL1989].
- *Back–tracing* starts from failing observation points and traces the sensitized paths towards the circuit inputs to find the defect sites [AbramB1980a, AbramMM1983].

- *SAT–based* approaches formulate the diagnosis problem as a Boolean satisfiability instance and a satisfying assignment gives a fault candidate [SmithVV2004].
- *Model allocation* techniques correlate internal signal values to observed fault activations to extract their behavior [MillmMA1990, DesinPB2006].

The former two concepts are *per–test* analysis approaches where each response is first analyzed individually and independently. All the results of the individual analyzes are then combined to obtain the final diagnosis result. The latter two concepts consider multiple or even all tests at the same time.

3.2.1. Effect–Cause and Cause–Effect

Cause–effect analysis [RichmB1985] starts with a set F of possible *causes* in the circuit. An element of this set $f \in F$ is called a fault and the set is generated according to a fault model. Diagnosis is performed by comparing for each test $\vec{t} \in T$ and each fault $f \in F$ the failures generated by the fault f with the observed response of the C_{ud}. The result of logic diagnosis is the set of faults, which show the same behavior as the C_{ud}:

$$\text{result} = \{f \in F \mid \forall \vec{t} \in T \;\; C_{ud}(\vec{t}) = C_f(\vec{t})\}$$

In order to avoid costly fault simulations during diagnosis, a *fault dictionary* is constructed. This dictionary is created once by simulating every fault $f_1, \ldots, f_n \in F$ with every pattern $\vec{t}_1, \ldots, \vec{t}_m \in T$ in the circuit C and storing the responses in a table. Two major disadvantages with this procedure were quickly identified. First, the size of the fault dictionary became a major issue and a lot of research effort was spend on reducing its size by various means [PomerR1992, ChessL1999]. As the size of the dictionary grows with the size of F, this approach is not very effective for more complex fault models. The second disadvantage is the a–priori selection of a fault model to generate F. Many defects cause a behavior of the C_{ud} that does not match with any of the faults in F. In this case, cause–effect diagnosis will fail.

To reduce the dependency on a single fault model as well as to handle fault models with very large number of faults, *effect–cause* diagnosis was proposed [BreueCS1976, AbramB1980, AbramB1980a]. Effect–cause based diagnosis algorithms derive candidate causes directly from the observed effects. The majority of modern diagnosis algorithms implement effect–cause methods and are discussed below based on their algorithmic principles.

There is also a trend of combining effect–cause and cause–effect ideas and diagnose a defect in two passes. First, a fast effect–cause analysis is performed to constrain the circuits region where possible culprits may be located. Second, for each of the possible fault sites and suspected fault model, a cause–effect approach is performed for identifying those faults which match the real observed behavior [DesinPB2006, AmyeeNV2006].

3.2.2. Inject–and–Validate

Inject–and–validate is a per–test analysis concept that generalized the cause–effect principle. Also here, a set of faults F is simulated and their responses are compared to the responses of the C_{ud}. But inject–and–validate procedures do not restrict themselves to exact matches between the simulated faults and the defective behavior of the C_{ud}. Instead, they determine fault candidates based on the similarity between the simulated and the observed responses.

The diagnosis approach in [WaicuL1989] was one of the first proposing this procedure to localize defects. This diagnosis method reports stuck–at faults that are able to explain most of the failing responses. Let F be the set of all stuck–at faults in the circuit C and T a test set. Let further $expl(T, f)$ be the subset of test patterns whose failing responses are explained by a fault $f \in F$:

$$expl(T, f) = \{\vec{t} \in T \mid C_f(\vec{t}) = C_{ud}(\vec{t}) \neq C(\vec{t})\}.$$

Then, this diagnosis approach returns:

$$\{\, f \in F \mid |expl(T, f)| \; = \max\{\, |expl(T, g)| \mid g \in F \,\}\,\}.$$

This basic idea was subsequently refined by a number of researchers by proposing more and more refined comparison metrics and candidate ranking heuristics.

One of the most prominent technique is the *single location at a time* (SLAT) method introduced in [BarteHH*2001, Huism2004]. A test pattern $\vec{t} \in T$ has the SLAT property if there is at least one observable stuck–at fault $f \in F$ which produces a response on that pattern identical with the response of the C_{ud}:

$\exists f \in F$ with $expl(\{\vec{t}\}, f) \neq \emptyset$. As diagnosis result, SLAT based algorithms report a set of *multiplets*. A multiplet M is a subset of F with the property that each SLAT pattern observed in the C_{ud} is explained by at least one fault in M. Let $T' \subseteq T$ the set of all SLAT patterns, then:

$$M = \text{ minimal set with } \bigcup_{f \in M} expl(T', f) = T'$$

By solving this set covering problem, single–line faults are combined to form candidates for multi–line faults such as bridging faults. The concept of constructing complex faults out of simpler components is also known as *composite signatures*. It was first proposed by [MillmMA1990] and can be found in various other diagnosis algorithms [VenkaD2000]. A rather similar concept to the SLAT property are *curable vectors* or *correctable vectors* [HuangCC*1997] used in design debug. A curable vector in this context is a failing response, which can be fixed by manipulating a single signal line within the circuit. This idea was applied to logic diagnosis of multiple defects at roughly the same time SLAT diagnosis was introduced [Huang2001].

The main drawback of the original SLAT method is the fact that information for fault location is only extracted from patterns with the SLAT property. All the other patterns are not taken into account, neither failing nor passing ones. To address this shortcoming, many works propose improved matching and scoring procedures to encompass non–SLAT patterns as well [LavoHL2002, WangMTR2006, Liu2007]. In [ZouCRT2006] not only single stuck–at faults are simulated, but also possible Byzantine effects at fanout branches in order to find possible explanations for otherwise unexplained non–SLAT tests. Several methods use SLAT to locate suspect signal lines and then proceed with specific fault model based analysis like considering activation conditions for opens and bridges [SatoSSY*2006], trapped charges in interconnect opens [ZouCR2006], or layout information for via defects [LiuZRCS*2007].

By using multi–valued logic simulation in inject–and–validate diagnosis algorithms, defects can be located in a circuit despite incomplete knowledge on their actual behavior [BoppaF1998]. Such a diagnosis algorithm reports the minimum set of signal lines that need to be set to an unknown value (X–value) in the fault simulator, such that fault simulation will produce a X for every failing response bit from the C_{ud}. The same idea was used more recently [PokuB2007] for diagnosing delay faults. Three–valued logic simulation is pessimistic and will result in X–values at outputs which are actually fault–free.

The works [WenTSK2003, WenKMYS*2006] alleviate this problem by using distinct X–symbols for every signal line and a more sophisticated X–propagation method that preserves some logic properties during X–propagation (inversion and branches). After location of the defect site, the unknown values at the site are resolved by 2–valued simulation exhaustively assigning the X–symbols to logic values.

Inject–and–validate diagnosis is computational expensive as it relies heavily on fault simulation of many possible candidates. Many proposals on high performance fault simulation are also applicable to inject–and–validate algorithms. Common techniques used in fast fault simulation can be found in [Schul1988]. Another canonical way is to trade computing time against memory storage by constructing a fault dictionary [RichmB1985, PomerR1992]. In contrast to cause–effect approaches, the dictionary is only used to replace some fault simulation runs by table lookups. The fault simulation problem can also be mapped to emerging data parallel architectures like *General Purpose Graphics Processing Units* (GPGPUs) [LiH2010, LiXHCL2010, HolstSW2012] or hybrid systems as the *Cell Broadband Engine* [KochtSW*2010] to gain speedup compared to a simulation on standard hardware.

3.2.3. Back–Tracing

Back–tracing based diagnosis strategies start with the failure information (test responses) at the output of the circuit and infer successively towards the inputs of the circuit internal signal values, possible fault propagation paths, and fault candidates.

The simplest technique, the so–called *back–coning* or *structural pruning*, considers all signal lines as possible fault locations, which have at least one structural path to a failing circuit output [WaicuL1989]. This alone is not sufficient for logic diagnosis, but it is used in numerous algorithms to provide the first initial pruning of the fault candidates.

Logic reasoning, first proposed by [AbramB1980a], searches for consistent values of all internal signals based on an observed response. Finding proper input values in order to generate a desired output is called *justification*. Two situations can arise at a gate. First, there is only one set of input values to the gate that produces the desired output. For example, to get a 1 at the output of a 2–input

AND–gate, the only possible input vector is $(1, 1)$. In this case, the input values are *implied* and the algorithm can proceed justifying the two new found values. Second, there is more than one possible input vector (e.g. justifying a 0 at a 2–input AND–gate leads to 3 possible input vectors). In this case, the algorithm takes a decision and branch into a search tree. This search tree is explored until a solution is found. This method was refined in [CoxR1988a] and used for multi–fault diagnosis. Justification is used to identify all fault–free lines in the circuit. The reported suspects are lines, which could not be shown to be fault–free. This way multiple faults can be diagnosed without exhaustive simulation of all possible multi–faults.

Critical path tracing techniques trace back error propagation paths from the outputs towards the possible culprits within the circuit. The concept of critical path tracing was first introduced in [AbramMM1983] as an efficient alternative to fault simulation and was first applied to delay fault diagnosis in [GirarLP1992, GirarLP*1992a]. The diagnosis procedure for delay faults uses a six–valued logic algebra to describe logic values, transitions and hazards in the circuit. This approach was later refined [RoussBG*2007], applied to bridging fault diagnosis [RoussBG*2007a, WangGB2006] as well as to crosstalk–induced delay faults [MehtaMT*2006].

Path tracing based diagnosis can be improved by taking into account the timing uncertainties (variation) of manufactured circuits [YangC2006]. The approach in [YenLLYL*2008] extends back–tracing over multiple time frames to analyze responses of functional tests.

Many path tracing based diagnosis techniques fail in the case of multiple interacting faults in the C_{ud} as they assume all the off–path signals of the currently traced fault propagation path to be fault–free or affected by the same fault in case of a re–convergence. If multiple faults are present, the effect of a fault x may sensitize a propagation path of a fault y at gate g. If fault x is not considered, path tracing would assume that the fault is located between g and the outputs, because a fault effect stemming from y would not propagate over g. This challenge was addressed in [YuB2008, YuB2008a, YuB2010] by using a more conservative path tracing approach in combination with fault simulation to diagnose multiple stuck–at, bridges, and transition faults. The situation described above is called an *underexplained propagation path* in these works.

3.2.4. SAT–Based Algorithms

The first formulation of the logic diagnosis problem as a Boolean satisfiability instance was proposed in [SmithVV2004]. Logic circuits are represented as Boolean formulas using the *Tseitin–transformation* [Tseit1968, Larra1992]. Each logic signal in the circuit corresponds to a variable in the formula, and the formula is satisfied if and only if the signals assignment is consistent with the logic function of the circuit. For logic diagnosis, first, fault injection circuits are added to the model to enable the SAT solver to activate faults in the circuit using some control variables $f_1, \dots, f_n$. Further constraints are added in order to limit the fault multiplicity and to assign a test pattern to the input signals and the observed response to the output signals. If the resulting SAT instance is satisfiable, possible fault locations are given by the assignment to the variables $f_1, \dots, f_n$.

More recent works also apply *maximum satisfiability solvers* to the diagnosis problem [SafarMV*2007, ChenSMV2010]. The maximum satisfiability problem is the problem of finding the maximum set of satisfiable clauses in a Boolean formula. In this case, no fault injection circuitry is necessary and the signals involved in unsatisfied clauses point to inconsistencies in the circuit and the possible fault locations.

More details on SAT–based diagnosis and a comparison simulation–based techniques can be found in [FeySVD2006].

3.2.5. Model Allocation

Model allocation techniques seek to extract more information about the behavior of the defect in order to further constrain the suspected chip area to be physically inspected and provide more insight into the actual defect mechanism.

A basic variant of model allocation is the concept of *composite signatures* for bridges [MillmMA1990] or open faults [VenkaD2000] already discussed with the inject–and–validate approaches. If both signal lines involved in a bridge can be identified for instance by a composite of stuck–at faults, only the area in the chip layout where these signals are close together is suspect. This fails for bridges with a dominant signal, which only influences a victim signal but always propagates a fault–free value itself.

One way to identify such aggressor lines is by correlating the defect activation to the signal values of the relevant neighborhood and extracting an activation function depending on the states of the neighboring signals [DesinPB2006, YuB2010a].

Cell–internal defects (i.e. bridges, opens, or transistor faults within a CMOS cell) can be identified if the observed output values of a cell are correlated to its input values [AmyeeNV2006, HigamST*2006, SharmCT*2007, LadhaM2010].

If a model is assumed in advance, signal correlations can also be used for initial pruning. The bridge fault diagnosis method proposed in [VogelMB2003] exploits the fact that bridge defects are only active if the two involved signal lines have different logic values. All failing tests $\vec{t}_1, \vec{t}_2, \ldots \in T'$ are simulated to obtain the values of all internal signal lines L for the fault–free case. With the simulation of the first pattern $\vec{t}_1$, L can be partitioned into two sets: L^0_{t1} the signals with the fault–free value 0 and L^1_{t1} the signals with the fault–free value 1. Because $\vec{t}_1$ is a failing test, a bridge can only be located between one signal of L^0_{t1} and one signal of L^1_{t1}. Let (L^0_{t1}, L^1_{t1}) denote all possible bridge candidates in this case. By simulating the next failing test $\vec{t}_2$, the sets L^0_{t2} and L^1_{t2} are obtained to further reduce the number of possible bridge candidates to:

$$\{\, (L^0_{t1} \cap L^0_{t2}, L^1_{t1} \cap L^1_{t2}), (L^0_{t1} \cap L^1_{t2}, L^1_{t1} \cap L^0_{t2}) \,\}$$

The remaining failing patterns are processed in the same way to obtain a reduced set of possible bridge candidates.

3.3. Conclusions

Defect oriented fault models describe the behavior of specific defect types. Applying such fault models to diagnosis adds restrictions to specific defect types and diagnosis algorithms may fail, if the assumptions on the defective behavior are not true. Instead, diagnosis algorithms should be based on more general fault models. Several general fault models and notations are available in the literature, and it has been shown that all faults in the most popular notations can be translated into a CLF representation. The generality of the CLF calculus allows it to encompass all previous notations and basing a diagnosis algorithm directly on this calculus provides a promising way to develop analysis methods with as few assumptions on defects as possible.

The goal is a diagnosis algorithm, which works in as many application areas as possible and is able to locate arbitrary defects. Only an effect–cause based algorithm is able to reach this goal. Among these algorithms, *back–tracing* is promising for its efficiency of finding time–related and multiple defects, but it has not been shown that this efficiency can be maintained in combination with test response compaction. *SAT–based* diagnosis techniques use heuristics of efficient SAT solvers to solve an NP–complete problem. The runtime of such algorithms on specific diagnosis cases are usually unpredictable and it has yet to be shown that these techniques are able to handle industrial–sized designs and large test sets. *Model allocation* techniques are applicable only if the location of a potential defect site is already known. This leaves *inject–and–validate* based algorithms as the only choice.

Exploiting the SLAT property of test responses has been proven very successful in diagnosing complex defects using fault simulation of much simpler faults. The original SLAT algorithm however uses just failing patterns with this SLAT property for diagnosis and needs to solve a set covering problem in order to compute the diagnosis result. Proposed extensions to this original algorithm are able to improve diagnosis results by passing pattern validation and analysis of failing patterns without SLAT property. However, these approaches add even more complexity to the SLAT algorithm and the diagnosis performance becomes very hard to predict.

A location–based effect–cause logic diagnosis approach, which

- exploits the SLAT property and yet extracts maximum diagnostic information out of all available failing and passing responses,
- provides predictable analysis times for easy deployment in volume production environments,
- and works at the same time on highly compacted response data

was not available in the literature and has been contributed to the state–of–the–art by the works published along with the research for this thesis.

4. POINTER

The *Partial Overlapping Impact couNTER* (POINTER)[HolstW2007, HolstW2007a, HolstW2009b] is a new location–based logic diagnosis algorithm based on inject–and–validate per–test analysis. Given the structure of a combinational circuit C, it infers from a known test set T and the observed responses $R_{ud} = C_{ud}(T)$ a sorted list of locations in C, which are likely to be defective in the C_{ud}. These locations are expressed as CLFs with additional measures to quantify how well the particular CLF explain the syndromes of the defect in the C_{ud}. Using the locations of the CLFs, the defect site is identified.

4.1. Fault–Model Independence

POINTER compares each response of the C_{ud} to a set of responses generated by a fault simulator. However, it is impossible to simulate every possible defect in the fault simulator because the set of all possible defects is infinite. Instead, only a few *defect representatives* are simulated and the criterions for the comparison of the syndromes from the C_{ud} and the fault simulation results accommodate for all possible defects in the C_{ud}.

The POINTER algorithm considers a certain defect representative f a candidate for the unknown defect in the C_{ud} as long as the set of failing responses from the fault simulation of f is a superset of the faulty syndromes of the C_{ud}:

$$\forall \vec{t} \in T \;\; C_{ud}(\vec{t}) \neq C(\vec{t}) \Rightarrow C_f(\vec{t}) \neq C(\vec{t})$$

So, f is a candidate if it predicts a faulty response for all tests $\vec{t}$ where $C_{ud}(\vec{t})$ shows a faulty syndrome. The representative f does not have to model the defect in the C_{ud} exactly. It is sufficient for f to satisfy the implication defined above.

The set of defect representatives F can have the same cardinality as the set of stuck–at faults in the circuit. In the simplest case, a defect representative of the form $s \oplus [1]$ (with s being an arbitrary signal line) satisfies above implication for all defects $s \oplus [c]$ with signal line s and an arbitrary condition c. The condition c is not restricted in any way. It may include Boolean conditions, timing and even random factors. By simulating the set of unconditional line flips $s \oplus [1]$ for all signals s in the circuit, POINTER will be able to include the correct defect location in the set of candidates as long as faulty responses from the C_{ud} exist and are caused by a single–line defect.

Defects may also influence more than one internal signal directly. More than one CLF may be necessary to model such defects and some responses of the C_{ud} might not match with any response of the fault simulation with single unconditional line flips. In many cases, some syndromes exist, which can be explained by a defect representative f and it will be shown in the experiments that these representatives most likely identify the defect location even in this situation. If there is no syndrome that matches the response of any defect representative exactly, POINTER will output defect representatives, which match the observed syndromes best. The reasoning behind the matching of two similar responses is explained in the next section.

4.2. Response Matching Criterions

Besides using the pass/fail information of complete tests, POINTER also compares the individual responses of the fault simulator and the C_{ud} bit–wise with each other to quantify their similarity. Not the output values themselves are compared to each other but the pass/fail information of the individual circuit outputs. This pass/fail information is obtained by computing the differences to a fault–free simulation $C_{ud}(\vec{t}) \oplus C(\vec{t})$ for $\vec{t} \in T$. POINTER considers two aspects while comparing two responses.

The first aspect is overlap in the failing bits of two responses as illustrated in figure 4.1. It shows a C_{ud} response compared to the responses from fault simulation of faults f and g. The number of failing outputs is the same in all three responses. However $C_g(\vec{t})$ has a larger overlap with the observed syndrome $C_{ud}(t)$ which indicates that the defect representative g is located nearer to the actual defect site than f. POINTER therefore will consider g as a better candidate compared to f.

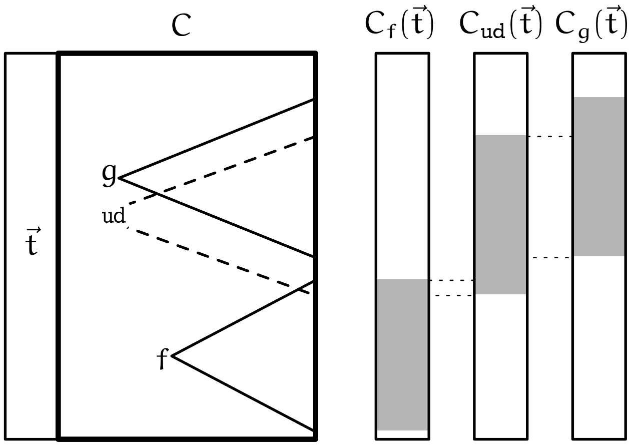


Figure 4.1.: Overlap between failing response bits.

One special case is a complete overlap or a perfect match between the syndrome and the effect of a defect representative. In this case, the syndrome of the C_{ud} can be explained completely by a defect representative $f \in F$ for a test $\vec{t} \in T$:

$$C_{ud}(\vec{t}) \oplus C(\vec{t}) = C_f(\vec{t}) \oplus C(\vec{t}) \neq \vec{0}$$

This condition is also known as SLAT property [BarteHH*2001] already discussed in section 3.2.2. In general, not all patterns will have the SLAT property, and not all the SLAT patterns will point to the same stuck–at faults but is has been shown that SLAT patterns provide very strong evidence towards the actual defect sites [BarteHH*2001, Huism2004].

Also in POINTER, the SLAT condition carries extra weight. If there is a defect representative, which matches all failing responses of the C_{ud} perfectly:

$$\exists f \in F \; \forall t \in T \;\; C_{ud}(t) \oplus C(t) \neq \vec{0} \Rightarrow C_{ud}(t) \oplus C(t) = C_f(t) \oplus C(t)$$

it will take precedence over all other representatives with partial matches, even if other candidates explain more response bits or lead to less additional failures. If no perfect match candidates are available, the amount of overlap is taken into account as previously explained. In this aspect, POINTER is a generalization of the original SLAT based diagnosis approach.

The second aspect taken into account in POINTER is the impact the defect has on the response under specific tests (see fig. 4.2). If the unknown defect leads only to a few failing outputs (low impact), the chance is high that many suspects in the output cone of the actual defect lead to similar responses. In contrast, if the impact of the defect is high, only a few fault candidates may produce similar responses and these candidates are close to the real defect site. Thus POINTER will rate the evidence produced in the latter case higher than the evidence in the former case.

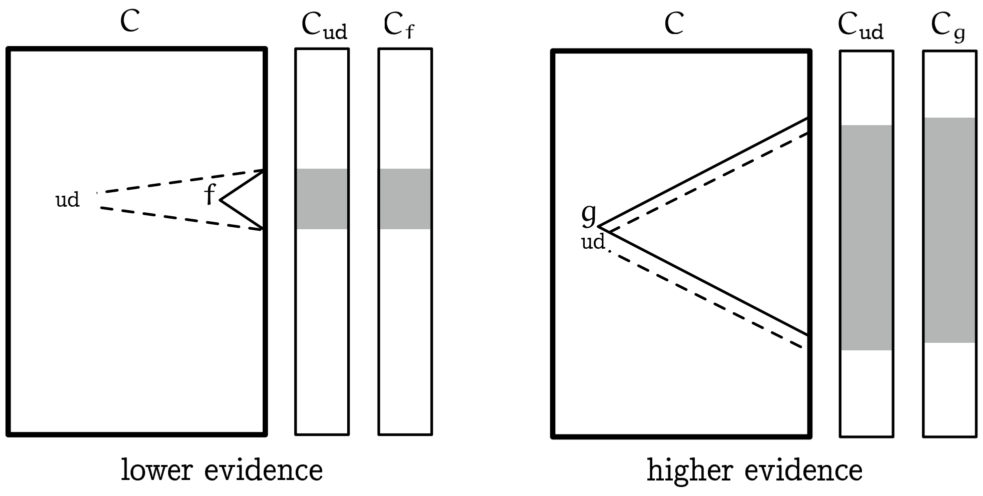


Figure 4.2.: Relation between evidence and impact of defects.

4.3. Evidence Calculation

As a per–test analysis approach, POINTER analyzes each C_{ud} response individually. A response $\vec{r}_{ud} = C_{ud}(\vec{t})$ to a test vector $\vec{t}$ is analyzed by comparing it to the responses $\vec{r}_f = C_f(\vec{t})$ of a set of defect representatives $F \ni f$ in the circuit. For each test $\vec{t}$, a comparison between $\vec{r}_{ud}$ and $\vec{r}_f$ yields a tuple of 4 integer numbers called *evidence* $e(\vec{t}, f) = (\sigma, \iota, \tau, \gamma)$. These numbers reflect the similarity between the two responses and are defined in the next section. The evidence is constructed in a way that the summation over the evidences of multiple tests component–by–component

$$e(T, f) = \sum_{\vec{t} \in T} e(\vec{t}, f)$$

preserves important properties. The accumulated evidence $e(T, f)$ is then used to produce a ranked list of defect candidates in section 4.3.2.

4.3.1. Response Comparison

Let $\vec{r}$ be the correct response, $\vec{r}_{ud}$ the response of the C_{ud}, and $\vec{r}_f$ the response of a circuit C_f with a known defect representative $f \in F$. The similarity between $\vec{r}_{ud}$ and $\vec{r}_f$ is quantified by four natural numbers σ, ι, τ and γ. The first three numbers σ, ι, τ are defined by the following functions:

$$\sigma(\vec{r}_{ud}, \vec{r}_f, \vec{r}) := \|(\vec{r}_{ud} \oplus \vec{r}) \wedge (\vec{r}_f \oplus \vec{r})\|$$

is the number of response bits that are faulty both in $\vec{r}_{ud}$ and $\vec{r}_f$. It can be interpreted as the number of explained fails or predictions by assuming f as the culprit. This function relates to the common term *tester–fail simulation–fail* (TFSF) [Huism2005].

$$\iota(\vec{r}_{ud}, \vec{r}_f, \vec{r}) := \|(\vec{r}_{ud} \equiv \vec{r}) \wedge (\vec{r}_f \oplus \vec{r})\|$$

is the number of response bits which are faulty in $\vec{r}_f$, but correct in $\vec{r}_{ud}$. This is the number of mispredictions by assuming fault f. It relates to the common term *tester–pass simulation–fail* (TPSF) [Huism2005].

$$\tau(\vec{r}_{ud}, \vec{r}_f, \vec{r}) := \|(\vec{r}_{ud} \oplus \vec{r}) \wedge (\vec{r}_f \equiv \vec{r})\|$$

is the number of response bits which are faulty in $\vec{r}_{ud}$ but correct in $\vec{r}_f$ (nonpredictions). These are failures which cannot be explained by f. This function is also known as *tester–fail simulation–pass* (TFSP) [Huism2005].

When C_{ud} and C are clear from the context, functions depending directly on a fault f and a test $\vec{t}$ are used for a more compact notation:

$$\sigma(f, \vec{t}) := \sigma(C_{ud}(\vec{t}), C_f(\vec{t}), C(\vec{t})),$$

$$\iota(f, \vec{t}) := \iota(C_{ud}(\vec{t}), C_f(\vec{t}), C(\vec{t})),$$

$$\tau(f, \vec{t}) := \tau(C_{ud}(\vec{t}), C_f(\vec{t}), C(\vec{t})).$$

Figure 4.3 visualizes these three comparison values. The C_{ud} on the left hand side and the C_f on the right hand side get the same test vector $\vec{t}$ as an input. The gray areas in the responses are the failing bits.

If a response $\vec{r}_f$ is identical to $\vec{r}_{ud}$, the two values ι and τ are 0:

$$\iota(\vec{r}_{ud}, \vec{r}_f, \vec{r}) = \tau(\vec{r}_{ud}, \vec{r}_f, \vec{r}) = 0 \Leftrightarrow \vec{r}_f = \vec{r}_{ud}$$

This is easily observed in figure 4.3 and follows from the function definitions with $\|(\vec{r}_f \oplus \vec{r}) \wedge (\vec{r}_f \equiv \vec{r})\| = 0$. With τ being 0, σ equals the number of failing bits in $\vec{r}_{ud}$ which is the maximum possible value σ can have for the current test $\vec{t}$. There is no other fault $\tilde{f} \in F$ with a higher number of predictions:

$$\forall \tilde{f} \in F \;\; C_{ud}(\vec{t}) = C_f(\vec{t}) \Rightarrow \sigma(f, \vec{t}) \geqslant \sigma(\tilde{f}, \vec{t})$$

Again, this perfect match with a single defect representative for a single test is the SLAT–condition and in the original SLAT algorithm [BarteHH*2001], only this special case is used for evidence.

If the C_{ud} response does not contain any fails in the current test ($\vec{r}_{ud} = \vec{r}$), σ is 0 for all faults $f \in F$. So all σ–values are equal and especially, $\sigma(f, \vec{t})$ is now maximum for all faults $f \in F$.

The fourth value is the minimum of σ and ι:

$$\gamma(f, \vec{t}) = \min\{\sigma(f, \vec{t}), \iota(f, \vec{t})\}.$$

γ will be zero, if the SLAT condition holds for the current test ($\iota = \gamma = 0$), or if the C_{ud} passes the current test ($\sigma = \gamma = 0$). In the case of $\gamma > 0$ like in figure 4.3, the corresponding defect representative is not a candidate for a single fault.

4.3.2. Accumulation and Ranking

For a fault f, each test of a test set $\vec{t} \in T$ yields the evidence

$$e(f, \vec{t}) = (\sigma(f, \vec{t}), \iota(f, \vec{t}), \tau(f, \vec{t}), \gamma(f, \vec{t})).$$

The evidence of a fault f and a test set T is

$$e(f, T) = (\sigma(f, T), \iota(f, T), \tau(f, T), \gamma(f, T))$$

with

$$\sigma(f, T) = \sum_{\vec{t} \in T} \sigma(f, \vec{t}),$$

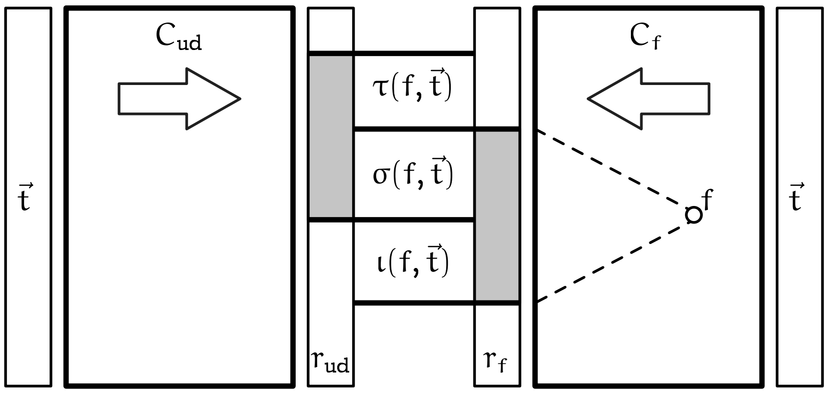


Figure 4.3.: Response comparison

$$\iota(f,T) = \sum_{\vec{t}\in T} \iota(f,\vec{t}),$$

$$\tau(f,T) = \sum_{\vec{t}\in T} \tau(f,\vec{t}),$$

$$\gamma(f,T) = \sum_{\vec{t}\in T} \gamma(f,\vec{t}).$$

While processing pattern after pattern, $\vec{t}_1,\dots,\vec{t}_i$, the knowledge base is constructed by the evidences $e(f,T_i)$, $T_i = \{\vec{t}_1,\dots,\vec{t}_i\}$ for all the defect representatives f. If a fault is not observable under a certain pattern, no value change takes place and this fault is not considered within this iteration. If the C_{ud} gives the correct output under a pattern $\vec{t}$, only $\iota(f,T)$ is increased for faults which are observable under this pattern and hence lead to a misprediction. In this way, candidates can be excluded using passing patterns, too. If the real culprit behaves exactly like a defect representative f, its evidence will show $\iota(f,T) = \tau(f,T) = 0$ and $\sigma(f,T)$ will be maximum. If the fault in the C_{ud} is not always active due to nondeterministic behavior or some unknown activation mechanism, the measure still provides consistent evidences.

For instance, let a defect d behave like a slow to rise gross delay fault. For some patterns $\vec{t}$, fault d will appear as a stuck–at 0 fault f, for others it is not observable. In both cases, $\sigma(f,\vec{t}) \geqslant \sigma(\tilde{f},\vec{t})$ with respect to all other faults $\tilde{f} \in F$ as shown in the previous section. Consequently, this also holds for the accumulated values: $\sigma(f,T) \geqslant \sigma(\tilde{f},T)$ and the evidence $e(f,T)$ still contributes information for locating the fault. However, the value $\iota(f, T)$ will not be zero anymore and can be used for ranking fault candidates.

For further analysis, the evidences in the knowledge base are ordered to create a ranking with the most suspicious fault sites at the beginning (lowest rank). Firstly, evidences are sorted by increasing γ, i.e.

$$\gamma(f, T) > \gamma(g, T) \Rightarrow \mathrm{rank}_T(f) > \mathrm{rank}_T(g)$$

moving defect representatives with only perfect matches in front. When assuming multiple faults, mutual fault masking is rather rare, and ranking the defect representatives according to the size of γ provides a good heuristic. Evidences with equal γ are then sorted by decreasing σ moving candidates in front, which explain most failures:

$$\sigma(f, T) > \sigma(g, T) \Rightarrow \mathrm{rank}_T(f) < \mathrm{rank}_T(g).$$

Finally evidences with equal γ and σ are ordered by increasing ι:

$$\iota(f, T) > \iota(g, T) \Rightarrow \mathrm{rank}_T(f) > \mathrm{rank}_T(g).$$

4.3.3. Example

For a brief example of the pattern analysis approach, consider the circuit in figure 4.4. It contains two gates and four exemplary stuck–at faults as defect representatives for fault simulation. The exhaustive test set and the response from the C_{ud} are shown in the first two columns of table 4.1. The erroneous bits are shown in bold, the C_{ud} has failed on output x in the third pattern.

Now, the four faults are simulated for the given pattern set and their signatures are shown in the remaining columns in table 4.1. The fault f_1 is observable in

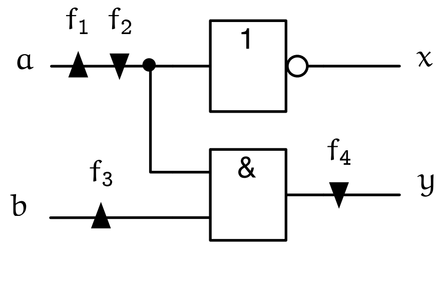


Figure 4.4.: Circuit model for fault simulation

Pattern ab	Syndrome xy	f_1 xy	f_2 xy	f_3 xy	f_4 xy
00	10	**00**	10	10	10
01	10	**01**	10	10	10
10	**10**	00	**10**	**01**	00
11	01	01	**10**	01	**00**

Table 4.1.: Syndrome and result from stuck–at fault simulation

three response bits, but it fails to explain the erroneous bit in the syndrome. This leads for this fault to an evidence of $e(f_1, T) = (\sigma, \iota, \tau, \gamma) = (0, 3, 1, 0)$. The evidence is derived for the other stuck–at faults as well; Table 4.2 shows the result.

Fault	σ_T	ι	τ	γ	Rank
f_1	0	3	1	0	4
f_2	1	2	0	0	1
f_3	0	1	1	0	2 or 3
f_4	0	1	1	0	3 or 2

Table 4.2.: Evidences and rank of the four faults

All evidences show $\gamma = 0$, so the ranking procedure continues with σ. Only f_2 has positive σ, so this fault is ranked above all other faults. The other faults are ranked by increasing ι. The top–ranked evidence f_2 shows positive σ and positive ι. Therefore, none of the simulated faults can explain the syndrome completely, but f_2 explains a subset of all fails. This leads to a CLF of the form $a \oplus [a \cdot \text{cond}]$ with some arbitrary condition.

4.4. Evidence Interpretation

After analysis of a test set T, each defect representative $f \in F$ has an evidence $e(f, T)$ associated with it. The evidence values reflect the relation of a defect representative to the defective behavior of the C_{ud} and certain evidence forms imply information about the unknown defect beyond the probable locations of victim signal lines. For further discussion of these forms, evidences are classified into so–called *match conditions*. Table 4.3 shows the definitions of the different match conditions. The table is sorted with respect to the *quality* of the match between defect representative and C_{ud} behavior with *perfect match* being the strongest match and *best guess* being the weakest. The ranking procedure in POINTER also ranks the candidates in the same way as shown in the table. For instance, if there is an evidence with perfect match condition, it will take precedence over an evidence with partial match condition.

Match Condition	$\iota(f,T)$	$\tau(f,T)$	$\gamma(f,T)$
Perfect Match	0	0	0
Perfect Tester Fail Match	>0	0	0
Partial Match	0	>0	0
Partial Tester Fail Match	>0	>0	0
Best Guess	>0	$\geqslant 0$	>0

Table 4.3.: Match conditions and evidence forms for $e(f,T)$

4.4.1. Perfect Match

If ι, τ, and γ are all zero in a $e(f,T)$, then the defect representative f is able to explain the C_{ud} behavior completely. In other words, under the test set T both the C_{ud} as well as C_f give exactly the same responses for all the tests. Thus, the model chosen for the defect representative is able to exactly model the defect in the C_{ud}. This is the strongest possible evidence provided by POINTER.

The value σ in the evidence equals the number of explained failing bits in the C_{ud} responses and can serve as additional confidence measure. One special case is $\sigma = 0$. In this case, the C_{ud} did not produce any failures under test set T and the evidences for all defect representatives f not observable during fault simulation will show a perfect match $e(f,T) = (0,0,0,0)$. These defect representatives identify all insufficiently tested circuit parts and the diagnosis result is completely valid also in this case.

There may be multiple defect representatives with perfect match conditions. The behavior of all these representatives is the same under the test set T and it is impossible to distinguish between these candidates with this test set.

4.4.2. Perfect Tester Fail Match

Evidences with $\tau = 0$, $\gamma = 0$ and positive ι are classified as *perfect tester fail match*, because in this case, all *failing* responses from the C_{ud} match completely with the behavior of the defect representative.

Because of $\tau = 0$, there are no unexplained failing response bits from the C_{ud}. The $\gamma = 0$ ensures that all failing responses from the C_{ud} are equal to the responses of the fault simulation of the associated defect representative. Suppose, there is a failing response of the C_{ud} that does not match fault simulation. Because there are no unexplained failing response bits ($\tau = 0$) the only difference can be that fault simulation leads to additional fails ($\iota > 0$). With γ being the maximum of σ and ι, it would be positive in this case.

As fault simulation of the defect representative match all failing responses, the difference to a perfect match are the existence of tests, which pass in the C_{ud} but not in fault simulation. The number of failing bits in these additional failing tests is given by $\iota > 0$.

If the defect representative can be expressed as a CLF of the form $f \oplus [c]$ with a victim signal f and a known, deterministic condition c, then the defect in the C_{ud} can be modelled by a CLF of the form $f \oplus [c \cdot u]$ with an additional, unknown condition u.

If $\sigma = 0$ in a perfect tester fail match evidence, the C_{ud} did not produce any failures. In contrast to defect representatives with perfect match condition, representatives with perfect tester fail match lead to failing responses and were tested by the given test set. If the C_{ud} passed all tests and the best ranked evidence is a perfect tester fail match, all defect representatives in the circuit were tested. In this case, the C_{ud} might contain a defect whose unknown condition u was never satisfied for tests that failed in fault simulation.

4.4.3. Partial Match

Evidences with *partial match* condition show $\iota = 0$, $\gamma = 0$ and a positive τ. The defect representative associated with such an evidence explains a subset of all tester fails, but some other faulty behavior is present in the C_{ud}.

If the best observed evidence is a partial match and defect representatives can be modeled with a single CLF of the form $f \oplus [c]$ with a victim signal f and a known, deterministic condition c, the defect in the C_{ud} can be modeled either

- by a single CLF on a signal line, which has no representative in the fault machine;
- by a single CLF of the form $f \oplus [c \vee u]$ with an unknown condition u and $u \not\rightarrow c$;
- by multiple CLFs on different signal lines.

If the defect representatives include CLFs of the form $f \oplus [1]$ on every signal line in the circuit, the first two modeling alternatives are not possible anymore and the behavior of the C_{ud} cannot be explained by any single fault.

Given multiple fault sites in the C_{ud}, the defect representative with the best ranked partial match evidence provides the best explanation for one of the fault sites. Also, there is a very high probability that the other fault sites are independent from the defect representative at hand, i.e. a failing output is either caused by the defect representative only or by some other fault sites.

4.4.4. Partial Tester Fail Match

A *partial tester fail match* is a evidence that shows $\gamma = 0$ and positive values for ι and τ. The basic reasoning for partial matches are also applicable to partial tester fail matches. Given that the defect representatives include CLFs of the form $f \oplus [1]$ on every signal line in the circuit, multiple faults must be present in the C_{ud}.

The difference to partial matches is that now, ι has a positive value. This means, combined with $\gamma = 0$, that fault simulation of the defect representative lead to additional failures not present in the syndromes of the C_{ud} in the following situations:

- a test passed in the C_{ud};
- a test failed in the C_{ud}, but the failing outputs caused by the defect representative are disjoint from the failing bits of the C_{ud}.

Again, there is a very high probability that the other fault sites are independent from the defect representative that show a partial tester fail match.

4.4.5. Best Guess

If all suspects show positive values in all components ι, τ, γ, all simplistic fault models would fail to explain the C_{ud} behavior.

This may happen in two situations:

- At least one response of the C_{ud} was caused by multiple interacting fault sites. I.e. an output of the C_{ud} was influenced by at least two faults on different signal lines at the same time.
- Error propagation was insufficiently modeled in fault simulation. If the defect is a time–related issue, which cannot be modeled as gross delay fault, and fault simulation employs the zero delay model for error propagation, no matching can be achieved. One typical example are *small delay faults* [TehraPC2011].

In this case, POINTER just reports the defect representatives, which explains many of the failing outputs of the C_{ud}. This heuristic still works well to identify the circuit regions, faults are located in. However, it is not any more true that the defect representative at one of the defective lines always explains the highest number of errors. To investigate this situation further, the ranking can serve as a starting point for more specialized diagnosis algorithms that target the cases described above.

4.5. Final Suspect List

Let $f_t \in F$ be the defect representative at the top of the ranking ($\text{rank}_T(f_t) = 1$). Depending on the match condition observed in the top–ranked evidence $e(f_t, T)$ of the final ranking, the following defect representatives are included in the reported list of suspects:

- Perfect match ($\iota(f_t, T) = \tau(f_t, T) = \gamma(f_t, T) = 0$): only defect representatives with equivalent evidence are included in the suspect list.

- Perfect tester fail match ($\tau(f_t, T) = \gamma(f_t, T) = 0$): only defect representatives f with $\sigma(f, T) = \sigma(f_t, T)$ are included in the suspect list and this list is sorted by increasing $\iota(f, T)$.
- In all other cases, all defect representatives with positive $\sigma(f, T)$ are included in the suspect list, which is then sorted first by decreasing σ then by increasing ι.

The number of suspects reported by POINTER gives a good indication on the confidence in the computed diagnosis result. The better the matching to a defect representative, the fewer suspects are reported. The experimental evaluation will show that, although the number of reported suspects can be quite high in the *best guess* case, the real defect site can be very often found by examining just the first few suspects at the top of the ranking.

4.6. Summary

POINTER is a location–based logic diagnosis algorithm that is based on an inject–and–validate approach. It is able to diagnose arbitrary spot defects and is not restricted to a specific fault model as it outputs locations of possible defects within the circuit structure directly with the help of defect representatives.

For each defect representative, an evidence containing four integer number is maintained: The number of predictions σ, the number of mispredictions ι, the number of nonpredictions τ, and $\gamma = \min\{\sigma, \iota\}$. The value of γ relates to the SLAT property of individual tests. While SLAT–based diagnosis algorithms only extract information out of failing responses with SLAT property, POINTER extracts diagnostic information from every available response. Failing responses are used to find defect locations, which lead to equal or similar syndromes. Fault–free responses are used to rule out defect locations, which likely would have failed the current test.

With every new response, the evidences are simply summed up and the final diagnosis result is generated just by sorting the set of evidences. Compared to SLAT, which solves a set covering problem for generating the diagnosis result, the runtime behavior of POINTER is very predictable and scales linearly with the size of the design and the size of the test set.

The forms of the evidences (matching conditions) describe the relationship between the defect in the C_{ud} and defect representatives. These matching conditions range from a *perfect match* between defect and representative, over *perfect tester fail match* and partial matches down to a *best guess*. The quality of the matching also reflects the confidence of the algorithm in the diagnosis result. Depending on this confidence, more or less suspects are reported as final result.

5. Adaptive Precision Diagnosis

High precision logic diagnosis is one of the tools for the in–depth analysis of malfunctioning chips in a lab environment. If the defect is suspected in a logic block of the chip, the tool is used to pin–point the defective structure as precisely as possible in order to guide physical failure analysis.

Very precise logic diagnosis needs besides an efficient response analysis method also tests with high diagnostic resolution. The production test pattern set is not an ideal choice, because they are often generated based on simple fault models like the stuck–at fault model and aim to test as many faults as possible at the same time in order to reduce the number of test patterns [BushnA2000]. For fault model independent diagnosis algorithms like POINTER, better options might be n–detect test sets [McCluT2000, LinPBNL*2008] or pseudo–exhaustive test sets [McClu1984]. However, these test sets are either still based on a fault model or contain such a large number of patterns that the resulting test time may not be acceptable anymore even in a lab environment.

Besides relaxed time and test data volume constraints, a lab environment provides additional flexibility that will be exploited by the diagnosis approach described in this chapter. While test sets are usually fixed during production test, now full interactive access to the C_{ud} is possible and tests may depend on the responses to previously applied patterns. After analysis of the first few test responses by a logic diagnosis algorithm, it is possible to generate new diagnostic tests specifically targeted to the defect in the C_{ud} based on collected evidence. This procedure is called *adaptive diagnosis* and has two advantages which lead to a more precise and faster diagnosis:

- Test patterns can be generated specifically for suspicious signals and gates in the circuit. If the suspicious region of the circuit is sufficiently small, it can even be tested exhaustively without any dependence of an a–priori chosen fault model.

- Compared to other fault model independent test sets, the time for test and diagnosis is greatly reduced as for every specific C_{ud} only patterns of high diagnostic value are applied.

This chapter starts with an introduction as well as a review of the state–of–the–art in adaptive diagnosis and diagnostic pattern generation. After this, a new adaptive diagnosis approach is introduced, which combines POINTER with a diagnostic pattern generator to obtain maximum diagnostic resolution.

5.1. Introduction to Adaptive Diagnosis

The paradigm adaptive diagnosis was first proposed for system diagnosis [Nakaj1981, HakimN1984] and applied for diagnosing bridging defects in [GongC1995]. The method in [GongC1995] uses I_{ddq} measurements for diagnosis and exploits the fact that a circuit draws more current if the two signals involved in a bridge are driven to different logic values. First, random patterns are applied to the circuit, then patterns targeting signals with low controllability and finally patterns to distinguish the remaining fault candidates. The pattern generation process is guided by a list of fault candidates which is in turn refined by the results of the patterns applied to the C_{ud}.

Figure 5.1 depicts this principle. A pattern analysis step extracts information from responses of the C_{ud} and accumulates it in a knowledge base. This knowledge in turn guides an automatic test pattern generator (ATPG) to generate relevant patterns for achieving high diagnostic resolution. The loop ends when an acceptable diagnostic resolution is reached.

Some works exploit this concept by generating additional high–resolution patterns after an initial diagnosis pass. In [LinLC2007], SLAT is used for the initial pass and new diagnostic patterns are adaptively generated to propagate the fault candidates individually to specific outputs. In [YangC2007] the tests are constructed in a way so that none of the propagation paths of a candidate passes through another candidate site.

Given two faults f and g, a diagnostic test generator calculates a test $\vec{t}$, which is able to distinguish between these faults: $C_f(\vec{t}) \neq C_g(\vec{t})$. Such diagnostic pattern generators are available for stuck–at faults [RothBS1967,

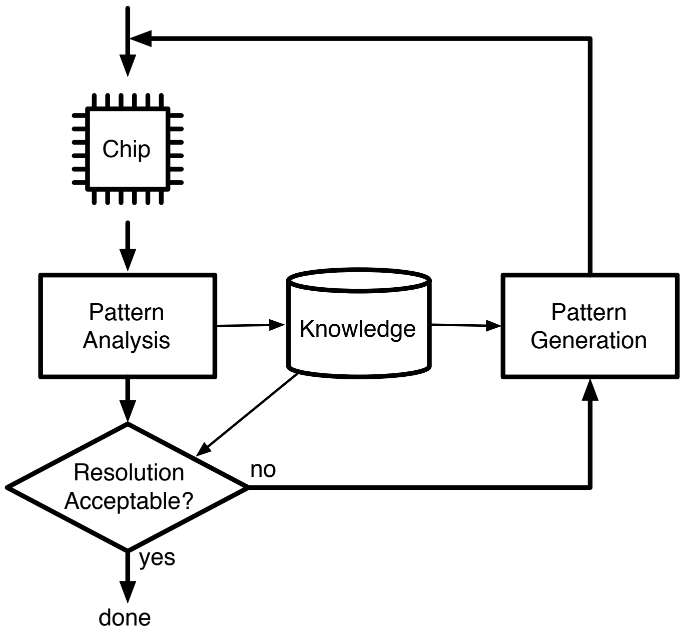


Figure 5.1.: Adaptive diagnosis flow.

CamurMP*1990, VenerCA*2004, Barte2000] as well as for more complex compound faults [WangHLL*2008].

Any diagnostic test generator can be combined with POINTER by choosing the model for the defect representatives accordingly. The following section describes a combination of POINTER with a pattern generator, which is able to generate patterns to detect a given defect representative or distinguish between two defect representatives.

5.2. Adaptive Diagnosis with POINTER

An *iteration* consists of pattern analysis and, based on its outcome, pattern generation. The number n of patterns analyzed and generated within an iteration is a trade–off between diagnostic value of each pattern and the time needed to apply the test set. If the application of a new test set to the C_{ud} involves a high constant overhead (setup time and time for response readout), iterations with larger sets are more favorable [GongC1995]. If the constant time overhead is negligible, each iteration may contain just very few patterns which are highly focused to the current analysis results.

With each response analysis, a ranked list of defect representatives are generated. Each representative is associated with evidence values as explained in chapter 4. Based on the matching condition of the best ranked defect representative, the strategy for pattern generation for the next iteration is chosen according to table 5.1. The newly generated patterns are applied to the C_{ud} and the new responses are analyzed to improve the evidences by adding the new values for σ, ι, τ, and γ to the existing ones for each defect representative. If the pattern generation strategy does not yield any new test patterns, the final result is reported.

$\sigma(f,T)$	$\iota(f,T)$	$\tau(f,T)$	$\gamma(f,T)$	Strategy
0	0	0	0	Fail Discovery
>0	0	0	0	Perfect Match Refinement
>0	>0	0	0	Perfect Tester Fail Match Refinement
>0	>0	$\geqslant 0$	$\geqslant 0$	Best Guess Refinement

Table 5.1.: Evidence forms for $e(f,T)$ and test generation strategies.

5.2.1. Fail Discovery

A top–ranked evidence of $e(f,T) = (0,0,0,0)$ implies the following situation:

- The C_{ud} did not fail any of the tests in T, because $\sigma(f,T) + \tau(f,T) = 0$.
- At least the defect representative f was not tested by T, because $\sigma(f,T) + \iota(f,T) = 0$.

This situation occurs especially at the very beginning, when no responses have been analyzed yet. Then, the evidences of all defect representatives $f \in F$ are empty: $e(f,\emptyset) = (0,0,0,0)$.

The goal of the discovery phase is to generate new patterns T', so that the C_{ud} eventually fails some tests in T'. If the C_{ud} shows at least one fail at one output for a test in T', the evidences for all defect representatives $f \in F$ will show $\sigma(f, T \cup T') + \tau(f, T \cup T') > 0$ and another test generation strategy will be used.

Until then, new tests are generated to target all defect representatives $f \in F$ with $\sigma(f, T) + \iota(f, T) = 0$.

Fail discovery may start with an existing production test set T_p. In this case, n new patterns from T_p are returned in each iteration until all patterns in T_p were analyzed. If no test pattern set is available, fail discovery starts by returning $\log(g)$ random test patterns (g is the number of gates in the C_{ud}) in the first iterations to test all random testable defect representatives. If a pattern detects a $f \in F$, its analysis will will yield an evidence with $\sigma(f, T \cup T') + \iota(f, T \cup T') > 0$.

Any untested defect representative left after applying the these patterns is targeted using a test pattern generator. If such a pattern can be generated for a $f \in F$, again its analysis will will yield an evidence with $\sigma(f, T \cup T') + \iota(f, T \cup T') > 0$. If the pattern generator fails to generate a test for a $f \in F$, this defect representative will be excluded from further consideration because it is deemed untestable and will not show $\sigma(f, T \cup T') + \iota(f, T \cup T') > 0$ for any T'.

Until some failing responses are observed, POINTER is used for fault dropping to continuously focus the pattern generation to the remaining undetected defect representatives. At the end of fail discovery, either the C_{ud} failed some tests or the test set cannot be improved any further. In both cases, the top–ranked evidence will not have the form $e(f, T \cup T') = (0, 0, 0, 0)$ anymore and adaptive diagnosis proceeds with one of the following pattern generation strategies.

5.2.2. Perfect Match Refinement

Let $f_t \in F$ be the top–ranked defect representative. The evidence of the form $e(f_t, T) = (\sigma(f_t, T), 0, 0, 0)$ with $\sigma(f_t, T) > 0$ implies the following situation:

- The C_{ud} failed at least one test in T.
- At least f is able to explain all responses of the C_{ud} completely.

Let F' be the set of all defect representatives f with the same evidence as f_t: $e(f, T) = (\sigma(f_t, T), 0, 0, 0)$. As all these defect representatives are located at the top of the ranking, this set is easily found by iterating over the top–ranked evidences. The goal of perfect match refinement is to generate tests in order to reduce the size of F'.

To reduce the size of F', tests can be used, which are able to distinguish between two defect representatives $f_a, f_b \in F'$, $f_a \neq f_b$. A test $\vec{t}$ distinguishes between f_a and f_b, if $C_{f_a}(\vec{t}) \neq C_{f_b}(\vec{t})$. If $C_{ud}(\vec{t})$ is analyzed and the unknown defect can indeed be modeled by a defect representative, the evidences will be different $e(f_a, \vec{t}) \neq e(f_b, \vec{t})$ and at least one of the defect representatives f_a, f_b will not be included in F' anymore.

Let P the set of all pairs of F':

$$P = \{\langle f_a, f_b \rangle | f_a, f_b \in F' \wedge f_a \neq f_b\}$$

To generate the test set of maximum size n for the next iteration, the pairs in P are given to a diagnostic test pattern generator. If the pattern generator returns a distinguishing test, it is added to the test set. If the pattern generator fails to generate a test, the pair is deemed undistinguishable and not considered further. All pairs given to the diagnostic pattern generator are also added to the set P'. The pairs are processed in arbitrary order until enough tests are generated for the next iteration or $P - P' = \emptyset$.

The termination of perfect match refinement is guaranteed by the maintenance of P' over all iterations so that a test is not generated twice for a specific pair. If the defect in the C_{ud} can be modelled completely by a defect representative, this refinement approach guarantees maximum diagnostic resolution.

5.2.3. Perfect Tester Fail Match Refinement

Let $f_t \in F$ be the top–ranked defect representative. The evidence of the form $e(f_t, T) = (\sigma(f_t, T), \iota(f_t, T), 0, 0)$ with $\sigma(f_t, T), \iota(f_t, T) > 0$ implies the following situation:

- The C_{ud} failed at least one test in T.
- At least f is able to explain all failing responses of the C_{ud} completely and no defect representative can explain all failing as well as passing responses of the C_{ud}.

Let F' be the set of all defect representatives f with the same evidence as f_t: $e(f, T) = (\sigma(f_t, T), \iota(f_t, T), 0, 0)$. Like in perfect match refinement, distinguishing tests are generated to reduce the size of F'. To guide pattern generation, again the set of all pairs of all $f \in F'$ is constructed:

$$P = \{\langle f_a, f_b \rangle | f_a, f_b \in F' \wedge f_a \neq f_b\}$$

and given to diagnostic test pattern generation.

There is evidence now that the defect in the C_{ud} has an additional activation condition not captured in the model for the defect representatives. Therefore, a diagnostic test pattern might not satisfy the activation conditions and fail to distinguish between the two considered representatives.

Several approaches can be used to increase the probability of successful distinction by generating patterns with additional constraints. For instance, multiple diagnostic patterns can be generated that distinguish between two defect representatives and at the same time set the neighboring signals to various logic values in order to improve the resolution for possible bridging faults. To provoke delay faults, an activation pattern can be generated to set the signal lines of f_a and f_b to values opposite to these in the distinguishing pattern. The consecutive application of the activation pattern and the distinguishing pattern ensures transitions at the suspected signal lines and possible activation of time dependent faults.

For demonstration purposes, only delay fault provoking pattern pairs will be used in perfect tester fail refinement during experimental evaluation. However, this refinement can easily be extended with more sophisticated approaches reported in literature [DesinPB2006] that also involve neighboring signal lines.

5.2.4. Best Guess Refinement

If the top–ranked evidence does not match any of the forms discussed above, best guess refinement is used. Best guess refinement follows the same strategy as perfect tester fail match refinement of distinguishing fault pairs with delay fault provoking pattern pairs. The difference lies in the considered set of fault pairs P.

In the case at hand, it is possible that the C_{ud} contains multiple interacting fault sites and not all defect representatives related to one of these sites are at the very top of the ranking. To refine the ranking as best as possible in this case, the m best ranked defect representatives with $\sigma > 0$ are considered:

$$F' = \{f \in F \mid \text{rank}_T(f) <= m \wedge \sigma(f, T) > 0\}$$

And pairs are formed between all $f \in F'$ with equal evidences:

$$P = \{\langle f_a, f_b \rangle | f_a, f_b \in F' \wedge f_a \neq f_b \wedge e(f_a, T) = e(f_b, T)\}$$

The elements of P are given to a diagnostic test pattern generator and added to the set P' like in the previously discussed strategies. The process terminates as soon as $P - P' = \emptyset$.

The selection of m allows a trade–off between runtime and refinement scope. A low m only refines the very top of the ranking and may miss a defect representative corresponding well to the unknown defect in the C_{ud} but ranked lower in early iterations. A high m considers many well ranked candidates, but may increase diagnosis runtime and the number of iterations until the final result considerably.

5.3. Summary

Adaptive diagnosis provides a way for high precision diagnosis without restricting fault assumptions. Based on partial diagnosis results, new diagnostic test patterns are generated to gradually refine them. This allows for fault model independent and very focused diagnostic testing of the suspicious circuit region.

A POINTER–based adaptive diagnosis approach was built by combining the response analysis method with a diagnostic test pattern generator which is able to generate fault distinguishing tests for defect representatives. The algorithm may start from scratch without any previous knowledge, or just refine a previous result (e.g., from production data or in–field data) further with a few focused tests. A simple scan over the sorted list of evidences identifies defect representatives, which have not been distinguished so far by the applied tests. Based on this information and the matching condition of the top–ranked evidence, a couple of patterns are generated in order to refine the ranking. This method yields the best possible diagnosis result in the *perfect match* case, and significant improvements are expected even if the defect in the C_{ud} is not perfectly modeled by a representative.

6. Production Diagnosis

Integrating diagnosis into the production test flow is very challenging, yet it is necessary for effective yield learning. The main challenges for implementing logic diagnosis in a production environment are:

- The high cost sensitivity of the production test. Any additional test time spend for diagnosis purposes adds cost.
- The storage of the diagnostic data gathered from the chip in production testers.
- The fail data from a large amount of chips have to be analyzed, which puts strict constraints on the computing time spend per failing chip.
- Production test patterns are fixed and optimized for pattern count rather than diagnostic resolution to reduce test time.

The first two challenges are typically addressed by the so–called stop–on–n^{th}–fail strategy in which a test of a logic block is stopped after the first n failing responses. In standard (non–diagnostic) production test of single chips at a time, the test can be aborted after observing the first failing response ($n = 1$) to increase the chip throughput per ATE. Another common way to increase the throughput is to test multiple chips at a time on the same tester in a *multi–site test* [VolkeKR*2002] (see figure 6.1). The test data is distributed to multiple chips and their responses are evaluated in parallel. Stopping the test for failing chips prematurely provides no throughput benefit in this case because the chips can only be exchanged after the completion of the test for all the chips on the tester. The benefit lies in the reduction of fail memory needed per chip. For standard production test, it is highly sufficient to store just the first failing response for each pass/fail decision. This reduced memory requirement per chip enables to test more chips in parallel on the same tester and reduce the test cost per chip.

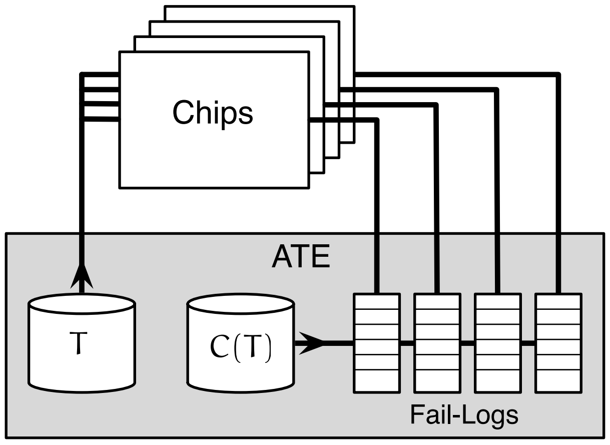


Figure 6.1.: Multi–site test setup.

For diagnostic production test, n can be set to a value, which provides the best trade–off between the additional test cost and the diagnostic data gathered per failing chip. The experimental evaluation will show that even a rather low number of $n = 8$ already provides near–optimal diagnostic resolution.

As test patterns are optimized towards low pattern count rather for high diagnostic resolution, it is even more important to extract as much diagnostic information as possible from all responses observed during test. At the same time, diagnosis must provide short and predictable runtimes in order to reduce the compute power needed for analysis of all the failing chip from a production line.

This chapter will present an extension to POINTER, which combines it with a small dictionary and a structural pruning approach for enhanced runtime performance without compromising on diagnosis quality.

6.1. Diagnostic Production Test

In standard production test for digital logic, the tester applies patterns to the C_{ud} and compares its responses to the expected ones. The expected responses are stored in the tester, too. If a response of the C_{ud} differs from the expected one, the test stops and the tester reports a fail. If all responses are correct, the test is continued until the end and the tester reports a pass.

To implement a stop–on–n^{th}–fail strategy for diagnostic production test, two modifications are necessary. First, not only pass/fail information needs to be reported from by the tester, but also the test pattern number and the incorrect response from the C_{ud} for this pattern. Second, the test must continue until n failing responses were observed or the test is finished. The outcome of such a diagnostic production test is a so–called *fail–log*, which is a list of at most n rows, each row containing the test pattern number of a failing response and the response itself.

The additional cost for enabling a diagnostic production test has two components. First, as failing responses are recorded instead of directly reporting a failing chip, more time is spend on faulty devices than during basic production test. Second, the failing responses need to be stored in the tester memory and transferred to the computing cluster performing the analysis.

The memory requirements and communication bandwidth needed is proportional to n. Also the additional tester time increases with larger n. Thus, all cost components can be controlled by setting this parameter to an appropriate value. Of course, diagnosis results will be of lower quality if n is chosen too low as not enough diagnostic information is available.

6.2. Passing Pattern Analysis using ι–Dictionary

The information in the fail–log from a failing chip implies the knowledge about some passing patterns as well. Let i be the test pattern index of the last record in a fail–log. Then, the C_{ud} passed all tests $\vec{t}_0, \dots, \vec{t}_{i-1} \in T$ except those present in the records of the fail–log. An example is shown in figure 6.2. The standard POINTER algorithm would analyze all i patterns one after another and summing up the evidences for each defect representative.

The analysis of a passing pattern $\vec{t} \in T$ only contributes to $\iota(f, \vec{t})$ of some defect representatives $f \in F$. All other components of the evidences are zero for this pattern as the C_{ud} has no failing outputs. The test pattern set is the same for all C_{ud} under consideration, so instead of simulating all defects representatives for the passing patterns in all failing chips, it is reasonable to store the values for ι in memory and re–use it for analysing each fail–log.

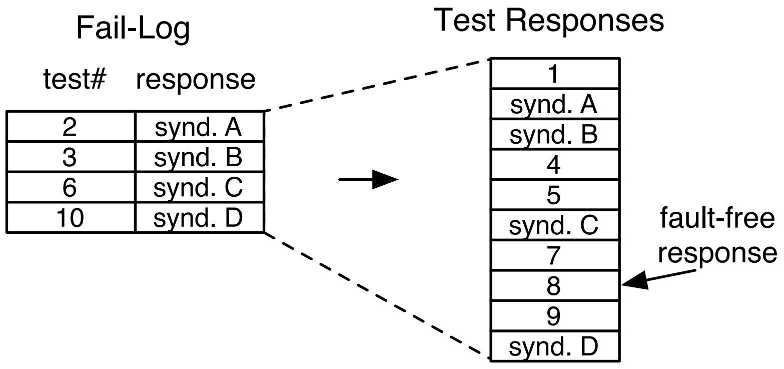


Figure 6.2.: Expansion of a fail–log into a set of test responses.

Let T_i denote the set of the first i tests of T $(i \leqslant |T|)$:

$$\{\vec{t}_0, \ldots, \vec{t}_{i-1}\} = T_i \subseteq T$$

The ι*–dictionary* contains for each defect representative $f \in F$ and test pattern index $0 < i \leqslant |T|$ the value $\iota(f, T_i)$. For convenience, we define further $\iota(f, T_0) = 0$. The number of values stored in the dictionary is $|F| \cdot |T|$ and can be handled by modern computing systems even with industrial–sized designs and naive encoding. Many techniques proposed for the compact storage of fault dictionaries are also applicable to the ι–dictionary proposed here to reduce its memory footprint.

The basic POINTER algorithm is modified in the following way for incorporating the ι–dictionary. Instead of iterating over each known response of the C_{ud}, now POINTER iterates just over all *failing* responses. Let i be the pattern index for the current failing response and i' the test pattern index of the previously analyzed failing response. If POINTER currently analyses the first failing response, we define $i' = -1$. For each failing response, the evidence for a $f \in F$ is:

$$e(f, \{\vec{t}_{i'+1}, \ldots, \vec{t}_i\}) = (\sigma(f, \vec{t}_i), \iota(f, \vec{t}_i) + \iota(f, T_{i-1}) - \iota(f, T_{i'+1}), \tau(f, \vec{t}_i), \gamma(f, \vec{t}_i))$$

Two accesses to the ι–dictionary are involved in order to incorporate the difference of the ι–values generated by the passing patterns between the previous failing pattern and the current failing pattern under consideration. The evidence calculated in this way is equivalent to the sum of evidences between i' and i explicitly computed in the basic approach:

$$e(f, \{\vec{t}_{i'+1}, \ldots, \vec{t}_i\}) = \sum_{\vec{t} \in \{\vec{t}_{i'+1}, \ldots, \vec{t}_i\}} e(f, \vec{t})$$

So the use of the ι–dictionary lead to the same diagnosis results as the basic POINTER algorithm, but provides a runtime proportional to the number of failing patterns. In particular, the runtime of this approach is independent from the distribution of the fails within the test. The basic algorithm needs more time if a failing response was recorded near the end of the complete test, compared with a fail–log containing only failures very early in the test. By using the ι–dictionary, the diagnosis runtime per fail–log becomes very predictable and can be tightly controlled by n, the maximum number of fails in a fail–log.

6.3. Structural Pruning

Structural pruning is a standard technique in simulation based diagnosis to speed up pattern analysis. The idea is to consider only these circuit parts during analysis of a failing test $\vec{t}$, which have at least one structural path to an output, that failed in the C_{ud}. This can speed up diagnosis especially for current industrial designs which tend to have short paths and rather small cones in the combinational logic.

Implementing structural pruning in POINTER just by skipping defect representatives, which do not have a structural path to a failing output, would affect the evidences and lead to lower diagnostic resolution. A simple example is the analysis of a passing pattern with structural pruning enabled. In this case, no evidence would be changed as there is no defect representative in the circuit having a structural path to a failing output and the passing patterns are simply ignored by the diagnosis algorithm.

In the general case, two components of the evidences are affected by pruning: ι and τ. The following approach generates these two values for evidences of defect representatives skipped by structural pruning without simulation. This way, structural pruning can be implemented to speed up diagnosis significantly and without compromising on the diagnosis quality.

Let $\vec{t_i} \in T$ be a test and let o be the number of failing bits in the response $C_{ud}(\vec{t_i})$. Let further $F_s(\vec{t_i}) \subseteq F$ be the set of defect representatives having at least one structural path to one of the failing outputs of the C_{ud} under test $\vec{t_i}$ (see also figure 6.3). The evidences for the set of defect representatives $F_p(\vec{t_i}) =$

$F - F_s(\vec{t}_i)$ are not determined by simulation, but just set as follows. For each $f_p \in F_p(\vec{t}_i)$:

$$e(f_p, \vec{t}_i) = (0, \iota(f, T_i) - \iota(f, T_{i-1}), o, 0).$$

A defect representative $f_p \in F_p(\vec{t}_i)$ cannot explain any failing bit in $C_{ud}(\vec{t}_i)$ because it lacks a structural path to any of the failing outputs. Therefore, the values $\sigma(f_p, \vec{t}_i)$ and $\gamma(f_p, \vec{t}_i)$ are both zero and all failing bits of f_p will contribute to $\iota(f_p, \vec{t}_i)$. This value can be calculated from the data in the ι–dictionary defined in the previous section. As a direct consequence of $\sigma(f_p, \vec{t}_i) = 0$, the number of unexplained failing bits is the same for all $f_p \in F_p(\vec{t}_i)$: $\tau(f_p, \vec{t}_i) = o$. The remaining evidences $f_s \in F_s(\vec{t}_i)$ are determined by simulation.

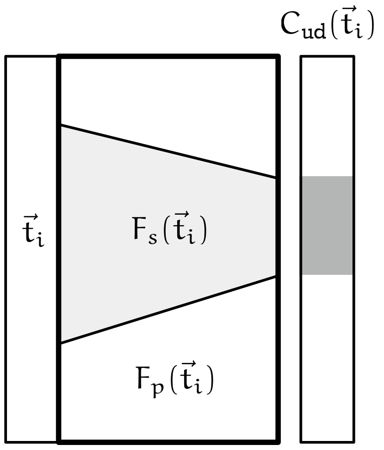


Figure 6.3.: Structural pruning for a failing response to the test pattern $\vec{t}_i$.

Structural pruning can be combined with the efficient passing pattern consideration described in the previous section in the following way. Let $t_i \in T$ now be a pattern with $o > 0$ failing outputs in $C_{ud}(\vec{t}_i)$ and i' the number of the previous failing pattern or, otherwise -1. For all defect representatives $f_s \in F_s(\vec{t}_i)$ the evidences are calculated as before:

$$e(f_s, \{\vec{t}_{i'+1}, \ldots, \vec{t}_i\}) = (\sigma(f, \vec{t}_i), \iota(f, \vec{t}_i) + \iota(f, T_{i-1}) - \iota(f, T_{i'+1}), \tau(f, \vec{t}_i), \gamma(f, \vec{t}_i))$$

All defect representatives in the pruned circuit part $f_p \in F_p(\vec{t}_i)$ cannot explain any of the o failing bits and the mispredictions can be calculated from the dictionary: $\iota(f, \vec{t}_i) = \iota(f, T_i) - \iota(f, T_{i-1})$. The evidence is therefore:

$$e(f_p, \{\vec{t}_{i'+1}, \ldots, \vec{t}_i\}) = (0, \iota(f, T_i) - \iota(f, T_{i'+1}), o, 0).$$

6.4. Summary

In production test, the test set is fixed and only the first n failing responses are recorded per chip for diagnosis purposes. The value of n controls the additional costs like increased test time and additional memory requirements, and is usually set as low as possible.

With the first n fails, also passing responses are known and the standard POINTER algorithm can already exploit this information. This advantage above other algorithms, which are only using failing responses for diagnosis, can be translated into a lower n for the same quality of diagnosis results.

The fixed test set allows the use of an ι–dictionary containing the number of failing response bits for each test and defect representative. This dictionary enables structural pruning and highly efficient passing pattern consideration in POINTER. Structural pruning greatly reduces the runtime for each failing response as fault simulation is not needed for circuit parts with no structural path to a failing output. With dictionary–based passing pattern consideration, the runtime becomes proportional to the number n of failing patterns. Now, the parameter n also controls the computing resources needed for logic diagnosis in high volume testing. The evidences calculated and the diagnosis results generated are still identical to these of the original POINTER algorithm.

7. Diagnosis and Extreme Space Compaction

Embedded testing including *logic built–in self–test* (LBIST) and *multi–site testing* are quite effective test cost reduction techniques. Compared to standard production test, more sophisticated on–chip test infrastructure is used to reduce the communication to external test equipment to a minimum. This reduces the test resources needed in the ATE and reduces test time by cutting the communication overhead over a slow, external test interface.

Two compaction principles are used in this context: *space compaction* and *time compaction*. A space compactor calculates a signature for each circuit response individually and independently. Typical examples for space compactors are parity trees [VrankGG*2006] or X–Compact [MitraK2004]. Time Compactors like *multiple–input shift registers* (MISR) [BardeM1982] on the other hand use multiple or even all responses of the C_{ud} for the calculation of a single signature. Time compactors can provide higher compaction ratios than space compactors but multiple test runs may be necessary in order to diagnose a failing signature as already discussed in section 1.2.2. To avoid the increased test time and the more complex test control necessary for the diagnostic test, a viable option might be the use of a space compactor paired with the direct diagnosis approach. Given an on–chip storage of s bits for diagnostic data, a very interesting question is how to use the available space most efficiently to maximize the quality of the diagnosis results obtained from these s bits.

This chapter will approach this question by introducing a very aggressive response compaction combined with an extended version of the POINTER algorithm. The experimental results will show that compacting responses aggressively can be more beneficial for diagnosis than storing less compacted data, given a limited storage of s bits. First, the approaches for enabling diagnosis in multi–site testing and BIST environments are discussed in more detail. Then, *extreme space compaction* is introduced as a technique for using the limited on–chip resources most efficiently in both embedded testing approaches. Finally, the modifications to POINTER are presented to enable the analysis of extremely compacted response data.

7.1. Diagnostic Embedded Testing

In multi–site testing, many chips are tested in parallel by the same ATE [VolkeKR*2002]. All chips receive identical input by the ATE, but the output side of the tested chips cannot be handled in the same straightforward way, as the defective chips will respond in many different, unpredictable ways. One solution is feeding all the dies with the correct output by the ATE, equipping them with an on–chip comparator and comparing the expected and computed response on chip (figure 7.1).

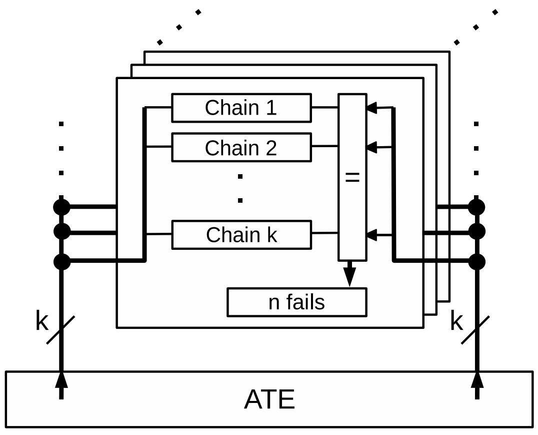


Figure 7.1.: Principle of multi–site testing with on–chip fail–logs.

In standard test, it would be sufficient for the chip to signal the ATE a fail after observing the first mismatch in the responses. To enable diagnosis, a small on–chip memory may be used to store information on the first n failing scan slices [PoehlRB*2006, KinsmON2006]. In contrast to the multi–site test approach discussed in chapter 6, the fail logs are now located on the chips themselves and the memory and communication requirements for the ATE are reduced further. After the parallel test of all chips, this data can be unloaded die by die and passed on as fail–logs to a logic diagnosis algorithm.

As with standard scan testing, throughput requirements and test application time are reduced, if each chip is equipped with test data decompression and test response compaction logic (figure 7.2) [PoehlRB*2006]. In addition, a higher number of fails n can be stored on the chip in a fail memory of the same size s.

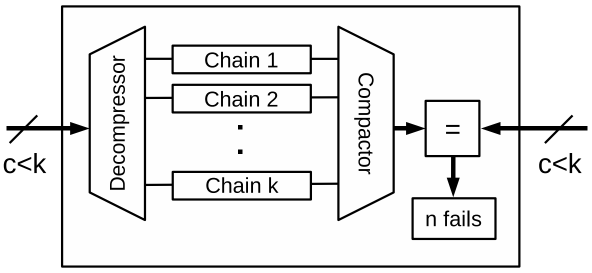


Figure 7.2.: Test data decompression and response compaction.

The standard approach for enabling diagnosis in BIST structures discussed so far is bypassing. Using bypassing in diagnostic production test would mean that each chip failing the BIST is re–tested with an external test for gathering diagnostic data. This would increase test cost dramatically especially if the yield is low and many failing chips needed to be diagnosed. An alternative without this disadvantage can be provided by extreme space compaction.

If the compaction ratio is sufficiently high, the same idea for diagnostic multi–site test can also be applied to BIST. Here, the correct responses are not provided externally, but must be stored internally as shown in figure 7.3. The memory requirement for the response memory decreases by the ratio of compaction used.

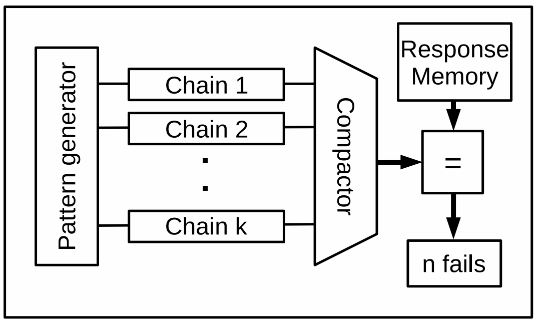


Figure 7.3.: Built–in self–diagnosis.

7.2. Extreme Space Compaction

The response compaction ratio determines the memory requirements for built–in diagnosis, and the bandwidth for multi–site testing. Given a budget s for the maximum size of fail data, implementing a stop–on–n^{th}–fail strategy is the natural choice.

While time compactors and convolutional compactors are highly effective in general [RajskTW*2005, LeiniGM2004, Touba2007], they are not the optimal choice for implementing a stop–on–n^{th}–fail strategy. After the occurrence of a faulty response for a single pattern, the signature gets contaminated over multiple or even all clock cycles filling up the fail memory with data not providing any additional diagnostic information. Space compaction techniques are more appropriate in the given context [LeiniMC*2005, MitraK2004]. The combinational compactor with highest compaction ratio is the parity tree. It was shown in [VrankGG*2006] that the amount of aliasing and masking between stuck–at faults introduced by a parity tree is negligible and fault coverage as well as diagnostic resolution for stuck–at faults are only marginally affected.

Extreme space compaction [HolstW2009, HolstW2009a] is provided by increasing the number of scan chains and hence the scan slice length, and by compacting complete scan slices into single parity bits. A *scan slice* is the set of bits that is shifted out of the scan chains in a single shift cycle. Figure 7.4 shows the scheme, which maps a complete scan slice into a single bit and provides the highest space compaction ratio possible.

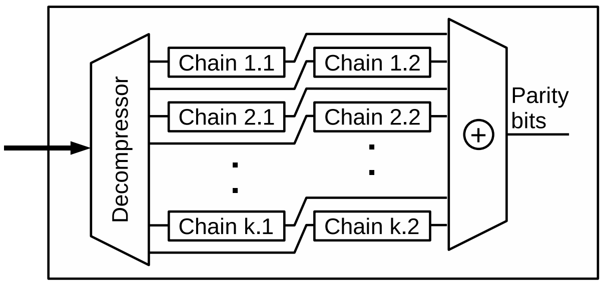


Figure 7.4.: Extreme response compaction.

Extreme space compaction greatly reduces test time and response data volume and can be applied to both test cost reduction approaches. Figure 7.5 shows the extreme compaction scheme applied to multi–site testing and embedded test.

Here, merely the expected parity bits are either sent by the ATE or stored on–chip. This significantly reduces the amount of on–chip storage and the ATE bandwidth requirements. The comparator is now a single XOR–gate controlling a memory which records the n first scan clock cycles for which the parity bits mismatch.

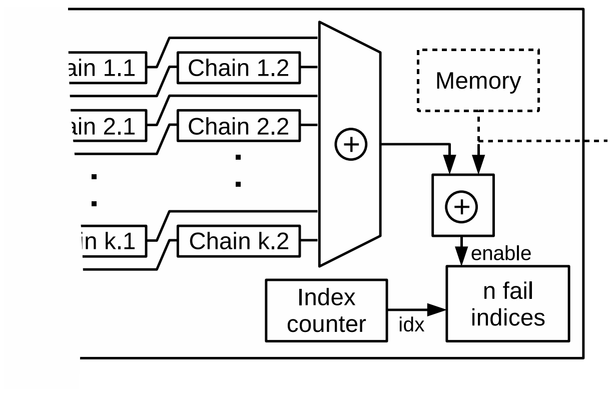


Figure 7.5.: Target scheme for extreme response compaction.

Let k' be the number of scan chains and l' the maximum length on a chain in the original design. Let k and l the number of scan chains and their maximum length after splitting them for extreme compaction. Given that the same number of patterns p are applied in both cases. Test time is reduced by a factor of nearly l/l', because less scan cycles are needed for loading patterns and reading out responses. The total response data to be distributed to the chips in multi–site test or to be stored on chip for BIST is reduced by a factor of $k' \cdot l'/l$.

In the fail memory, only the scan cycles numbers with mismatching parity bits need to be stored. This information from the fail memory together with the known, expected parities is sufficient to obtain the parity bits produced by the C_{ud} for diagnosis. Given a straight–forward binary encoding of the failing scan cycles, the correspondence between the number of fails n and the fail memory size s in bits is:

$$s = n \cdot \log_2(p \cdot l)$$

Besides the very high compaction ratio, the main advantage of extreme space compaction over the compactors proposed in the literature is that, in combination with a variant of POINTER, the diagnostic success for arbitrary defects can be maintained. In the next sections, the necessary extensions to the basic POINTER algorithm are presented. The experimental results in chapter 8 will show that the diagnosis results are only marginally affected by the extreme compaction.

7.3. Combinational Representation

For extreme space compaction, only the direct diagnosis approach is applicable. Parity information is insufficient to reconstruct the scan cell failures and make the indirect diagnosis approach impossible to implement. Bypassing based diagnosis defies the purpose of this approach tuned to production test. However, it may be implemented also in this infrastructure in order to assist high–precision diagnosis in lab environments.

For direct diagnosis as well as for pattern generation, the combinational equivalent of a circuit with extreme compaction is constructed as shown in figure 7.6. Let k be the number of scan chains, and l the maximum length of a scan chain after splitting the default scan configuration. All flip–flops which are at the same positions in the scan chains are compressed into a single parity. Hence the combinational representation contains l parity trees each compacting at most k pseudo–primary outputs.

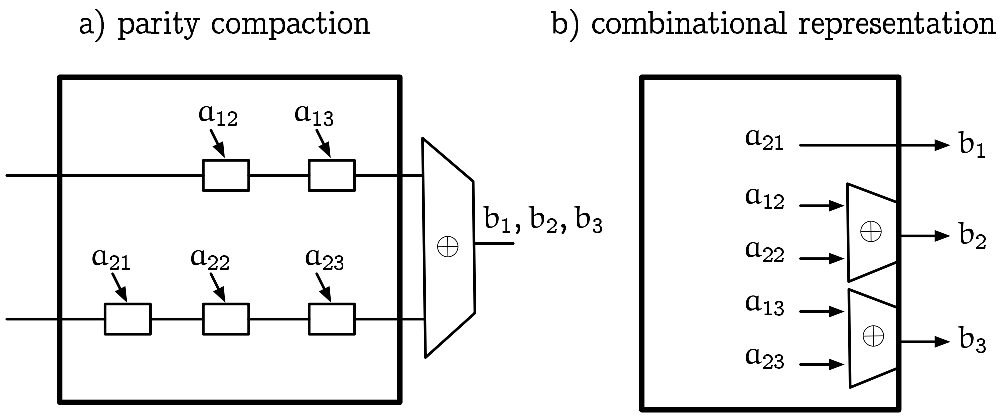


Figure 7.6.: Combinational representation for extreme response compaction, example with $k = 2$ and $l = 3$.

The circuit 7.6b has a higher global reconvergent fanout in general, a significantly higher fan–in, and is more difficult to test. But the approach rarely introduces new redundancies which would lead to fault masking, and the negative impact can be reduced by an appropriate scan chain organization [ElmW2008]. However the unusually high fan–in of compactor structures poses a great challenge especially for fault model independent diagnosis approaches. The effect–cause approaches relying on cone intersections or path tracing may loose effectiveness. Since the input cone of a failing parity bit spans all flip–flops in a scan slice, a single failing parity bit can be caused by a much larger number of possible defect candidates than a single failing bit in the uncompacted response. All paths in a parity tree are sensitized regardless of the inputs, so back–tracing also need to consider all flip–flops in a scan slice with an incorrect parity bit. For inject–and–validate based algorithms however, the impact of extreme compaction is minimal. Although there may be a slowdown because structural pruning is not as effective, it will be shown, that the loss in diagnostic resolution is only marginal.

7.4. Response Parity Analysis

From the fail memory, the failing scan cycles are known. This information is translated into the stream of the first $m = n + p$ parity bits (denoted as P_{ud}) of the response with p being the number of passing scan slices before the n^{th} failing one.

The main difference to the diagnosis applications discussed so far is, that m may not be a multiple of l, i.e., the parity bits contain a partial response from the C_{ud}. One possibility is to discard partial responses and apply POINTER in the same way as discussed in the previous chapters using the combinational equivalent for extreme compaction. This reduces the quality of the diagnosis results because not all available information is used. If the first failing pattern contain more than n failing scan slices, diagnosis would even lead to no result at all.

Incorporating partial responses in the analysis need some special precautions described in the following. Figure 7.7 depicts the principle for obtaining the components for the evidence $e(f, \vec{t})$ for a test $\vec{t}$ with a partially known response $C_{ud}(\vec{t})$ and a defect representative f. Only the known response bits are compared to the ones from $C_f(\vec{t})$ and contribute to the evidence. The remaining response bits from $C_f(\vec{t})$ must be discarded because the information to which components of the evidence they contribute to is missing.

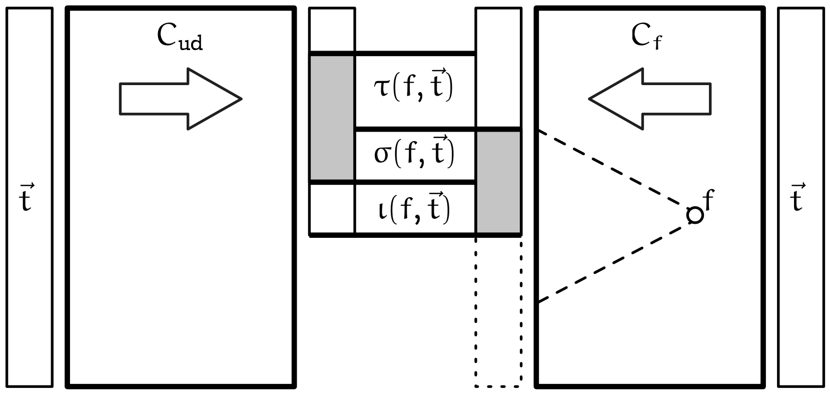


Figure 7.7.: Evidence calculation for partial responses.

The properties of evidences as well as the ranking and match conditions are still valid with the modified calculation. For the evidence components σ, ι and τ, this follows from the associativity of addition. Regardless in what granularity the evidences are summed up (e.g. response–wise or output–wise), the resulting values don't change and the properties hold in every partial sum. This fact was also used in adaptive diagnosis, where additional patterns are generated and new evidences were added to the existing ones. The value of $\gamma(f,\vec{t})$ for the partial response is always less or equal to $\gamma(f,\vec{t})$ generated for the full response. This means that POINTER will reduce the matching condition for a defect representative to *best guess* only in cases, where a partial matching within a single response is evident, and all potential candidates for better matching conditions will maintain $\gamma = 0$.

7.5. ι–Dictionary and Pruning

The runtime optimization methods based on ι–dictionary and structural pruning introduced in the previous chapter is in principle also applicable for the analysis of partial responses. However, they have to be generalized in order to still provide valid evidences in this case.

Consider a defect representative f propagating to known as well as unknown outputs like in figure 7.7. The ι–dictionary cannot be used for processing a passing partial response in this case, because the actual $\iota(f,\vec{t})$ may be lower than the value in the dictionary. Let i be the pattern index of the current failing or partially known response $C_{ud}(\vec{t}_i)$ and i' the index of the previous failing or partially known response $C_{ud}(\vec{t}_{i'})$. For the patterns $\vec{t}_{i'+1}, \ldots, \vec{t}_{i-1} \in T$, completely known, passing responses of the C_{ud} are available. Let further $o \neq 0$ the number of failing bits in the current response $C_{ud}(\vec{t}_i)$. Based on the structural output cone starting at a defect representative f its evidence $e(f, \vec{t}_i)$ is calculated in the following way:

- If f can only propagate to known outputs, which are correct in $C_{ud}(\vec{t}_i)$, structural pruning is valid and the evidence is:

$$e(f, \{\vec{t}_{i'+1}, \ldots, \vec{t}_i\}) = (0, \iota(f, T_i) - \iota(f, T_{i'+1}), o, 0)$$

- If f can only propagate to unknown outputs, $\iota(f, \{\vec{t}_{i'+1}, \ldots, \vec{t}_i\})$ is zero and must not be read from the dictionary:

$$e(f, \{\vec{t}_{i'+1}, \ldots, \vec{t}_i\}) = (0, 0, o, 0)$$

- All other cases (f may propagate to at least one failing output, or may propagate to known and unknown outputs at the same time) require explicit simulation and the evidence is:

$$e(f, \{\vec{t}_{i'+1}, \ldots, \vec{t}_i\}) = (\sigma(f, \vec{t}_i), \iota(f, \vec{t}_i) + \iota(f, T_{i-1}) - \iota(f, T_{i'+1}), \tau(f, \vec{t}_i), \gamma(f, \vec{t}_i))$$

The evidences calculated in this way are still equivalent to the ones obtained by explicit simulation in every case. Only in cases where a defect representative may propagate to known passing outputs and unknown outputs at the same time, additional simulations are required. For the given application, the performance decrease is negligible, as there is only one partial pattern possible (the last one) and the known outputs in this partial patterns always contain failing bits.

7.6. Summary

Extreme space compaction uses a broad scan configuration and compacts complete scan slices into single parity bits. Direct diagnosis is the only viable option for analyzing the extremely compacted responses. The high fan–in of the compactor structure poses a great challenge to fault model independent diagnosis algorithms. POINTER as an inject–and–validate approach is able to handle such extremely compact diagnostic data without performance degradation. The proposed extensions enable POINTER to extract maximum diagnostic information out of partial responses. This is necessary, because in combination with a stop–on–n^{th}–fail strategy, extreme response compaction may lead to responses, which are not completely known.

The diagnostic fail data generated after extreme space compaction is small enough to fit into on–chip memories. This enables highly efficient diagnostic multi–site testing setups as well as diagnostic in–field tests.

8. Experimental Evaluation

The goal of the experimental evaluation is to show the effectiveness of the proposed diagnosis approaches in terms of runtime performance and the quality of the diagnosis results. Most experiments follow the general principle of simulating the application environments for the logic diagnosis approaches as closely as possible in all important aspects. Within the setups, large numbers of Monte–Carlo experiments with randomly injected defects are performed to obtain the performance values, which can be expected in actual deployments of these approaches.

The overall experimental setup is depicted in figure 8.1. The fail data is generated by a $\mathrm{C_{ud}}$–*simulator*, which simulates the combinational equivalent of a circuit under consideration containing a defect. The defects are chosen randomly from a set of defects generated on the basis of a selected *defect model*. The *test patterns* are either production test patterns generated by a commercial ATPG or provided by the adaptive diagnosis approach. The *logic diagnosis* determines from the test patterns and the fail data the evidences for the set of *defect representatives* generated on the basis of a selected *defect representative model*. The *diagnosis result*s are ranked lists of defect representatives with their respective evidence values. Before the *result evaluation*, the injected defects have to be translated into identifying defect representatives. This translation will be further detailed together with the used defect models in section 8.1. Using the identifying defect representatives of the injected defect, the diagnosis result is evaluated with various metrics introduced in section 8.2. After the introduction of the used benchmark circuits in section 8.3, the conducted experiments are presented and discussed.

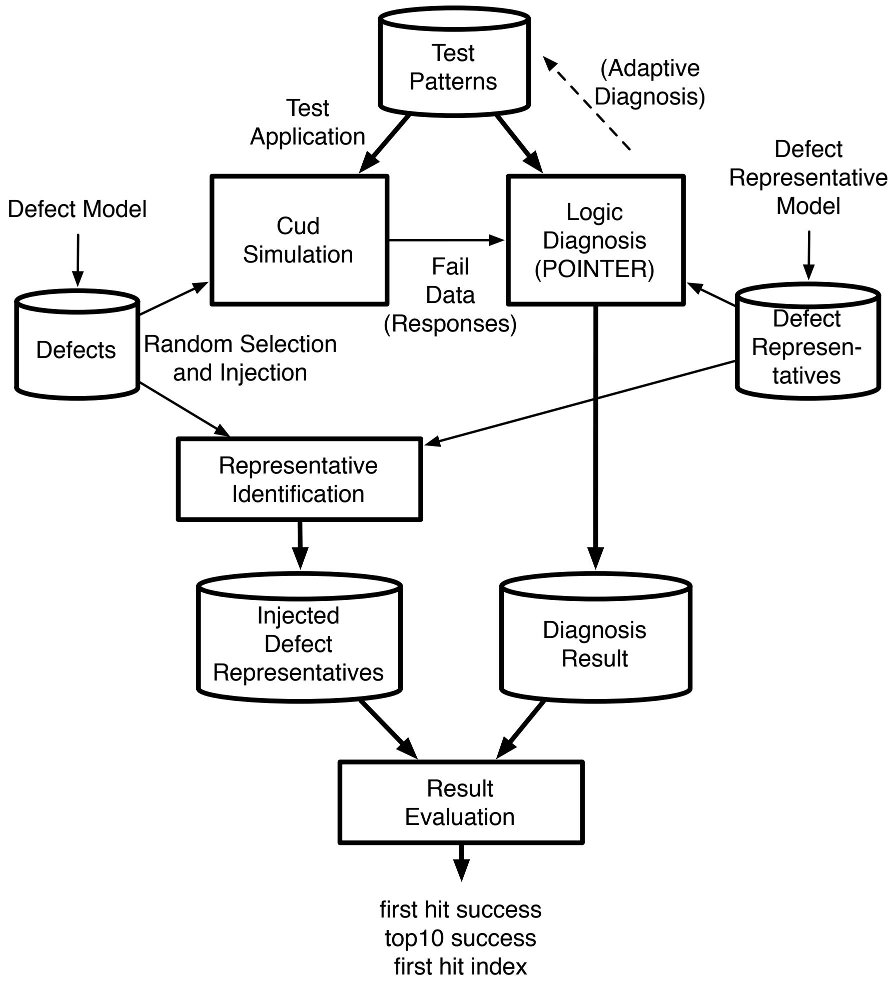


Figure 8.1.: Overall experimental setup.

8.1. Defect Models

Various models for defect representatives are applicable to the POINTER approach. POINTER is then able to locate arbitrary defects in the C_{ud} based on the assumption that the error propagation in the surrounding logic is sufficiently modelled in the fault simulation. To demonstrate this capability in the experiments, the stuck–at fault model is chosen for generating the set of defect representatives, and a zero delay model is used for circuit simulation. By this model selection, POINTER cannot effectively diagnose small delay faults for instance, because the simulation of their fault effect propagation need to incorporate the timing of the different paths. Diagnosis of such timing related defects is possible by choosing the appropriate model for fault simulation, but again, the goal here is to assess the matching capabilities of POINTER under a sufficient error propagation model. This goal is achieved by choosing the model for defect representatives and the injected defects accordingly.

The following fault models are used for generating the C_{ud} instances in the Monte–Carlo experiments. They challenge the matching approach in various ways.

- Single stuck–at fault

  The experiments with this fault model represent all diagnosis cases, in which a defect representative in fault simulation is able to completely model the defect in the C_{ud}. This fault model is used to validate that POINTER diagnosis will always report *perfect match* candidates in these cases and to assess the performance of POINTER in the best case.

  The faults used for generating the C_{ud} are randomly sampled from the structurally collapsed set of all stuck–at faults in the circuit.

  The identifying defect representative is, of course, the corresponding stuck–at fault in the fault simulator.

- Single gross delay fault

  The experiments with gross delay faults represent the diagnosis cases, in which the defect in the C_{ud} is partially modeled by a defect representative in fault simulation. It is reasonable to expect that diagnosis results obtained here are comparable to diagnoses on C_{ud}s containing defects, which can be described by the CLF of a defect representative in combination with an additional activation condition. This includes a large range of defects like bridges with a single victim signal or crosstalk faults.

  The faults used for generating the C_{ud} are randomly sampled from the structurally collapsed set of all stuck–at faults in the circuit. The stuck–at faults sampled in this way are used to determine the location and the polarity of the gross delay faults to be injected.

  The identifying defect representative is again the corresponding stuck–at fault in the fault simulator.

- Single swap bridge fault

  This fault model swaps the drivers of two randomly chosen signal lines within the circuit. Corresponding to the bridge fault models presented in section 3.1.2, such a bridge between signal lines v and w can be described by the following tuple of two CLFs: $v \oplus [v \oplus w]$, $w \oplus [v \oplus w]$. If the bridge is excited (v and w have different values), both line flips are active and possibly propagate to the outputs. This represents the most challenging defect type in these experiments, because in most cases, the behavior of the C_{ud} can not be explained by any single defect representative in fault simulation.

  Four defect representatives identify a swap bridge between v and w: $v \oplus [v]$, $v \oplus [\overline{v}]$, $w \oplus [w]$, $w \oplus [\overline{w}]$. The identifying defect representative among them with the best rank determines the diagnosis success as well as the estimated number of PFA attempts.

  A C_{ud} with a swap bridge is generated by the random selection of two distinct signal lines within the circuit. The calculation of the syndrome is performed in three steps. First a good simulation is performed, then the faulty values at the two victim lines are calculated based on the good circuit state. If there are any faulty values, they are propagated as far as possible towards the outputs in the last step.

8.2. Quality and Performance Metrics

To measure the quality of a diagnosis result, the best ranked evidence corresponding to the injected defect is determined. Several metrics are used to report the performance of POINTER.

- Top1 success, top1 hit, first hit success

  A diagnosis of a defect is counted as *top1 success*, if and only if the top–ranked evidence corresponds to the defect and no non–identifying evidence has the same confidence. If the resulting candidate ranking is not perfect in this sense, the diagnosis failed. This rigid definition of diagnostic success is suitable for benchmarking the algorithm because the resulting figures form a lower bound to the success rates expected in different applications.

Depending on the diagnosis environment, additional candidates from top of the ranking can be considered. This consideration may only improve the success rates.

- Top10 success, top10 hit

  A diagnosis of a defect is a *top10 success* if an identifying defect representative is among the ten best ranked candidates. The number of ten is somewhat arbitrarily chosen, but is generally believed that no more than ten failure analysis attempts will be performed on a single chip.

- PFA attempts

  The second parameter which is important for any diagnosis approach is the average number of low level investigations that have to be invoked. One defect representative corresponds to one PFA attempt. If diagnosis is successful (top10 success), this number is the rank, otherwise it is 10.

- Suspect count

  This is the total number of suspects reported by diagnosis.

All numbers reported are averaged over multiple defect injection experiments giving the success probability, the average number of PFA attempts and the average suspect count.

To express the relative runtime performance between different designs and simulation acceleration approaches, a metric called *million evidences per second* (MEPS) is used. This is the number of patterns that can be analyzed in a second with one Million evidences considered in the circuit. One MEPS corresponds therefore to the performance for evaluating 1000 patterns for 1000 evidences in one second. All MEPS values reported in this thesis were measured on the same hardware so that they are comparable to each other.

8.3. Benchmark Circuits

To obtain statistically relevant data, diagnosis must be performed on a wide range of different logic circuits. This work will use the public benchmark circuits sets ISCAS'89 [BrgleBK1989] and ITC'99 [David1999, CornoRS2000] as well as industrial designs provided by NXP which are named NXP'08 in here.

All circuits are synthesized into flat combinational equivalents of full–scan designs containing only the two–input gates AND, NAND, OR, NOR, XOR, XNOR and the one–input gates NOT and BUF. From the ISCAS'89 and ITC'99 benchmark sets only circuits with 10000 logic gates or more are considered. For the NXP'08 circuits, also the original scan chain configurations are available. Tables A.1 and A.2 show various statistics on the designs like gate counts, structurally longest paths, the number of inputs and outputs as well as the original scan chain configuration where available. Production test sets targeting stuck–at faults in these benchmarks were generated using a commercial ATPG. The number of test patterns and the corresponding stuck–at fault coverage are reported in columns p' and *fc*. ISCAS'89 circuit names start with "s", ITC'99 circuits start with "b" and NXP'08 circuits start with "p". It can be seen that these three benchmark sets span the complete range of designs from very small ones up to one Million gates.

8.4. Pattern Analysis Performance

The runtime performance of POINTER is determined by the efficiency of the fault simulator used. The operations performed beside simulation like sorting of the evidence only contribute very little to the overall runtime and are neglected in the following. The fault simulator is based on the *parallel–pattern single–fault propagation* (PPSFP) paradigm [WaicuEF*1985, Schul1988]. Multiple patterns (64 in the implementation used here) are simulated in parallel to optimally exploit bitwise operations on machine words. Only defect representatives on branch–points need to be simulated explicitly. The remaining evidences are determined without simulation by considering the activation conditions and the propagation conditions to the next branch–point.

The first experiment shows the baseline performance of the fault simulator for all benchmarks and the performance gain obtained by using the ι–dictionary and structural pruning. For each benchmark circuit, multiple C_{ud} are generated each containing a randomly selected single stuck–at fault. For each C_{ud}, the previously generated test pattern set is applied and the responses are analyzed using the PPSFP fault simulator. The time spend for fault simulation and the number of C_{ud} analyzed is measured over a time of at least one minute. Then, the MEPS value is calculated.

Tables A.3 and A.4 show the performance characteristics. By comparing the number of evidences and the number of branch–points in the benchmarks, it can be observed, that only 11.6% of the defect representatives need to be simulated explicitly. The performance degrades for larger circuits due to multiple factors. First, very small circuits allow for cache local simulation and lead to very high performance. The larger a benchmark gets, the more time is spend on simulating fault propagations and the MEPS value decreases. On average 4.89 MEPS can be achieved without using a dictionary. The use of a ι–dictionary improves the average simulation performance by a factor of 28.8X to 140.97 MEPS.

The performance also depends on the general observability of the stuck–at faults within the circuit. This is the reason, why the use of the ι–dictionary does not show the same effectiveness on all benchmarks and the variation in the performance values are much higher. One extreme case is benchmark p469k, where almost complete observability for every fault leads to a very low performance with and without dictionary.

8.5. Adaptive Precision Diagnosis

The adaptive diagnosis was conducted on all benchmark circuits and the three different defect types. A C_{ud} was obtained by randomly injecting a defect into a benchmark circuit and performing a production test using a pattern set targeting stuck–at faults previously generated by a commercial ATPG. If the potential C_{ud} passes the production test, no diagnosis will be performed on this circuit. If the potential C_{ud} fails the production test, adaptive diagnosis is performed starting with the fail data generated by the production test. For each defect type and each benchmark circuit, 1000 test cases are diagnosed and the average diagnostic success is reported.

Adaptive Stuck–At Fault Diagnosis

The average diagnostic success for stuck–at faults in all benchmarks is shown in figure 8.2. The exact numeric results shown in this graph are reported in the tables A.5 and A.6.

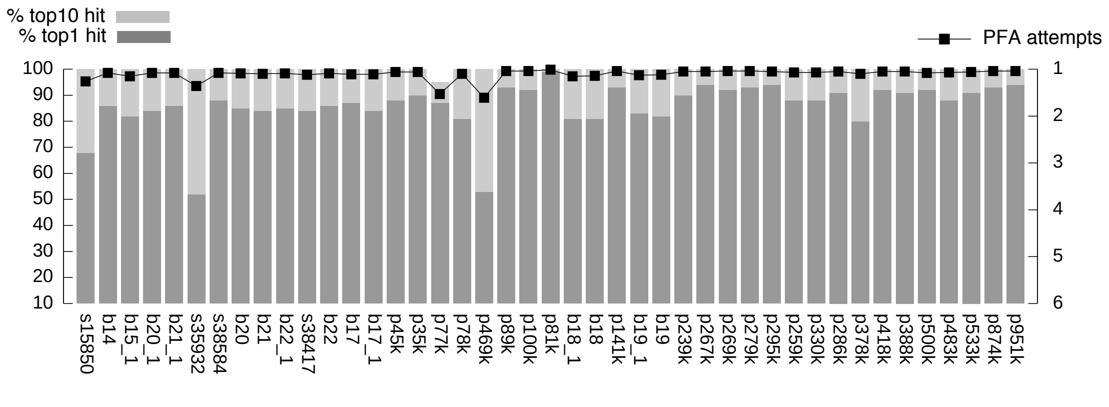


Figure 8.2.: Diagnostic success for adaptive diagnosis of stuck–at faults.

The fault model used for the defects is the same as the model for the defect representatives. The adaptive diagnosis approach will therefore always perform a perfect match refinement in which all the fault candidates f with $e(f, T) = (\sigma, 0, 0, 0)$ and a maximum number σ of predictions are distinguished as far as possible using diagnostic patterns. The diagnosis in this case is complete, i.e. POINTER provides an optimal resolution given that the diagnostic ATPG is able to generate all required distinguishing patterns.

The average number of PFA attempts over all circuits is 1.11, the average top10 success probability is 99.9% and the probability for a perfect diagnosis is 85.9%. The average number PFA attempts is larger than one, because the set of defect representatives is determined using simple structural fault collapsing. During diagnosis, the diagnostic pattern generator has either proven the functional equivalence for more fault pairs or aborted some distinguishing attempts. The average number of equivalent stuck–at faults is especially large in the benchmarks s35932, p77k and p469k and causes a reduction in diagnostic success. In the case of p77k, the number of equivalent fault candidates at the top of the ranking even exceeded 20 sometimes, thus leading to a top10 success probability of less than 100%.

The average number of diagnostic patterns generated in addition to the production test set is 0.15. This shows, that the production test set already provides a reasonable diagnostic resolution for stuck–at faults.

Adaptive Gross Delay Fault Diagnosis

The average diagnostic success for gross delay faults in all benchmarks is shown in figure 8.3. The exact numeric results shown in this graph are reported in the tables A.7 and A.8.

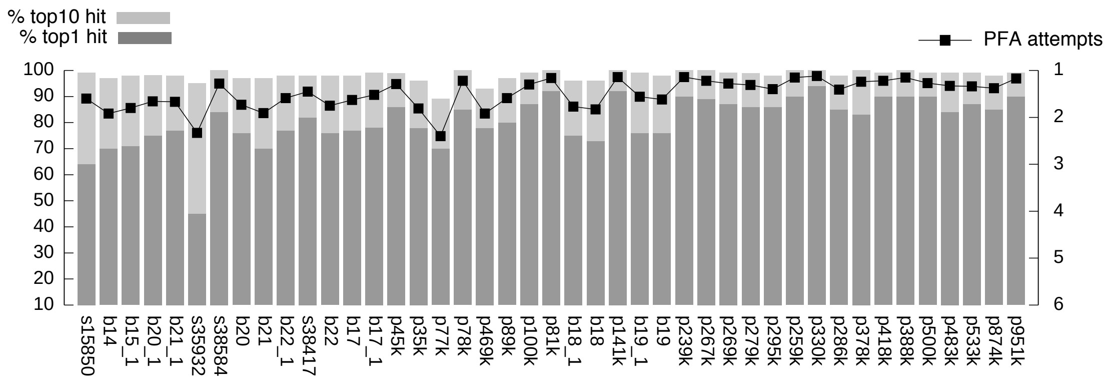


Figure 8.3.: Diagnostic success for adaptive diagnosis of gross delay faults.

Gross delay faults can be regarded as stuck–at faults with an additional activation condition. With stuck–at faults as defect representatives, POINTER will discover either *perfect match* or *perfect tester fail match* defect candidates. The implementation used in these experiments generates gross delay fault tests for perfect tester fail match refinement. The same adaptive diagnosis approach can also be combined to layout–aware diagnostic pattern generation [DesinPB2006] in order to provoke the activation of bridge defects and crosstalk issues in the same way.

The average number of PFA attempts (1.50), the average top10 success probability (98.1%) and the probability for a perfect diagnosis (80.8%) is still almost as good as for stuck–at fault diagnosis. It is noticeable, that the results for the industrial designs of NXP’08 (av. PFA attempts 1.36) are better than the results for the synthetic benchmark circuits (av. PFA attempts 1.70). The drop in diagnostic performance for the benchmarks s35932, p77k and p469k is also observable in this experiment.

Adaptive Swap Bridge Diagnosis

The average diagnostic success for swap bridge faults in all benchmarks is shown in figure 8.4. The exact numeric results shown in this graph are reported in the tables A.9 and A.10.

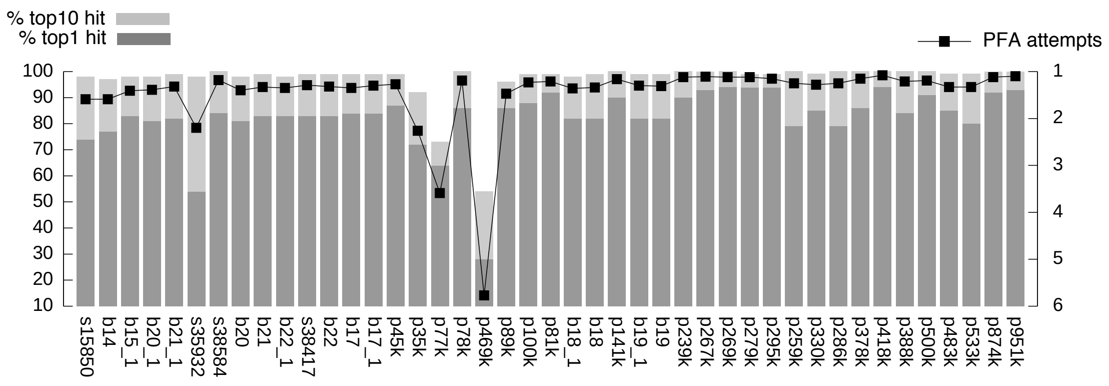


Figure 8.4.: Diagnostic success for adaptive diagnosis of swap bridge faults.

Whenever a swap bridge is active, two faulty signal lines are present in the C_{ud}. For POINTER, which compares the behavior of the C_{ud} with the behavior of single line faults, this defect type is most challenging. Depending on the observed behavior of the C_{ud}, the algorithm may decide on any of the refinement types from perfect match refinement to best guess refinement. In most cases, best guess refinement is used, because the behavior of the C_{ud} can not be explained by a fault on a single signal line. The resulting average number of suspect defect representatives is much higher in this case, because every evidence with $\sigma(f,T) > 0$ is included into this list. The average number of additional diagnostic test patterns generated is with 3.60 also higher, because more pairs of defect representatives near the top of the ranking are targeted. Still the number of additional patterns is very low compared to the number of production test patterns. This shows, that a diagnostic ATPG can be focused very well on the suspected circuit parts using the proposed adaptive diagnosis approach. POINTER provides consistently good results with all benchmarks except of four: p35k, and the circuits already identified as hard to diagnose (s35932, p77k and p469k). The average number of PFA attempts (1.47), the average top10 success probability (97.2%) and the probability for a perfect diagnosis (82.6%) is still comparable to diagnosis of single line faults.

8.6. Production Diagnosis

On each C_{ud}, a production test is performed with the stuck–at fault test set previously generated by a commercial ATPG. The production test is aborted after observing the n^{th} failing response from the C_{ud}. For each C_{ud}, three tests are performed with different abort limits: $n = 1, 4$, and 8. After each production test, diagnosis is performed on the fail data. For each benchmark circuit, defect model and test abort limit, the averages over 1000 diagnosis runs are reported.

The results of POINTER are compared to the original SLAT algorithm [BarteHH*2001]. Both algorithms are provided with the same fail data generated by the production test. The outcome of the SLAT approach is not a ranked list of suspects but an unordered set of multiplets. Each multiplet is a minimal set of evidences which can explain all the SLAT patterns. For the SLAT algorithm, the rank is defined as the expected number of drawings from these multiplets until the real culprit is found, however the maximum is set to 10 as for POINTER.

The diagnosis results for single stuck–at faults are shown in tables A.11 and A.12, for gross delay faults in tables A.13 and A.14, and for swap bridge faults in tables A.15 and A.16. Since the diagnostic performance of both algorithms is independent from the size of the used benchmark circuits, unweighted averages over all benchmarks can be used to compare their results.

The average diagnosis results over all benchmark circuits are shown in figure 8.5. For stuck–at faults, SLAT is always able to produce a set of multiplets with the victim line included, but the resulting rank often exceeds the top 10 and leads to a fail even if the number n of analyzed failing patterns is increased. The average ranks and success rates of POINTER mark the maximum achievable diagnostic resolution because all remaining candidates are equivalent under the pattern set applied. Since a gross delay fault is also a single line fault, similar results are obtained as in single stuck–at fault diagnosis. Again, POINTER provides maximum diagnostic resolution and needs a smaller number of failed responses to be analyzed. For the hard to diagnose swap bridges, the results of both algorithms are lower than for the previous two defect models. Still, POINTER provides better results than SLAT and already after the analysis of the 8th fail, less than two PFA attempts are necessary on average to find the defect.

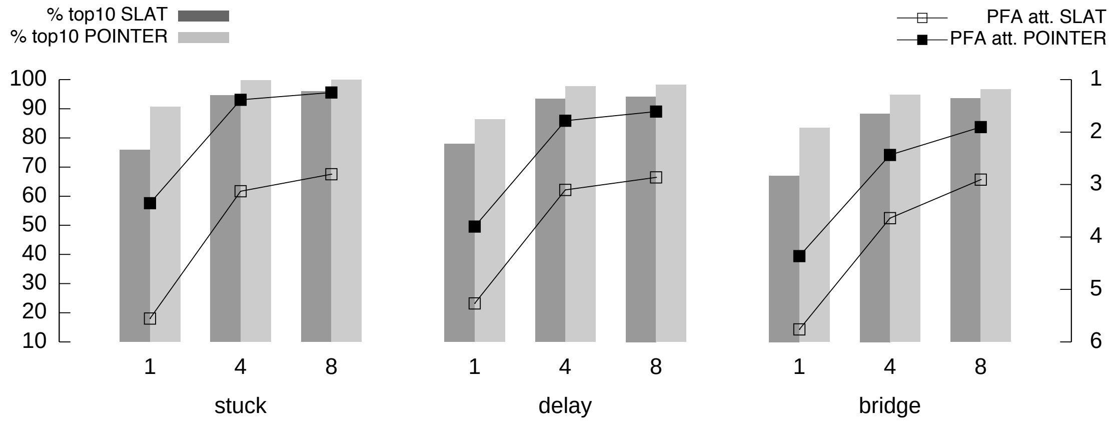


Figure 8.5.: Average diagnostic success of SLAT and POINTER after the first, the 4th and the 8th failing response.

After analysis of at least 8 fails, POINTER provides a near–perfect success rate in almost every test case. The average ranks show, that only one or two physical inspections are required in average to find the real defect. This outcome is highly sufficient for deciding about further adaptive diagnosis in a second step. One of the mayor factors contributing to the clear advantage of POINTER is, that POINTER considers passing pattern as well as all failing patterns from the fail–log. SLAT on the other hand only considers failing patterns with the SLAT–property. The results of SLAT can be refined towards the performance of POINTER by performing a passing pattern validation [BhattB2006].

In most cases, the runtimes of both diagnosis approaches are almost the same, because they are dominated by fault simulation. The SLAT approach needs to solve a set covering problem in order to generate the diagnosis result. If there are many patterns with SLAT–property or the failing responses can be explained by many stuck–at faults, this covering problem might add significantly to the overall runtime. POINTER on the other hand only uses simple sorting for generating the diagnosis result, which is less complex than solving a covering problem. Possible extensions to SLAT like passing pattern validation add even more to the runtime of SLAT. It can be concluded, that POINTER is able to provide better diagnosis results is less time compared to SLAT–based diagnosis algorithms.

8.7. Extreme Space Compaction

This series of experiments analyzes the impact of extreme space compaction on industrial designs and validates the performance of POINTER in this application. Only the NXP'08 benchmark set is used here, as realistic scan configurations are not available for the other benchmarks.

The test time is determined by the number of shifting cycles necessary to apply the test set. For the original designs, this can be calculated with the data reported in tables A.1 and A.2. Let p' the number of test patterns, l' the longest scan chain, and k' the number of scan chains for the original designs. The number of shifting cycles equals the number of test patterns p' multiplied by the length of the longest scan chain l', reported in column $p'l'$ of table A.17. In each shifting cycle, k' response bits are observed at the output of the scan chains. Therefore, the original number of response bits is $p'l'k'$ in table A.17. For example, over 7 Million shift cycles are necessary for p295k, and the amount of response data rises up to 257Mbit for p951k.

The original scan chains have been split into multiple shorter chains in order to reach a ratio of approximately $k \sim 5l$. This wide scan chain organization with rather short chains will reduce test time significantly. The next columns of table A.17 show the number of scan chains k and maximum scan chain length l of the new configurations. Now, parity trees were attached to the designs and ATPG was performed on the corresponding combinational representations.

The commercial ATPG was not able to compact the test sets as much as with the original designs and a higher abort limit had to be used in order to maintain the original fault coverage. Column Δfc shows the difference to the stuck–at fault coverage in the original designs. As expected, the fault masking introduced by the compactor is negligible. Except in three cases, the fault coverage was maintained or even improved. The slight reduction in stuck–at fault coverage for some circuits is mainly attributed to a larger number of faults aborted by the ATPG. The number of test patterns (column p) is higher in almost every case. However, as the scan chain lengths are much shorter now, the number of shifting cycles *decreases* in almost every case. Figure 8.6 shows the ratio $\Delta t = \frac{p'l'}{pl}$, the exact numbers are given in column Δt in table A.17. An average test time improvement of 6.3X is obtained. The test time did not improve for p378k, because the original scan chain configuration of this design already satisfies $k \sim 5l$ and no splitting was performed.

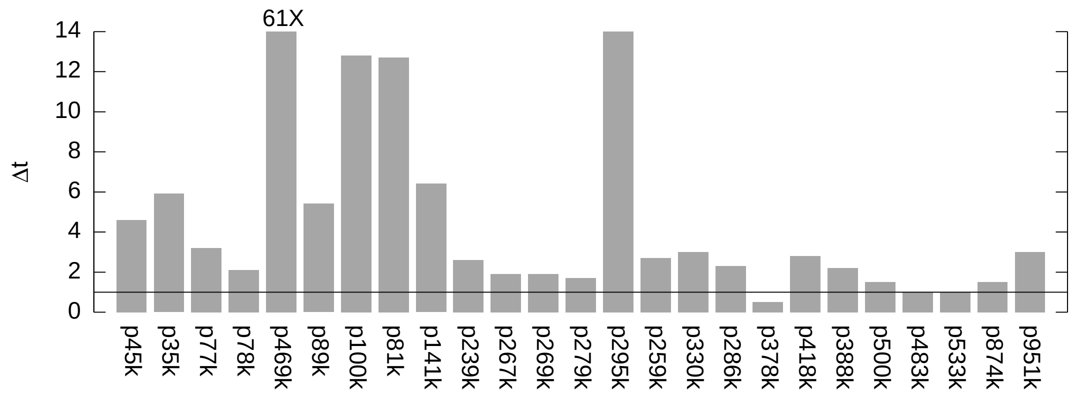


Figure 8.6.: Test time reduction achieved by the extreme response compaction scheme.

In each shifting cycle, only one response bit has to be observed at the output, hence the number of response bits equals the number of shifting cycles (column pl in table A.17). The figures in column Δr show the compaction ratio $\frac{p'l'k'}{pl}$ and are plotted in figure 8.7. The response data volume is reduced by several orders of magnitude with an average ratio of 135X.

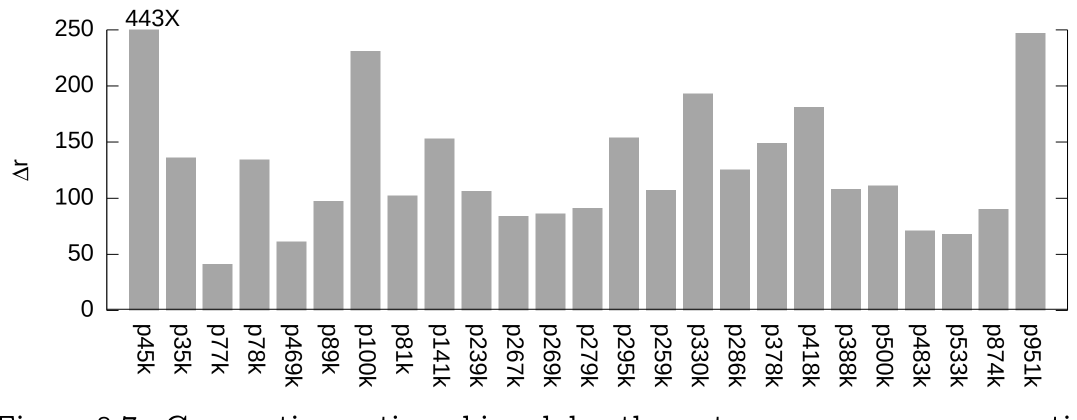


Figure 8.7.: Compaction ratio achieved by the extreme response compaction scheme.

8.7.1. Diagnosis on Parity Bits

Each C_{ud} is tested two times to generate two different fail–logs. The first fail–log contains the uncompacted responses of the test with the original test pattern set. The second fail–log contains only the parity information for the responses of the test with the newly generated test pattern set. Each fail–log is analyzed by POINTER and diagnosis results are compared between the uncompacted syndromes and the parity data. Again, for each benchmark and each defect model, the averages over 1000 diagnosis runs are reported.

Figure 8.8 compares the diagnostic success rates on the full uncompacted response with the diagnosis on the parity bits. The exact numeric values for each benchmark circuit are shown in tables A.18, A.19 and A.20 for stuck–at faults, gross delay faults, and swap bridges respectively. For single line faults, the diagnostic success can be maintained even with only 1/100th of the response data available. This shows, that the analysis of a single parity bit provides the same diagnostic resolution as 100+ uncompacted response bits. For non–target faults like gross delay faults, the diagnostic resolution even improves because the larger test set has higher defect coverage.

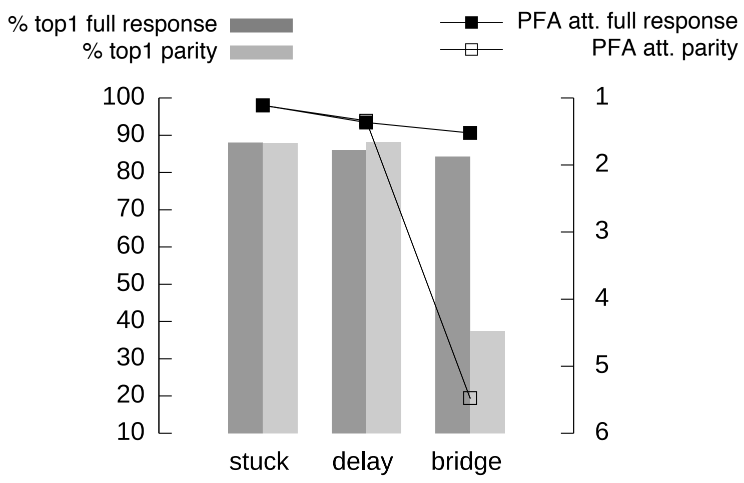


Figure 8.8.: Comparison of diagnostic success on full response and parity data.

For multi–line faults like swap bridges, the diagnostic performance drops noticeably with the use of extreme compaction. This is to be expected, because the high fan–in of the compactor increases the probability of the effects of the two faulty signal lines interacting in the syndrome. Under this difficult condition, still the real defect can be found on the top of the ranking in 37.4% is all cases and in under the top10 in 57.2% of all cases.

If a multi–line defect is encountered during analysis, the reduced confidence in the diagnosis result is clearly indicated by the reported matching conditions and the increased number of suspects reported. This data is still very usable for volume diagnosis. If a more reliable result is needed for PFA guidance, bypassing and adaptive diagnosis can be used.

8.7.2. Diagnostic Success on Limited Fail Data

This experiment aims to approach the question, whether it is more beneficial for diagnosis to store just parity bits or more elaborate signatures in a limited amount of fail memory. If more bits per failing scan slice are stored, more information is available for this particular fail, and fewer signatures may be needed to reach a certain diagnostic success. However, as more elaborate signatures take also more memory space per fail than simple parity information, less fails can be stored in the same amount of memory.

For generating the elaborate signatures, the *X–Compact* [MitraK2004] scheme is used. The X–Compactor can be parametrized by the number u of tolerated unknowns in a scan slice. As a parity tree does not tolerate any unknowns, u is set to 0 for a fair comparison. The resulting compactor guarantees error detection, if a scan slice contains one, two or any odd number of fails. In addition, if only one or two failing bits are present, these bits can be located by the signature. For a number of chains between 257 and 512, the X–Compact signature is 10 bits long.

To compare the fails storage requirements a straight forward encoding is used. For each fail, a 20–bit scan cycle index and the signature is stored. In the case of X–Compact, 30 bits are needed to store a fail, and for parity compaction, only the 20 bits for the scan cycle index are needed.

Each C_{ud} is now tested two times generating two fail–logs of limited size; one containing only failing scan slice indices and the other one containing indices and X–Compact signatures. Figure 8.9 show the average top1 success probability for single–line defects to be expected with a given fails memory size in bits.

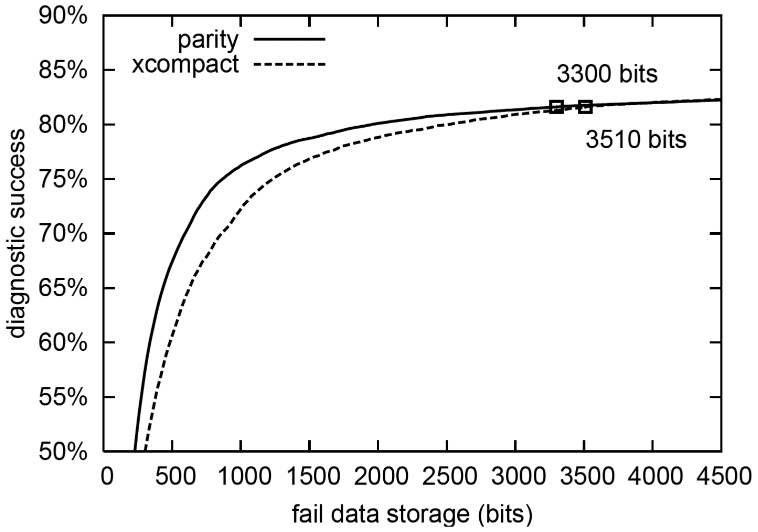


Figure 8.9.: Diagnostic success with limited amount of fail data [HolstW2009].

With limited amount of fails memory, the success rate of extreme response compaction is significantly higher than the one obtained by X–Compact. The two markings in figure 8.9 denote the break–even points where the diagnostic success with compaction reaches the diagnostic success on the full, uncompacted response data of the original designs. For extreme compaction, $n = 165$ fails are need to be stored which requires 3300 bits of memory. At these break–even points the fails memory for parity compaction is still smaller than for X–Compact. If unlimited on chip resources are available, the diagnostic success rate is higher with X–Compact as expected. In conclusion, storing just the indices of failing scan slices in a limited fails memory is more efficient than spending additional space for X–Compact signatures.

8.8. Summary

A series of experiments were performed on a representative set of benchmark circuits and industrial designs ranging from 10000 to 1 Million gates in size. The POINTER algorithm was parameterized with the stuck–at model for generating the defect representatives and a zero–delay model for circuit simulation. Then, it was benchmarked by generating C_{ud}s with randomly selected defects and performing logic diagnosis on their responses. The models chosen for the random defects represent a large class of spot defects encountered in real chips: the stuck–at model for defects, that perfectly match a defect representative in fault simulation; the gross delay fault model for all single–line defects with an unknown activation condition; and the swap bridge for multi–line defects or multiple interacting defects. For each benchmark, each defect model and each setting, 1000 test cases were diagnosed to obtain well–founded numeric results.

In a lab setting, adaptive diagnosis with POINTER provided very good results across all defect models and the vast majority of benchmark circuits. For defects with stuck–at fault behavior, the achieved results mark the best possible ones because POINTER analysis is complete in this case and diagnostic ATPG was not able to resolve the remaining equivalences among the suspects.

In production diagnosis, POINTER performs better than the SLAT algorithm and yields acceptable results with just the first 8 failing responses available. The clear advantage here, stems from the inherent passing response consideration in POINTER.

Extreme space compaction provides shorter test times and reduces the response data volume by several orders of magnitude. The diagnostic success for single–line defects was maintained or even improved by extreme response compaction. For multi–line defects, the diagnostic success drops, but an identifying defect representative was still among the top 10 in the majority of the test cases. Such weak call–outs are clearly communicated by POINTER by a higher number of suspects reported.

If only limited on–chip storage is available for diagnostic fail data, it has been shown, that storing the parity information of a higher number of responses is more beneficial than storing bigger signatures and a lower number of fails. To reach the same diagnostic quality as with the complete, uncompacted responses, a fail memory of only 3300 bits is needed in average.

9. Conclusions

Every semiconductor company competing in the field of leading edge technologies has to face the challenge of building more powerful and more complex silicon chips in large quantities and high quality in a very short time. In order to improve production efficiency and margins, one has to learn from the flawed chips directly through *failure analysis*, and essential tools for failure analysis are *logic diagnosis algorithms*.

The two principle challenges for logic diagnosis algorithms are the reduction of a–priori fault assumptions and test data economy. Constraints on response data storage and test time force diagnosis algorithms to use the available data from production test, lab test, or in–field self–test most effectively. The effective use of all these sources calls for a versatile diagnosis algorithm, which is able to use and combine fail data of various sources and properties to compute the best possible diagnosis result despite the lack of specific fault models.

The challenge of reducing fault assumptions was addressed in this work by introducing a new notation for generalized fault modeling, the *Conditional Line Flip* (CLF) calculus. The CLF calculus is able to describe arbitrarily complex defective behavior and provides a clear distinction between a defect site and a fault–free circuit surrounding this area. The generality of the CLF calculus allows it to encompass all previous notations and basing a diagnosis algorithm directly on this calculus was the way taken in this work to develop a location–based diagnosis method with as few assumptions on defects as possible.

The proposed diagnosis algorithm POINTER is based on this calculus and the inject–and–validate paradigm as the natural choice for an efficient algorithm with predictable runtimes, reduced fault assumptions and the ability to work with highly compacted response data. POINTER draws from the concepts of SLAT–based diagnosis and provides at the same time several key advantages over previously proposed algorithms and extensions. The core algorithm is based on fault simulation, summing up of integer numbers and sorting. This provides stable and predictable runtime performance important for efficient deployments in high volume diagnosis environments. Maximum diagnostic information is extracted from every available response—failing and passing ones—with a runtime linear in the number of failing responses. POINTER can be parameterized with various models for defect representatives and circuit simulation. Advanced matching conditions describe the relation between the defect under investigation and their representatives in fault simulation and provides information on the quality of the diagnosis result.

POINTER has been extended to an adaptive diagnosis method, that is able to provide the best possible diagnosis result in many cases. Only a few diagnostic test patterns are needed in most cases to reach this result. The fixed test set in production test allows the use of an ι–dictionary to enable structural pruning and highly efficient passing pattern consideration without compromising the quality of the diagnosis results. The achieved speedup of almost 30 is especially useful for deployments in high volume production lines. Moreover, due to efficient passing pattern consideration, 8 failing patterns recorded per case is already enough for highly accurate diagnosis results.

To enable diagnosis for built–in self–test in the field and for multi–site test, extreme space compaction was proposed. Given a small fail memory of fixed size, storing just the parity of failing scan slices was shown to be more beneficial for diagnosis than storing more complex signatures. The diagnostic success after analyzing just the parity bits degrades slightly for multi–line defects, but is maintained or even improved for single–line defects. Again, POINTER clearly points out, which of these two cases are present in a chip. The results are highly sufficient to decide on further investigations. All applications use the same base algorithm and results gained from production diagnosis and in–field diagnosis can be re–used in a thorough lab analysis using adaptive diagnosis.

9.1. Further Research Directions

The results of this work may provide the foundation for a couple of further research directions.

The CLF calculus can be applied to other fields of testing like the investigation of its relation to higher–level fault models or the analysis of fault model independent test generation approaches in the same way it was done with partial pseudo–exhaustive testing [MumtaIH*2011, MumtaIH*2011a].

The findings on the relation between extreme space compaction and diagnosis results have stimulated research towards even higher compaction ratios and to further improve diagnostic information per bit of response data [KochtHE*2009, Elm2011, CookEWA2011].

The available matching criterions are already sufficient for applications of POINTER to graceful degradation [DalirHE*2011, DalirHE*2012], where not a fault candidate, but the *fault–free* circuit parts have to be identified. Other groups have based their candidate matching and ranking methods on some key aspects of POINTER to further optimize the results on multiple interacting faults [YeHL2011, TangCGR2010].

Other research directions may include extracting CLF conditions by correlating their activation frequency with the activities of neighboring signal lines, or the parameterization of POINTER with timing–aware CLFs and circuit models. A timing–aware POINTER approach could diagnose small–delay faults or other timing–related issues more precisely and even on compacted signatures. However, timing simulation is much more compute–intensive and new ways have to be found to reduce simulation effort as much as possible. Promising approaches could include focusing the fault simulation more precisely on the suspicious region by using the evidence collected so far, or the use of data–parallel architectures for accelerating the core tasks.

A. Additional Result Tables

circuit	g	t	i	o	s	k'	l'	p'	*fc*
s15850	10211	87	611	684	597			176	96.80
b14	10735	66	277	299	245			830	99.34
b15_1	14122	55	485	519	449			1186	99.42
b20_1	15354	74	522	512	490			651	99.50
b21_1	15460	70	522	512	490			683	99.51
s35932	16353	29	1763	2048	1728			48	89.81
s38584	21462	59	1464	1730	1452			208	95.03
b20	21599	73	522	512	490			1037	99.32
b21	22055	74	522	512	490			1060	99.21
b22_1	23210	72	767	757	735			682	99.50
s38417	23537	48	1664	1742	1636			190	99.48
b22	32090	74	767	757	735			964	99.50
b17	35549	103	1452	1512	1415			1281	97.52
b17_1	42879	55	1452	1512	1415			1529	99.43
p45k	43190	60	3739	2550	2331	97	333	2133	99.69
p35k	46435	71	2912	2229	2173	23	130	4036	98.39
p77k	72370	569	3487	3400	3386	13	304	588	92.43
p78k	74243	46	3148	3484	2977	65	64	81	100.00
p469k	75572	174	635	403	332	1	706	314	98.80
p89k	88726	110	4632	4557	4301	18	963	1083	98.74
p100k	96685	105	5902	5829	5735	18	792	2054	99.50

Table A.1.: Small benchmark circuit characteristics. g: number of gates, t: length of structurally longest combinational path, i: number of inputs (primary + pseudo primary), o: number of outputs (primary + pseudo primary), s: number of scan elements, k': number of scan chains, l': length of the longest scan chain, p': stuck–at test set size, *fc*: stuck–at fault coverage.

circuit	g	t	i	o	s	k'	l'	p'	fc
p81k	108991	55	4029	3952	3877	8	513	1336	99.47
b18_1	118263	174	3357	3342	3320			1455	99.45
b18	124886	174	3357	3342	3320			1469	99.42
p141k	172686	79	11290	10502	10501	24	486	1609	98.87
b19_1	238810	179	6666	6669	6642			1703	99.13
b19	251560	179	6666	6669	6642			1680	99.13
p239k	259241	184	18692	18495	18382	40	541	1081	98.79
p267k	271538	74	17332	16621	16528	45	494	1135	99.60
p269k	272630	74	17333	16621	16528	45	494	1152	99.60
p279k	287935	150	18074	17831	17524	55	409	1288	97.90
p295k	291022	114	18508	18521	18465	11	1852	3877	99.16
p259k	334524	187	18713	18495	18398	40	541	1248	99.08
p330k	355642	71	18010	17468	16775	64	317	5306	98.95
p286k	364343	154	18347	17831	17713	55	416	2156	98.41
p378k	371215	46	15732	17420	14885	325	64	84	100.00
p418k	439198	206	30430	29809	28616	64	830	2335	98.36
p388k	489271	225	25005	24065	23789	50	525	991	99.47
p500k	495544	170	30768	30840	29312	76	446	2352	98.45
p483k	515717	109	33264	32610	32307	71	900	481	98.84
p533k	662730	113	33373	32610	32409	71	900	737	99.15
p874k	717268	239	42899	42243	41803	59	780	1833	92.43
p951k	1002883	140	91994	104747	91410	82	1381	2270	99.03

Table A.2.: Large benchmark circuit characteristics. See table A.1 for column definitions.

circuit	e	fo	base MEPS	pruning MEPS
s15850	12150	1518	13.76	83.44
b14	23716	2409	13.37	41.21
b15_1	30508	2910	18.91	93.22
b20_1	34710	3668	10.18	45.21
b21_1	34510	3477	9.42	39.01
s35932	39094	5295	7.58	119.47
s38584	38358	3946	9.45	299.24
b20	47376	4645	8.62	43.66
b21	48182	4613	9.62	42.13
b22_1	52172	5430	9.08	52.34
s38417	32320	4569	8.52	172.40
b22	70464	6876	7.76	58.33
b17	81330	8145	8.32	90.47
b17_1	92794	8767	10.49	139.96
p45k	72164	7625	6.48	258.55
p35k	70382	8020	5.64	46.43
p77k	124572	14033	2.34	19.61
p78k	163310	18462	3.64	173.21
p469k	169590	14399	0.26	0.37
p89k	155900	14560	5.00	215.95
p100k	168102	19057	3.54	265.18

Table A.3.: Pattern analysis performance for small benchmarks. *e*: number of evidences (=number of structurally collapsed stuck–at faults), *fo*: number of fanout stems explicitly simulated, *base MEPS*: baseline analysis performance, *pruning MEPS*: analysis performance with pruning and ι–dictionary.

circuit	e	fo	base MEPS	pruning MEPS
p81k	224424	29440	4.13	152.90
b18_1	264022	29934	4.42	161.49
b18	277756	31022	5.10	161.23
p141k	291546	33080	1.65	38.81
b19_1	533736	60815	2.62	171.34
b19	560256	63006	2.92	200.51
p239k	456982	46960	1.43	275.02
p267k	375958	40614	1.29	171.90
p269k	378142	40629	1.29	170.17
p279k	499146	55090	1.26	149.85
p295k	488172	46293	1.52	51.80
p259k	643436	83400	1.05	218.39
p330k	566386	65730	1.52	198.44
p286k	677868	86365	1.35	139.10
p378k	816534	92302	1.68	184.39
p418k	694172	75163	1.14	249.48
p388k	913724	122962	1.20	146.40
p500k	852646	96983	0.85	345.14
p483k	943864	117055	0.80	216.23
p533k	1299188	180328	0.53	209.39
p874k	1022380	108020	0.38	92.06
p951k	1610146	178289	0.09	58.25

Table A.4.: Pattern analysis performance for large benchmarks. See table A.3 for column definitions.

circuit	dp	s	%t10	%t1	#a
s15850	0.0	1.6	100	68	1.26
b14	0.1	1.2	100	86	1.08
b15_1	0.1	1.3	100	82	1.15
b20_1	0.1	1.2	100	84	1.08
b21_1	0.0	1.2	100	86	1.08
s35932	0.0	1.7	100	52	1.36
s38584	0.0	1.2	100	88	1.08
b20	0.2	1.2	100	85	1.09
b21	0.2	1.2	100	84	1.10
b22_1	0.1	1.2	100	85	1.09
s38417	0.1	1.2	100	84	1.12
b22	0.1	1.2	100	86	1.09
b17	0.0	1.2	100	87	1.11
b17_1	0.2	1.2	100	84	1.11
p45k	0.0	1.1	100	88	1.06
p35k	0.2	1.1	100	90	1.06
p77k	4.3	4.0	95	87	1.53
p78k	0.0	1.2	100	81	1.10
p469k	0.0	2.2	100	53	1.61
p89k	0.1	1.1	100	93	1.04
p100k	0.0	1.1	100	92	1.04

Table A.5.: Adaptive diagnosis of single stuck–at faults in small benchmarks. *dp*: average number of additional diagnostic patterns generated, *s*: number of suspects reported, *%t10*: probability of top10 hit, *%t1*: probability of perfect diagnosis, *#a*: average number of PFA attempts.

circuit	dp	s	%t10	%t1	#a
p81k	0.0	1.0	100	99	1.01
b18_1	0.0	1.3	100	81	1.15
b18	0.0	1.3	100	81	1.14
p141k	0.0	1.1	100	93	1.04
b19_1	0.1	1.3	100	83	1.13
b19	0.0	1.2	100	82	1.12
p239k	0.0	1.1	100	90	1.05
p267k	0.1	1.1	100	94	1.05
p269k	0.1	1.1	100	92	1.04
p279k	0.0	1.1	100	93	1.04
p295k	0.0	1.1	100	94	1.05
p259k	0.0	1.1	100	88	1.07
p330k	0.0	1.1	100	88	1.07
p286k	0.0	1.1	100	91	1.05
p378k	0.0	1.2	100	80	1.10
p418k	0.1	1.1	100	92	1.05
p388k	0.0	1.1	100	91	1.05
p500k	0.1	1.2	100	92	1.08
p483k	0.0	1.1	100	88	1.07
p533k	0.0	1.1	100	91	1.06
p874k	0.0	1.1	100	93	1.04
p951k	0.0	1.1	100	94	1.04

Table A.6.: Adaptive diagnosis of single stuck–at faults in large benchmarks. See table A.5 for column definitions.

circuit	dp	s	%t10	%t1	#a
s15850	0.1	4.4	99	64	1.60
b14	0.1	6.8	97	70	1.92
b15_1	0.1	4.7	98	71	1.80
b20_1	0.1	7.2	98	75	1.66
b21_1	0.2	8.3	98	77	1.67
s35932	1.2	5.8	95	45	2.33
s38584	0.0	3.1	100	84	1.28
b20	0.1	6.7	97	76	1.73
b21	0.2	10.0	97	70	1.91
b22_1	0.2	6.6	98	77	1.59
s38417	0.3	3.5	98	82	1.45
b22	0.1	7.0	98	76	1.75
b17	0.0	4.9	98	77	1.63
b17_1	0.1	3.9	99	78	1.52
p45k	0.0	3.6	99	86	1.29
p35k	0.1	11.9	96	78	1.81
p77k	2.5	218.5	89	70	2.40
p78k	0.1	3.2	100	85	1.22
p469k	0.0	7.8	93	78	1.92
p89k	0.3	8.1	97	80	1.59
p100k	0.1	4.7	99	87	1.30

Table A.7.: Adaptive diagnosis of single gross delay faults in small benchmarks. See table A.5 for column definitions.

circuit	dp	s	%t10	%t1	#a
p81k	0.0	3.4	100	92	1.16
b18_1	0.1	8.1	96	75	1.77
b18	0.1	8.0	96	73	1.83
p141k	0.0	3.0	100	92	1.14
b19_1	0.2	5.4	99	76	1.56
b19	0.1	5.6	98	76	1.62
p239k	0.0	3.0	100	90	1.14
p267k	0.1	3.3	100	89	1.22
p269k	0.1	3.5	99	87	1.28
p279k	0.1	4.2	99	86	1.31
p295k	0.1	7.0	98	86	1.40
p259k	0.0	3.1	100	90	1.15
p330k	0.0	3.5	100	94	1.12
p286k	0.1	5.9	98	85	1.41
p378k	0.0	3.2	100	83	1.24
p418k	0.1	3.0	99	90	1.22
p388k	0.0	3.7	100	90	1.15
p500k	0.1	4.1	99	90	1.27
p483k	0.1	4.7	99	84	1.33
p533k	0.1	5.7	99	87	1.34
p874k	0.1	4.1	98	85	1.38
p951k	0.1	3.3	99	90	1.17

Table A.8.: Adaptive diagnosis of single gross delay faults in large benchmarks. See table A.5 for column definitions.

circuit	dp	s	%t10	%t1	#a
s15850	20.4	195.4	98	74	1.59
b14	2.8	998.2	97	77	1.59
b15_1	2.1	614.8	98	83	1.41
b20_1	0.7	1204.4	98	81	1.39
b21_1	1.2	1250.6	99	82	1.32
s35932	33.7	83.4	98	54	2.20
s38584	10.9	85.6	100	84	1.18
b20	0.6	1363.1	98	81	1.40
b21	0.7	1537.5	99	83	1.33
b22_1	0.5	1287.9	98	83	1.35
s38417	4.1	210.7	99	83	1.29
b22	0.4	1565.2	99	83	1.32
b17	0.5	793.2	99	84	1.35
b17_1	2.5	704.8	99	84	1.30
p45k	2.5	429.9	99	87	1.27
p35k	0.7	6097.8	92	72	2.26
p77k	32.0	7933.1	73	64	3.59
p78k	2.1	296.3	100	86	1.19
p469k	0.4	7252.5	54	28	5.77
p89k	3.6	465.9	96	86	1.47
p100k	4.4	597.1	99	88	1.23

Table A.9.: Adaptive diagnosis of single swap bridge faults in small benchmarks. See table A.5 for column definitions.

circuit	dp	s	%t10	%t1	#a
p81k	0.0	833.9	99	92	1.21
b18_1	0.9	1241.2	98	82	1.36
b18	0.9	1270.3	99	82	1.34
p141k	0.4	841.1	100	90	1.16
b19_1	1.1	1809.7	99	82	1.30
b19	2.4	1852.4	99	82	1.31
p239k	1.2	399.6	100	90	1.12
p267k	4.2	347.5	100	93	1.11
p269k	3.4	372.6	100	94	1.12
p279k	1.0	473.7	99	94	1.12
p295k	1.4	658.9	99	94	1.15
p259k	0.2	419.6	100	79	1.25
p330k	0.3	651.5	99	85	1.28
p286k	0.3	608.7	100	79	1.25
p378k	1.7	307.5	100	86	1.15
p418k	0.7	321.5	100	94	1.08
p388k	0.1	632.6	100	84	1.21
p500k	5.6	432.2	100	91	1.19
p483k	0.5	652.0	99	85	1.33
p533k	0.2	638.3	99	80	1.33
p874k	0.4	438.2	100	92	1.12
p951k	1.0	304.2	100	93	1.10

Table A.10.: Adaptive diagnosis of single swap bridge faults in large benchmarks. See table A.5 for column definitions.

	1st fail				4th fail				8th fail			
	SLAT		POINTER		SLAT		POINTER		SLAT		POINTER	
circuit	%t10	#a	%t10	#a	%t10	#a	%t10	#a	%t10	#a	%t10	#a
s15850	88	4.4	96	2.8	98	3.0	100	1.5	98	2.8	100	1.3
b14	71	6.9	94	3.4	90	3.8	100	1.3	91	3.4	100	1.2
b15_1	92	4.7	98	2.5	98	3.1	100	1.4	98	3.0	100	1.3
b20_1	70	6.7	92	3.6	91	3.4	100	1.4	93	3.0	100	1.2
b21_1	72	6.6	90	3.5	93	3.2	100	1.3	94	3.0	100	1.2
s35932	60	6.2	86	4.4	100	2.6	100	1.5	100	2.4	100	1.4
s38584	97	3.6	99	2.4	100	2.4	100	1.4	100	2.3	100	1.3
b20	66	7.1	92	3.5	90	3.6	100	1.4	92	3.3	100	1.2
b21	66	7.1	93	3.3	90	3.7	100	1.3	91	3.5	100	1.2
b22_1	74	6.6	92	3.6	93	3.2	100	1.3	94	3.0	100	1.2
s38417	86	4.3	95	2.8	98	2.5	100	1.3	98	2.3	100	1.2
b22	66	7.1	92	3.5	94	3.3	100	1.3	95	3.0	100	1.2
b17	86	5.0	97	2.7	97	3.0	100	1.4	98	2.8	100	1.2
b17_1	92	4.5	97	2.6	98	3.0	100	1.4	99	2.7	100	1.3
p45k	68	5.8	87	3.6	95	3.2	100	1.4	96	2.7	100	1.3
p35k	39	7.0	69	4.9	71	5.1	100	1.5	84	4.1	100	1.3
p77k	63	5.9	74	4.4	76	4.5	89	2.5	77	4.3	93	2.1
p78k	86	5.6	95	3.7	100	2.1	100	1.2	100	1.9	100	1.1
p469k	41	8.1	83	4.9	84	5.2	100	1.7	86	4.9	100	1.6
p89k	72	5.6	92	3.1	91	3.7	100	1.3	94	3.3	100	1.1
p100k	67	5.5	85	3.7	96	3.0	100	1.3	98	2.4	100	1.2

Table A.11.: Stuck–at fault diagnosis with limited failure information in small benchmarks. *%t10*: probability of top10 hit, *#a*: average number of PFA attempts.

	1st fail				4th fail				8th fail			
	Slat		Pointer		Slat		Pointer		Slat		Pointer	
circuit	%t10	#a	%t10	#a	%t10	#a	%t10	#a	%t10	#a	%t10	#a
p81k	58	7.2	89	3.7	97	3.0	100	1.3	97	2.6	100	1.2
b18_1	76	5.5	86	3.6	94	3.5	99	1.5	98	2.9	100	1.3
b18	77	5.7	88	3.5	93	3.6	100	1.5	98	3.0	100	1.3
p141k	84	4.8	95	2.9	99	2.6	100	1.3	99	2.3	100	1.2
b19_1	74	5.8	86	3.6	90	3.8	99	1.5	95	3.2	100	1.3
b19	75	5.7	87	3.6	90	3.7	99	1.6	94	3.2	100	1.3
p239k	72	5.1	87	3.5	99	2.5	100	1.2	99	2.2	100	1.1
p267k	92	4.2	98	2.5	98	2.5	100	1.3	98	2.3	100	1.2
p269k	90	4.1	97	2.6	99	2.4	100	1.3	99	2.3	100	1.2
p279k	92	4.1	97	2.4	98	2.6	100	1.3	98	2.5	100	1.2
p295k	86	4.5	95	2.5	93	3.4	100	1.4	94	3.2	100	1.3
p259k	76	5.4	88	3.6	98	2.8	100	1.3	99	2.4	100	1.1
p330k	76	5.5	90	3.4	97	2.9	100	1.4	98	2.5	100	1.2
p286k	83	5.4	95	3.1	98	3.0	100	1.3	98	2.7	100	1.2
p378k	81	5.9	92	3.9	100	2.1	100	1.2	100	1.9	100	1.1
p418k	92	4.0	98	2.5	99	2.4	100	1.3	99	2.3	100	1.2
p388k	77	5.7	89	3.5	99	2.6	100	1.3	99	2.3	100	1.1
p500k	79	5.0	91	3.2	94	3.0	100	1.3	96	2.7	100	1.2
p483k	61	6.2	81	4.2	95	2.9	100	1.3	97	2.5	100	1.1
p533k	66	6.1	84	4.0	96	3.0	100	1.3	98	2.5	100	1.2
p874k	83	4.4	93	2.8	96	2.9	100	1.3	97	2.5	100	1.2
p951k	84	4.3	94	2.8	98	2.7	100	1.3	98	2.4	100	1.2

Table A.12.: Stuck–at fault diagnosis with limited failure information in large benchmarks. *%t10*: probability of top10 hit, *#a*: average number of PFA attempts.

	1st fail				4th fail				8th fail			
	Slat		Pointer		Slat		Pointer		Slat		Pointer	
circuit	%t10	#a	%t10	#a	%t10	#a	%t10	#a	%t10	#a	%t10	#a
s15850	90	3.9	96	2.8	96	2.8	99	1.7	96	2.7	99	1.6
b14	75	6.6	86	4.4	89	3.9	97	2.0	89	3.7	98	1.9
b15_1	92	4.3	92	3.4	97	3.1	98	2.0	97	3.0	98	1.9
b20_1	74	6.4	85	4.3	90	3.6	98	1.9	90	3.4	98	1.8
b21_1	77	6.1	85	4.3	92	3.4	98	1.9	92	3.3	98	1.8
s35932	61	6.1	78	4.9	88	3.6	94	2.5	89	3.5	94	2.4
s38584	98	3.2	99	2.4	100	2.3	100	1.5	100	2.3	100	1.4
b20	72	6.7	86	4.4	90	3.8	97	1.9	90	3.7	97	1.8
b21	67	6.9	84	4.4	86	4.0	96	2.1	86	3.9	97	2.0
b22_1	77	6.2	86	4.4	93	3.3	98	1.8	93	3.2	98	1.7
s38417	89	4.0	93	3.0	95	2.5	98	1.6	95	2.4	98	1.5
b22	69	6.8	84	4.5	90	3.7	98	1.9	90	3.6	98	1.8
b17	89	4.6	91	3.5	96	3.1	98	1.9	96	3.0	98	1.7
b17_1	94	4.1	93	3.1	98	2.8	99	1.8	98	2.6	99	1.6
p45k	70	5.4	84	3.9	95	3.0	99	1.7	96	2.6	99	1.5
p35k	43	6.7	57	5.7	73	5.0	92	2.7	84	3.9	96	2.0
p77k	66	5.7	71	4.7	74	4.6	82	3.2	75	4.4	87	2.7
p78k	85	5.6	92	4.2	99	2.2	100	1.4	99	2.1	100	1.2
p469k	62	7.1	76	4.5	89	4.2	93	2.0	89	4.1	93	1.9
p89k	79	5.1	88	3.3	88	3.8	97	1.8	89	3.6	97	1.7
p100k	69	5.3	82	4.0	94	2.9	98	1.6	95	2.5	99	1.4

Table A.13.: Gross delay fault diagnosis with limited failure information in small benchmarks. *%t10*: probability of top10 hit, *#a*: average number of PFA attempts.

	1st fail				4th fail				8th fail			
	Slat		Pointer		Slat		Pointer		Slat		Pointer	
circuit	%t10	#a	%t10	#a	%t10	#a	%t10	#a	%t10	#a	%t10	#a
p81k	56	7.1	82	4.6	97	2.7	100	1.4	97	2.3	100	1.3
b18_1	78	5.3	81	4.2	91	3.5	95	2.2	93	3.1	96	1.9
b18	78	5.3	82	4.1	91	3.4	96	2.0	92	3.2	96	1.9
p141k	87	4.4	94	3.1	98	2.4	100	1.4	98	2.2	100	1.3
b19_1	75	5.4	79	4.3	90	3.6	96	2.1	95	3.1	98	1.8
b19	76	5.5	80	4.2	90	3.7	95	2.3	94	3.2	98	1.8
p239k	71	5.1	84	3.8	98	2.5	100	1.4	98	2.1	100	1.2
p267k	91	4.0	94	2.8	97	2.4	100	1.5	98	2.3	100	1.3
p269k	92	4.0	95	2.9	98	2.5	99	1.5	98	2.3	99	1.4
p279k	91	4.0	94	2.9	97	2.6	99	1.5	98	2.4	99	1.4
p295k	88	4.0	94	2.7	93	3.2	97	1.8	94	3.0	98	1.6
p259k	74	5.3	83	4.0	98	2.5	100	1.4	98	2.2	100	1.3
p330k	79	5.2	90	3.5	97	2.6	100	1.4	98	2.3	100	1.3
p286k	81	5.2	89	3.6	95	3.1	98	1.7	95	2.9	98	1.6
p378k	85	5.6	90	4.0	99	2.2	100	1.4	99	2.1	100	1.3
p418k	92	3.8	96	2.7	98	2.4	99	1.5	98	2.2	99	1.3
p388k	77	5.5	85	3.9	98	2.6	99	1.4	98	2.3	100	1.2
p500k	81	4.7	90	3.3	95	2.8	98	1.6	96	2.5	99	1.4
p483k	61	6.1	78	4.4	94	2.9	99	1.6	94	2.6	99	1.4
p533k	69	5.8	81	4.3	94	2.9	98	1.6	94	2.5	99	1.4
p874k	84	4.2	90	3.1	95	2.8	98	1.7	96	2.6	98	1.5
p951k	87	4.1	92	3.0	98	2.5	99	1.4	98	2.2	99	1.3

Table A.14.: Gross delay fault diagnosis with limited failure information in large benchmarks. *%t10*: probability of top10 hit, *#a*: average number of PFA attempts.

	1st fail				4th fail				8th fail			
	SLAT		POINTER		SLAT		POINTER		SLAT		POINTER	
circuit	%t10	#a	%t10	#a	%t10	#a	%t10	#a	%t10	#a	%t10	#a
s15850	69	5.2	92	3.4	87	3.3	96	2.2	91	2.8	97	2.0
b14	72	6.4	84	4.7	88	4.1	96	2.3	92	3.4	96	1.9
b15_1	89	4.4	96	3.3	98	2.8	98	2.0	98	2.5	97	1.8
b20_1	73	6.1	86	4.6	90	3.8	97	2.3	96	3.0	99	1.7
b21_1	73	6.0	85	4.5	93	3.7	97	2.2	96	2.8	99	1.6
s35932	38	7.5	66	5.7	75	4.6	93	3.0	86	3.7	96	2.4
s38584	72	5.2	98	3.3	88	3.3	99	2.0	92	2.8	100	1.6
b20	73	6.2	89	4.3	90	4.0	96	2.4	94	3.3	98	1.8
b21	74	6.1	87	4.4	90	3.9	97	2.2	95	3.0	99	1.7
b22_1	75	6.0	86	4.5	94	3.6	97	2.2	96	2.9	99	1.7
s38417	65	5.5	91	3.6	88	3.3	97	2.2	94	2.5	98	1.8
b22	69	6.6	83	4.8	91	3.9	98	2.2	95	3.1	99	1.6
b17	81	4.9	93	3.7	95	3.1	98	2.0	97	2.6	98	1.7
b17_1	85	4.7	93	3.5	97	2.8	98	2.0	98	2.4	99	1.7
p45k	65	5.5	82	4.3	87	3.6	96	2.5	92	3.0	98	2.0
p35k	54	5.8	64	5.1	70	4.6	82	3.5	81	3.9	87	2.9
p77k	61	5.7	71	4.9	78	4.4	69	4.5	83	3.8	69	4.3
p78k	47	7.5	83	5.6	81	4.7	98	2.2	91	3.2	100	1.4
p469k	42	7.8	52	6.8	84	4.9	78	4.4	96	3.6	75	4.2
p89k	70	5.3	86	3.9	92	3.4	96	2.2	96	2.9	96	1.8
p100k	63	5.6	79	4.5	89	3.5	94	2.6	94	2.8	97	1.9

Table A.15.: Swap bridge fault diagnosis with limited failure information in small benchmarks. *%t10*: probability of top10 hit, *#a*: average number of PFA attempts.

	1st fail				4th fail				8th fail			
	Slat		Pointer		Slat		Pointer		Slat		Pointer	
circuit	%t10	#a	%t10	#a	%t10	#a	%t10	#a	%t10	#a	%t10	#a
p81k	51	7.2	73	5.5	88	4.0	98	2.3	95	2.9	100	1.7
b18_1	76	5.4	83	4.4	90	3.6	94	2.5	96	2.9	96	2.0
b18	71	5.8	79	4.7	89	3.8	93	2.7	96	3.0	96	2.0
p141k	77	4.9	94	3.5	93	2.9	98	2.0	97	2.3	99	1.6
b19_1	64	6.1	73	5.1	87	3.8	89	2.9	94	3.1	95	2.2
b19	69	5.8	78	4.6	86	3.9	92	2.7	93	3.2	96	2.0
p239k	54	6.2	75	4.8	85	3.8	94	2.5	93	2.8	98	1.7
p267k	75	4.7	94	3.3	93	2.8	98	2.0	94	2.4	99	1.6
p269k	74	4.9	93	3.4	93	2.8	98	2.0	96	2.3	99	1.6
p279k	72	5.1	92	3.4	92	2.8	99	1.8	95	2.3	99	1.4
p295k	75	4.7	89	3.4	91	3.3	94	2.4	94	2.7	99	1.7
p259k	63	5.8	78	4.8	86	3.8	95	2.5	92	3.0	99	1.8
p330k	66	5.7	82	4.3	89	3.5	96	2.5	93	2.9	98	2.0
p286k	70	5.6	87	4.2	90	3.5	98	2.1	95	2.8	99	1.6
p378k	52	6.9	76	5.5	83	4.4	99	2.3	93	2.9	100	1.5
p418k	72	5.0	94	3.5	91	3.0	98	2.0	95	2.4	99	1.5
p388k	67	5.9	81	4.7	89	3.6	96	2.3	94	2.9	98	1.7
p500k	71	5.3	88	4.0	91	3.3	96	2.3	95	2.7	99	1.7
p483k	51	6.5	71	5.2	81	4.1	90	3.1	92	3.0	96	2.1
p533k	69	5.4	80	4.5	87	3.8	95	2.5	93	3.0	99	1.8
p874k	69	5.2	88	3.6	90	3.2	98	2.1	94	2.6	99	1.7
p951k	62	5.7	88	3.9	86	3.5	97	2.1	90	2.9	99	1.6

Table A.16.: Swap bridge fault diagnosis with limited failure information in large benchmarks. *%t10*: probability of top10 hit, *#a*: average number of PFA attempts.

circuit	$p'l'$	$p'l'k'$	k	l	p	pl	Δt	Δr	Δfc
p45k	710289	68898033	485	67	2323	155641	4.6X	443X	-0.01
p35k	524680	12067640	138	22	4027	88594	5.9X	136X	0.00
p77k	178752	2323776	143	28	2023	56644	3.2X	41X	-0.28
p78k	5184	336960	195	22	114	2508	2.1X	134X	0.00
p469k	221684	221684	60	12	304	3648	60.8X	61X	0.22
p89k	1042929	18772722	306	57	3391	193287	5.4X	97X	0.00
p100k	1626768	29281824	270	53	2393	126829	12.8X	231X	-0.02
p81k	685368	5482944	144	29	1857	53853	12.7X	102X	0.00
p141k	781974	18767376	264	45	2723	122535	6.4X	153X	0.00
p239k	584821	23392840	360	61	3628	221308	2.6X	106X	0.00
p267k	560690	25231050	360	62	4819	298778	1.9X	84X	0.03
p269k	569088	25608960	360	62	4822	298964	1.9X	86X	0.03
p279k	526792	28973560	385	59	5367	316653	1.7X	91X	0.04
p295k	7180204	78982244	330	62	8247	511314	14.0X	154X	0.00
p259k	675168	27006720	360	61	4147	252967	2.7X	107X	0.00
p330k	1682002	107648128	320	64	8700	556800	3.0X	193X	0.07
p286k	896896	49329280	385	60	6560	393600	2.3X	125X	0.00
p378k	5376	1747200	325	65	181	11765	0.5X	149X	0.00
p418k	1938050	124035200	576	93	7384	686712	2.8X	181X	0.00
p388k	520275	26013750	400	66	3653	241098	2.2X	108X	0.02
p500k	1048992	79723392	456	75	9597	719775	1.5X	111X	0.08
p483k	432900	30735900	568	113	3806	430078	1.0X	71X	0.23
p533k	663300	47094300	568	113	6126	692238	1.0X	68X	0.04
p874k	1429740	84354660	531	87	10756	935772	1.5X	90X	0.05
p951k	3134870	257059340	820	139	7500	1042500	3.0X	247X	0.02

Table A.17.: Reduction of test time and response data volume with extreme compaction. $p'l'$: original number of scan cycles, $p'l'k'$: original response data in bits, k: new scan chain count, l: new maximum scan chain length, p: new stuck–at test set size, pl: new number of scan cycles (=response data in bits), Δt, Δr, Δfc: change in test time, response data volume and fault coverage.

	full response				parity data			
circuit	s	%t10	%t1	#a	s	%t10	%t1	#a
p45k	1.1	100	88	1.06	1.1	100	88	1.06
p35k	1.2	100	86	1.10	1.2	100	86	1.09
p77k	10.4	94	79	1.70	10.3	94	79	1.73
p78k	1.2	100	81	1.09	1.2	100	80	1.10
p469k	2.2	100	53	1.60	2.2	100	53	1.61
p89k	1.2	100	87	1.08	1.2	100	88	1.08
p100k	1.1	100	90	1.05	1.1	100	91	1.05
p81k	1.0	100	98	1.02	1.0	100	98	1.02
p141k	1.1	100	92	1.04	1.1	100	93	1.04
p239k	1.1	100	91	1.05	1.1	100	90	1.05
p267k	1.1	100	93	1.05	1.1	100	94	1.04
p269k	1.1	100	92	1.05	1.1	100	92	1.04
p279k	1.1	100	92	1.05	1.1	100	92	1.07
p295k	1.1	100	94	1.05	1.1	100	92	1.06
p259k	1.1	100	87	1.07	1.1	100	89	1.06
p330k	1.1	100	88	1.07	1.2	100	87	1.09
p286k	1.1	100	92	1.05	1.1	100	91	1.05
p378k	1.2	100	80	1.10	1.2	100	80	1.10
p418k	1.1	100	91	1.05	1.1	100	90	1.05
p388k	1.1	100	90	1.05	1.1	100	91	1.05
p500k	1.2	100	91	1.09	1.2	100	91	1.09
p483k	1.1	100	89	1.07	1.2	100	86	1.08
p533k	1.1	100	90	1.06	1.1	100	90	1.06
p874k	1.1	100	92	1.05	1.1	100	93	1.04
p951k	1.1	100	94	1.04	1.2	100	92	1.08

Table A.18.: Diagnostic success on full response data and on parity bits for stuck–at faults. *s*: Number of suspects, *%t10*: Top10 success rate, *%t1*: Perfect diagnosis rate, *#a*: Number of PFA attempts.

	full response				parity data			
circuit	s	%t10	%t1	#a	s	%t10	%t1	#a
p45k	3.5	99	86	1.28	11.0	99	89	1.21
p35k	12.1	96	78	1.83	19.2	96	77	1.79
p77k	237.3	88	68	2.51	266.7	87	70	2.55
p78k	3.2	100	85	1.22	27.9	95	77	1.79
p469k	7.8	93	78	1.94	13.4	93	76	1.93
p89k	7.6	97	78	1.59	35.1	97	81	1.56
p100k	4.9	99	87	1.31	23.2	99	89	1.25
p81k	3.4	100	92	1.17	19.4	99	93	1.20
p141k	3.0	100	92	1.14	20.8	99	93	1.16
p239k	3.1	100	90	1.15	14.7	100	92	1.12
p267k	3.4	100	89	1.23	27.8	99	93	1.13
p269k	3.5	99	86	1.30	20.7	100	92	1.13
p279k	4.2	99	87	1.32	34.8	98	89	1.33
p295k	6.9	98	86	1.40	84.4	97	86	1.46
p259k	3.1	100	90	1.16	28.1	100	92	1.13
p330k	3.4	100	93	1.12	14.4	100	94	1.10
p286k	6.3	98	85	1.42	48.9	99	90	1.28
p378k	3.2	100	83	1.23	12.8	99	86	1.31
p418k	2.9	99	90	1.22	47.2	99	93	1.17
p388k	3.6	100	90	1.15	28.3	100	94	1.09
p500k	4.5	98	90	1.29	23.1	99	94	1.16
p483k	4.8	99	83	1.32	12.8	99	88	1.19
p533k	6.1	98	86	1.34	19.4	99	92	1.13
p874k	4.1	99	85	1.38	38.9	99	92	1.17
p951k	3.2	99	90	1.16	17.1	100	91	1.14

Table A.19.: Diagnostic success on full response data and on parity bits for gross delay faults. See table A.18 for column definitions.

	full response				parity data			
circuit	s	%t10	%t1	#a	s	%t10	%t1	#a
p45k	421.3	99	87	1.30	9309.5	76	50	3.84
p35k	6024.9	93	74	2.19	14438.6	65	36	4.99
p77k	7914.5	73	64	3.57	21680.8	52	28	6.10
p78k	296.6	100	86	1.19	22091.0	50	36	5.83
p469k	7335.4	53	28	5.81	12188.7	46	23	6.49
p89k	473.4	96	85	1.52	15789.9	67	39	4.89
p100k	609.6	99	88	1.22	16369.6	62	42	5.05
p81k	800.5	99	93	1.17	28104.3	57	50	5.02
p141k	807.2	100	90	1.19	41698.8	51	26	6.33
p239k	387.6	100	90	1.12	36866.0	54	37	5.63
p267k	341.8	100	93	1.11	51486.2	51	24	6.34
p269k	392.8	100	93	1.12	53533.6	51	28	6.25
p279k	431.8	100	96	1.07	56333.4	51	35	5.99
p295k	690.7	99	93	1.17	38364.5	70	47	4.42
p259k	431.7	100	79	1.25	51165.7	59	39	5.41
p330k	675.3	99	84	1.31	63141.2	63	39	5.03
p286k	554.1	100	80	1.23	71109.2	57	41	5.35
p378k	307.0	100	86	1.16	57097.7	75	61	3.51
p418k	318.7	100	92	1.09	60087.6	45	30	6.56
p388k	640.1	100	84	1.22	86383.7	61	43	5.09
p500k	435.5	100	91	1.19	67505.0	45	30	6.47
p483k	626.1	99	86	1.30	47641.8	65	44	4.72
p533k	627.5	99	80	1.32	64437.7	56	40	5.44
p874k	441.1	100	92	1.12	84874.5	50	33	6.20
p951k	305.1	100	92	1.11	71294.4	51	34	5.91

Table A.20.: Diagnostic success on full response data and on parity bits for swap bridge faults. See table A.18 for column definitions.

B. Bibliography

[AbramB1980a] M. Abramovici and M. A. Breuer. Multiple fault diagnosis in combinational circuits based on an effect-cause analysis. *IEEE Trans. on Computers*, C-29(6):451–460, June 1980. doi: 10.1109/TC.1980.1675604.

[AbramBF1990] M. Abramovici, M. A. Breuer, and A. D. Friedman. *Digital Systems Testing and Testable Design*. IEEE Press, 1990. ISBN 0-7803-1062-4.

[AbramMM1983] M. Abramovici, P. R. Menon, and D. T. Miller. Critical path tracing - an alternative to fault simulation. In *Proc. 20th IEEE/ACM Design Automation Conference (DAC)*, 1983, pp. 214–220. doi: 10.1109/DAC.1983.1585651.

[AbramB1980] M. Abramovici and M. A. Breuer. Fault diagnosis based on effect-cause analysis: An introduction. In *Proc. 17th IEEE/ACM Design Automation Conference (DAC)*, 1980, pp. 69–76. doi: 10.1145/800139.804514.

[AckenM1991] J. M. Acken and S. D. Millman. Accurate modeling and simulation of bridging faults. In *Proc. IEEE Custom Integrated Circuits Conference (CICC)*, 1991, pp. 17.4/1–17.4/4. doi: 10.1109/CICC.1991.164111.

[AckenM1992] J. M. Acken and S. D. Millman. Fault model evolution for diagnosis: Accuracy vs precision. In *Proc. IEEE Custom Integrated Circuits Conference (CICC)*, 1992, pp. 13.4/1–13.4/4. doi: 10.1109/CICC.1992.591298.

[Aitke1995] R. C. Aitken. Finding defects with fault models. In *Proc. IEEE International Test Conference (ITC)*, 1995, pp. 498–505. doi: 10.1109/TEST.1995.529877.

[AmyeeNV2006] M. E. Amyeen, D. Nayak, and S. Venkataraman. Improving precision using mixed-level fault diagnosis. In *Proc. IEEE Interna-*

tional Test Conference (ITC), 2006, p. 22.3. doi: 10.1109/TEST.2006.297661.

[ArumiRF2008] D. Arumí, R. Rodríguez-Montañés, and J. Figueras. Experimental characterization of CMOS interconnect open defects. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 27(1):123–136, January 2008. doi: 10.1109/TCAD.2007.907255.

[BardeM1982] P. H. Bardell and W. H. McAnney. Self-testing of multichip logic modules. In *Proc. IEEE International Test Conference (ITC)*, 1982, pp. 200–204.

[Barte2000] T. Bartenstein. Fault distinguishing pattern generation. In *Proc. IEEE International Test Conference (ITC)*, 2000, pp. 820–828. doi: 10.1109/TEST.2000.894285.

[BarteHH*2001] T. Bartenstein, D. Heaberlin, L. M. Huisman, and D. Sliwinski. Diagnosing combinational logic designs using the single location at-a-time (SLAT) paradigm. In *Proc. IEEE International Test Conference (ITC)*, 2001, pp. 287–296. doi: 10.1109/TEST.2001.966644.

[BastaWA2008] P. Bastani, L.-C. Wang, and M. S. Abadir. Linking statistical learning to diagnosis. *IEEE Design & Test of Computers*, 25(3):232–239, May 2008. doi: 10.1109/MDT.2008.79.

[BhattB2006] N. K. Bhatti and R. D. Blanton. Diagnostic test generation for arbitrary faults. In *Proc. IEEE International Test Conference (ITC)*, 2006, p. 19.2. doi: 10.1109/TEST.2006.297647.

[Black1967] J. Black. Mass transport of aluminum by momentum exchange with conducting electrons. In *Proc. International Reliability Physics Symposium*, 1967, pp. 148–159.

[BlantDD2006] R. D. Blanton, K. N. Dwarakanath, and R. Desineni. Defect modeling using fault tuples. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 25(11):2450–2464, 2006. doi: 10.1109/TCAD.2006.870836.

[BoppaF1998] V. Boppana and M. Fujita. Modeling the unknown! towards model-independent fault and error diagnosis. In *Proc. IEEE International Test Conference (ITC)*, 1998, pp. 1094–1100. doi: 10.1109/TEST.1998.743310.

[BreauC2008] L. Breaux and S. Collins. *Handbook of Semiconductor Manufacturing Technology*, chapter 27: Yield Management, p. 27.1. CRC Press London, 2008. ISBN 9781574446753.

[BreueCS1976] M. A. Breuer, S.-J. Chang, and S. Y. H. Su. Identification of multiple stuck-type faults in combinational networks. *IEEE Trans. on Computers*, C-25(1):44–54, January 1976. doi: 10.1109/TC.1976.5009204.

[BrgleBK1989] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. In *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, 1989, vol. 3, pp. 1929–1934. doi: 10.1109/ISCAS.1989.100747.

[BushnA2000] M. L. Bushnell and V. D. Agrawal. *Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits*. Springer, November 2000. ISBN 0792379918.

[CamurMP*1990] P. Camurati, D. Medina, P. Prinetto, and M. Sonza Reorda. A diagnostic test pattern generation algorithm. In *Proc. IEEE International Test Conference (ITC)*, 1990, pp. 52–58. doi: 10.1109/TEST.1990.114000.

[CarteIR1987] J. L. Carter, V. S. Iyengar, and B. K. Rosen. Efficient test coverage determination for delay faults. In *Proc. IEEE International Test Conference (ITC)*, 1987, pp. 418–427.

[Case1976] G. R. Case. Analysis of actual fault mechanisms in CMOS logic gates. In *Proc. 13th IEEE/ACM Design Automation Conference (DAC)*, 1976, pp. 265–270. doi: 10.1145/800146.804823.

[ChenRPR2006] G. Chen, S. M. Reddy, I. Pomeranz, and J. Rajski. A test pattern ordering algorithm for diagnosis with truncated fail data. In *Proc. 43rd IEEE/ACM Design Automation Conference (DAC)*, 2006, pp. 399–404. doi: 10.1145/1146909.1147015.

[ChenGB1997] W. Chen, S. K. Gupta, and M. A. Breuer. Analytic models for crosstalk delay and pulse analysis under non-ideal inputs. In *Proc. IEEE International Test Conference (ITC)*, 1997, pp. 809–818. doi: 10.1109/TEST.1997.639695.

[ChenSMV2010] Y. Chen, S. Safarpour, J. Marques-Silva, and A. Veneris. Automated design debugging with maximum satisfiability. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 29(11):1804–1817, November 2010. doi: 10.1109/TCAD.2010.2061270.

[Cheng2004] W.-T. Cheng, K.-H. Tsai, Y. Huang, N. Tamarapalli, and J. Rajski. Compactor independent direct diagnosis. In *Proc. 13th Asian Test Symposium (ATS)*, 2004, pp. 204–209. doi: 10.1109/ATS.2004.32.

[ChessL1999] B. Chess and T. Larrabee. Creating small fault dictionaries. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 18(3):346–356, March 1999. doi: 10.1109/43.748164.

[ChianK2007] C. C. Chiang and J. Kawa. *Design for Manufacturability and Yield for Nano–Scale CMOS*. Springer, 2007. ISBN 978-1-4020-5187-6.

[CookEWA2011] A. Cook, M. Elm, H. Wunderlich, and U. Abelein. Structural in-field diagnosis for random logic circuits. In *Proc. 16th IEEE European Test Symposium (ETS)*, 2011, pp. 111–116. doi: 10.1109/ETS.2011.25.

[CookHW2012] A. Cook, S. Hellebrand, and H.-J. Wunderlich. Built-in self-diagnosis exploiting strong diagnostic windows in mixed-mode test. In *Proc. 17th IEEE European Test Symposium (ETS)*, 2012, pp. 146–151. doi: 10.1109/ETS.2012.6233025.

[CornoRS2000] F. Corno, M. S. Reorda, and G. Squillero. RT-level ITC'99 benchmarks and first ATPG results. *IEEE Design Test of Computers*, 17(3):44–53, Jul.-Sep. 2000. doi: 10.1109/54.867894.

[CoxR1988a] H. Cox and J. Rajski. A method of fault analysis for test generation and fault diagnosis. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 7(7):813–833, July 1988. doi: 10.1109/43.3952.

[DalirHE*2012] A. Dalirsani, S. Holst, M. Elm, and H.-J. Wunderlich. Structural test for graceful degradation of NoC switches. *Journal of Electronic Testing – Theory and Applications (JETTA), published online*, October 2012. doi: 10.1007/s10836-012-5329-9.

[DalirHE*2011] A. Dalirsani, S. Holst, M. Elm, and H.-J. Wunderlich. Structural test for graceful degradation of NoC switches. In *Proc. 16th European Test Symposium (ETS)*, 2011, pp. 183–188. doi: 10.1109/ETS.2011.33.

[Das2000] S. R. Das, T. F. Barakat, E. M. Petriu, M. H. Assaf, and K. Chakrabarty. Space compression revisited. *IEEE Trans. on Instrumentation and Measurement*, 49(3):690–705, June 2000. doi: 10.1109/19.850416.

[David1999] S. Davidson. ITC'99 benchmark circuits - preliminary results. In *Proc. IEEE International Test Conference (ITC)*, 1999, p. 1125. doi: 10.1109/TEST.1999.805857.

[DegraKD*2001] R. Degraeve, B. Kaczer, A. De Keersgieter, and G. Groeseneken. Relation between breakdown mode and location in short-channel nMOSFETs and its impact on reliability specifications. *IEEE Trans. on Device and Materials Reliability*, 1(3):163–169, September 2001. doi: 10.1109/7298.974832.

[DesinPB2006] R. Desineni, O. Poku, and R. D. Blanton. A logic diagnosis methodology for improved localization and extraction of accurate defect behavior. In *Proc. IEEE International Test Conference (ITC)*, 2006, p. 12.3. doi: 10.1109/TEST.2006.297627.

[EicheW1977] E. B. Eichelberger and T. W. Williams. A logic design structure for LSI testability. In *Proc. 14th IEEE/ACM Design Automation Conference (DAC)*, 1977, pp. 462–468.

[Eldre1959] R. D. Eldred. Test routines based on symbolic logical statements. *J. ACM*, 6(1):33–37, 1959. doi: 10.1145/320954.320957.

[Elm2011] M. Elm. *Embedded Hardware Structures for Efficient Volume and In-Field Diagnosis of Random Logic Circuits.* VDE Verlag, 2011. ISBN 978-3-8007-3396-5.

[ElmW2008] M. Elm and H.-J. Wunderlich. Scan chain organization for embedded diagnosis. In *Proc. IEEE Design, Automation and Test in Europe (DATE)*, 2008, pp. 468–473. doi: 10.1109/DATE.2008.4484725.

[EmmerSB2000] J. M. Emmert, C. E. Stroud, and J. R. Bailey. A new bridging fault model for more accurate fault behavior. In *Proc. IEEE*

AUTOTESTCON, 2000, pp. 481–485. doi: 10.1109/AUTEST.2000.885628.

[Engel2009] P. Engelke. *Resistive Bridging Faults - Defect-Oriented Modeling and Efficient Testing.* Ph.D. thesis, Albert-Ludwigs-Universität Freiburg im Breisgau, 2009.

[FavalDO*1993] M. Favalli, M. Dalpasso, P. Olivo, and B. Ricco. Analysis of dynamic effects of resistive bridging faults in CMOS and BiCMOS digital ICs. In *Proc. IEEE International Test Conference (ITC)*, 1993, pp. 865–874. doi: 10.1109/TEST.1993.470614.

[FeySVD2006] G. Fey, S. Safarpour, A. G. Veneris, and R. Drechsler. On the relation between simulation-based and SAT-based diagnosis. In *Proc. Design, Automation and Test in Europe (DATE)*, 2006, pp. 1139–1144. doi: 10.1145/1131796.

[GanesK2010] K. Ganeshpure and S. Kundu. On ATPG for multiple aggressor crosstalk faults. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 29(5):774–787, May 2010. doi: 10.1109/TCAD.2010.2043589.

[GhoshT2000] J. Ghosh-Dastidar and N. A. Touba. Diagnosing resistive bridges using adaptive techniques. In *Proc. IEEE Custom Integrated Circuits Conference (CICC)*, 2000, pp. 79–82. doi: 10.1109/CICC.2000.852622.

[GhoshT2000a] J. Ghosh-Dastidar and N. A. Touba. A rapid and scalable diagnosis scheme for BIST environments with a large number of scan chains. In *Proc. 18th IEEE VLSI Test Symposium*, 2000, pp. 79–85. doi: 10.1109/VTEST.2000.843830.

[GieleWM*2008] G. Gielen, P. D. Wit, E. Maricau, J. Loeckx, J. Martin-Martinez, B. Kaczer, G. Groeseneken, R. Rodríguez, and M. Nafría. Emerging yield and reliability challenges in nanometer CMOS technologies. In *Proc. IEEE Design, Automation and Test in Europe (DATE)*, 2008, pp. 1322–1327. doi: 10.1109/DATE.2008.4484862.

[GirarLP1992] P. Girard, C. Landrault, and S. Pravossoudovitch. Delay-fault diagnosis by critical-path tracing. *IEEE Design Test of Computers*, 9(4):27–32, December 1992. doi: 10.1109/54.173329.

[GirarLP*1992a] P. Girard, C. Landrault, and S. Pravossoudovitch. A novel approach to delay-fault diagnosis. In *Proc. 29th ACM/IEEE Design Automation Conference (DAC)*, 1992, pp. 357–360. doi: 10.1109/DAC.1992.227778.

[GongC1995] Y. Gong and S. Chakravarty. On adaptive diagnostic test generation. In *Proc. IEEE International Conference on Computer-Aided Design (ICCAD)*, 1995, pp. 181–184. doi: 10.1109/ICCAD.1995.480010.

[HakimN1984] S. L. Hakimi and K. Nakajima. On adaptive system diagnosis. *IEEE Trans. on Computers*, 33(3):234–240, 1984. doi: 10.1109/TC.1984.1676420.

[HakmiHW*2009] A.-W. Hakmi, S. Holst, H.-J. Wunderlich, J. Schloeffel, F. Hapke, and A. Glowatz. Restrict encoding for mixed-mode BIST. In *Proc. 27th VLSI Test Symposium (VTS)*, 2009, pp. 179–184. doi: 10.1109/VTS.2009.43.

[HigamST*2006] Y. Higami, K. K. Saluja, H. Takahashi, S. Kobayashi, and Y. Takamatsu. Diagnosis of transistor shorts in logic test environment. In *Proc. 15th Asian Test Symposium (ATS)*, 2006, pp. 354–359. doi: 10.1109/ATS.2006.260955.

[HillePE*2008] S. Hillebrecht, I. Polian, P. Engelke, B. Becker, M. Keim, and W.-T. Cheng. Extraction, simulation and test generation for interconnect open defects based on enhanced aggressor-victim model. In *Proc. 39th IEEE International Test Conference (ITC)*, 2008, p. 33.3. doi: 10.1109/TEST.2008.4700642.

[HolstW2007a] S. Holst and H.-J. Wunderlich. Adaptive debug and diagnosis without fault dictionaries. In *19. GI/GMM/ITG Workshop für Testmethoden und Zuverlässigkeit von Schaltungen und Systemen (TuZ)*, 2007, pp. 82–86.

[HolstW2009b] 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, Aug 2009. doi: 10.1007/s10836-009-5109-3.

[HolstSW2012] S. Holst, E. Schneider, and H.-J. Wunderlich. Scan test power simulation on GPGPUs. In *Proc. 21st IEEE Asian Test Symposium (ATS), to appear*, 2012.

[HolstW2007] S. Holst and H.-J. Wunderlich. Adaptive debug and diagnosis without fault dictionaries. In *Proc. 12th European Test Symposium (ETS)*, 2007, pp. 7–12. doi: 10.1109/ETS.2007.9.

[HolstW2009a] S. Holst and H.-J. Wunderlich. Diagnose mit extrem kompaktierten Fehlerdaten. In *21. Workshop für Testmethoden und Zuverlässigkeit von Schaltungen und Systemen (TuZ)*, 2009, pp. 15–20.

[HolstW2009] S. Holst and H.-J. Wunderlich. A diagnosis algorithm for extreme space compaction. In *Proc. 12th Design, Automation and Test in Europe (DATE)*, 2009, pp. 1355–1360.

[HoraSEL2002] C. Hora, R. Segers, S. Eichenberger, and M. Lousberg. An effective diagnosis method to support yield improvement. In *Proc. IEEE International Test Conference (ITC)*, 2002, pp. 260–269. doi: 10.1109/TEST.2002.1041768.

[Huang2001] S.-Y. Huang. On improving the accuracy of multiple defect diagnosis. In *Proc. 19th IEEE VLSI Test Symposium (VTS)*, 2001, pp. 34–39. doi: 10.1109/VTS.2001.923415.

[HuangCC*1997] S.-Y. Huang, K.-T. Cheng, K.-C. Chen, and D. I. Cheng. ErrorTracer: A fault simulation-based approach to design error diagnosis. In *Proc. IEEE International Test Conference (ITC)*, 1997, pp. 974–981. doi: 10.1109/TEST.1997.639713.

[Huism2004] L. M. Huisman. Diagnosing arbitrary defects in logic designs using single location at a time (SLAT). *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, 23(1):91–101, January 2004. doi: 10.1109/TCAD.2003.816206.

[Huism2005] L. M. Huisman. *Data Mining and Diagnosing IC Fails.* Springer, September 2005. ISBN 0387249931.

[Hutch2008] G. D. Hutcheson. *Handbook of Semiconductor Manufacturing Technology*, chapter 35: Economics of Semiconductor Manufacturing, p. 35.1. CRC Press London, 2008. ISBN 9781574446753.

[Intel2011a] Intel. 22nm announcement pres. http://download.intel.com/newsroom/kits/22nm/pdfs/22nm-Announcement_Presentation.pdf, April 2011.

[Intel2011] Intel. Press release: Intel to invest more than $5 billion to build new factory in arizona. http://newsroom.intel.com/community/intel_newsroom/blog/2011/02/18/intel-to-invest-more-than-5-billion-to-build-new-factory-in-arizona, February 2011.

[ITRS2011] ITRS. International technology roadmap for semiconductors. http://www.itrs.net/Links/2011ITRS/Home2011.htm, 2011.

[JainA1985a] S. K. Jain and V. D. Agrawal. Modeling and test generation algorithms for MOS circuits. *IEEE Trans. on Computers*, C-34(5):426–433, May 1985. doi: 10.1109/TC.1985.1676582.

[KeimTTS*2006] M. Keim, N. Tamarapalli, H. Tang, M. Sharma, J. Rajski, C. Schuermyer, and B. Benware. A rapid yield learning flow based on production integrated layout-aware diagnosis. In *Proc. IEEE International Test Conference (ITC)*, 2006, pp. 7.1/1–7.1/10. doi: 10.1109/TEST.2006.297715.

[Kelle1996] B. L. Keller. Hierarchical pattern faults for describing logic circuit failure mechanisms. *US Patent No 5546408*, August 1996.

[KhursRA*2008] S. Khursheed, P. Rosinger, B. M. Al-Hashimi, S. M. Reddy, and P. Harrod. Bridge defect diagnosis for multiple-voltage design. In *Proc. 13th IEEE European Test Symposium (ETS)*, 2008, pp. 99–104. doi: 10.1109/ETS.2008.14.

[KinsmON2006] A. B. Kinsman, S. Ollivierre, and N. Nicolici. Diagnosis of logic circuits using compressed deterministic data and on-chip response comparison. *IEEE Trans. on VLSI Systems*, 14(5):537–548, May 2006. doi: 10.1109/TVLSI.2006.876109.

[KochtHE*2009] M. Kochte, S. Holst, M. Elm, and H.-J. Wunderlich. Test encoding for extreme response compaction. In *Proc. 14th European Test Symposium (ETS)*, 2009, pp. 155–160. doi: 10.1109/ETS.2009.22.

[KochtSW*2010] M. A. Kochte, M. Schaal, H.-J. Wunderlich, and C. G. Zoellin. Efficient fault simulation on many-core processors. In *Proc. 46th*

IEEE/ACM Design Automation Conference (DAC), 2010, pp. 380–385. doi: 10.1145/1837274.1837369.

[KrstiC1998] A. Krstic and K.-T. Cheng. *Delay Fault Testing for VLSI Circuits*. Springer, 1998. ISBN 978-0-7923-8295-9.

[KuhnLK2010] K. J. Kuhn, M. Y. Liu, and H. Kennel. Technology options for 22nm and beyond. In *Proc. International Workshop on Junction Technology (IWJT)*, 2010. doi: 10.1109/IWJT.2010.5475000.

[KunduSG2006] S. Kundu, S. Sengupta, and D. Goswami. Generalized fault model for defects and circuit marginalities. *US Patent No 7036063*, April 2006.

[KunduZC*2005] S. Kundu, S. T. Zachariah, Y.-S. Chang, and C. Tirumurti. On modeling crosstalk faults. *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, 24(12):1909–1915, 2005. doi: 10.1109/TCAD.2005.852670.

[LadhaM2010] A. Ladhar and M. Masmoudi. An effective and accurate methodology for the cell internal defect diagnosis. *Journal of Electronic Testing*, 26:621–639, 2010. doi: 10.1007/s10836-010-5181-8.

[Larra1992] T. Larrabee. Test pattern generation using boolean satisfiability. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 11(1):4–15, January 1992. doi: 10.1109/43.108614.

[LavoHL2002] D. B. Lavo, I. Hartanto, and T. Larrabee. Multiplets, models, and the search for meaning: Improving per-test fault diagnosis. In *Proc. IEEE International Test Conference (ITC)*, 2002, pp. 250–259. doi: 10.1109/TEST.2002.1041767.

[LeiniGM2004] A. Leininger, M. Gössel, and P. Muhmenthaler. Diagnosis of scan-chains by use of a configurable signature register and error-correcting codes. In *Proc. Design, Automation and Test in Europe (DATE)*, 2004, vol. 2, pp. 1302–1307. doi: 10.1109/DATE.2004.1269075.

[LeiniMC*2005] A. Leininger, P. Muhmenthaler, W.-T. Cheng, N. Tamarapalli, W. Yang, and H. Tsai. Compression mode diagnosis enables high volume monitoring diagnosis flow. In *Proc. IEEE International Test Conference (ITC)*, 2005, p. 7.3. doi: 10.1109/TEST.2005.1583972.

[LiXHCL2010] H. Li, D. Xu, Y. Han, K.-T. Cheng, and X. Li. nGFSIM: A GPU-based fault simulator for 1-to-n detection and its applications. In *Proc. IEEE International Test Conference (ITC)*, 2010, pp. 12.1/1–12.1/10. doi: 10.1109/TEST.2010.5699235.

[LiTM2001] J. C. M. Li, C.-W. Tseng, and E. J. McCluskey. Testing for resistive opens and stuck opens. In *Proc. IEEE International Test Conference (ITC)*, 2001, pp. 1049–1058. doi: 10.1109/TEST.2001.966731.

[LiH2010] M. Li and M. S. Hsiao. FSimGP^2: An efficient fault simulator with GPGPU. In *Proc. 19th IEEE Asian Test Symposium (ATS)*, 2010, pp. 15–20. doi: 10.1109/ATS.2010.12.

[Liebm2003] L. W. Liebmann. Layout impact of resolution enhancement techniques: impediment or opportunity? In *Proc. ACM International Symposium on Physical Design (ISPD)*, 2003, pp. 110–117. doi: 10.1145/640000.640026.

[LinPBNL*2008] Y.-T. Lin, O. Poku, R. D. Blanton, P. Nigh, P. Lloyd, and V. Iyengar. Evaluating the effectiveness of physically-aware n-detect test using real silicon. In *Proc. 39th IEEE International Test Conference (ITC)*, 2008, p. 21.3. doi: 10.1109/TEST.2008.4700606.

[LinLC2007] Y.-C. Lin, F. Lu, and K.-T. Cheng. Multiple-fault diagnosis based on adaptive diagnostic test pattern generation. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 26(5):932–942, May 2007. doi: 10.1109/TCAD.2006.884486.

[LiuC2003] C. Liu and K. Chakrabarty. Failing vector identification based on overlapping intervals of test vectors in a scan-BIST environment. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 22(5):593–604, May 2003. doi: 10.1109/TCAD.2003.810739.

[LiuZRCS*2007] C. Liu, W. Zou, S. M. Reddy, W.-T. Cheng, M. Sharma, and H. Tang. Interconnect open defect diagnosis with minimal physical information. In *Proc. IEEE International Test Conference (ITC)*, 2007, p. 7.3. doi: 10.1109/TEST.2007.4437580.

[Liu2007] C. Liu. Improve the quality of per-test fault diagnosis using output information. *Journal of Electronic Testing*, 23(1):11–24, 2007. doi: 10.1007/s10836-006-9442-5.

[MaxweA1993] P. C. Maxwell and R. C. Aitken. Biased voting: A method for simulating CMOS bridging faults in the presence of variable gate logic thresholds. In *Proc. IEEE International Test Conference (ITC)*, 1993, pp. 63–72. doi: 10.1109/TEST.1993.470717.

[McClu1984] E. J. McCluskey. Verification testing - a pseudoexhaustive test technique. *IEEE Trans. on Computers*, 33:541–546, 1984. doi: 10.1109/TC.1984.1676477.

[McCluT2000] E. J. McCluskey and C.-W. Tseng. Stuck-fault tests vs. actual defects. In *Proc. IEEE International Test Conference (ITC)*, 2000, pp. 336–343. doi: 10.1109/TEST.2000.894222.

[McPhe2006] J. W. McPherson. Reliability challenges for 45nm and beyond. In *Proc. 43rd IEEE/ACM Design Automation Conference (DAC)*, 2006, pp. 176–181. doi: 10.1145/1146909.1146959.

[MehtaMT*2006] V. J. Mehta, M. Marek-Sadowska, K.-H. Tsai, and J. Rajski. Timing defect diagnosis in presence of crosstalk for nanometer technology. In *Proc. IEEE International Test Conference (ITC)*, 2006, p. 12.2. doi: 10.1109/TEST.2006.297626.

[Mei1974] K. C. Y. Mei. Bridging and stuck-at faults. *IEEE Trans. on Computers*, 23:720–727, July 1974. doi: 10.1109/T-C.1974.224020.

[MillmMA1990] S. D. Millman, E. J. McCluskey, and J. M. Acken. Diagnosing CMOS bridging faults with stuck-at fault dictionaries. In *Proc. IEEE International Test Conference (ITC)*, 1990, pp. 860–870. doi: 10.1109/TEST.1990.114104.

[MitraK2004] S. Mitra and K. S. Kim. X-compact: an efficient response compaction technique. *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, 23(3):421–432, March 2004. doi: 10.1109/TCAD.2004.823341.

[Moore1965] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117, April 1965.

[Moore1975] G. E. Moore. Progress in digital integrated electronics. In *Proc. International Electron Devices Meeting*, 1975, vol. 21, pp. 11–13.

[MumtaIH*2011] A. Mumtaz, M. E. Imhof, S. Holst, and H.-J. Wunderlich. Eingebetteter Test zur hochgenauen Defekt-Lokalisierung. In *5. GMM/GI/ITG-Fachtagung Zuverlässigkeit und Entwurf (ZuE)*, 2011. VDE Verlag.

[MumtaIH*2011a] A. Mumtaz, M. E. Imhof, S. Holst, and H.-J. Wunderlich. Embedded test for highly accurate defect localization. In *20th IEEE Asian Test Symposium (ATS)*, 2011, pp. 213–218. doi: 10.1109/ATS.2011.60.

[Nakaj1981] K. Nakajima. A new approach to system diagnosis. In *Proc. 19th Annu. Allerton Conf. Commun., Contr. and Comput.*, 1981, pp. 697–706.

[ParkMW1988] E. S. Park, M. R. Mercer, and T. W. Williams. Statistical delay fault coverage and defect level for delay faults. In *Proc. IEEE International Test Conference (ITC)*, 1988, pp. 492–499. doi: 10.1109/TEST.1988.207761.

[PatelLR2003] J. H. Patel, S. S. Lumetta, and S. M. Reddy. Application of saluja-karpovsky compactors to test responses with many unknowns. In *Proc. 21st IEEE VLSI Test Symposium (VTS)*, 2003, pp. 107–112. doi: 10.1109/VTEST.2003.1197640.

[PoehlRB*2006] F. Poehl, J. Rzeha, M. Beck, M. Gössel, R. Arnold, and P. Ossimitz. On-chip evaluation, compensation, and storage of scan diagnosis data - a test time efficient scan diagnosis architecture. In *Proc. 11th IEEE European Test Symposium (ETS)*, 2006, pp. 239–246. doi: 10.1109/ETS.2006.34.

[PokuB2007] O. Poku and R. D. Blanton. Delay defect diagnosis using segment network faults. In *Proc. IEEE International Test Conference (ITC)*, 2007, p. 15.1. doi: 10.1109/TEST.2007.4437602.

[PomerR1992] I. Pomeranz and S. M. Reddy. On the generation of small dictionaries for fault location. In *Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 1992, pp. 272–279. doi: 10.1109/ICCAD.1992.279361.

[PramaR1988] A. K. Pramanick and S. M. Reddy. On the detection of delay faults. In *Proc. IEEE International Test Conference (ITC)*, 1988, pp. 845–856. doi: 10.1109/TEST.1988.207872.

[Prior1957] A. N. Prior. *Time and Modality.* Oxford University Press, 1957. ISBN 978-0198241584.

[RajskT1999] J. Rajski and J. Tyszer. Diagnosis of scan cells in BIST environment. *IEEE Trans. on Computers*, 48(7):724–731, July 1999.

[RajskTW*2005] J. Rajski, J. Tyszer, C. Wang, and S. M. Reddy. Finite memory test response compactors for embedded test applications. *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, 24(4):622–634, 2005. doi: 10.1109/TCAD.2005.844111.

[ReddyRA1984] S. M. Reddy, M. K. Reddy, and V. D. Agrawal. Robust tests for stuck-open faults in CMOS combinational logic circuits. In *Proc. 14th International Fault-Tolerant Computing Symposium*, 1984, p. 44–49.

[RenovHB1994] M. Renovell, P. Huc, and Y. Bertrand. CMOS bridging fault modeling. In *Proc. 12th IEEE VLSI Test Symposium (VTS)*, 1994, pp. 392–297. doi: 10.1109/VTEST.1994.292283.

[RenovHB*1994a] M. Renovell, P. Huc, and Y. Bertrand. A unified model for inter-gate and intra-gate CMOS bridging fault: the configuration ratio. In *Proc. 3rd Asian Test Symposium (ATS)*, 1994, pp. 170–175. doi: 10.1109/ATS.1994.367235.

[RichmB1985] J. Richman and K. R. Bowden. The modern fault dictionary. In *Proc. IEEE International Test Conference (ITC)*, 1985, pp. 696–702.

[RodriAF*2007] R. Rodríguez-Montañés, D. Arumí, J. Figueras, S. Eichenberger, C. Hora, and B. Kruseman. Impact of gate tunnelling leakage on CMOS circuits with full open defects. *Electronics Letters*, 43(21):1140–1141, November 2007. doi: 10.1049/el:20072117.

[RodriAF*2008] R. Rodríguez-Montañés, D. Arumí, J. Figueras, S. Eichenberger, C. Hora, and B. Kruseman. Time-dependent behaviour of full open defects in interconnect lines. In *Proc. 39th IEEE International Test Conference (ITC)*, 2008, p. 10.1. doi: 10.1109/TEST.2008.4700575.

[RodriAF*2010] R. Rodríguez-Montañés, D. Arumí, J. Figueras, S. Eichenberger, C. Hora, and B. Kruseman. Diagnosis of full open defects in interconnect lines with fan-out. In *Proc. 15th IEEE European Test Symposium (ETS)*, 2010, pp. 233–238. doi: 10.1109/ETSYM.2010.5512752.

[RodriAF*2007] R. Rodríguez-Montañés, D. Arumí, J. Figueras, S. Eichenberger, C. Hora, B. Kruseman, M. Lousberg, and A. K. Majhi. Diagnosis of full open defects in interconnecting lines. In *Proc. 25th IEEE VLSI Test Symposium (VTS)*, 2007, pp. 158–166. doi: 10.1109/VTS.2007.28.

[Roth1966] J. P. Roth. Diagnosis of automata failures: A calculus and a method. *IBM Journal of Research and Development*, 10(4):278–291, July 1966. doi: 10.1147/rd.104.0278.

[RothBS1967] J. P. Roth, W. G. Bouricius, and P. R. Schneider. Programmed algorithms to compute tests to detect and distinguish between failures in logic circuits. *IEEE Trans. on Electronic Computers*, EC-16(5):567–580, October 1967. doi: 10.1109/PGEC.1967.264743.

[RoussBG*2007a] A. Rousset, A. Bosio, P. Girard, C. Landrault, S. Pravossoudovitch, and A. Virazel. Fast bridging fault diagnosis using logic information. In *Proc. 16th Asian Test Symposium (ATS)*, 2007, pp. 33–38. doi: 10.1109/ATS.2007.75.

[RoussBG*2007] A. Rousset, A. Bosio, P. Girard, C. Landrault, S. Pravossoudovitch, and A. Virazel. DERRIC: A tool for unified logic diagnosis. In *Proc. 12th European Test Symposium (ETS)*, 2007, pp. 13–20. doi: 10.1109/ETS.2007.16.

[RuppS2011] K. Rupp and S. Selberherr. The economic limit to Moore's law. *IEEE Trans. on Semiconductor Manufacturing*, 24(1):1–4, February 2011. doi: 10.1109/TSM.2010.2089811.

[SafarMV*2007] S. Safarpour, H. Mangassarian, A. Veneris, M. H. Liffiton, and K. A. Sakallah. Improved design debugging using maximum satisfiability. In *Proc. Formal Methods in Computer Aided Design (FMCAD)*, 2007, pp. 13–19. doi: 10.1109/FAMCAD.2007.26.

[SalujK1983] K. K. Saluja and M. Karpovsky. Testing computer hardware through data compression in space and time. In *Proc. IEEE International Test Conference (ITC)*, 1983, p. 83–89.

[Sapat1994] S. Sapatnekar. *Timing*. Kluwer Academic Publishers, 1994. ISBN 1-4020-7671-1.

[SatoSSY*2006] Y. Sato, K. Sugiura, R. Shimoda, Y. Yoshizawa, K. Norimatsu, and M. Sanada. Defect diagnosis - reasoning methodology. In *Proc. 15th Asian Test Symposium (ATS)*, 2006, pp. 209–214. doi: 10.1109/ATS.2006.261022.

[Savir1998] J. Savir. Salvaging test windows in BIST diagnostics. *IEEE Trans. on Computers*, 47(4):486–491, April 1998. doi: 10.1109/12.675718.

[Schul1988] M. H. Schulz. *Testmustergenerierung und Fehlersimulation in digitalen Schaltungen mit hoher Komplexität*, vol. 173 of *Informatik-Fachberichte*. Springer, 1988. ISBN 3-540-50051-0.

[Segura2004] J. Segura and C. F. Hawkins. *CMOS Electronics: How It Works, How It Fails*. John Wiley & Sons, 2004. ISBN 0-471-47669-2.

[SharmCT*2007] M. Sharma, W.-T. Cheng, T.-P. Tai, Y. S. Cheng, W. Hsu, C. Liu, S. M. Reddy, and A. Mann. Faster defect localization in nanometer technology based on defective cell diagnosis. In *Proc. IEEE International Test Conference (ITC)*, 2007, p. 15.3. doi: 10.1109/TEST.2007.4437604.

[SharmBL*2008] M. Sharma, B. Benware, L. Ling, D. Abercrombie, L. Lee, M. Keim, H. Tang, W.-T. Cheng, T.-P. Tai, Y.-J. Chang, R. Lin, and A. Man. Identifying physical root causes for yield excursions from test fail data. In *Proc. 13th IEEE European Test Symposium (ETS)*, 2008.

[SmithVV2004] A. Smith, A. Veneris, and A. Viglas. Design diagnosis using boolean satisfiability. In *Proc. IEEE Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2004, pp. 218–223. doi: 10.1109/ASPDAC.2004.1337569.

[SodenTT*1989] J. M. Soden, R. K. Treece, M. R. Taylor, and C. F. Hawkins. CMOS IC stuck-open-fault electrical effects and design considerations.

In *Proc. IEEE International Test Conference (ITC)*, 1989, pp. 423–430. doi: 10.1109/TEST.1989.82325.

[SpinnJP*2007] S. Spinner, J. Jiang, I. Polian, P. Engelke, and B. Becker. Simulating open-via defects. In *Proc. 16th Asian Test Symposium (ATS)*, 2007, pp. 265–270. doi: 10.1109/ATS.2007.72.

[TakahPH*2001] H. Takahashi, M. Phadoongsidhi, Y. Higami, K. K. Saluja, and Y. Takamatsu. Simulation-based diagnosis for crosstalk faults in sequential circuits. In *Proc. 10th Asian Test Symposium (ATS)*, 2001, pp. 63–68. doi: 10.1109/ATS.2001.990260.

[TamPB2010] W. C. Tam, O. Poku, and R. D. Blanton. Systematic defect identification through layout snippet clustering. In *Proc. IEEE International Test Conference (ITC)*, 2010, p. 13.2. doi: 10.1109/TEST.2010.5699239.

[TangCGR2010] X. Tang, W.-T. Cheng, R. Guo, and S. M. Reddy. Diagnosis of multiple physical defects using logic fault models. In *Proc. 19th IEEE Asian Test Symposium (ATS)*, 2010, pp. 94–99. doi: 10.1109/ATS.2010.25.

[TehraPC2011] M. Tehranipoor, K. Peng, and K. Chakrabarty. *Test and Diagnosis for Small-Delay Defects.* Springer, 2011. ISBN 978-1-4419-8296-4. doi: 10.1007/978-1-4419-8297-1.

[Touba2007] N. A. Touba. X-canceling MISR — an X-tolerant methodology for compacting output responses with unknowns using a MISR. In *Proc. IEEE International Test Conference (ITC)*, 2007, p. 6.2. doi: 10.1109/TEST.2007.4437576.

[Tseit1968] G. S. Tseitin. On the complexity of derivation in propositional calculus. *Studies in Constructive Mathematics and Mathematical Logic, Part II*, p. 115–125, 1968.

[VenerCA*2004] A. G. Veneris, R. Chang, M. S. Abadir, and M. Amiri. Fault equivalence and diagnostic test generation using ATPG. In *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, 2004, pp. 221–224. doi: 10.1109/ISCAS.2004.1329502.

[VenkaD2000a] S. Venkataraman and S. B. Drummonds. A technique for logic fault diagnosis of interconnect open defects. In *Proc. 18th IEEE VLSI*

Test Symposium (VTS), 2000, pp. 313–318. doi: 10.1109/VTEST.2000.843860.

[VenkaD2000] S. Venkataraman and S. B. Drummonds. POIROT: A logic fault diagnosis tool and its applications. In *Proc. IEEE International Test Conference (ITC)*, 2000, pp. 253–262. doi: 10.1109/TEST.2000.894213.

[VogelMB2003] T. J. Vogels, W. Maly, and R. D. S. Blanton. Progressive bridge identification. In *Proc. IEEE International Test Conference (ITC)*, 2003, pp. 309–318. doi: 10.1109/TEST.2003.1270853.

[VolkeKR*2002] E. H. Volkerink, A. Khoche, J. Rivoir, and K. D. Hilliges. Test economics for multi-site test with modern cost reduction techniques. In *Proc. 20th IEEE VLSI Test Symposium (VTS)*, 2002, pp. 411–416. doi: 10.1109/VTS.2002.1011173.

[VrankGG*2006] H. P. E. Vranken, S. K. Goel, A. Glowatz, J. Schlöffel, and F. Hapke. Fault detection and diagnosis with parity trees for space compaction of test responses. In *Proc. 43rd IEEE/ACM Design Automation Conference (DAC)*, 2006, pp. 1095–1098. doi: 10.1145/1146909.1147185.

[Wadsa1978] R. Wadsack. Fault modeling and logic simulation of CMOS and MOS integrated circuits. *Bell Systems Techn. Journal*, 57:1449–1488, 1978.

[Wagne2008] L. C. Wagner. *Handbook of Semiconductor Manufacturing Technology*, chapter 29: Failure Analysis, p. 29.1. CRC Press London, 2008. ISBN 9781574446753.

[WaicuEF*1985] J. A. Waicukauski, E. B. Eichelberger, D. O. Forlenza, E. Lindbloom, and T. Mc-Carthy. Fault simulation for structured VLSI. *VLSI Systems Design*, 6(12):20–32, December 1985.

[WaicuL1989] J. A. Waicukauski and E. Lindbloom. Failure diagnosis of structured VLSI. *IEEE Design & Test of Computers*, 6(4):49–60, August 1989. doi: 10.1109/54.32421.

[WangCHW2003] C.-W. Wang, K.-L. Cheng, C.-T. Huang, and C.-W. Wu. Test and diagnosis of word-oriented multiport memories. In *Proc. 21st VLSI*

Test Symposium (VTS), 2003, pp. 248–253. doi: 10.1109/VTEST.2003.1197658.

[WangHLL*2008] F. Wang, Y. Hu, H. Li, X. Li, J. Ye, and Y. Huang. Deterministic diagnostic pattern generation (DDPG) for compound defects. In *Proc. 39th IEEE International Test Conference (ITC)*, 2008, p. 14.1. doi: 10.1109/TEST.2008.4700587.

[WangST2008] L.-T. Wang, C. E. Stroud, and N. A. Touba. *System-on-Chip Test Architectures Nanometer Design for Testability*. Morgan Kaufmann Publishers Inc., 2008. ISBN 0-12-373973-X.

[WangWW2006] 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.

[WangGB2006] L. Wang, S. K. Gupta, and M. A. Breuer. Diagnosis of delay faults due to resistive bridges, delay variations and defects. In *Proc. 15th Asian Test Symposium (ATS)*, 2006, pp. 215–224. doi: 10.1109/ATS.2006.261023.

[WangMTR2006] Z. Wang, M. Marek-Sadowska, K.-H. Tsai, and J. Rajski. Analysis and methodology for multiple-fault diagnosis. *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, 25(3):558–575, 2006. doi: 10.1109/TCAD.2005.854624.

[WenTSK2003] X. Wen, H. Tamamoto, K. K. Saluja, and K. Kinoshita. Fault diagnosis for physical defects of unknown behaviors. In *Proc. 12th Asian Test Symposium (ATS)*, 2003, pp. 236–241. doi: 10.1109/ATS.2003.1250816.

[WenKMYS*2006] X. Wen, S. Kajihara, K. Miyase, Y. Yamato, K. K. Saluja, L.-T. Wang, and K. Kinoshita. A per-test fault diagnosis method based on the X-fault model. *IEICE Transactions*, 89-D(11):2756–2765, 2006. doi: 10.1093/ietisy/e89-d.11.2756.

[WilliA1973] M. J. Y. Williams and J. B. Angell. Enhancing testability of large-scale integrated circuits via test points and additional logic. *IEEE Trans. on Computers*, C-22(1):46–60, January 1973. doi: 10.1109/T-C.1973.223600.

[WilliB1981] T. W. Williams and N. C. Brown. Defect level as a function of fault coverage. *IEEE Trans. on Computers*, C-30(12):987–988, December 1981. doi: 10.1109/TC.1981.1675742.

[WohlWPM2002] P. Wohl, J. A. Waicukauski, S. Patel, and G. Maston. Effective diagnostics through interval unloads in a BIST environments. In *Proc. ACM/IEEE Design Automation Conference (DAC)*, 2002, pp. 249–254. doi: 10.1109/DAC.2002.1012630.

[WuA1999] Y. Wu and S. Adham. Scan-based bist fault diagnosis. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 18(2):203–211, February 1999. doi: 10.1109/43.743733.

[Wunde1991] H.-J. Wunderlich. *Hochintegrierte Schaltungen: Prüfgerechter Entwurf und Test.* Springer, 1991. ISBN 3-540-53456-3.

[Wunde2010] H.-J. Wunderlich, editor. *Models in Hardware Testing.* Springer, 2010. ISBN 978-90-481-3281-2. doi: 10.1007/978-90-481-3282-9.

[WundeEH2007] H.-J. Wunderlich, M. Elm, and S. Holst. Debug and diagnosis: Mastering the life cycle of nano-scale systems on chip (invited paper). In *Proc. 43rd International Conference on Microelectronics, Devices and Materials (MIDEM)*, 2007, pp. 27–36.

[WundeEH*2007a] H.-J. Wunderlich, M. Elm, and S. Holst. Debug and diagnosis: Mastering the life cycle of nano-scale systems on chip (invited paper). *Informacije MIDEM*, 37(4(124)):235–243, Dec 2007.

[WundeH2010] H.-J. Wunderlich and S. Holst. *Models in Hardware Testing*, chapter Generalized Fault Modeling for Logic Diagnosis, pp. 133–156. Springer, 2010. ISBN 978-90-481-3281-2. doi: 10.1007/978-90-481-3282-9_5.

[XueDJ1994] H. Xue, C. Di, and J. A. G. Jess. Probability analysis for CMOS floating gate faults. In *Proc. European Design and Test Conference (EDTC)*, 1994, pp. 443–448. doi: 10.1109/EDTC.1994.326838.

[YangC2006] K. Yang and K.-T. Cheng. Timing-reasoning-based delay fault diagnosis. In *Proc. Design, Automation and Test in Europe (DATE)*, 2006, pp. 418–423. doi: 10.1145/1131595.

[YangC2007] K. Yang and K.-T. Cheng. Silicon debug for timing errors. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 26(11):2084–2088, November 2007. doi: 10.1109/TCAD.2007.906479.

[YeHL2011] J. Ye, Y. Hu, and X. Li. On diagnosis of multiple faults using compacted responses. In *Proc. Design, Automation Test in Europe Conference Exhibition (DATE)*, 2011, pp. 679–684.

[YenLLYL*2008] C.-C. Yen, T. Lin, H. Lin, K. Yang, T. Liu, and Y.-C. Hsu. A general failure candidate ranking framework for silicon debug. In *Proc. 26th IEEE VLSI Test Symposium (VTS)*, 2008, pp. 352–358. doi: 10.1109/VTS.2008.60.

[YuB2010] X. Yu and R. D. Blanton. Diagnosis of integrated circuits with multiple defects of arbitrary characteristics. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 29(6):977–987, June 2010. doi: 10.1109/TCAD.2010.2048352.

[YuB2010a] X. Yu and R. D. Blanton. Estimating defect-type distributions through volume diagnosis and defect behavior attribution. In *Proc. IEEE International Test Conference (ITC)*, 2010, p. 22.3. doi: 10.1109/TEST.2010.5699270.

[YuB2008a] X. Yu and R. D. S. Blanton. An effective and flexible multiple defect diagnosis methodology using error propagation analysis. In *Proc. 39th IEEE International Test Conference (ITC)*, 2008, p. 17.1. doi: 10.1109/TEST.2008.4700595.

[YuB2008] X. Yu and R. D. S. Blanton. Multiple defect diagnosis using no assumptions on failing pattern characteristics. In *Proc. 45th IEEE/ACM Design Automation Conference (DAC)*, 2008, pp. 361–366. doi: 10.1145/1391469.1391567.

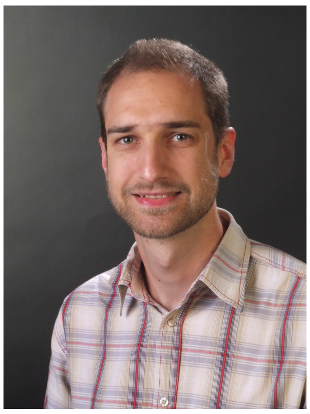
[ZachaCK*2003] S. Zachariah, Y.-S. Chang, S. Kundu, and C. Tirumurti. On modeling cross-talk faults. In *Proc. Design, Automation and Test in Europe (DATE)*, 2003, pp. 490–495. doi: 10.1109/DATE.2003.1253657.

[ZouCR2006] W. Zou, W.-T. Cheng, and S. M. Reddy. Interconnect open defect diagnosis with physical information. In *Proc. 15th Asian Test Sym-*

posium (ATS), 2006, pp. 203–209. doi: 10.1109/ATS.2006.261021.

[ZouCRT2006] W. Zou, W.-T. Cheng, S. M. Reddy, and H. Tang. On methods to improve location based logic diagnosis. In *Proc. 19th International Conference on VLSI Design*, 2006, pp. 181–187. doi: 10.1109/VLSID.2006.123.

C. Curriculum Vitae of the Author



Stefan Holst received his Diploma in Computer Science from the University of Stuttgart in 2005 before he joined the Institute of Computer Architecture and Computer Engineering of Prof. Dr. rer. nat. habil. H.–J. Wunderlich at the same University in 2006.

From 2006 to 2012 he was involved in the research projects "DIADEM: Embedded Diagnosis and Debug Methods for Nanoscale VLSI Systems" and "ROCK: Robust On–Chip Communication" supported by the German Research Foundation (DFG) as well as "DIANA: End–to–End Diagnostic Capabilities for Automotive Electronics Systems" and "MAYA: New Methods for the Massive–Parallel–Test for High Volume, Yield Learning and Best Test Quality" sponsored by the German ministry of education and research (BMBF) in cooperation with several industrial partners including AUDI, Infineon, NXP, and Mentor Graphics.

He supported some of the core courses including "Hardware Lab", "Advanced Processor Architecture", "Design and Test of Systems–on–a–Chip", and "Hardware Verification and Quality Assessment" taught in English as well as German. He also supervised students in 4 seminars, 6 Master's and Diploma theses as well as 10 Bachelor's theses and undergraduate projects.

His research interests include—but are not limited to—logic diagnosis, test compression, logic simulation, EDA algorithms on emerging hybrid architectures.

D. Diagnosis and Test Publications

Book Chapters

Hans–Joachim Wunderlich and Stefan Holst. Generalized Fault Modeling for Logic Diagnosis. *Chapter 5 in: Models in Hardware Testing* ISBN 978-90-481-3281-2, pp. 133–156. Springer, 2010.

Journal Publications

Hans–Joachim Wunderlich, Melanie Elm, and Stefan Holst. Debug and diagnosis: Mastering the life cycle of nano–scale systems on chip (invited paper). *Informacije MIDEM*, 37(4(124)):235–243, Dec 2007.

Stefan Holst and Hans–Joachim Wunderlich. Adaptive debug and diagnosis without fault dictionaries. *Journal of Electronic Testing – Theory and Applications (JETTA)*, 25(4-5):259–268, Aug 2009. doi: 10.1007/s10836-009-5109-3.

Atefe Dalirsani, Stefan Holst, Melanie Elm, and Hans-Joachim Wunderlich. Structural test for graceful degradation of NoC switches. To appear in *Journal of Electronic Testing – Theory and Applications (JETTA)*, 2012.

Conference Publications

Stefan Holst and Hans–Joachim Wunderlich. Adaptive debug and diagnosis without fault dictionaries. In *Proc. 12th European Test Symposium (ETS)*, 2007, pp. 7–12. IEEE. doi: 10.1109/ETS.2007.9.

Hans–Joachim Wunderlich, Melanie Elm, and Stefan Holst. Debug and diagnosis: Mastering the life cycle of nano–scale systems on chip (invited paper). In *Proc. 43rd International Conference on Microelectronics, Devices and Materials (MIDEM)*, 2007, pp. 27–36.

Stefan Holst and Hans–Joachim Wunderlich. A diagnosis algorithm for extreme space compaction. In *Proc. 12th Design, Automation and Test in Europe (DATE)*, 2009, pp. 1355–1360. IEEE/ACM. ISBN 978-3-9810801-3-1.

Abdul–Wahid Hakmi, Stefan Holst, Hans–Joachim Wunderlich, Juergen Schloeffel, Friedrich Hapke, and Andreas Glowatz. Restrict encoding for mixed–mode BIST. In *Proc. 27th VLSI Test Symposium (VTS)*, 2009, pp. 179–184. IEEE. doi: 10.1109/VTS.2009.43.

Michael Kochte, Stefan Holst, Melanie Elm, and Hans–Joachim Wunderlich. Test encoding for extreme response compaction. In *Proc. 14th European Test Symposium (ETS)*, 2009, pp. 155–160. IEEE. doi: 10.1109/ETS.2009.22.

Atefe Dalirsani, Stefan Holst, Melanie Elm, and Hans–Joachim Wunderlich. Structural test for graceful degradation of NoC switches. In *Proc. 16th European Test Symposium (ETS)*, 2011, pp. 183–188. IEEE. doi: 10.1109/ETS.2011.33.

Abdullah Mumtaz, Michael E. Imhof, Stefan Holst, and Hans–Joachim Wunderlich. Eingebetteter Test zur Hochgenauen Defekt–Lokalisierung. In *5. GMM/GI/ITG–Fachtagung Zuverlässigkeit und Entwurf (ZuE)*, 2011.

Abdullah Mumtaz, Michael E. Imhof, Stefan Holst, and Hans–Joachim Wunderlich. Embedded test for highly accurate defect localization. In *Proc. 20th Asian Test Symposium (ATS)*, 2011, pp. 213–218. IEEE. doi: 10.1109/ATS.2011.60.

Stefan Holst, Eric Schneider, and Hans-Joachim Wunderlich. Scan test power simulation on GPGPUs. To appear in *Proc. 21st IEEE Asian Test Symposium (ATS)*, 2012

Workshop Contributions

Stefan Holst and Hans–Joachim Wunderlich. Adaptive debug and diagnosis without fault dictionaries. In *19. GI/GMM/ITG Workshop für Testmethoden und Zuverlässigkeit von Schaltungen und Systemen (TuZ)*, 2007, pp. 82–86.

Stefan Holst and Hans–Joachim Wunderlich. Diagnose mit extrem kompaktierten Fehlerdaten. In *21. Workshop für Testmethoden und Zuverlässigkeit von Schaltungen und Systemen (TuZ)*, 2009, pp. 15–20.

Atefe Dalirsani, Stefan Holst, Melanie Elm, and Hans–Joachim Wunderlich. Structural test for graceful degradation of NoC switches. In *23. GI/GMM/ITG Workshop für Testmethoden und Zuverlässigkeit von Schaltungen und Systemen (TuZ)*, 2011, pp. 33–38.

Index

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.

Stefan Holst