POZNAŃ UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTING AND TELECOMMUNICATIONS

SYLWESTER MILEWSKI

HYPERCOMPRESSION OF TEST DATA

Ph. D. Thesis

Supervisor:

prof. dr hab. inż. Jerzy Tyszer

Poznań, Poland, 2021

Abstract

Testing of digital circuits has always been a vibrant area of research and development, primarily due to imperfect manufacturing processes and staggering complexity of semiconductor devices. It can be easily observed that each shrinking technology node brings new challenges for manufacturing test due to, for example, new types of failure mechanisms and defects. These defects require more complex fault models, and thus test patterns whose number is steadily increasing. This, in turn, leads to long test application times and elevated test data volumes. Furthermore, modern safety-critical applications require reliable and robust in-system tests of quality matching that of manufacturing tests. To alleviate and temper these problems, new testing methods are required despite significant advancements made in the past. Furthermore, power consumption during test must be seriously taken into account so that functional power limits are not exceeded. To satisfy current and anticipated VLSI test requirements, the thesis introduces a number of methods that target two important aspects of a test preparation process: test set compaction and test data compression.

First, a novel test set compaction methodology employing a state-of-the-art SAT-based automatic test pattern generation (ATPG) algorithm is presented. In principle, it is based on a dimensionality reduction paradigm that works with a meaningful representation of test patterns using external and internal necessary assignments to determine small groups of potentially compatible faults. These faults are subsequently retargeted by the robust SAT-based ATPG and its solvers producing a single test pattern for the entire group, thus making the resultant test set smaller in size.

The subsequent part of the thesis presents a new test compression scheme that aims at achieving encoding efficiency higher than any best-to-date sequential compression methods. The approach is based on a key observation that among care bits populating test cubes only a very few have a status of necessary assignments (their locations cannot be changed), whereas the remaining ones have alternative sites. These test cubes are used to form circular test templates which synergistically control a decompressor and guide back ATPG to find assignments yielding highly compressible test patterns.

The last part of this work proposes a next step in developing a new class of test data compression schemes. Although it builds on the paradigm recalled in the previous paragraph, it further limits silicon overhead, remains non-intrusive to the core logic, and elevates compression ratios. More importantly, however, it is inherently a low-power test solution that alleviates problems related to fault coverage drop and pattern count inflation caused by power constraints. Extremely small test templates make that scheme very flexible – it is capable of attaching a template to every pattern at a negligible cost related to test time and data volume, especially when working with a very few ATE channels.

All solutions presented in the thesis have been thoroughly verified through experimental results obtained for a variety of industrial and open-source benchmark circuits representing the latest technology nodes while varying with respect to design styles and scan methodologies, to name just a few factors that were taken into account.

Streszczenie

Testowanie układów i systemów cyfrowych pozostaje dynamicznie rozwijanym obszarem badań naukowych i praktyki inżynierskiej, przede wszystkim ze względu na niedoskonałości procesów wytwarzania układów półprzewodnikowych i ich bezprecedensową złożoność. Kolejne generacje cyfrowych układów scalonych wielkiej skali integracji wprowadzają nowe typy uszkodzeń, których wykrycie wymaga złożonych modeli oraz stale rosnącej liczby wektorów testowych. W konsekwencji rośnie konieczna ilość danych testowych oraz wydłuża się czas testowania. Istotny wpływ na kształt nowych technologii testowania ma także powszechna już obecność układów elektronicznych w urządzeniach o krytycznym znaczeniu dla zdrowia lub bezpieczeństwa. Ewentualne uszkodzenia układów zajmujących centralne miejsce w takich systemach mogą prowadzić do katastrofalnych następstw. Systemy te wymagają periodycznych testów wysokiej jakości (zwykle porównywalnej z jakością testów produkcyjnych) wykonywanych także w trakcie normalnej eksploatacji układu. Kolejny problem stanowi kilkukrotny wzrost, w trakcie testowania, zużycia energii. Ponadnormatywna aktywność układu w trakcie testu wymusza zatem bardzo staranne zarządzanie wykorzystaniem energii za pomocą dedykowanych, energooszczędnych metod testowania. W związku z przedstawionymi wymaganiami w pracy podjęto próbę rozwiązania dwóch istotnych zagadnień: minimalizacji liczby wektorów testowych (co w efekcie oznacza skrócenie czasu testowania) oraz redukcji wolumenu danych niezbędnych w procesie aplikacji właściwych testów.

W pierwszej części rozprawy podano metodę ograniczenia liczby wektorów testowych, początkowo otrzymanych za pomocą konwencjonalnych algorytmów automatycznej generacji testów. W zaproponowanym podejściu wykorzystano technikę redukcji wymiaru, tj. proces zmniejszania liczby zmiennych branych pod uwagę w trakcie analizy wektorów testowych. Zgodnie z tym podejściem, początkowy zbiór wektorów testowych, a w szczególności zbiór jego wartości koniecznych, jest wykorzystywany do zdefiniowania małych grup uszkodzeń potencjalnie zgodnych (wykrywanych za pomocą tego samego pobudzenia). Następnie wektory testowe są generowane ponownie za pomocą heurystycznych algorytmów stosowanych tradycyjnie w rozstrzyganiu problemu spełnialności formuł logicznych.

W dalszej części rozprawy przedstawiono metodę redukcji (kompresji) danych testowych, której celem jest osiągnięcie wysokiej efektywności kodowania. Zaproponowane podejście opiera się na obserwacji, że wśród wyspecyfikowanych bitów wektorów testowych tylko nieliczne są konieczne (ich wartości oraz przypisania do wejść układu nie można zmienić), podczas gdy pozostałe można zastąpić innymi wartościami przypisanymi do alternatywnych wejść. Tak interpretowane wektory testowe są następnie wykorzystywane do utworzenia cyklicznych szablonów sterujących procesem kompresji (i dekompresji) danych testowych.

Ostatni rozdział pracy to kolejny krok w ewolucji metod kompresji danych testowych. Bazując na opisanej we wcześniejszych rozdziałach rozprawy metodzie redukcji danych, nowy algorytm umożliwia dalsze ograniczenie infrastruktury testującej, zwiększa stopień kompresji oraz znacząco zmniejsza pobór mocy w trakcie podawania testów. W szczególności redukuje także rozmiar szablonów, które są dodawane do każdego wektora bez istotnego wpływu na czas testu i wynikową ilość danych testowych.

Wszystkie rozwiązania przedstawione w pracy zweryfikowano w badaniach eksperymentalnych przeprowadzonych za pomocą opracowanego przez autora oryginalnego oprogramowania będącego rozszerzeniem istniejących narzędzi komercyjnych. W eksperymentach wykorzystano współczesne scalone cyfrowe układy przemysłowe oraz układy testowe typu open-source.

Contents

Figures

Tables

List of abbreviations

Abbreviation	Description
ATE	Automatic test equipment
ATPG	Automatic test pattern generation
BIST	Built-in self-test
CAD	Computer-aided design
CMOS	Complementary metal-oxide semiconductor
CNF	Conjunctive normal form
CUT	Circuit under test
DFT	Design for testability
DPM	Defects per million
DUT	Design under test
EDA	Electronic design automation
EDT	Embedded deterministic test
EUV	Extreme ultraviolet
FFR	Fanout-free region
HTL	Hypercompression test logic
IC	Integrated circuit
IECEJ	Institute of Electronics and Communication Engineers of Japan
LBIST	Logic built-in self-test
LFSR	Linear feedback shift register
LP	Low power
LP-HTL	Low power hypercompression test logic
MISR	Multiple-input signature register
NA	Necessary assignment
PCB	Printed circuit boards

PRPG	Pseudorandom pattern generator
ROM	Read-only memory
SCOAP	Sandia Controllability/Observability Analysis Program
SAT	Boolean satisfiability problem
SoC	System on a chip
SPIE	Société Parisienne pour l'Industrie Electrique
STUMPS	Self-testing using MISR and parallel shift register sequence generator
TPG	Test pattern generator
VLSI	Very-large-scale integration
WSM	Weighted switching activity
WTM	Weighted transition metric

The conference names:

ATS	IEEE Asian Test Symposium
DAC	ACM/IEEE Design Automation Conference
DATE	Design Automation and Test in Europe
ETS	IEEE European Test Symposium
FTCS	IEEE International Symposium on Fault-Tolerant Computing
ICCAD	ACM/IEEE International Conference on Computer-Aided Design
ICCD	IEEE International Conference on Computer Design
IEDM	IEEE International Electron Devices Meeting
ISSCC	IEEE International Solid-State Circuits Conference
ITC	IEEE International Test Conference
VTS	IEEE VLSI Test Symposium

Chapter 1
Introduction

1.1 Preamble

On April 25, 1961, the US patent office awarded the first patent for a monolithic integrated circuit (IC) to Robert Noyce of Fairchild Semiconductor startup company. Although those so-called "unitary circuits" comprised just a few transistors [65], they were seminal signs of the shift in technology that was moving the entire world into the third industrial revolution, which would be dominated by electronics, computing, information, and digital advances. From that moment on, microelectronic devices have been increasingly shaping every aspect of our lives: the way we work, communicate, travel, or entertain. Interestingly, humankind is again facing a time of significant change. During the last two decades, we have witnessed a perfect storm of technology convergence that includes the dominance of data, massive computing, progress in algorithms and processing methodologies, and the integration of disparate technologies. Clearly, one of the key factors that makes all of this possible is the exponential miniaturization of chips and other components, predicted inexorably by Moore's Law [59]. Despite of several concerns, this 60-year old observation "is still there" as evidenced by Apple releasing, in 2020, one of the most powerful processors made of more than 16 billion transistors [94].

Semiconductor chips housing large circuitries capable of executing complex tasks and functions are prone to defects as any other result of product engineering. However, the reliability and robustness of electronic systems is no longer a concern limited to certain industries, where a failure may have severe (or even catastrophic) consequences. On the contrary, reliability and test techniques have become of increasing interest to a countless number of applications. As a result, the challenge of testing electronic systems has been growing rapidly over the last decades driven by unprecedented technological advances resulting in the technology feature dimensions shrinking such that state-of-the-art ultra-tiny interconnects between transistors are now matter of tens of atoms, whereas the thickness of insulating layers of transistor gates is equal to 3-5 atoms [13]. Even though the introduction of extreme ultraviolet (EUV) lithography

[92] made this miniaturization possible, it also made a complex defects inevitable in nanometer-scale devices. A list of principal failure mechanisms includes surface and bulk effects, metallization, process instabilities, package-related problems, and human errors. In addition to popular types of defects such as extra and missing material, oxide breakdowns, or electromigration (Fig. 1.1) [56], [91], one needs to consider more frequent stochastic printing failures such as microbridges, broken lines, and missing or merging contacts (Fig. 1.2) [9], [91], all of them resulting in faulty circuits. These defects require robust test procedures to assure delivery of impeccable products, regardless of whether the product is a single IC or an electronic system composed of many VLSI devices. Nevertheless, the wide variety of defects in chips makes it virtually impossible to create test patterns that would detect all actual physical failures. As a solution, abstract fault models are employed to represent a defective circuit behavior, including the all-time favorite single stuck-at fault where one of the signal lines in a circuit is assumed to be stuck at a fixed logic value. Since new manufacturing techniques generate new types of defects, many fault models were gradually introduced to better represent behavior of a faulty IC [1].

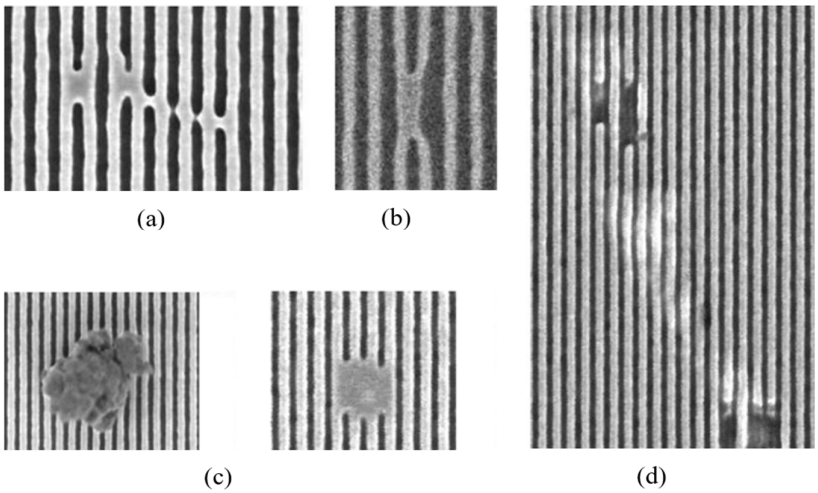

Fig. 1.1. “Traditional” types of defects: (a) cluster, (b) line collapse, (c) particles, (d) process defect [5].

Defects in microelectronic systems may occur at any stage of their production process. Consequently, various types of test routines are deployed several times beginning with manufacturing tests for individual ICs, then tests involving the same ICs soldered on printed circuit boards (PCBs), and subsequently all the way to a complete system assembled and sent to the end user. At each stage, the device must be tested thoroughly because the later a defect is detected, the higher the cost of fault detection is going to be. As the rule of ten states [83],

detecting a defective IC further in the process increases that cost by order of magnitude per stage. Consequently, it is crucial to spot a defective circuit as soon as possible.

Given a circuit, the basic objective of the IC testing is to produce patterns that excite (provoke) faults and propagate their effects (errors) to observable outputs (or other observation points). To put it in a different way, a test stimulus must yield different responses depending on whether a circuit is fault-free or faulty. Apart from very simple designs, it is virtually impossible to get such patterns manually. In 1966 and 1967, P. Roth published two seminal papers [74], [75] on D-algorithm – the method to automate test pattern generation that, for the first time, introduced a calculus and methods for *automatic test pattern generation* (ATPG) for combinational circuits. It was also shown [44] that although the D-algorithm can find a test pattern for any detectable fault, it belongs to the class of NP-complete problems. Because of such complexity, several combinational ATPG heuristics were proposed in the following years [83] to reduce ATPG processing times. Although most of them used the concepts introduced by P. Roth, these techniques may not guarantee a solution. Instead, they exploit structural information or sophisticated learning processes to improve ATPG performance (as detailed in the next chapter). As a result, the test generation algorithms can maximize the number of detected faults, but they may not optimize the corresponding pattern count. Since it determines test application time and a tester memory size (and thus cost of test), a lot of research effort was spent on developing test set reduction methods, also known as *test set compaction*. In principle, these algorithms transform test sets in such a way that each (compact) test vector detects as many faults as possible [83].

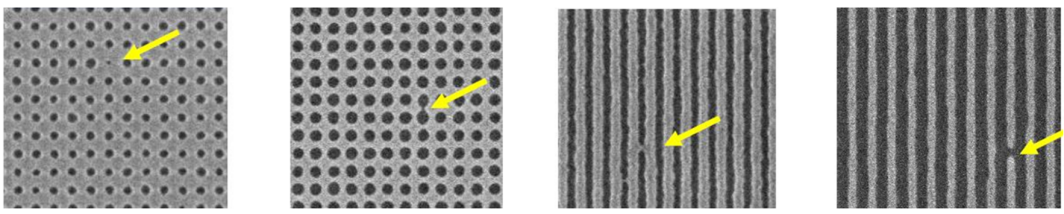

Fig. 1.2. Stochastic printing defects: (a) missing contact, (b) merging contact, (c) microbridge, (d) broken line [5].

The challenge of test generation is clearly more cumbersome and intricate for sequential circuits. Although there were several attempts to develop a fully functional sequential ATPG [61], [64], its computational complexity makes generating patterns for large sequential circuits

an excruciating and daunting task. Even worse, those algorithms were handling successfully only fairly regular structures. The inability to easily control and/or observe internal state variables of sequential circuits led finally to adoption of design techniques that facilitate testing. These so-called design for test (DFT) schemes advocate the use of additional on-chip constructs that make testing easier with respect to test generation, test application, and many other test-related activities. The DFT started with the ad-hoc insertion of control and observation test points into a design. However, it has required manual implementation and verification which quickly became unfeasible because of complexity of VLSI circuits. This created a particularly pressing need for automation. In the 70s, IBM and NEC have introduced independently a scan-based design methodology, which unified and automated the process of DFT insertion [25], [27], [49]. According to this paradigm, a synchronous sequential circuit works in two modes: (1) a functional mode, where a circuit executes a mission it was designed for, whereas in (2) a test mode, all or almost all flip-flops form one or more shift registers (*scan chains*), which are used to gain access to internal nodes of a chip. As a result, test generation is reduced to that of combinational circuits which can be directly driven and observed. The resultant patterns and responses can be shifted-in and shifted-out via scan chains, respectively. This test paradigm was successfully and firmly in place for the next five decades, and there are no contenders on the horizon even though scan designs incur certain hardware overhead and performance degradation, not to mention certain IP security concerns.

Despite the overwhelming deployment of efficient ATPG and DFT schemes, new test challenges surfaced in the early nineties. In particular, smaller and smaller technology nodes with the corresponding new fault models have caused the magnitude of test sets produced by contemporary ATPG tools to grow at a pace visibly surpassing Moore's Law. This, in turn, has resulted in a significant increase in test cost because of lengthy test times and large tester memory requirements. In 1999, at his International Test Conference (ITC) keynote speech, Pat Gelsinger, the CEO of Intel at that time, warned that in 10 years, the cost of testing a transistor would become greater than its manufacturing cost [29]. The test community responded to this challenge promptly with the introduction of test data compression, another milestone in reducing the test cost [6], [72], [79]. In accordance with this new paradigm, ATE stores compressed test patterns and delivers them to an on-chip decompressor, which drives scan chains with the actual test stimuli. Similarly, test responses are shifted-out via scan chains to an on-chip compactor and sent back to a tester to be compared against golden references. This approach reduces

test application time, ATE memory as well as I/O channels. Moreover, the test throughput is increased while maintaining the high-quality of test.

The trend to integrate various functionalities on a single chip continues and brings more and more functionalities to a single die. Massive integration is even stronger in the mobile, space, home automation, or automotive electronics, where systems-on-a-chip (SoC) bring together complex computing, communications, and entertainment functions on a single die. As a result, durability and reliability over their expected lifetime become a significant concern. In these applications, devices must be thoroughly tested, not only before their application but also during their lifespan; it requires monitoring of material aging and wear-out. This was partially resolved by deployment of logic built-in self-test (LBIST) [4], [80]. However, the resultant test quality and test application times were not satisfactory, mainly due an inherent inability to successfully tackle random pattern resistant faults. This bottleneck was overcome by combining LBIST with test compression. New compression techniques allowed sharing specific on-chip resources and created hybrid test schemes as a new and promising direction in the embedded test. Several hybrid LBIST schemes were proposed to store deterministic top-up patterns (detecting random pattern resistant faults) on a tester in a compressed form, and then use the existing LBIST hardware to decompress them. Nowadays, many manufacturers go even further and perform full deterministic in-system tests. It takes more memory resources but assures the desired test quality.

The drive to pack more functions into a small space leads also to power delivery and heat flux issues affecting supply integrity and chip packaging. Power issues, however, affect not only the mission mode but the test mode as well, as toggling rates and the resultant power consumption can be much higher than a circuit is rated for (typically, the goal of structural scan-based test is to activate as many nodes as possible in a very short period of time [30]). This trend is only expected to get worse. The resulting higher junction temperature and increased peak power lead to overheating or supply voltage noise - either of which can cause a device malfunction and thus yield loss, chip reliability degradation, shorter product lifetime, or device permanent damage.

In spite of many techniques used to arrive with high quality and compact tests, the aforementioned limitations and constraints are still shaping development of testing schemes for VLSI devices. Therefore, new generation ATPG and test compression schemes must take those

factors into account. In response to these challenges, the thesis proposes new techniques targeting test pattern generation, test set compaction, and test data compression.

1.2 Motivation

Although combinational ATPG, the associated test compaction schemes, and test data compression methods are considered a very mature area of science and engineering by many, the magnitude of test sets produced by contemporary ATPG tools continues to grow, as already mentioned in the previous chapter. The inflated test sets and lengthy test application times are commonly attributed, although not limited, to: (1) pattern-intensive transistor-level test generation techniques aimed at reducing test escapes and handling many clock domains, (2) circuits of large combinational depths with staggeringly complex clocking schemes, (3) excessive tail pattern counts featuring very few specified bits, yet difficult to merge due to mutual conflicts, and (4) automatically generated RTL whose tricky control logic poses non-trivial test challenges. As a result, large test sets become efficiency limiting and cost-increasing factors in testing embedded systems, system-on-a-chip designs, or any other complex designs brought to the market.

To address these challenges, the first part of this work proposes a new test set compaction technique that follows a long list of earlier contributions in the same area, including reverse order fault simulation [78], [82], methods setting unspecified values so as to detect more faults by a single pattern [32], or techniques that target faults in a particular order to further decrease a test set [66]. A method presented here replaces a complete set of fully specified test patterns by its meaningful yet reduced representation. It integrates synergistically several techniques, such as computation of necessary assignments, a comprehensive fault profiling, a fault grouping, and a customized version of SAT-based ATPG, to reduce effectively the original test set while preserving all benefits of modern ATPG tools.

It is worth noting that even a compact test set can still be a major source of test complexity due to the explosive pace of test data growth. Clearly, this problem is not new [50]. However, since the late nineties, it is growing much more acute. As a result, various test data compression schemes have started having a significant impact on the test landscape. One of the most straightforward solutions was to broadcast test data to scan chains through hardwired fanouts of a few ATE channels [54], as shown in Fig. 1.3b. The Illinois scan [36] used the same principle but

allowed reconfiguration of input fanouts. Clearly, these techniques can only succeed provided they are tightly coupled with a compression-aware ATPG. More flexible solutions belong to a linear compression class that involves only linear operations to decompress test vectors. These techniques are based on combinational linear expansion circuits comprising XOR gates [7], [8] (Fig. 1.3c), or various forms of linear finite state machines, such as linear feedback shift registers (LFSRs), ring generators, or cellular automata (Fig. 1.3d)[1]. The chronologically first static

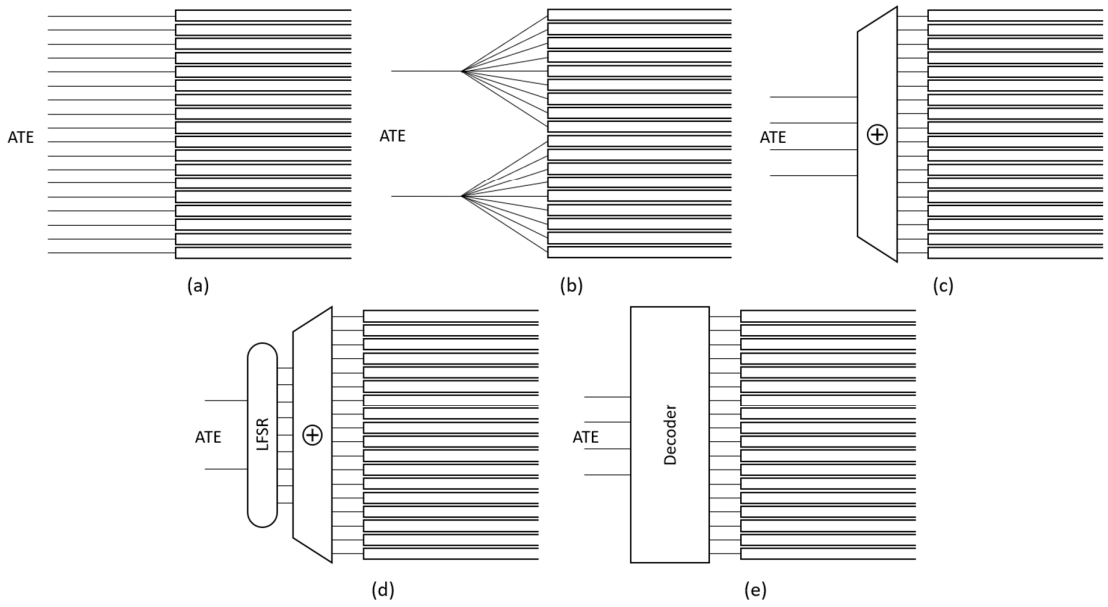

Fig. 1.3. Evolution of test data compression: (a) no compression, (b) broadcast scan, (c) combinational linear decompressor, (d) sequential linear decompressor, (e) code-based decompressor.

LFSR reseeding [40], [50], was eventually replaced by a dynamic LFSR reseeding streaming data into a decompressor as it loads the scan chains. This continuous-flow paradigm is probably best represented by the embedded deterministic test (EDT) [72] that has gained broad acceptance as a reliable industrial solution. Its sequential decompressor has much higher encoding capacity, so the encoding works for arbitrary test cubes. The other major advantage of EDT is that its decompressor can be reused as pseudorandom test pattern generator (PRPG) in compression/LBIST hybrids. Any conventional dynamic LFSR reseeding scheme, however, cannot compress test data to less than the total number of specified bits. *Isometric compression* [51]

[1] Another group of methods uses on-chip devices to handle various forms of code-based test pattern compression (Fig. 1.3e). However, these methods have a limited usage in the industrial test realm due to a non-trivial encoding process and a complex on-chip hardware needed to decompress test data [81].

elevates the encoding beyond this limit by synergistically engineering both ATPG and test encoding. Nevertheless, it might be inconvenient to implement a generic isometric decompressor, as proposed in [51]. Furthermore, since the scheme works exclusively with low-power patterns affecting all scan chains, it may also compromise test coverage.

The above concerns are addressed by a novel test data compression paradigm called *hypercompression* which is presented in the second part of this work. It aims at extreme test data compression ratios and, additionally, it offers inherent low-power capabilities by reducing switching activity during shift-in of test data. While it builds on the isometric compression principles, the native isometric decompressor is rearchitected to keep a silicon real estate of test logic at an acceptable level. As a result, its area becomes a tiny fraction of the entire DFT infrastructure. Moreover, the hypercompression test power management scheme deploys a programmable selection of a very few scan chains that should be put into a full-toggle mode. As a result, it avoids periods of elevated toggling in scan chains and reduces scan load switching activity.

The remainder of the thesis is organized as follows. Chapter 2 recalls and outlines the basic VLSI test concepts as well as state-of-the-at solutions related to problems being tackled in this work. Chapter 3 describes the methods of computing necessary assignments. What follows in Chapter 4 is a presentation of a static compaction algorithm that combines necessary assignments with SAT-based ATPG. The second part of the thesis begins with Chapter 5 that introduces the test hypercompression technique. Chapter 6 demonstrates how this method can be enhanced by further reducing the decompressor size and lowering a power dissipation. The thesis concludes with Chapter 7. All solutions proposed in this work are thoroughly verified by means of experimental results obtained for large and complex industrial designs.

1.3 Acknowledgments

First and foremost, I would like to express my sincere gratitude to my supervisors Prof. Janusz Rajski and Prof. Jerzy Tyszer for their invaluable guidance and encouragement throughout the span of this thesis. Their enthusiasm and high standards have fueled the motivation behind my work and helped me acquire valuable skills with carry over to my future endeavors. I am also thankful for scientific advice, knowledge, and many insightful suggestions and discussions we

had. Without their assistance and dedication in every step throughout the project it would have never been successfully completed.

My special gratitude goes to Dr. Grzegorz Mrugalski of Siemens Digital Industries Software, Poland, whose stimulating suggestions and constructive feedback helped me many times resolve various technical problems. I extend my thanks to Dr. Justyna Zawada and Dr. Jędrzej Solecki who have accompanied me on my PhD journey. I also would like to acknowledge the members of DFT group at Siemens Digital Industries Software. In particular, I appreciate very much insightful suggestions and support received from Dr. Yu Huang, Dr. Chen Wang and Dr. Elham Moghaddam. They all have been a source of friendship as well as good advice and collaboration. Finally, I gratefully acknowledge funding that made my PhD work possible. The Siemens Digital Industries Software scholarship I received during my PhD journey allowed me to attend international conferences and to successfully join the international test community.

Lastly, I would like to thank my family: my parents who supported me in all my pursuits, and most of all, my loving, supportive, encouraging, and patient wife Karolina whose faithful support during all stages of this PhD process is so appreciated. Thank you.

Chapter 2
Preliminaries

As the thesis tackles problems related to test set compaction and the resultant test data compression, this chapter, for the sake of self-containment and completeness, recalls and reviews both the useful key terms and the most relevant state-of the-art solutions in these areas. We begin with principles of algorithmic generation of test patterns and the corresponding test compaction methods. Subsequently, the basic concepts that govern the process of designing circuits to make them easy to test are briefly revisited to allow a review of the isometric test compression – a technique that lays foundations for the hypercompression of test data, a new test paradigm presented in the following parts of this work.

2.1 Deterministic test pattern generation

In 1966, P. Roth introduced the first-ever published algorithmic procedure for test pattern generation [74], [75]. His D algorithm uses a five-value algebra comprising symbols 0, 1, D, $\overline{D}$ and X. In particular, symbols D and $\overline{D}$ stand for a faulty effect on a line, i.e., they allow one to distinguish a good circuit behavior and its faulty counterpart: D represents the logic value of 1 in the good circuit and 0 in the faulty one, whereas $\overline{D}$ is the opposite case. The algorithm injects the faulty effect D or $\overline{D}$ (depending on a fault type) at the fault site, and then tries to propagate it to an observable output. Furthermore, all gates necessary to propagate the faulty effects must be justified with the required values. Since the test generation problem belongs to the NP-complete class, D algorithm's runtime for larger circuits becomes quickly unacceptable. Consequently, the following years brought several attempts to reduce the test generation CPU time. Authors of PODEM [31] – one of the first methods proposed at that time – noticed that D algorithm added every node to a decision tree. However, in combinational circuits, it suffices to consider *primary input* (PI) nodes only. As a result, a fewer number of decisions makes a test generation process much faster. Still, backtracking from decisions involving PIs needs to recover states of all gates driven by the same PIs, which again might be a fairly time-consuming

process. This problem is tackled by FAN [26], which introduced the concept of fanout-free regions (FFR). It finds headlines, i.e., the outputs of FFRs, and uses that information to reduce decision trees. TOPS [48] further improves that idea by using basis-nodes instead of headlines. Along different lines, in the middle of eighties, new techniques were explored to increase awareness of a circuit structure. In particular, SOCRATES [78] introduced concepts of static and dynamic learning which reveal indirect implications allowing to identify wrong decisions made earlier. In the meantime, a number of tools, such as BACK [14], EBT [57], FASTEST [47], GATTO [16], or HITEC [64] made attempts to address the problem of test pattern generation for sequential circuits. Moreover, ATPG tools producing diagnostic test patterns were introduced in [10], [34], [67], [93]. Another direction in test generation is comprised of techniques that employ SAT solvers to generate, in a rather CPU-time-intensive manner, compact test sets (see, for example, NEMESIS [52] or [20]). On top of those techniques, several algorithms were proposed to generate test patterns capable of minimizing power dissipated during test application, e.g., [17], [84], [85]. Nowadays, the state-of-the-art ATPG algorithms include FastScan [95], TestMAX [96], or Modus [97], which continue to reduce search space by introducing new heuristics while adding, at the same time, support for novel fault models and parallel processing.

In parallel with test pattern generation schemes, a lot of research effort was spent on developing test compaction methods that allow reducing counts of ATPG-produced test patterns. Typically, these methods are either *static* or *dynamic*. The former ones use pre-generated test sets and remove redundant test patterns while trying to merge the remaining test cubes. Probably the most prominent approach here is the reverse order fault simulation and its derivatives [1], [68], [69] that handle tests in the opposite order of generation. If a test pattern does not detect any new faults when simulated, it is discarded with no impact on the final fault coverage. Other techniques are discussed, for example, in [42], [55], [58], [62], [63], [66]. In particular, relaxation-based post-processing schemes (see, for instance, [58]) try to determine unspecified values which are not needed to detect essential faults. As a result, a test set may include more unspecified values, and thus static compaction can be applied more effectively. Yet another approach leverages SAT-based ATPG to determine compatible fault groups which can be then detected by the same test [24]. As a result, the scheme of [24] is capable of producing lower pattern counts than those of state-of-the-art ATPG tools. On the other hand, dynamic test com-

paction techniques such as [12], [18], [37], [45], [62], [66] are integral parts of the ATPG process itself where they work with heuristics that are likely to yield fewer test patterns by assigning certain unspecified bits of generated tests with values allowing detection of additional faults.

2.2 Design for testability

As the size of VLSI circuits was expanding exponentially (including sequential elements), controlling and observing internal state variables using only primary inputs and primary outputs became unwieldy. In fact, sequential test generation algorithms were unable to efficiently handle large and continuously growing designs. At some point, it became clear that a synergistic integration of design and test processes is needed to make testing feasible. It was the advent of DFT. It started with ad-hoc methods such as partitioning [3] or test points insertion [2], [38], [88]. In particular, the latter one adds extra easily accessible gates placed on lines hard to control or observe. Identification of such locations in large circuits is a complex problem, hence various procedures and measures, e.g., SCOAP [33], have been used to compute and quantify *controllability* and *observability*, and to assist in finding suitable test point locations. Recall that controllability is a measure of how easily a signal line can be set to a particular value (subsequently used to excite a fault). At the same time, observability indicates the effort needed to determine the logic value of a given node by properly setting inputs and observing outputs.

Although the aforementioned methods have substantially improved testability of designs and made sequential ATPG less severe, they were still circuit-dependent and generally difficult to automate. Thus, more systematic and structured techniques were incorporated into the design

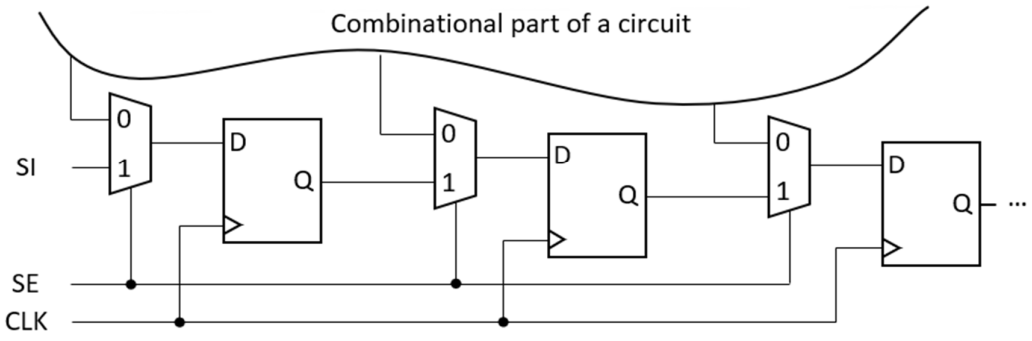

Fig. 2.1. D-multiplexed scan cells.

process. Clearly, the most pronounced structured DFT technology is a scan [49]. In this approach, all memory elements are transformed into *scan cells* (see, for example, Fig. 2.1) forming shift registers or *scan chains*. According to this paradigm, a design has two modes of operations: a functional (mission) mode, where the circuit works as originally intended, and a test mode, where all memory elements form scan chains whose serial inputs and outputs are used to shift-in test patterns and to receive shifted-out test responses, respectively. Scan makes all state elements directly accessible and observable, and thus the complex problem of testing sequential circuits boils down to much simpler testing of their combinational parts (integrity of scan chains is usually verified by applying so-called flush tests).

With the deployment of scan-based designs, a structured design for test approach has gained wide industrial acceptance. The resultant high controllability and observability of internal nodes made it possible to automatically generate high quality tests for large designs and use verification techniques to debug the first silicon. Moreover, simple architecture of scan chains enables their automated stitching and insertion. Scan, supported by many electronic design automation tools, offers a systematic way to manufacture testable and reliable semiconductor devices. Overall, due to its advantages, it has become one of the most influential and industry-proven DFT techniques.

2.3 Embedded Deterministic Test

The scan-based design-for-test paradigm was firmly in place for four decades. With circuits growing in size, however, it became more and more expensive to retain high level of test coverage. It was due to prohibitively large volumes of test data and long test application times. With cost-effective test data reduction techniques as the foundation for maintaining the high efficiency of a testing scheme, on-chip test compression has quickly established itself as a next DFT milestone and a mainstream DFT methodology [46], [81], with all major EDA companies and some semiconductor manufacturers regularly announcing products and technologies in that area. Embedded Deterministic Test [71] is one of the most representative solutions in the area of test data compression.

EDT inserts a decompressor and a compactor on chip to drive and observe scan chains (Fig. 2.2). The design does not require any modifications other than a configuration with a large number of short scan chains. The compression logic is placed in the scan path, so it does not

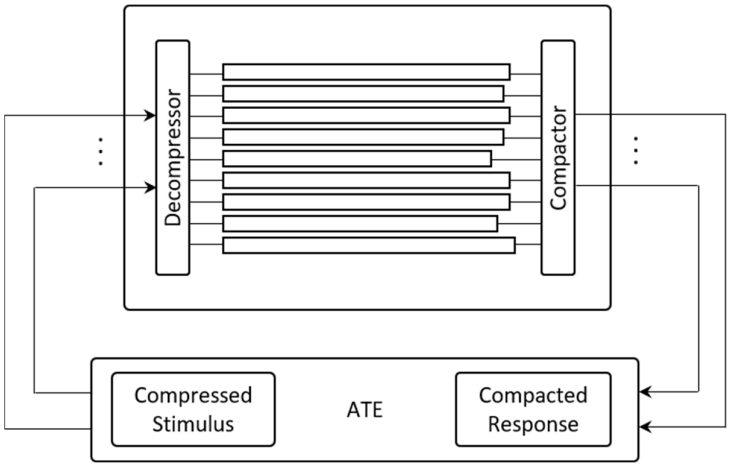

Fig. 2.2. General scheme of test data compression and compaction.

affect the functional operation. The EDT decompressor consists of a ring generator and a phase shifter (Fig. 2.3). Compressed data from ATE goes, through the input channels, to the injector sites of the ring generator. It is a form of LFSR [60] with much shorter propagation delays and internal fan-outs limited to two branches. Data circulating within the ring generator feeds a phase shifter, whose outputs are directly connected to scan chains. A phase shifter delivers tests to scan chains whose number is typically much larger than the size of the ring generator.

EDT is an example of a continuous flow compression, i.e., a dynamic LFSR reseeding, where successive seed variables are delivered to test logic in a per-cycle fashion (in a vivid contrast to a static reseeding of LFSRs [35], [40], [41], [50], [73], [87], [89] or counters [11], [39]). The actual compression algorithm works as follows. ATPG-produced test cubes (sparsely specified tests targeting certain faults) are encoded using a system of linear equations, two examples of which are shown in Fig. 2.3. Clearly, all scan cells are assigned respective linear expressions that form a system of linear equations once locations of specified bits are known. Solving these equations maps specified bits into a compressed stimulus. If the solver succeeds in compressing a test cube, ATPG targets additional faults and adds more specified bits to the current test cube. The solver and ATPG iterate incrementally increasing gradually the fill rate until a predefined number of encoding failures.

Although test compression has been a successful methodology of the last two decades, the current and future technology nodes keep introducing new test challenges. In order to sustain low defect per million (DPM) rates, new fault models are used, which consequently increases the pattern counts. It directly translates into increased ATE memory volume and test

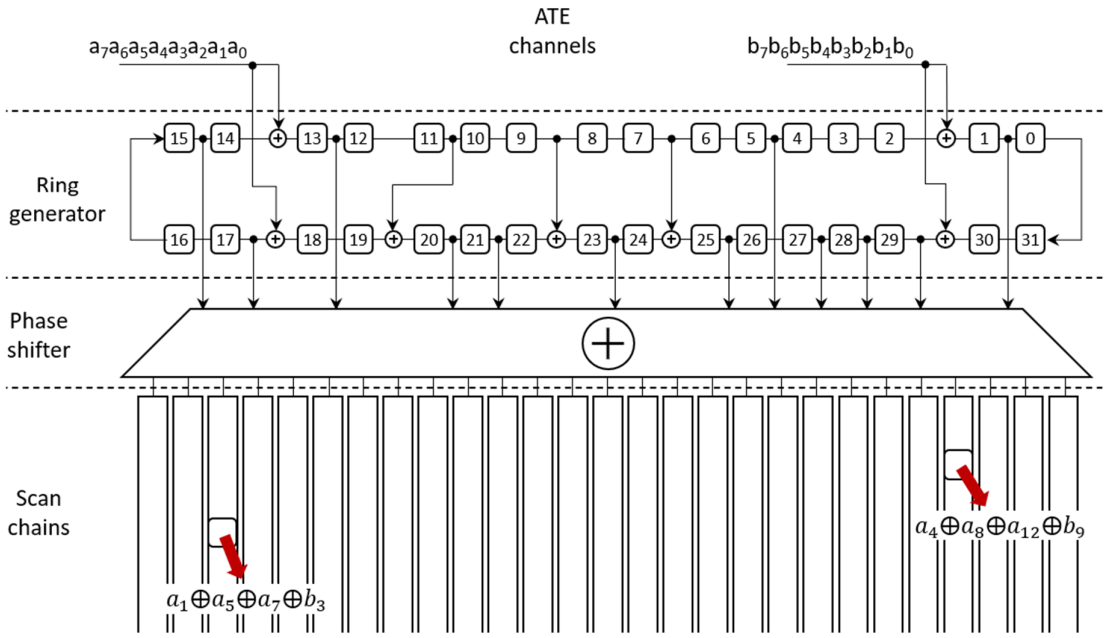

Fig. 2.3. Example of the EDT decompressor.

application time. The requirements for new test technologies are also shaped by safety-critical applications which require high-quality, in-system deterministic tests. In particular, it applies to automotive designs, where the ISO 26262 standard defines high test quality requirements, quite difficult to meet by conventional LBIST schemes. Thus, deterministic test patterns need to be stored in a system, whereas current solutions may not be able to reduce test data to the desired levels. One solution capable of outperforming conventional sequential compression is based on the isometric compression paradigm.

2.4 Isometric compression

Isometric compression [51] takes the on-chip test data compression to a new level by having more interactions between test generation (ATPG) and test encoding. This approach is aimed at elevating compression ratios to values unachievable through the conventional static [40], [50], and dynamic LFSR reseeding [5], [72], which can reach, at their best, the value of f^{-1}, where f is a test pattern fill rate.

It is worth noting that the number of care bits does not have to constrain neither compression nor a desired low toggling (in low power test solutions). On the contrary, having several

cells assigned the same value and hosted by the same chain may actually ease problems related to both data reduction and switching activity. Indeed, only specified bits occurring at certain locations would be then encoded, while bits of the same value make it possible to deliver identical test data to scan chains for several shift cycles, thereby reducing the resultant toggling. Consider a test cube shown in Fig. 2.4. It detects a stuck-at-1 fault (for the sake of the presentation, stuck-at faults are represented by diamonds, while nets set to the logic values of 0 and 1 are printed in blue and red, respectively) by feeding a 25-input XOR gate G_1 and a 3-input OR gate G_3. Alternatively, one can apply a test pattern listed at the bottom of the figure. The number of specified bits that must be encoded within the first vector is equal to 25 + 3 = 28, whereas it suffices to target only 13 + 1 = 14 care bits to encode the second test cube. Indeed, the specified pairs 00 and 11 can be obtained by encoding just the first value of each pair (indicated by arrows) and then sustaining the decompressor outputs to deliver the identical value during the next shift cycle.

Fig. 2.6 recalls a generic architecture of the isometric test data decompressor [51] implementing the above concept. In addition to a ring generator and a phase shifter driving scan chains, a hold register is placed between these two devices. It occasionally captures certain states of the ring generator. As a result, the toggling-free data can be provided to the scan chains for several continuous scan shift cycles, while the generator keeps advancing to the next states needed to encode another group of specified bits. As in many other schemes, compressed test patterns are delivered to the decompressor through *c* external channels in such a way that a new *c*-bit vector is injected into the ring generator every scan shift cycle.

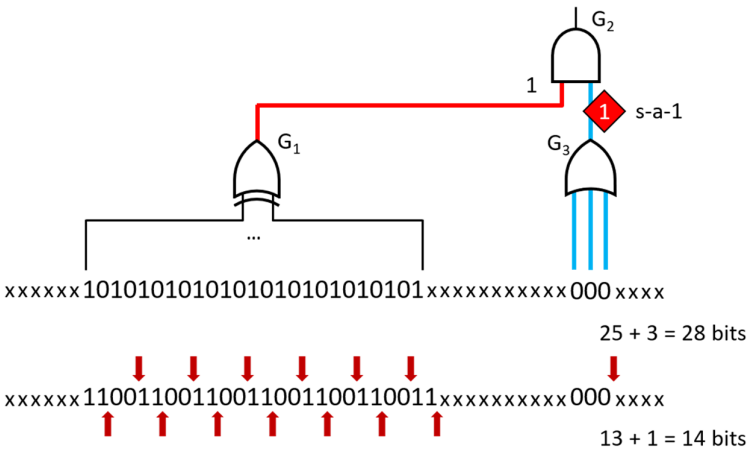

Fig. 2.4. Example of isometric compression.

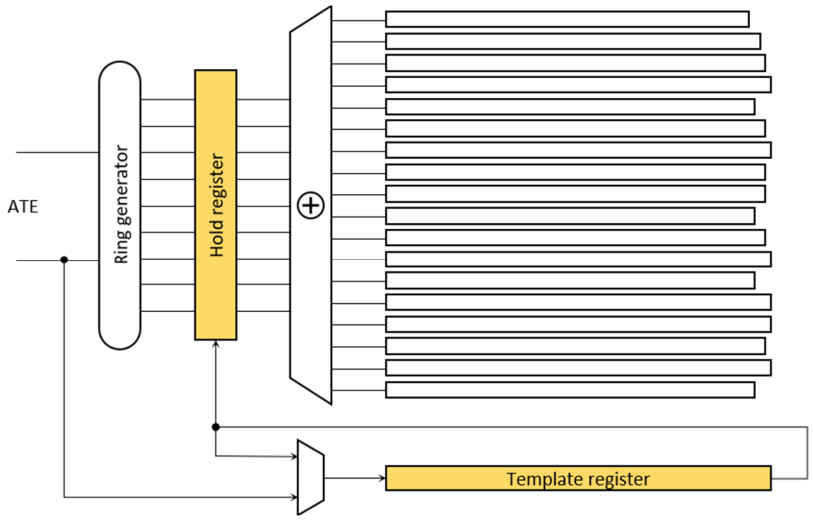

Fig. 2.6. Isometric test data decompressor.

The isometric decompressor houses also its key component – a circular template register whose size matches the longest scan chain, as shown in Fig. 2.6. The template register controls the decompressor by providing a control bit to the hold register every shift cycle to indicate whether this register is to be updated by the ring generator. If so, such a time is referred to as a toggle point, with two successive toggle points forming a hold segment. Thus, a given test cube is partitioned into several transition-free hold segments. One can repeat, therefore, a given decompressor state many times in succession by using the hold register storing a state that the ring generator entered at the beginning of the hold segment. Locations of all toggle points form a test template (Fig. 2.5), where two colors indicate the values of 0 and 1 applied to scan cells

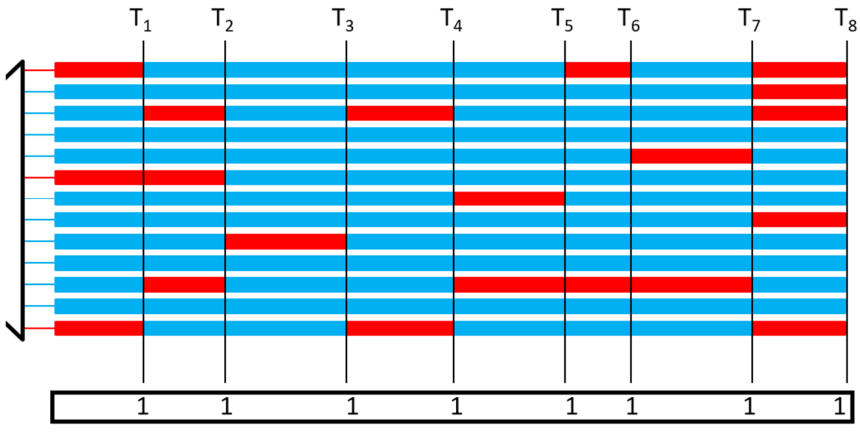

Fig. 2.5. Isometric test pattern.

between toggle points T_1, T_2, …, T_8. Clearly, ATPG tries to generate test cubes that fit to a given test template and segments it defines.

As can be seen, the isometric test compression provides a coherent way to generate and apply test patterns compressed beyond the limits achievable by the earlier solutions while substantially reducing power dissipation during all scan operations (other methods tackling low power operations within the framework of test compression can be found in [19], [53], [73], [90], to recall just a few representative works). However, its sizeable template register (or registers) is a non-negligible footprint of the entire DFT infrastructure. Moreover, multiple reloads of a template register might take a substantial amount of time which would be unacceptable in many practical applications. These concerns will be addressed in Chapter 5 that revisits the concept of isometric compression and proposes a new and more effective solution.

Chapter 3
Computing necessary assignments

The thesis tackles two key aspects of testing VLSI circuits: test set compaction and test data compression. The next two chapters focus on the former problem. A proposed test set compaction method works with a precomputed list of assignments mandatory to detect every target fault. These necessary assignments (NAs) are deployed, to a large degree, for the purpose of fault grouping. Such groups of faults are subsequently retargeted by a robust SAT-based ATPG whose solver is trying to arrive with a single test pattern for the entire group, thus making the resultant test set smaller in size. This is why techniques used to determine NAs are detailed herein with respect to their algorithmic capabilities and execution efficiency. It is worth noting

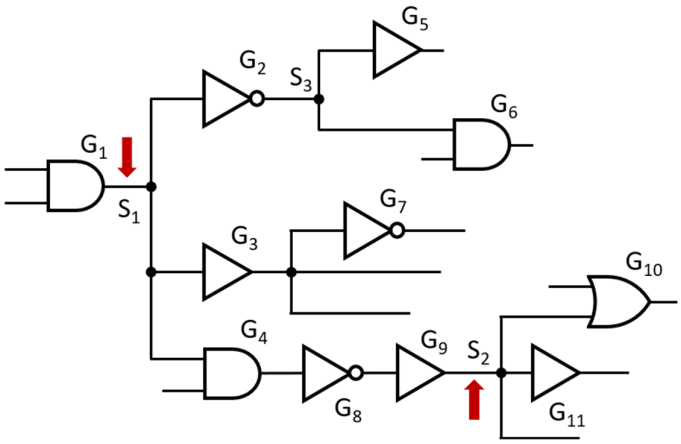

Fig. 3.1. Primary fanout stems.

that NAs are obtained for primary inputs, pseudo-primary inputs (scan cells), and all internal primary fanout stems (or simply primary stems). The latter are defined as fanout stems driven either by (1) a gate with two or more inputs, or by (2) a sequence of single-input gates only such that a gate in this sequence with a lowest input level is not driven by another stem. Fig. 3.1 illustrates a simple circuit with two primary stems S_1 and S_2 at the output of gates G_1 and G_9, respectively. Note that processing stem S_3 will not yield any new NA because it is functionally the same as primary stem S_1. Essentially, in order to determine NAs

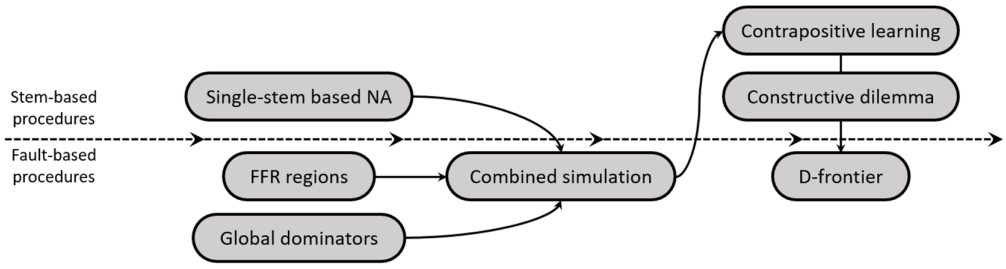

Fig. 3.2. Necessary assignments computation flow.

in a computationally feasible manner, six different techniques are used that either handle all faults in parallel or target every fault in a more individual fashion, as shown in Fig. 3.2. These techniques, in varying forms and to various degrees, have been deployed earlier by several ATPG systems [70], [77], [78]. However, while preserving their basic functionalities, they are run here either individually or in a combined effort to identify as many internal NAs as possible within the most critical parts of a design through low-cost processing procedures. They are presented in the subsections to follow.

3.1 Single-stem-based necessary assignments

This approach is run independently for every primary stem (plus inputs regardless of their fan-out count). These nets are systematically set to 0 and 1, and then an event-driven fault-free logic simulation is used to learn what other signal values it implies. Given the logic value $v(s)$ assigned to net s, every simulation run consists of three steps: (1) finding forward implications of $v(s)$, (2) carrying out a backward justification of $v(s)$, and (3) finding additional forward implications of signals implied at branches of stems encountered during (2). The simulation terminates once it cannot determine any more logic values (the even list becomes empty). In addition to finding logic values, the logic simulation keeps track of all visited gates and, in particular, records gates whose inputs have been set to a controlling value. It also applies to some other devices such as multiplexers, where setting select inputs to a particular combination blocks signals reaching data inputs that have not been selected.

Consider now gates, visited by the above process, whose inputs and outputs are sites of faults. These gates are checked to determine if the presence of their local faults causes the affected terminals to have the same values as those of a fault-free circuit. If so, such faults would clearly be test escapes. As a result, an inversion $\overline{v(s)}$ of a given primary stem value is assigned

to these faults as a NA required for their detection. Furthermore, for every gate G with an input having a controlling value, all propagation paths are traced back leading to the remaining inputs of G. This process is aimed at finding faults whose propagation is blocked by those controlling values. In particular, if a fanout branch came across, it is marked as blocked. If the remaining branches of the same fanout are also blocked, then tracing towards primary inputs continues. Again, all encountered faults are assigned the value of $\overline{v(s)}$ as their NA.

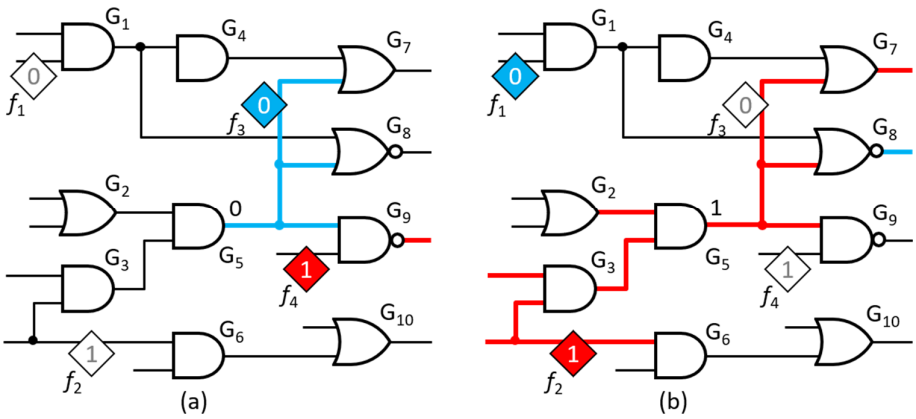

Fig. 3.3. Single-stem-based necessary assignments. Simulation of (a) logic 0 and (b) logic 1.

The above process is illustrated in Fig. 3.3. First, the stem at the output of gate G_5 is set to 0 (Fig. 3.3a). As can be seen, the fault site of fault f_3 has the same value as the fault's polarity, therefore it cannot be excited. On the other hand, f_4 is blocked on gate G_9. Consequently, both have NA on this node of the value 1. In the second step, the same line is set to 1 (Fig. 3.3b). Now, excitation of fault f_2 is impossible and fault f_1 could not be propagated to any observable output. It is worth noting that even though f_1 has two propagation paths through gates G_7 and G_8, both are blocked. As a result, the output of gate G_5 set to 0 became NA for both f_1 and f_2.

3.2 Fanout-free regions and global dominators

Computing NAs within FFRs is a relatively straightforward task which is repeated for every fault. First, a unique propagation path is established from a fault site to the FFR output. The fault site is assigned a test value (fault excitation) which is subsequently implied forward until the FFR output. The same process determines off-path signals (within FFR) that enable fault effect propagation. Because of its simplicity, this is the only case where NAs are assigned to

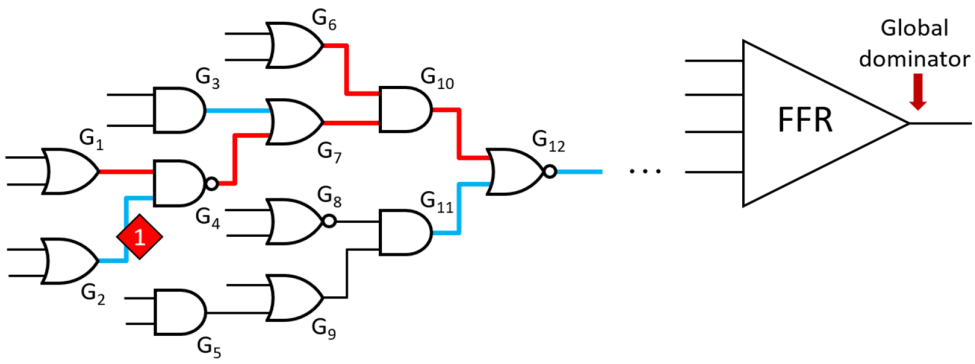

Fig. 3.4. Fanout-free regions and global dominator.

some internal nets of a circuit different than PIs or fanout stems. It is worth noting that in some cases selecting off-path values may not be possible – a two-input XOR gate propagating a faulty effect may serve here as an example. Additional NAs are determined by going beyond the FFR hosting the current fault. This is accomplished by taking advantage of global (absolute) dominators, i.e., lines through which the fault effect has to pass to be detected at any primary output [48]. Moreover, one can move then to another FFR associated with the closest dominator (considering its inversion parity) and essentially repeat operations detailed above. The process terminates once primary outputs are reached or there are no more global dominators that could be used.

The whole process is summarized in Fig. 3.4. First, G_2 set to 0 is added to the list of NAs because it is a value needed to excite a fault. Next, as the analysis moves forward to the FFR output, all visited gates are included in the list of NAs: $G_4 = 1$, $G_7 = 1$, $G_{10} = 1$, and $G_{12} = 0$, respectively. Moreover, the NAs list is complemented by the first part of each off-path set to the non-controlling values as follows: $G_1 = 1$, $G_3 = 0$, $G_6 = 1$, and $G_{11} = 0$.

3.3 Combined simulation

NAs gathered in the two previous steps (see Sections 3.1 and 3.2) are now reused jointly to launch, for each fault separately, the event-driven fault-free logic simulation again. The objective here is to discover additional NAs that can only be computed provided several NAs are analyzed simultaneously rather than individually. Initially, the event list includes all NAs obtained so far for a given fault. Subsequently, forward implications, backward justifications, and further forward implications are run in a manner similar to that of the first phase discussed

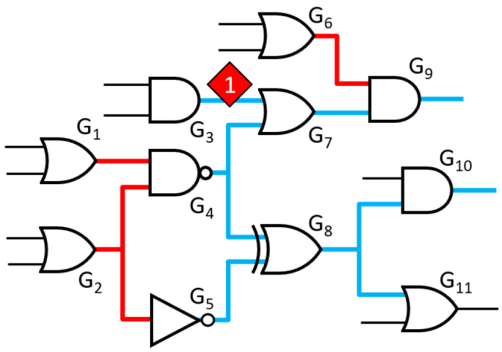

Fig. 3.5. Combined simulation.

earlier. Whenever the simulation hits a primary stem, its value is saved as a new NA for a given fault.

Consider, for example, a fault shown in Fig. 3.5. Gates G_2 and G_4 are among those for which NAs have been computed in the previous steps. Propagation of these signals individually would stop at XOR gate G_8 because only a single input would always be specified, and thus the output of G_8 could not be determined. When working synergistically, however, simulation can proceed further, and logic values associated with gate G_8 can also be added to the list of NAs for the indicated fault. Moreover, because logic 0 is the controlling value of gate G_{10}, the procedure may continue to find more NAs.

3.4 Contrapositive learning

In the process of collecting NAs as shown above, a learning procedure is used to reveal additional relationships between gates set currently to controlling values and other signals that they may imply. Given such a gate, one can set its output to a non-controlling value, and then can run the backward justification followed by forward implications of signals implied at branches

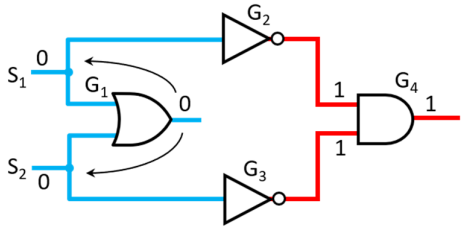

Fig. 3.6. Contrapositive learning.

of stems encountered during the backward justification. For each gate set to its newly obtained output non-controlling value, the Boolean contrapositive relation is used the same way it is done in [77], i.e.,

$$(x \Rightarrow y) \Leftrightarrow (\bar{y} \Rightarrow \bar{x}) \tag{3.1}$$

to learn contrapositive implications towards the initial controlled gate. As an example, consider a circuit shown in Fig. 3.6. If gate G_1 was originally set to 1 (a controlling value), now its output is set to a non-controlling value of 0 which implies $S_1 = 0$ and $S_2 = 0$. As a result, gate G_4 has 1 on its output. Using now the contrapositive learning, the reasoning is as follows:

$$(G_1 = 0 \Rightarrow G_4 = 1) \Leftrightarrow (G_4 = 0 \Rightarrow G_1 = 1) \tag{3.2}$$

In other words, having G_4 set to 0 implies G_1 being set to 1. Such a relation is saved and subsequently used as a NA for G_1 whenever other procedures will arrive, for example, with the value of 0 assigned to G_4.

3.5 Constructive dilemma

This technique employs a logic rule of inference, also used in [77], to get further NAs in a cost-effective manner. It is known as the constructive dilemma:

$$[(x \Rightarrow y) \wedge (\bar{x} \Rightarrow y)] \Rightarrow y \tag{3.3}$$

This approach is used in conjunction with the combined simulation step for a given fault. Once the simulation is completed, one can collect unate gates with only one input x left unspecified. In such a manner, only a small fraction of all simulated gates is processed without visibly reducing the number of new NAs. Input x is first set to 0, and then to 1. What follows in both cases is again the 3-step logic simulation, as done before. Its second run allows us to identify line(s) y whose value remains the same in both simulation steps, that is, the status of y can be

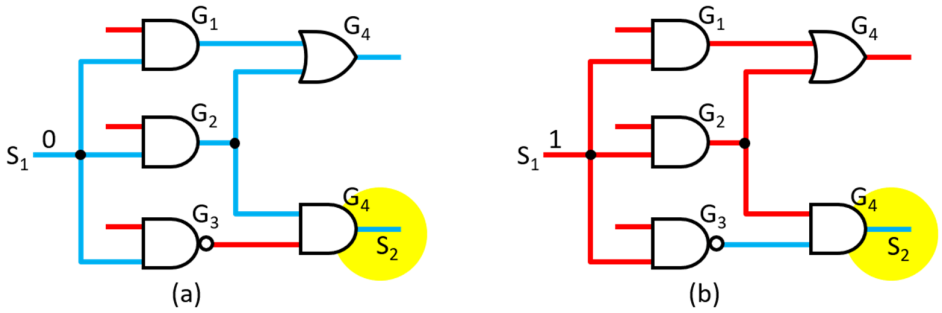

Fig. 3.7. Constructive dilemma. Simulation of (a) logic 0 and (b) logic 1.

deduced independently of a value assigned to *x*. The value of line *y* becomes then the next NA for the current fault. Consider an example of Fig. 3.7. After the combined simulation, gates G_1, G_2, and G_3 have only one input left unspecified. Therefore, simulating stem S_1 in conjunction with formula (3.3) leads to the following result:

$$[(S_1 = 0 \Rightarrow S_2 = 0) \wedge (S_1 = 1 \Rightarrow S_2 = 0)] \Rightarrow (S_2 = 0) \tag{3.4}$$

Node S_2 remains intact in both simulation runs of S_1. As a result $S_2 = 0$ becomes a new NA.

3.6 D-frontier

Finally, for every fault being analyzed, the well-known ATPG-originated concept of a D-frontier is adapted. Starting with the output of FFR hosting a given fault, one can check every branch of this fanout stem to determine its status as a potential fault propagation path. If all of them but one are blocked, for example, due to unate gates having controlling values on their inputs, the only propagation path is further examined by setting off-path non-controlling values as the fault's NAs and by moving on to the following FFR, as described earlier.

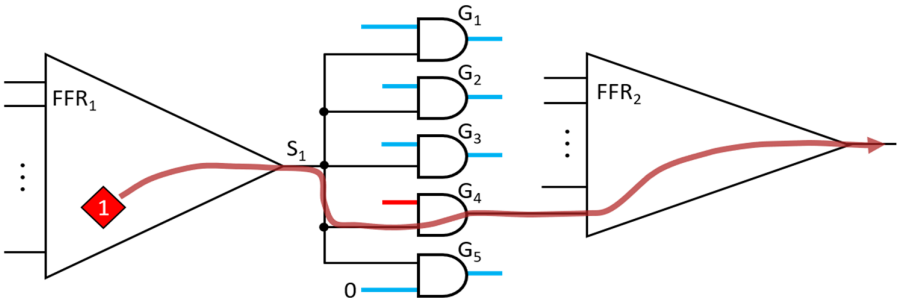

Fig. 3.8. D-frontier.

Fig. 3.8 shows an example of the above process. A stuck-at-1 fault propagates through FFR_1 to stem S_1. Now, there are five possible propagation paths. Since gates G_1, G_2, G_3, and G_5 are blocked due to NAs obtained in the previous phases, the fault effect propagates through gate G_4, and the analysis of Section 3.2 can be repeated for FFR_2.

Chapter 4
Multitarget cube generation

This chapter proposes a new test set compaction technique. It can be briefly summarized as follows. Given a fault list, the corresponding ATPG-produced test patterns are fault simulated with a high and user-defined *n*-detect threshold. As a result, each fault is assigned a list of patterns detecting this particular defect. These multiple-detect profiles are used synergistically with sets of external and internal necessary assignments associated with every fault (as presented in Chapter 3) to form clusters of likely compatible faults. It is worth noting that this process is based on the principle of dimensionality reduction [52], an approach that has grown rapidly in recent years, where high dimensional data (here a complete set of fully specified test patterns) is replaced by a meaningful representation of reduced dimensionality (here a subset of tests with necessary assignments obtained in a time-efficient manner for selected faults). Subsequently, the clusters of faults are retargeted by the SAT-based ATPG to generate a single test pattern per cluster, what effectively reduces the original test set. This method has been presented, for the first time, at the IEEE International Test Conference in 2021 [23].

4.1 Fault detection profile

Having determined the comprehensive sets of NAs for every fault, the test set compaction procedure proposed in this chapter enters now the first phase of its main flow. It runs a conventional ATPG with dynamic compaction to produce an initial set of test patterns that detect all target faults. These patterns are then fault simulated with virtually no fault dropping. As a result, it allows to count how many times each fault is detected, and thus to form a fault detection profile. For the sake of brevity, a fault detected by *n* test patterns will be designated as *n*D, e.g., 2D, 3D, etc. Furthermore, a fault *n*D will also be linked with a list of *n* associated test patterns that detect it. What follows is creation of a much more detailed breakdown of faults that are detected only once (1D). These faults, detected by just a single test pattern, are often referred to as essential faults [12], [24], [45].

Table 4.1: Fault detection profile.

	f_1	f_2	f_3	f_4	f_5	f_6
p_1	×			×		×
p_2		×		×	×	×
p_3			×	×		
	1D1	1D2	1D1	3D	1D2	2D

The breakdown of 1D faults is used to find out how many faults of this kind share a test pattern. In other words, given a 1D fault f, it is of interest to learn how many other 1D faults are detected by a test pattern detecting f. Let 1Dk denote a fault whose sole test pattern is also detecting other $k - 1$ 1D faults. Clearly, nothing prevents test patterns targeting 1D faults from detecting other faults of type nD, where $n > 1$. For the sake of illustration, consider three test patterns p_1, p_2, and p_3 detecting six faults from f_1 to f_6 as follows:

$$p_1 \rightarrow \{f_1, f_4, f_6\}, \qquad p_2 \rightarrow \{f_2, f_4, f_5, f_6\}, \qquad p_3 \rightarrow \{f_3, f_4\}.$$

The above relations can be represented in a tabular form, as shown in Table 4.1. Indeed, faults f_1, f_2, f_3, and f_5 are 1D as they are detected exclusively by patterns p_1, p_2, p_3, and p_2, respectively. Since f_1 is the only 1D fault detected by pattern p_1, it is labeled as 1D1. The same rule applies to f_3. Fault f_2 is 1D as well, but pattern p_2 detects another 1D fault (f_5), and thus both faults get label 1D2. The remaining two faults are detected three times (f_4) and twice (f_6).

A simple experimental multiple-detect profile is shown in Table 4.2. An entry in column nD gives the number of faults that are detected by exactly n test patterns. These results were obtained for ATPG-produced test patterns targeting stuck-at faults in 9 industrial designs (their

Table 4.2: Breakdown of faults nD.

	1D	2D	3D	4D	5D	> 5D
D1	204,074	142,323	96,972	84,662	69,228	749,416
D2	61,881	43,982	26,773	17,871	11,052	221,669
D3	147,410	96,111	72,988	60,464	48,600	898,614
D4	132,414	79,781	65,735	53,981	45,762	699,182
D5	136,765	108,387	86,817	69,856	56,962	709,767
D6	381,265	148,160	48,424	19,740	8,508	134,910
D7	31,616	14,791	16,381	17,898	16,022	294,131
D8	16,266	8,913	7,031	6,209	5,273	6,320
D9	152,838	145,019	143,318	150,483	152,481	2,044,979

Table 4.3: Breakdown of faults 1D.

	1D1	1D2	1D3	1D4	1D5	1D > 5
D1	6,748	536	477	1,576	295	51,778
D2	3,930	3,226	1,344	988	615	51,778
D3	9,435	16,314	5,370	4,804	4,195	107,292
D4	3,225	2,116	1,770	1,668	1,475	122,160
D5	635	1,356	2,127	1,368	1,225	130,054
D6	5,750	4,414	2,490	1,760	1,405	365,446
D7	915	470	270	200	110	29,651
D8	4,369	4,826	2,523	1,560	790	2,198
D9	1,080	8,176	9,978	7,860	4,320	121,424

detailed characteristics can be found in Section 4.3). Table 4.3 illustrates a more detailed breakdown of 1D faults (their total number is listed in column 1D of Table 4.2). Each entry in column 1D*k* gives the number of 1D faults that belong to this particular category, i.e., a class of faults being uniquely detected by a pattern that detects, in total, *k* 1D faults. As can be seen, class 1D1 is typically the most populated one with some noticeable exceptions though (designs D3 or D5). It is a good starting point for the actual test pattern reduction, as shown in the next sections. It will also be shown that having a healthy population of 1D2 faults can be beneficial in some cases as well.

4.2 Reduction of patterns

4.2.1 Fault grouping

The ranking of faults from the previous section as well as their necessary assignments (obtained by applying the methods of Chapter 3) can now facilitate grouping of faults in such a way that subsequent processing of the corresponding test patterns may lead to a more compact test set. It begins by sorting 1D1 faults such that they are in order by largest NA counts (note that this approach may prefer faults located in large FFRs as NAs within an FFR hosting a given fault are not limited to primary stems – see Chapter 3). The fault grouping procedure works as follows (see also a flowchart in Fig. 4.1 for 1D1 faults). First, pick two compatible 1D1 faults f_1 and f_2 with the highest NA scores. For the sake of this work, two faults are assumed compatible if their NAs do not contradict each other. Moreover, these faults can only be accepted for further processing provided their test patterns do not share any 2D fault f_3 they both detect. This is to

avoid a situation in which a novel test that is to be generated does not cover f_3 anymore. If fault f_1 cannot be paired with any other fault f_2, then f_1 is moved to the end of the 1D1 faults list while the process continues by trying other pairs of 1D1 faults.

A pair of faults (f_1, f_2) that has been accepted becomes the subject of the optimized SAT-based multiple-target test generation [20]. If the SAT solver succeeds in finding a single test pattern that covers both faults f_1 and f_2, then a conventional dynamic compaction tries to add more faults that could be detected by a newly created test pattern (see also Section 4.2.2). The next step is to discard the former test patterns detecting f_1, f_2, and other faults added by the dynamic compaction, replace these patterns with the SAT-generated vector, and drop all the corresponding faults. After deleting the test patterns, one important task has to be accomplished: check to see whether deleting a given test pattern brought any non-1D faults closer to class 1D. This task is implemented with a single scan through the list of non-1D faults. It updates status of faults that were covered so far by the just-deleted test patterns. In particular, a 2D fault may now become a member of a class 1D*k*, respective 3D faults are elevated to the rank of 2D, and

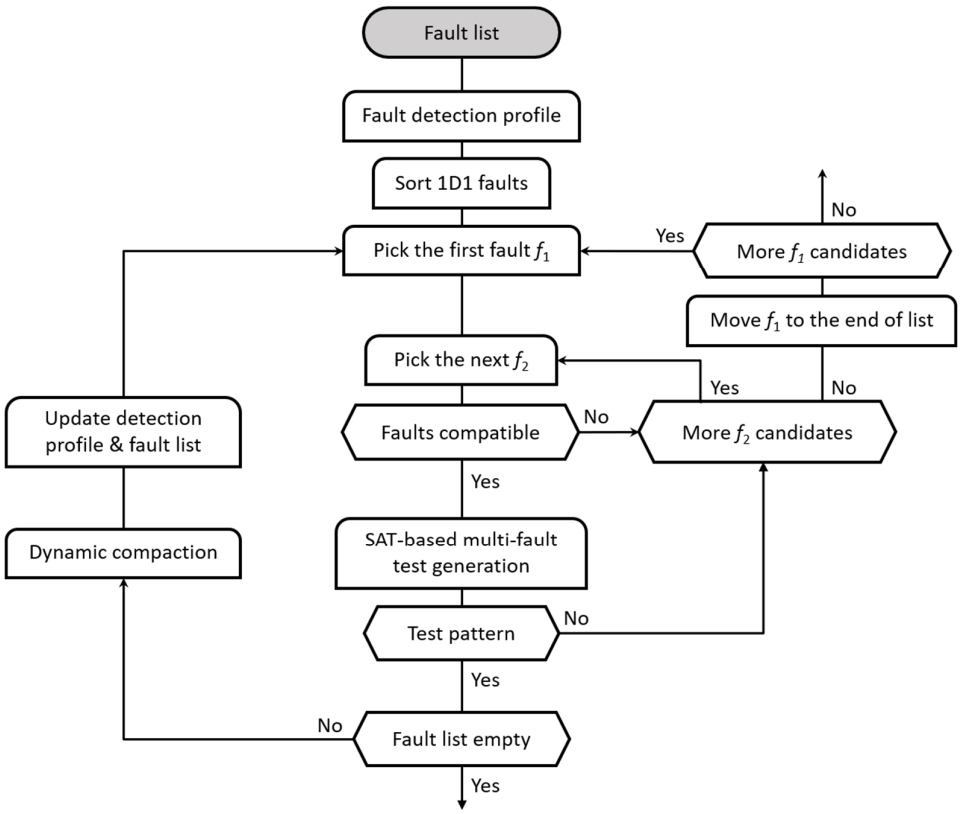

Fig. 4.1. 1D1 fault grouping flow.

so on. At this point, the process is ready to pick another pair of 1D1 faults, and to carry on, as described above. Continuing in this way, it is ensured that there are no two 1D1 faults left that could be paired. On the other hand, certain 1D1 faults may still remain on the list as standalone items that cannot be coupled with any other 1D1 fault, unless other 1D faults would be tried instead, as shown below.

Consider now 1D2 faults. As in the approach for 1D1 faults, the first step is to rearrange 1D2 faults so that they are ordered according to the number of their NAs. Let f_1 be now a 1D2 fault with the largest NA count. Next, the 1D2 fault f_2 that shares a test pattern with f_1 is picked. Clearly, these faults are compatible by definition. Having combined their NAs, a 1D1 fault f_3 is selected that is compatible with both f_1 and f_2. Moreover, as shown earlier, test patterns p_{12} and p_3 detecting faults f_1, f_2, and f_3, respectively, cannot share any 2D fault they both detect. The SAT solver is then deployed to find a single test pattern that covers faults f_1, f_2, and f_3. If it succeeds, all relevant data regarding faults is updated, and the process is repeated for other possible choices of two 1D2 faults. If the SAT solver fails to deliver a desired test pattern, another attempt to pair one more suitable 1D1 fault with the 1D2 faults f_1 and f_2 is made. This method may continue through several selections of a 1D1 fault. Eventually, it will run out of possible choices and will terminate. In this case, it is possible to investigate potential solutions with 1D1 faults being paired with faults belonging to classes 1D3, 1D4, etc. How far this method may go depends on the constraints regarding resultant run time, diminishing returns effects, and other circuit-dependent factors.

The fault coupling process can be generalized in a twofold manner. First of all, one may attempt to pair 1Dk faults with other 1Dk faults, for $k > 1$. It might be followed by coupling of 1Dk faults with faults of type 1Dn, where $n = k + 1, k + 2, \ldots$ Moreover, given the ability of the SAT-based ATPG to produce a single test pattern for several faults in parallel, another technique goes beyond the limits set by the process of replacing just two original test patterns with a single (new) one. From the very beginning, it picks 3, 4 or more faults to be processed simultaneously. In principle, the approach works as described in this section. There are, however, some differences. If three 1D1 faults are selected, then their original test patterns cannot solely detect 2D and 3D faults. Working with four 1D1 faults would preclude from merging faults whose patterns are sole tests for some 2D, 3D, and 4D faults, and so on. Clearly, there is a trade-off between the degree of compaction that can be achieved here and the effort spent on selecting

successive *k*-tuples of faults, where $k > 2$, and running the SAT solver in each case. Consequently, to prevent excessive run time, a parameter whose exact value depends upon the implementation is deployed to bound the number of faults that can be targeted by one instance of a test generator.

4.2.2 SAT-based ATPG formulation for multiple targets

A SAT solver, such as MiniSat [21], accepts as input a Boolean formula in conjunctive normal form (CNF), also called the SAT instance φ. A CNF φ is a conjunction of clauses, and a clause is a disjunction of literals. A literal is a Boolean variable x or its inverse $\bar{x}$. Clearly, to satisfy a CNF, all clauses have to be satisfied, and to satisfy a clause, at least one literal has to be satisfied. This homogeneous problem formulation enables application of very effective reasoning and learning techniques.

Invoking the SAT solver to generate test patterns is preceded by the following steps:

1. Each signal line in the circuit is assigned a Boolean variable to represent the logic state 0 or 1 of this net.
2. Each gate or cell g is transformed into a set of clauses φ_g using the Boolean variables of the connected signal lines.
3. The CNF of φ_C of a circuit or circuit part C is then formed by combining the CNF formulas of all its gates (cells). It represents the characteristic function of C.
4. Fault detection implications (constraints) for a fault *f* are transformed into a CNF formula φ_f. This consists of the faulty part of the circuit, miter and/or D-chain structures establishing links between the good and the faulty circuit.

The final SAT instance for test generation φ_f^{TG} consists of the conjunction of the CNF for the circuit part and the fault detection constraints:

$$\varphi_f^{TG} = \varphi_c \cdot \varphi_f \tag{4.1}$$

This SAT instance is given to the solver. If a solution is found, a test can be directly extracted through the solution assignments. Otherwise, the fault is untestable. Typically, a comprehensive structural analysis is deployed to reduce the number of variables, the SAT instance size, and the resultant test cube.

The presented formulation for single faults can now be extended to target multiple defects. For that, the faulty parts of each fault $f_1, f_2, \dots, f_n$ and their detection constraints $\varphi_1, \varphi_2, \dots, \varphi_n$ have to be added to the overall SAT instance. While the CNF of the good circuit

φ_c can be shared by all target faults, the faulty parts have to be defined for each single fault leading to the following formula:

$$\varphi^{TG}_{f_1 \dots f_n} = \varphi_c \cdot \varphi_{f_1} \cdot \dots \cdot \varphi_{f_n} \tag{4.2}$$

The SAT solver returns a test that detects $f_1, f_2, \dots, f_n$ provided all faults are mutually compatible. However, this approach fails if at least two faults are incompatible with each other. To alleviate this deficiency, the above decision problem can be reformulated to an optimization one with the objective of finding a test that detects the largest possible number of faults out of a given fault list. The corresponding optimization function θ can be written as follows:

$$\theta = d_1 + d_2 + \dots + d_n \tag{4.3}$$

where $d_1, d_2, \dots, d_n$ are Boolean variables representing the detection status of faults $f_1, f_2, \dots, f_n$, respectively. They are linked to the faulty parts in the CNF.

Assigning the value of 1 to variable d_i forces detection of fault f_i, i.e., it triggers a D-chain for this fault. Setting d_i to 0 ignores detection of the fault. Clearly, having all variables set to 1 would yield the largest number of detected faults. As this may not lead to a test (see above), an incremental SAT solving strategy is used to find an optimal solution. During this process, the detection variables are dynamically reassigned to find the optimal test. Furthermore, the SAT solver takes advantage of its conflict-driven learning strategy. It allows the solver to reuse information in a series of calls to prune the search space [22]. Eventually, the optimization procedure returns an assignment of the fault detection variables as well as the satisfying assignment of the SAT instance from which a test can be extracted.

The SAT-based ATPG of [24] has been adapted to work with the proposed test compaction method by including additional information into the SAT instance. In particular, the solving process has been strengthened with NAs obtained for each target fault (see Chapter 3). These assignments allow the solver to prune the search space and to reduce the complexity of the SAT instance. In the following, it is shown how NAs are modelled.

Let a necessary assignment be a triple (*f, x, v*), where *f* is a fault affecting a signal line *x*, and *v* is a logic value. Each NA can be transformed into an implication which is then added to the SAT instance $\varphi^{TG}_{f_1 \dots f_n}$. The following steps illustrate how to carry out this transformation. In addition to the detection variable d_f, the Boolean variable *x* representing the necessary state of line *x* is used. With these two variables, the implications are formulated depending on the logic value of *v* as follows:

$$\begin{aligned} v = 0 &: d_f = 1 \Rightarrow x = 0 \\ v = 1 &: d_f = 1 \Rightarrow x = 1 \end{aligned} \tag{4.4}$$

Each implication is transformed into a clause:

$$\begin{aligned} v = 0 &: \overline{d_f} \vee \bar{x} \\ v = 1 &: \overline{d_f} \vee x \end{aligned} \tag{4.5}$$

Note that each implication is unidirectional only. When fault *f* is not detected, the implication will be dynamically ignored during the solving process since the assignment $d_f = 0$ already satisfies the clauses. On the other hand, the assignment $d_f = 1$ causes the literal to be false, and thus the clause can only be satisfied by the correct assignment of *x*. Once the implications/clauses are formulated and added to the SAT instance $\varphi^{TG}_{f_1 \ldots f_n}$, the regular solving process can be applied to find a test.

A newly generated test pattern goes back to the dynamic compaction procedure. It attempts to add more faults by following the conventional rules, i.e., it first reclaims all necessary assignments and their implications that the new test pattern consists of, and then checks, if tests for additional faults can be added without violating earlier assignments. A distinct feature of this phase is an order in which fault candidates are tried. Dynamic compaction in our case starts with faults of type 1D, followed by faults 2D, 3D, etc. Within class 1D, faults are ordered as discussed in Section 4.2, i.e., 1D1, 1D2, and so forth.

4.3 Experimental results

The new test set compaction scheme has been verified by conducting a series of experiments with several industrial cores ranging in size between 218K and 7.8M gates. The basic data regarding these circuits such as the number of gates, the number of scan cells, the number of scan chains, the size of the longest chain, and the number of stuck-at faults are listed in Table 4.4. In all experiments, the original test patterns are produced by a state-of-the-art commercial ATPG tool. Furthermore, every test pattern set is split into two unequal parts: a relatively small group G_1 of tests which nevertheless detect a significant percentage (usually greater than 90%) of faults, and a much larger set G_2 of patterns that cover the fault list tail end. The experiments presented in this section focus primarily on the second group of patterns (faults). It is beneficial in two ways: one can still expect a visible reduction of the total pattern count by working exclusively with patterns of group G_2. At the same time, a procedure handling a relatively small

Table 4.4: Circuit characteristics.

	Gates	Scan cells	Scan chains	Longest chain	Stuck-at faults
D1	2.4M	181K	1,365	134	4,049,753
D2	218K	14.2K	54	263	349,286
D3	2.1M	143K	400	360	3,053,898
D4	2.5M	174K	114	1,964	4,731,360
D5	2.1M	148K	70	2,579	3,612,124
D6	1.2M	97.8K	300	327	2,421,874
D7	3.1M	169K	69	2,456	3,977,620
D8	103K	1,140	25	46	229,550
D9	7.8M	429K	857	502	10,874,455

subset of the entire fault list is likely to run significantly faster than any other method tackling the complete fault list.

Table 4.5 summarizes the results of the experiments for stuck-at faults, including a detailed breakdown of test patterns produced by the SAT-based ATPG of Section 4.2.2 invoked for faults being clustered, as shown in Section 4.2.1. The successive columns of the table list the following data:

- test coverage (TC) of deterministic patterns produced by a conventional ATPG,
- the number of test patterns (TP_1) in group G_1; recall that those patterns are not directly subjected to the test compaction process,
- the number of test patterns (TP_2) in group G_2 which forms the primary source of input data for the compaction method presented in the thesis,
- the total number of patterns obtained after running the SAT-based ATPG (SAT TP); note that this number is also taking account of patterns that the SAT solver was unable to replace with new patterns detecting some additional faults,
- the next six columns provide a detailed breakdown of those patterns that were specifically produced by the SAT-based ATPG; every column shows the number of test patterns obtained by using one of the corresponding (and the simplest) fault-pairing schemes, from pairs (1D1, 1D1) up to pairs of patterns detecting solely three faults each, i.e., (1D3, 1D3),
- the resultant test pattern reduction, i.e., a difference between the numbers reported in columns TP_2 and SAT TP divided by TP_2.

Table 4.5: Experimental results – stuck-at faults.

	TC [%]	TP_1	TP_2	SAT TP	1D1-1D1	1D1-1D2	1D1-1D3	1D2-1D2	1D2-1D3	1D3-1D3	Reduction
D1	99.19	1,894	7,290	2,196	2,176	12	0	0	0	0	69.88%
D2	99.81	1,806	6,841	2,248	1,718	211	29	9	4	2	67.14%
D3	96.37	5,265	20,843	18,049	2,538	171	5	0	0	0	13.40%
D4	90.72	5,504	5,556	4,873	240	215	42	5	34	16	12.29%
D5	98.10	2,047	4,543	4,033	236	190	2	2	23	2	11.23%
D6	97.45	19,543	5,750	4,137	1,674	0	0	0	0	0	28.05%
D7	93.53	272	1,573	1,501	50	11	1	0	10	0	4.58%
D8	97.89	1,252	8,226	6,007	1,256	0	0	308	297	41	26.98%
D9	93.00	3,691	11,158	10,439	668	0	0	0	0	0	6.44%

The last column of Table 4.5 clearly indicates that the test set compaction scheme proposed in the thesis is capable of producing compact test sets, and in several cases it compares favorably with the state-of-the-art ATPG that was used to deliver the original test sets of size listed in columns TP_1 and TP_2. Interestingly, for some designs, the pattern counts of the new approach are significantly lower than those of the standard ATPG and its compaction techniques, while test coverage remains uncompromised. In other test cases, the pattern count reduction is not so spectacular. Although the pattern count reduction appears to be the case across all designs, its degree remains a strong circuit-specific factor. Our analysis (see Section 4.1) and experiments clearly confirm that there is a class of circuits whose structural properties allow the contemporary commercial ATPG tools to produce near-optimal test sets. However, there do exist other industrial cores, circuits, and designs that may challenge the state-of-the-art ATPG algorithms. The method presented in this chapter provides the ability to thrive in such complex scenarios and to push new limits of the ATPG technology.

Chapter 5
Hypercompression of test data

The chapter presents a next-generation test data compression scheme. It builds on the isometric compression paradigm but makes it more flexible and elevates encoding efficiency to values unachievable through state-of-the-art sequential compression schemes. Furthermore, its programmable selection of full toggle scan chains ensures high test coverage and virtually eliminates compression aborts. A redesigned low-silicon-area decompressor is also capable of reducing switching rates in scan chains with a new test power control scheme, as was initially shown in [43].

5.1 Decompressor architecture

Recall that the isometric decompressor houses a circular template register whose size matches the longest scan chain, as shown in Fig. 2.6. The template register controls the decompressor by providing a control bit to the hold register every shift cycle to indicate whether this register is to be updated by the ring generator. To avoid a sizeable template register (or registers) of the isometric decompressor, a new decompressor architecture (Fig. 5.1) is proposed here that deploys a much smaller template register, typically no longer than 32 bits. Again, this circular register provides a control bit to the hold register every scan shift cycle to indicate whether this register should be reloaded with the current content of the ring generator. Because of its size, however, the very same short test template is now going to be used multiple times within duration of the same test pattern. Back to Fig. 2.4, the repetitive use of a 4-bit test template 0101 may suffice to handle the second test cube, and to designate all necessary toggle points.

Although the use of short circular test templates can still deter abnormal scan toggling and reduce the number of specified bits that need to be encoded, applying the same control bits several times may require additional ATPG constraints to secure compression of a given test cube, as further discussed in Section 5.3. It is also important to observe that short test templates make the new scheme very flexible. It is now capable of deploying a test template for

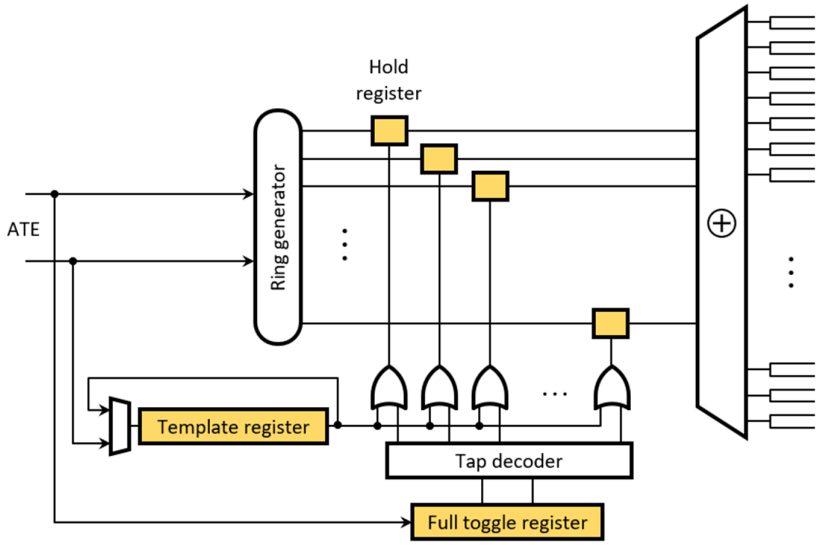

Fig. 5.1. Hypercompression architecture.

any number of test patterns, including a single vector. This is because an update of a small test template register requires no additional patterns and/or ATE channels, and time needed to do this is a negligible fraction of a regular test pattern upload period.

The function of the new scheme to control every segment of the hold register individually is another feature, for which the approach of [51] has no comparable capability. The hold register can be implemented, for example, as a set of latches, where every data input is fed directly by the corresponding stage of the ring generator, while enable inputs are driven by the corresponding outputs of a *tap decoder*. If the output k of the tap decoder is asserted, then the kth segment of the hold register becomes transparent allowing the kth output of the ring generator to feed directly the phase shifter, and thus scan chains driven by XOR gates connected to the kth stage of the ring generator. As a result, these scan chains enter the full toggle mode regardless of the current test template. The *full toggle register* interfacing the tap decoder with an external tester is loaded once per pattern. Its binary-coded content determines stages of the hold register that should remain transparent. Consequently, the same content defines the fraction of scan chains that may receive a full toggle stimulus.

The functionality offered by the circular template register and the tap decoder allows one to select dynamically certain stages (taps) of the hold register to stay in the transparent mode, while feeding scan chains with test patterns. As a result, a subset of scan chains – called *full toggle scan chains* – driven by such taps can toggle every scan shift cycle rather than at selected toggle points. This approach prevents scenarios where certain faults escape detection (leading

to a coverage drop) because they need more frequent changes in a given scan chain than a test template could permit. For the remaining scan chains, toggle points and hold cycles are determined by the content of the template register. When a 1 reaches the rightmost position of the register, these scan chains capture the ring generator content processed by the phase shifter, otherwise they hold their current state.

Experimental evidences indicate that typically it suffices to have at most two hold latches in the transparent mode. To accommodate just two transparent stages of the hold register in an n-bit test data decompressor, a pair of 1-out-of-n decoders driven by two associated full toggle registers can be deployed, as illustrated in Fig. 5.2. For each test pattern, this circuitry selects two of n hold latches to be fed directly by the ring generator (such latches will also be referred to as *full toggle taps*).

An important figure of merit when introducing a new DFT scheme is its test logic silicon real estate. As shown above, the hypercompression test logic (HTL) requires one hold latch and one 3-input OR gate per a single bit of a ring generator. Furthermore, it comprises a template register, typically a 32-bit device, two $\log_2 n$-bit full toggle registers, where n is the number of scan chains, and two 1-out-of-n decoders using a certain number (depending on n) of 5-input AND gates and inverters.

Table 5.1 reports the silicon footprint taken up by HTL in terms of equivalent area of 2-input NAND gates (measured also in mm^2). The presented numbers were computed with a commercial synthesis tool for four industrial circuits. All components of test logic were synthesized

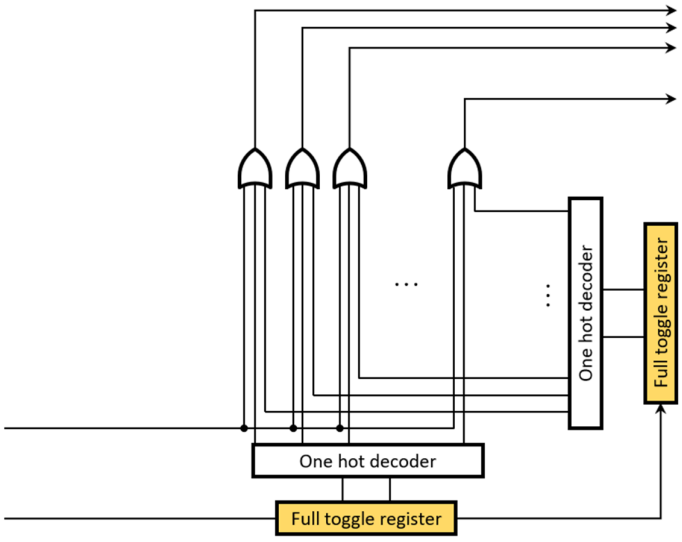

Fig. 5.2. Full toggling decoder with two active outputs.

Table 5.1: Area overhead – 2-input NAND equivalent (and mm^2).

	Conventional scan			Designs with EDT logic		Hypercompression (HTL)	
	Sequential	Combinational	Total	Total	D_E	Total	D_H
C1	462,203 (0.54)	724,627 (0.85)	1,186,830 (1.39)	1,197,765 (1.40)	0.92%	1,200,893 (1.41)	1.18%
C2	749,037 (0.88)	1,057,120 (1.24)	1,806,157 (2.12)	1,812,503 (2.12)	0.35%	1,814,766 (2.13)	0.48%
C3	213,204 (0.25)	71,419 (0.08)	284,623 (0.33)	286,439 (0.33)	0.64%	287,191 (0.34)	0.90%
C4	713,196 (0.83)	1,614,419 (1.89)	2,327,615 (2.72)	2,335,625 (2.73)	0.34%	2,338,450 (2.74)	0.46%

using a 65nm CMOS standard cell library under 2.5ns timing constraint. The table reports the following quantities: the resultant silicon area with respect to combinational and sequential devices for conventional scan-based designs (the first three columns), the total area taken by circuits with on-chip EDT-based test compression, and then the percentage area increase (Δ_E). Subsequently, the total HTL-based area is presented and compared with the corresponding area occupied by conventional scan-based designs (Δ_H). The results of Table 5.1 do not account for a routing cost. Besides two signals to control template and full toggle registers, however, it remains similar to that of conventional scan. As can be seen, the resultant area is comparable to other scan-based DFT methods. Indeed, in testing with power constraints there is typically additional hardware required to: activate a high-speed scan enable signal, moderate *di/dt* through a scan burst capability, gate scan cells to reduce power dissipation during shift, and gate scan out signal to reduce the power consumption during normal operations. Clearly, having HTL logic on a chip may result in slightly more complicated designs with respect to the placement and routing, but, in turn, the new approach further reduces test data volume, power consumption, and it allows for more efficient handling of new types of defects.

Having discussed the new decompressor architecture, an explanation of how to determine a desired circular test template and how to designate 2-out-of-*n* full toggle taps is provided in the following sections.

5.2 Test template synthesis

5.2.1 Toggle ranges

The approach presented in the following works with a set of test cubes corresponding to a subset of faults. It begins by processing each test cube individually, and then moves gradually to

the process of merging these test cubes based on their ability to form a circular test template and to deploy a certain fraction of full toggle scan chains.

The hypercompression can encode a test cube in a low toggling fashion by selecting toggle points in places where scan chains change their content values. Recall that the toggle point requires a hold register update by the ring generator. If two successive specified values in a test cube are separated by scan cells with don't cares, a toggle point can occur anywhere between these scan cells provided they differ in values. Consider the following test cube:

$$x\ x\ 1\ x\ 0\ x\ x\ x\ 0\ x\ x\ x\ 1\ x\ x\ x\ x\ x\ 1\ x\ x\ 0\ x\ x\ x\ x\ 0\ 1\ x\ x\ x\ x\ x$$
$$\uparrow\uparrow \qquad \uparrow\uparrow\uparrow\uparrow \qquad \uparrow\uparrow\uparrow \qquad \uparrow$$

As can be seen, the rightmost toggle point is already fixed. However, locations of three other toggle points remain to be determined as they can occur anywhere between the values of 0 and 1 or vice versa. A sequence of adjacent cells where a toggle point can be located is referred to as a *toggle range*. Let v_1 and v_2 be specified values assigned to scan cells c_1 and c_2, respectively. All cells between c_1 and c_2 are don't cares. If $v_1 \neq v_2$, then the corresponding toggle range, denoted by $[c_1, c_2)$, consists of c_1 and all subsequent cells until c_2 which is, by definition, excluded. If scan cells are numbered from the left starting with 0, then, for the above example, the following toggle ranges are as follows: [2, 4), [8, 12), [18, 21), [26, 27).

The next step is to convert, for every scan chain, its toggle ranges into reduced toggle ranges. For the sake of simplicity, let us assume that given a range $[x, y)$, a toggle point can occur at one of the following sites: $x, x + 1, x + 2, \ldots, y - 1$. Let s be the size of a test template. If a given range consists of more than s cells, it is ignored. Otherwise, every toggle range $[x, y)$ is mapped into a toggle range $[x \bmod s, y \bmod s)$ with potential sites of toggle points converted accordingly. The resultant ranges and the corresponding toggle points become now the subject of a *reduce* operation.

The reduce step transforms the set of converted toggle ranges into another set of toggle ranges by either computing non-empty intersections of toggle ranges or just leaving certain ranges intact, if they do not overlap with any other range. As an example, consider a 15-bit-long scan chain shown in Fig. 5.3. The following list of toggle ranges can be easily determined: [1, 4), [7, 9), [12, 13), assuming that scan cells are numbered from the left, starting with 0. Let $s = 5$. Now the scan chain is divided into three equal parts (black vertical bars), and any toggle range between sections is dropped. The remaining is conversed using modulo s operation resulting in the following ranges: [1, 4), [2, 4), [2, 3). Hence, the resultant toggle point candidates

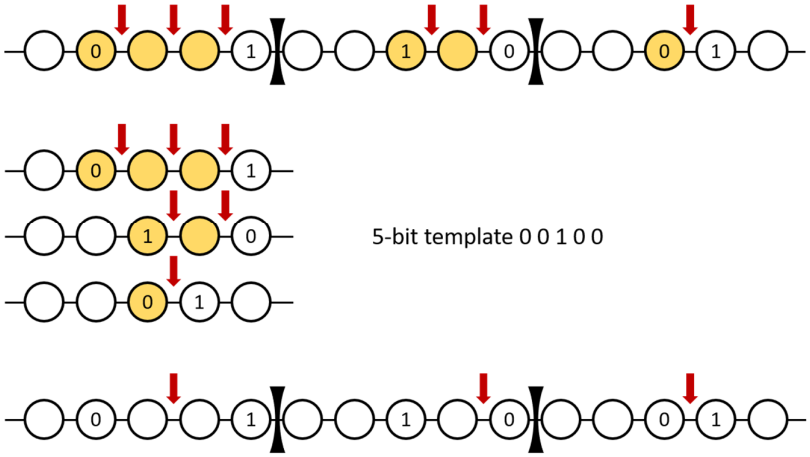

Fig. 5.3. Reduce operation on toggle ranges.

are as follows: {1, 2, 3}, {2, 3}, and {2}. Clearly, their intersection is {2}, and this would be the toggle point location for the scan chain being analyzed here.

In a general case, toggle ranges after reduction are processed as follows. Let r_1 and r_2 be two toggle ranges. If r_1 and r_2 are disjoint, then there is no reduction of toggle point candidates, and both ranges are kept for further processing. If $r_1 \cap r_2 = \emptyset$, then the largest overlap between r_1 and r_2 is picked. The remaining toggle point candidates are discarded. If an intersection comprises more than a single toggle point (or it consists of disjoint subsets), we do not pick any specific toggle points yet. Finally, if more than two ranges overlap in a pairwise fashion, they are processed in an arbitrary order. For example, given ranges r_1, r_2 and r_3, a new range $r_4 = r_1 \cap r_2 \cap r_3$ can be determined. As a result, we obtain a list of reduced toggle ranges for every scan chain. Every list is referred to as a *prime toggle group*.

Prime toggle groups have to be now merged to obtain a set of toggle points that can be shared by as many scan chains as possible provided a desired toggle ratio is not exceeded. First, all prime toggle groups are sorted in ascending order by using the number of reduced toggle ranges every group features as a sort key. Starting from a prime toggle group with the smallest number of ranges, we keep combining prime toggle groups until the number of potential toggle points becomes greater than a predefined threshold representing a target toggling (recall that every reduced toggle range is eventually represented by a single toggle point). We follow here the very same reduction rules as those presented earlier. If a given prime toggle group cannot be merged because of the resultant threshold, a next group is tried until all groups have been examined. Scan chains whose prime groups have not been merged become *full toggle scan*

chains (the green chains in Fig. 5.4), where every care bit will be set individually. The above procedure repeats for every test cube.

5.2.2 Test templates

To make full toggle scan chains operational, one needs to select full toggle taps, typically no more than two – see Section 5.1 and Fig. 5.4. Clearly, identifying the smallest number of taps that drive 3-input XOR gates feeding full toggle scan chains is equivalent to solving a maximum covering problem [15]. To determine full toggle taps, the hypercompression makes use of a greedy approach which first sorts all full toggle scan chains in descending order according to the number of test cubes a given full toggle scan chain is involved. During next iterations, it picks successive full toggle chains and determines their tap drivers. The highest-ranked chain will have three such taps to choose from (if driven by a 3-input XOR gate), and all subsequent chains will either share certain taps with the previously selected ones or narrow down the space of available taps. The procedure keeps trying different full toggle chains to maximize the number of covered faults (or test cubes) until there are no more than two taps available. All test cubes whose full toggle chains cannot be served by the selected taps (the second chain from the bottom in Fig. 5.4) are returned to the pool of test cubes to be merged in next iterations.

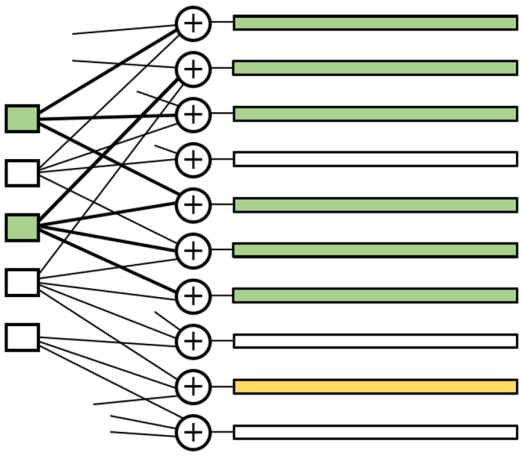

Fig. 5.4. Selection of full toggle taps.

Admittedly, certain full toggle taps can make additional scan chains full toggle ones. Consider a 32-bit hold register feeding 960 scan chains through 3-input XOR gates. As a result, every hold latch has its fan-out equal to 90. Consequently, some of these scan chains have likely been controlled by a test template in the first place. Since they do not need toggle points any

more, the corresponding test template may become the subject of further adjustments. As such a flow is usually complex, a relaxation approach presented below is used to avoid further corrections of test templates. Another scenario occurs when certain test cubes are not covered because not all of their full toggle chains can be enabled. These test cubes remain in the pool of cubes to be merged in subsequent iterations.

We now turn to the problem of forming an s-bit test template that could be shared by non-full-toggle scan chains of several test cubes. Initially, each cube in a buffer is assigned the number of bits to encode with no templates involved. Such cubes can be obtained by using a conventional ATPG. Subsequently, a *relaxation algorithm* is used that begins with an s-bit test template set to the all-1 vector (with such a template, the scheme is equivalent to the conventional EDT-based compression). It then iteratively determines test cubes that could still be encoded if toggle points corresponding to position k were not used, when $k = 0, \ldots, s - 1$ (bit k of a test template is set to 0). If a given cube remains encodable, we determine a difference G between the original number of its specified bits and the number of specified bits that need to

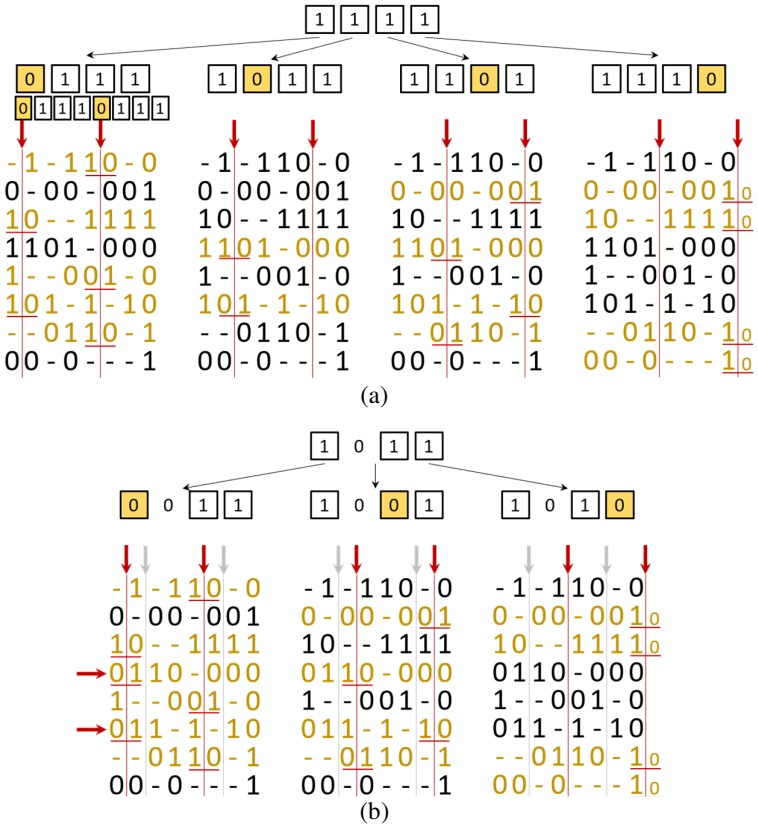

Fig. 5.5. Relaxation: (a) the first phase, (b) the second phase.

be EDT encoded after relaxing a template bit (note that the latter number represents just the leading specified bits of every hold segment). If the cube cannot be encoded anymore (a deleted toggling point has been separating two cells with the opposite values), its gain L from the last iteration is retrieved. Once all cubes in a buffer are examined, their values of G and L are summed. A test-template bit corresponding to the largest value of $\Sigma G - \Sigma L$ is subsequently reset, and all potential toggle points on this position for all test cubes are discarded. Cubes that cannot be encoded are removed from the buffer. One can now refill vacated positions with new cubes under the current intermediate template and carry on looking for next template bits to be flipped to 0. It is important to observe that new test cubes are now produced by a constrained ATPG aware of disabled toggle points.

The whole process continues until the number of bits set to 1 in the test template becomes smaller than a predefined threshold that corresponds to user's low power test requirements. For example, if a scan chain shift toggle rate is to be 25%, then relaxing 50% of template bits can achieve this goal. This procedure can be summarized by the following algorithm (the meaning of variables G and L remains the same as defined above):

```
set all bits in the test template to 1
    while number of 1s in the test template is above threshold do
        for each 1 in the test template do
            set bit to 0
            set score S to 0
            for each test cube do
                if test cube can be encoded, then
                    add G to S
                else
                    subtract L from S
            reset bit with the highest S
            remove all cubes which cannot be encoded from the buffer
            refill test cube buffer
```

Example 5.1: Consider an 8-bit-long scan chain fed by 8 test cubes and a 4-bit test template. Initially, the template is set to 1111 (Fig. 5.5a). In four identical steps, we verify which test cubes could still be encoded if a given template bit were set to 0 (green boxes in the figure). As can be checked, the lack of two toggling points on bits indicated by arrows (recall that the template is circular) precludes encoding of test cubes printed in yellow. For example, the first cube –1–110–0 cannot be encoded because of a disabled toggling point between bits having values of 1 and 0, i.e., exactly where a hold register should be reloaded with a new content of the decompressor. As can be seen, if the first bit of the template were reset, then five test cubes

could not be encoded. Similarly, we verify if the same test cubes can or cannot be encoded when successive template bits are set to 0. It appears that resetting the second template bit leaves the largest number (6) of test cubes unaffected, and thus the test template becomes 1011. The cube buffer is refiled now with two new cubes (replacing those that could not be encoded) and the second round begins (Fig. 5.5b, the horizontal arrows point to new cubes). Again, the impact of setting the template bits to 0 on the ability of the scheme to encode the test cubes is examined. This is done in a similar manner as before. As can be easily verified, de-asserting the third and fourth template bits leaves four test cubes in each case. However, the encodable test cubes in the last column feature one more specified bit than those of the second column, so the template 1010 is picked. For larger test templates, the same process continues until the number of toggle points is smaller than a threshold corresponding to low power test requirements.

Example 5.2. Suppose there are three scan chains and three test cubes with toggle ranges as follows:

Test cube 0:
Chain 0: [7, 9) [14, 17)
Chain 1: [10, 12)
Chain 2: [13, 14)

Test cube 1:
Chain 0: [0, 5) [8, 10) [16, 17)
Chain 1: [0, 10)
Chain 2: [1, 2) [2, 3) [3, 4)

Test cube 2:
Chain 0: [7, 9) [9, 11)
Chain 1: [2, 3) [3, 4) [4, 5)
Chain 2: [6, 7) [7, 8) [8, 9)

Let us assume that there is an 8-bit test template and at most two toggle points are allowed per template. Let us also assume that at most one full toggle scan chain is permitted. After reduction, the prime toggle groups for each scan chain are:

Test cube 0:
Chain 0: [7, 1) [2, 4); prime toggle group
Chain 1: only 2 toggle points allowed, so no more points
Chain 2: full toggle scan chain

Test cube 1:
Chain 0: [0, 1); prime toggle group
Chain 1: toggle range larger than a template
Chain 2: full toggle scan chain

Test cube 2:
Chain 0: [7, 1) [1, 3); prime toggle group

Chain 1: full toggle scan chain
Chain 2: full toggle scan chain

Finally, after computing prime toggle groups and finding a full toggle scan chain, “Test cube 2” is dropped (too many full toggle scan chains). The test template becomes 10100000 with the third scan chain working in the full toggle mode.

5.3 Test compression flow

Test templates of the previous section can guide ATPG to produce highly compressible test cubes. Fig. 5.6 sketches out the proposed test compression flow. As can be seen, the first phase produces test cubes by means of a conventional ATPG for randomly selected faults, forms a corresponding test template, and finds full toggle taps. These steps are controlled by a desired toggle ratio. Test cubes with more full toggle scan chains than allowed or not compliant with the template are dropped, and the corresponding faults are returned to the fault list. The test template is now passed to ATPG aware of recently added constraints. The guided ATPG and

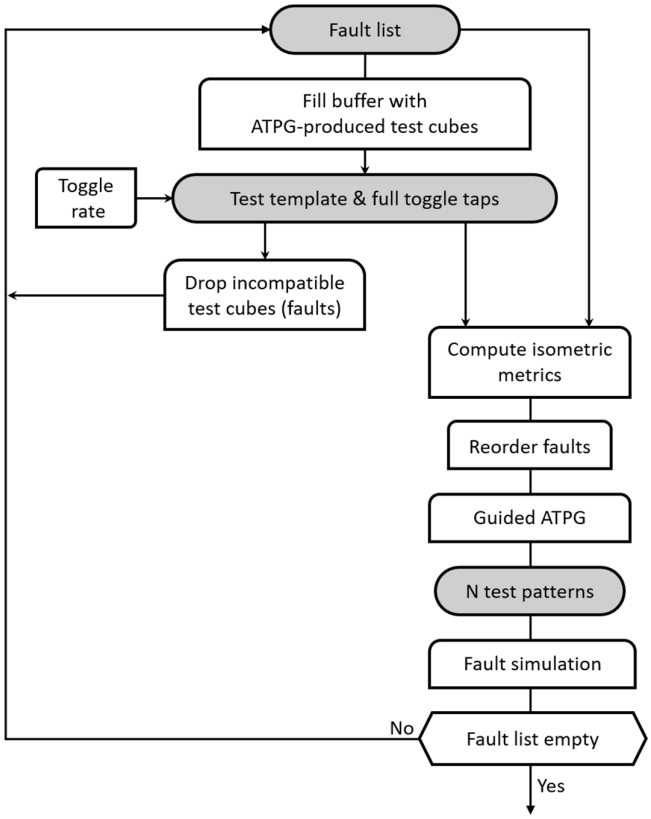

Fig. 5.6. High-level test flow.

EDT-based compression iterate until a given number of test patterns compliant with the newly created test template are produced or a user-defined abort limit is reached. Note that all faults still on the fault list can be used in this phase. The resultant test patterns are fault simulated, and the fault list is updated accordingly. If there are still faults on the list, the method goes back to pick new faults and to produce new test cubes. Otherwise, it terminates.

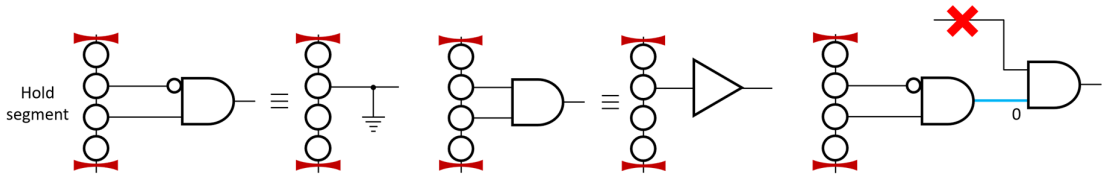

Fig. 5.7. Examples of guided ATPG: (a) buffer surrogate, (b) tie-0 surrogate, and (c) blocked path.

A conventional ATPG working with testability measures may remain unaware of hold segments where several scan cells in a row assume the same value. Ignoring this additional constraint may lead to wrong decisions. Consider a 2-input AND gate driven by cells x and y of the same hold segment in such a way that one of the inputs receives data through an inverter (Fig. 5.7a). Since either $xy = 00$ or $xy = 11$, the output value of this gate will always be equal to 0 as along as this particular test template is employed. Effectively, the AND gate becomes a constant 0 that can be regarded as a surrogate. Clearly, ATPG should not try to justify 1 over there. If there are no inverters on the inputs of an AND gate, then it behaves like a buffer (Fig. 5.7b). Furthermore, hold segments may also impact observability of certain nets. If the AND gate, replaced by the constant 0 surrogate, drives another AND gate, then this particular gate is blocked as well (Fig. 5.7c). Consequently, ATPG should not propagate faults toward this logic.

In order to guide the ATPG in a test-template-aware manner, first an identification of all surrogates and blocked gates is needed. This is handled by the following algorithm:

---

for each hold segment **do**
  set all segment's scan cells to 0
  run logic simulation
  find all blocked gates G_0
  set all segment's scan cells to 1
  run logic simulation
  find all blocked gates G_1
for each gate g whose output is either 0 or 1 **do**
  if gate g has the same value in both simulations **then**
    replace g with its proper constant surrogate
  else
    replace g with its buffer surrogate

```
Simulate design several times using surrogates inserted earlier
Recompute testability measures with surrogates
```

The above procedure uses an auxiliary function to find all blocked gates due to asserted or de-asserted scan cells of a given hold segment. The basic steps of this function are as follows:

```
for each gate g with one input set to a controlling value do
    trace backward from each non-controlling input of g
    if all fan-out branches of a visited gate v are blocked
        gate v is blocked
        trace all inputs of v
    else
        stop tracing the branch leading to v
if gate g is blocked in both simulations
    gate g is always blocked
    set observability of g to 0
Use blocked gates in subsequent simulations
```

Experiments run on several large industrial designs indicate that fault reordering paving the way for the guided ATPG can yield better compression results. In particular, one can assign weights to successive faults and sort the entire fault list by using these weights. Every fault weight is computed as a ratio of the fault's new testability (taking into account rules like those of Fig. 5.7) and the same fault's testability determined in a conventional fashion, e.g., by using the SCOPE-like metrics. A potential testability increase is only taken into account if its value is greater than a predefined threshold r (experiments reported in the thesis use $r = 1.1$).

5.3.1 Implied values

Detection of so-called implied values is another mechanism that allows the hypercompression to foster a close interaction between ATPG and encoding. This approach is aimed at reducing CPU time needed to generate and compress test cubes. It guides back ATPG in such a way that ATPG does not assign certain nodes values that otherwise lead to conflicts, compression aborts, and backtracks. These objectives are accomplished by trimming the number of possible test cubes ATPG may produce and merge by using values that ATPG must consider necessary assignments as they become gradually available [28]. Clearly, savings due to a reduced number of decision nodes ATPG is to go through might be partially nullified by the time needed to determine all implied values. Within the EDT framework, however, this problem is solved by working with linear equations to quickly arrive with implied values given the current status of a compression solver.

Recall that the EDT compression treats the external test data as variables forming linear expressions assigned to scan cells [72]. A compressed pattern is then obtained by solving a system of linear equations in GF(2). Typically, Gaussian elimination determines the reduced row-echelon form of an equation system by picking leading variables. Suppose there are k specified scan cells at certain point. To make sure that these assignments can be encoded, one has to solve a partially specified set of equations. If there is no solution, an ATPG backtrack is needed since a test cube in its current form cannot be compressed.

If the k specified bits can be encoded, then one can add another specified cell, thus forming a set of $k + 1$ equations. There are now 3 possible scenarios. (1) The newly added equation is independent, and thus a cube with $k + 1$ specified values can be encoded. As a result, ATPG carries on. (2) The newly added equation is linearly dependent on others, but it remains solvable. As before, ATPG can continue. Note, however, that the new value could be determined prior to the actual ATPG assignment as it is a linear combination of the earlier assignments. (3) The newly added equation is linearly dependent on others, and it is not solvable. Clearly, ATPG has to backtrack in this case. As previously, the value of cell $k + 1$ could be implied (here with the opposite value to the desired one) once the k scan cells have become specified. Being aware of this relation, one could constrain ATPG to avoid setting a scan cell to a value that cannot be encoded.

The hold segments add a new dimension to the above approach. This is illustrated in Fig. 5.8 depicting two scan chains. Here a 9-bit test template forms three consecutive hold segments comprising 4, 3, and 2 bits, respectively. As can be seen, the template is applied twice. Consider two faults f_1 and f_2. Let fault f_1 require 3 scan cell assignments, as shown in Fig. 5.8a. After these assignments are done, the corresponding values can fill the remaining cells within the same hold segments, as depicted in Fig. 5.8b. Furthermore, solving equations representing specified bits of the first step may result, through already determined seed variables, in additional (implied) values, such as those in Fig. 5.8c. As they occur in different hold segments, the remaining cells of the same segments must assume exactly the same values, as shown in Fig. 5.8d. At this point, ATPG attempts to find a test cube detecting fault f_2. Suppose it requires a logic value of 0 on the output of the AND gate shown in the figure (plus another 0 in the same chain). Since one of the inputs of this gate is already set to 1, the only solution is to reset the other input, as in Fig. 5.8e. Note that ignoring implied values could misguide ATPG and result in a destructive attempt to assign 0 to the other input, which would cause a conflict and invoke

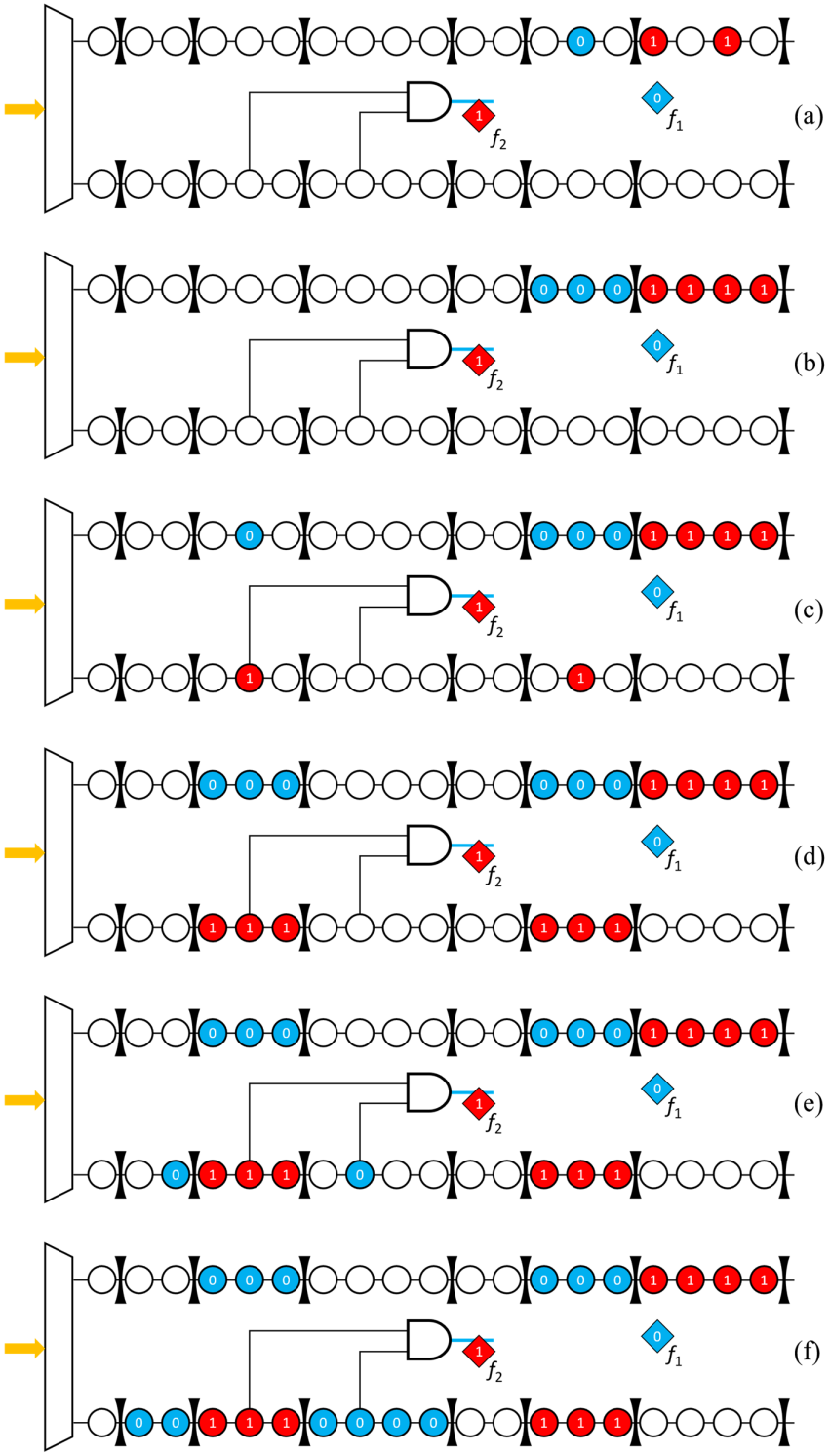

Fig. 5.8. Implied values and hold segments. (a) Assignments for fault f_1, (b) Implied values in hold segments, (c) Additional implied values, (d) Filling hold segments for additional implied values, (e) ATPG assignments for fault f_2, and (f) Final assignments.

time-consuming backtracks. Finally, the remaining scan cells of two hold segments affected by the test cube are filled with the same values, as demonstrated in Fig. 5.8f.

It is worth noting that all implied values discussed above are computed, updated, and passed to ATPG as dynamic constraints. It simplifies the task of test generation as ATPG will trim the search space based on these new constrains and generate compact test patterns with less run time.

5.4 Experimental results

The hypercompression approach has been verified by conducting a series of experiments with several industrial designs and circuits from the IWLS'05 benchmark suit. The basic data regarding the designs such as the number of gates, the number of scan cells, the number of scan chains, the size of the longest chain, the number of EDT input channels, and the size of decompressor are listed in Table 5.2. All experiments are performed with a 30-bit test template and at most two full toggle taps. It is also assumed that the number of toggle points per this particular test template is equal to 15, virtually for all test cases. A test session consists of two repeatable steps: (1) shifting in a test template and the content of two full toggle registers, and (2) applying 64 test patterns corresponding to this template.

Table 5.3 summarizes the results of the experiments for stuck-at faults, obtained for both the conventional EDT-based compression and the hypercompression presented in the thesis. The successive columns of the table list the test coverage (TC), the number of test patterns (TP),

Table 5.2: Circuit characteristics.

	Gates	Scan cells	Scan chains	Longest chain	EDT channels	EDT size
D1	1.2M	72K	400	181	1	22
D2	3.2M	281K	960	293	2	32
D3	4.8M	287K	960	299	2	32
D4	1.3M	52K	200	260	1	32
D5	3.2M	213K	500	426	2	33
D6	2.4M	182K	1,365	134	6	55
D7	4.0M	421K	1,500	281	10	56
leon3mp	1.2M	109K	800	137	1	29
leon2	2.5M	149K	400	374	4	32

Table 5.3: Experimental results – stuck-at faults.

	TC [%]	Standard EDT		Hypercompression		TP reduction
		TP	DV [Mb]	TP	DV [Mb]	
D1	96.92	31,246	6.34	19,870	4.03	1.57x
D2	96.95	57,037	37.07	24,137	15.69	2.36x
D3	94.84	33,013	21.85	21,160	14.01	1.56x
D4	91.34	45,523	13.73	11,219	3.38	4.06x
D5	97.87	17,153	15.75	10,237	9.40	1.68x
D6	99.78	15,551	15.12	8,118	7.89	1.92x
D7	98.50	39,226	122.78	15,589	48.80	2.52x
leon3mp	99.83	13,352	2.54	7,907	1.50	1.69x
leon2	98.64	13,310	5.66	9,472	4.02	1.40x

the corresponding input data volume (DV) in megabits, and the resultant reduction of test pattern counts over the conventional EDT-based solution. Note that the hypercompression-based DV includes test templates data and binary-coded full toggle tap labels.

As the “TP reduction” column of Table 5.3 indicates, the hypercompression produces appreciable results. It compares favorably with earlier solutions as far as pattern and compression numbers are concerned. In all test cases, compression rates are significantly higher than those of the standard EDT, while the test coverage remains virtually unaffected. The observed pattern reduction relative to the EDT varies from 1.5x to 4.0x, and its average value computed

Table 5.4: Experimental results – transition faults.

	TC [%]	Standard EDT		Hypercompression		TP reduction
		TP	DV [Mb]	TP	DV [Mb]	
D1	88.81	33,331	6.77	22,272	4.52	1.50x
D2	80.22	15,742	10.23	13,439	8.73	1.17x
D3	82.55	18,880	12.50	17,408	11.52	1.08x
D4	69,63	11,010	3.31	5,632	1.69	1.95x
D5	96.17	21,664	19.89	12,224	11.22	1.77x
D6	90.41	12,130	11.79	6,270	6.09	1.93x
D7	77.72	19,561	61.22	16,640	52.08	1.17x
leon3mp	97.70	27,574	5.24	19,926	3.78	1.38x
leon2	99.64	36,331	15.44	27,947	11.88	1.30x

Table 5.5: Avarage fill rate for stuck-at patterns [%].

	Standard EDT	Hypercompression
D1	0.34	0.59
D2	0.23	0.45
D3	0.22	0.40
D4	1.64	1.11
D5	0.41	0.74
D6	0.67	0.66
D7	0.09	0.42
leon3mp	0.18	0.32
leon2	0.11	0.17

across the examined designs is virtually equal to 2x. Clearly, the proposed scheme raises compression beyond what the state-of-the-art sequential schemes can achieve. In particular, it successfully handles test cubes with incidentally high fill rate, whereas earlier solutions would either inflate pattern counts or declare compression aborts. Table 5.4 provides similar experimental results for transition faults. As can be seen, results shown in Table 5.4 are in line with those of Table 5.3. Here, the average reduction computed across all examined benchmark circuits is close to 1.5x.

As one may expect, the hypercompression is capable of handling test patterns with higher fill rates than those of test vectors that the EDT can encode. This is illustrated in Table 5.5 for

Table 5.6: Power metrics [%].

	Standard EDT			Hypercompression		
	Load	Capture	Unload	Load	Capture	Unload
D1	49.34	10.79	49.94	32.90	10.48	36.36
D2	49.38	11.70	44.37	28.71	11.54	29.68
D3	49.53	10.69	42.01	30.25	10.54	26.09
D4	49.52	14.69	42.04	30.00	14.40	33.48
D5	49.65	15.96	44.78	33.47	15.37	33.16
D6	49.24	36.55	46.75	28.32	31.58	43.40
D7	49.23	15.83	44.77	24.63	10.36	26.20
leon3mp	49.39	7.59	49.38	23.38	7.49	22.83
leon2	49.70	6.76	49.69	24.91	9.66	25.19

stuck-at tests. In many test cases, the acceptable fill rates can be doubled. A noticeable exception is design D4 where the actual fill rate served by the hypercompression drops to 1.11 compared to the corresponding value of 1.64 for the EDT-based compression. Interestingly, however, this is exactly the same design where the hypercompression achieves the highest reduction of the test pattern count (4.06x) and the test data volume for otherwise the same test coverage.

Finally, Table 5.6 offers the switching activity numbers. In all examined test cases the resultant scan-shift-induced switching activity was measured by the normalized weighted transition metric (WTM) [76]. As can be easily verified, these particular figures of merit were reduced to approximately 30% compared to the reference value of nearly 50% obtained as the average value over all examined designs for the standard EDT scheme. The toggling activity in the capture mode was recorded by means of a weighted switching activity (WSA) [86]. Since the average power dissipated during test is proportional to a shift-clock frequency, almost 2-fold reduction of a test power envelope creates a significant margin that allows for acceleration of scan shifting at a rate that maintains the same power consumption as that of conventional test solutions.

Chapter 6
Low power hypercompression

This chapter is the next evolutionary step in developing a new class of test data compression schemes. The presented low-silicon-area scheme builds on a test hypercompression paradigm but is capable of handling stringent power requirements in a more flexible fashion. Moreover, it deploys, on the average, fewer control registers than its hypercompression counterpart. However, additional processing needed to evaluate the most suitable decompressor configuration might increase the overall runtime by 50%. Consequently, the proposed scheme offers a trade-off between the ability to resolve problems related to test power dissipation and the resultant processing overhead.

6.1 Low power architecture

Test logic deployed by the new scheme follows the principles that serve as foundations for the original hypercompression, including the use of test templates. However, low power hypercompression test logic (LP-HTL) employs a different test template circuitry than that of Fig. 5.1. The new scheme is shown in Fig. 6.1. A very short circular register (seldom longer than 4 bits) is controlled by an initialization decoder that updates the register based on a template ID which directly precedes test pattern seeds in data streamed by ATE. Clearly, as there is no need to store templates any longer, time needed to channel a template ID is a negligible fraction of a regular test pattern upload period. It makes the new scheme very flexible since it is now capable of attaching a test template to every pattern. Back to Fig. 6.1, one can pick one of the four test templates: 1000, 0100, 0010, or 0001, just by attaching a 2-bit ID to the corresponding test pattern seeds.

To further reduce test data, it is assumed that every test pattern can put at most two hold latches in the transparent mode (the feasibility of this assumption is strongly supported by experimental evidences). To accommodate two transparent stages of the hold register, a pair of 1-out-of-n decoders driven by two associated full toggle latch ID registers are deployed, as

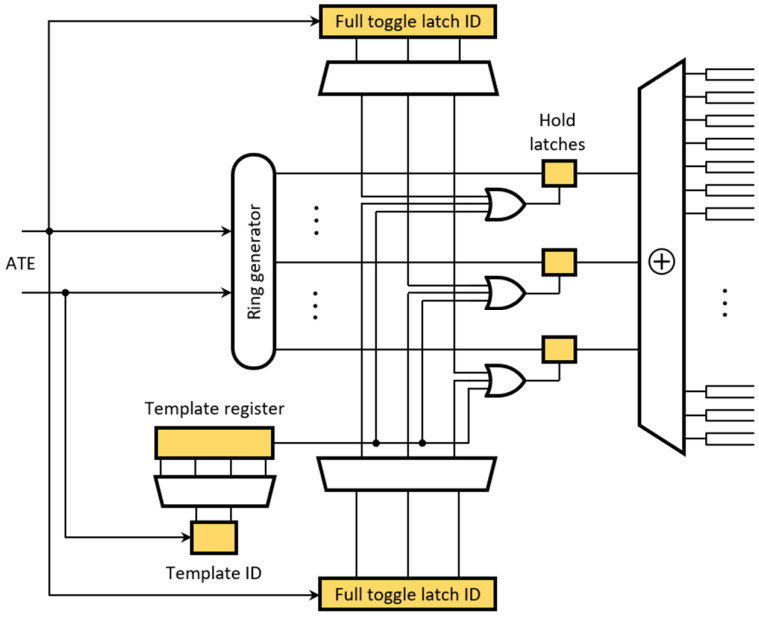

Fig. 6.1. Low power hypercompression test logic.

illustrated in Fig. 6.1, where n is the size of the ring generator. For each test pattern, this circuitry selects two out of n hold latches to be fed directly by the ring generator. Similarly to the previous scheme, such latches will also be referred to as full toggle taps.

6.2 Essential test cubes

The first step to determine short hypercompression test templates is the computation of the corresponding essential test cubes. Given a fault, the *essential test cube* is a partially specified vector that retains only those positions of an ATPG-produced test cube that correspond to primary inputs and scan cells whose likelihood of being specified is greater than a predetermined threshold. In particular, all necessary assignments associated with certain scan cells become unconditional parts of any essential test cube. Once a fault is injected, the process to determine an essential test cube keeps track of assignments made by APTG and computes the corresponding signal odds along fault's propagation and justification paths. A noteworthy observation is that a signal probability is computed regardless of the actual logic value (0 or 1) used to set a given net. This is why every specified net gets just a single value. Clearly, as long as there is a unique fault propagation path, the corresponding probabilities are set to 1. Probabilities associated with branches of a fan-out are equal fractions (inversely proportional to the number of branches) of the probability that the fault can reach the stem. However, the probability is only

assigned to a branch that was actually selected by ATPG and follows that path. In a backward implication process, inputs assume the values based on the corresponding outputs; if they are NAs, then the corresponding probabilities are equal to 1. Otherwise, the number of (remaining) inputs divides the output signal probability. Again, the resultant probability is just assigned to an input that has been set to a specific value by ATPG. In the off-path implication phase, while inputs assume the same non-controlling value as that of the fault propagation, we use the same rules as stated above to determine and assign the corresponding probabilities. Having determined test cubes and their probabilistic weights, the essential test cubes are obtained by keeping only specified bits with the corresponding signal probabilities greater than or equal to a given limit.

The above description highlights the main features of the method used to determine probabilities for all relevant nets in a circuit until it hits both primary inputs and scan cells (pseudo-primary inputs). The following examples illustrate the algorithm in operation on two simple circuits.

Example 6.1: Consider a stuck-at-0 fault shown in Fig. 6.2. To excite this fault, ATPG assigns the value of 1 (red nets) to the fault site *c*, and thus the probability p_c associated with this net becomes 1.0, i.e., $p_c = 1.0$. An off path input *f* of gate G_3 is assigned a non-controlling yet necessary value (0 – blue nets), and hence $p_f = 1.0$. Since the output *g* of gate G_3 is uniquely determined, the related probability is $p_g = 1.0$. Furthermore, as probabilities associated with branches of this fan-out are equal fractions of the stem probability, we have $p_h = 0.33$, $p_i = 0.33$, and $p_j = 0.33$. However, only p_h is recorded, as representing a path selected by ATPG. Now, gate G_2 should output 0. Thus the number of its inputs divides the output signal probability such that $p_d = 0.5$ and $p_e = 0.5$. Only p_d is stored, again because of the ATPG-based decision. Since gate G_1 yields 1 by setting both of its inputs to 1, we get $p_a = 1.0$ and $p_b = 1.0$.

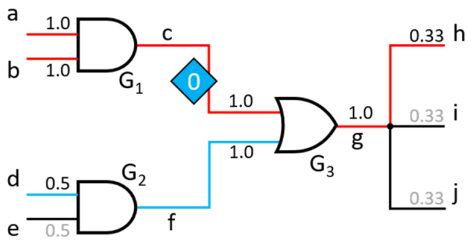

Fig. 6.2. Circuit for Example 6.1.

Example 6.2: A stuck-at-1 fault affecting the output of gate G_1 (Fig. 6.3) forces ATPG to assign the value of 0 to all lines printed in blue. Note that as far as gate G_1 is concerned, all these assignments are necessary, and thus the corresponding signal probabilities are all equal to 1.0. The same applies to stem G_3. Here, ATPG selects the upper branch as a fault propagation path. Since the probabilities associated with branches of this fan-out are inversely proportional to the number of branches (3), the input of gate G_4 gets 0.33 as its signal probability. Consequently, the same applies to the remaining pins of the same gate, and then a scan cell driving G_4. At the same time, as one of the inputs of G_2 is already driven by a set-to-0 necessary assignment, a desired value of 1 is assigned (arbitrarily) to the second input of the gate. As the number of inputs (here the remaining 2) divides the output signal probability, this particular input and its scan cell driver get 1.0 / 2 = 0.5 probability of being specified. As a result, four scan cells are specified with different associated probabilities. If the acceptance threshold is set to 0.5, the essential test cube is as follows: *xxx*00*xx*1*xxxx*. However, if the threshold is lowered to 0.25, the essential test cube becomes *x*1*x*00*xx*1*xxxx*.

6.3 Test template synthesis

The approach presented in this section works with a set of essential test cubes corresponding to certain faults. Initially, essential test cubes are processed individually, and then they are gradually merged based on their ability to form a circular test template and to deploy, if necessary, a small fraction of full toggle scan chains.

The method begins by selecting toggle points, i.e., locations where scan chains change their content values. The same concept is used in the previous technique, as presented in Section 5.2.1. However, in addition to the computations introduced earlier, essential test cubes are merged while preserving all necessary toggle points and toggle ranges.

Example 6.3: Consider an essential test cube and its toggle ranges (see Fig. 6.4, cube 1): [0, 3), [4, 6), [6, 11). Let $s = 4$ and the number of toggle points per template be confined to 1. After conversion modulo s, we get two toggle ranges: [0, 3), [0, 2); note that toggle range [6, 11) is rejected as too long. The resultant toggle point candidates are, therefore, {0, 1, 2} and {0, 1}. As their intersection is {0, 1}, the current single-1 test templates at this stage of processing are 1 0 0 0 and 0 1 0 0. The next essential test cube (cube 2) is first merged with cube 1. The resultant cube comprises three toggle ranges: [1, 3), [4, 5), and [9, 10). The conversion modulo

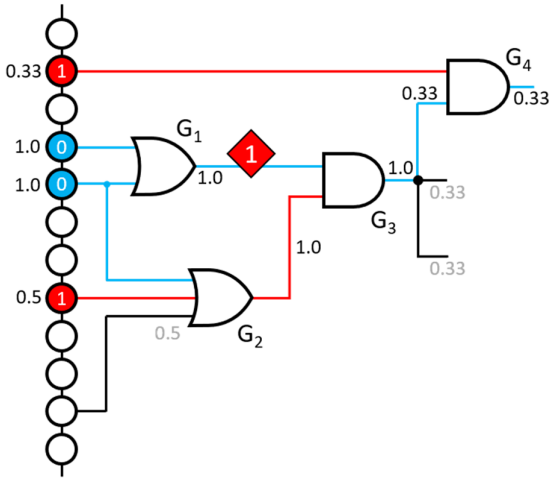

Fig. 6.3. Circuit for Example 6.2.

4 gives the following ranges: [1, 3), [0, 1), and [1, 2). They would need a template 1 1 0 0, which is rejected as not having a single 1. Consequently, cube 2 is not merged with cube 1. Merging the third essential test cube (cube 3) with cube 1 yields eventually three toggle ranges: [0, 3), [1, 2), and [1, 3). The intersection of the corresponding toggle points leads to a desired template 0 1 0 0.

Having test templates with a single one is two-fold beneficial. It reduces the total number of templates, and thus allows uploading their IDs rather than the entire templates. Moreover, it elongates non-toggling periods of scan-in shifting, thus reducing the total switching activity. As a result, the presented scheme becomes an efficient low power solution. If one uses slightly longer test templates, for example 8-bit long, their preferable form comprises combinations with two adjacent 1s to reduce the resultant template count. Furthermore, it addresses several cases when one needs to reload scan chains twice in a row with different values. The basic purpose of this operation is to shift-in a sequence having the same logic values but passing through scan cells connected via inverters.

In summary, given an essential test cube, its modular toggle ranges corresponding to a single scan chain are processed as follows. Let r_1 and r_2 be two toggle ranges. If r_1 and r_2 are disjoint, then there is no reduction of toggle point candidates, and both ranges are kept with no changes. If $r_1 \cap r_2 \neq \varnothing$, then the largest overlap between r_1 and r_2 is picked, with the remaining toggle point candidates being discarded. No specific toggle points are finalized yet, if an intersection comprises more than a single toggle point candidate. Generally, given a list $\{r_1, r_2, \ldots,$

$r_n\}$ of toggle ranges, they are processed in an arbitrary order by computing $r_k \cap r_{k+1} \cap \ldots$ as long as they are not empty. Every such product becomes a prime toggle group obtained for every scan chain.

Prime toggle groups are combined to obtain toggle points that can be shared by as many scan chains as possible. First, all lists are sorted by using the number of toggle ranges every list features as a sort key. Starting from a list with the smallest number of ranges, we keep combining lists until the number of potential toggle points becomes greater than a predefined threshold.

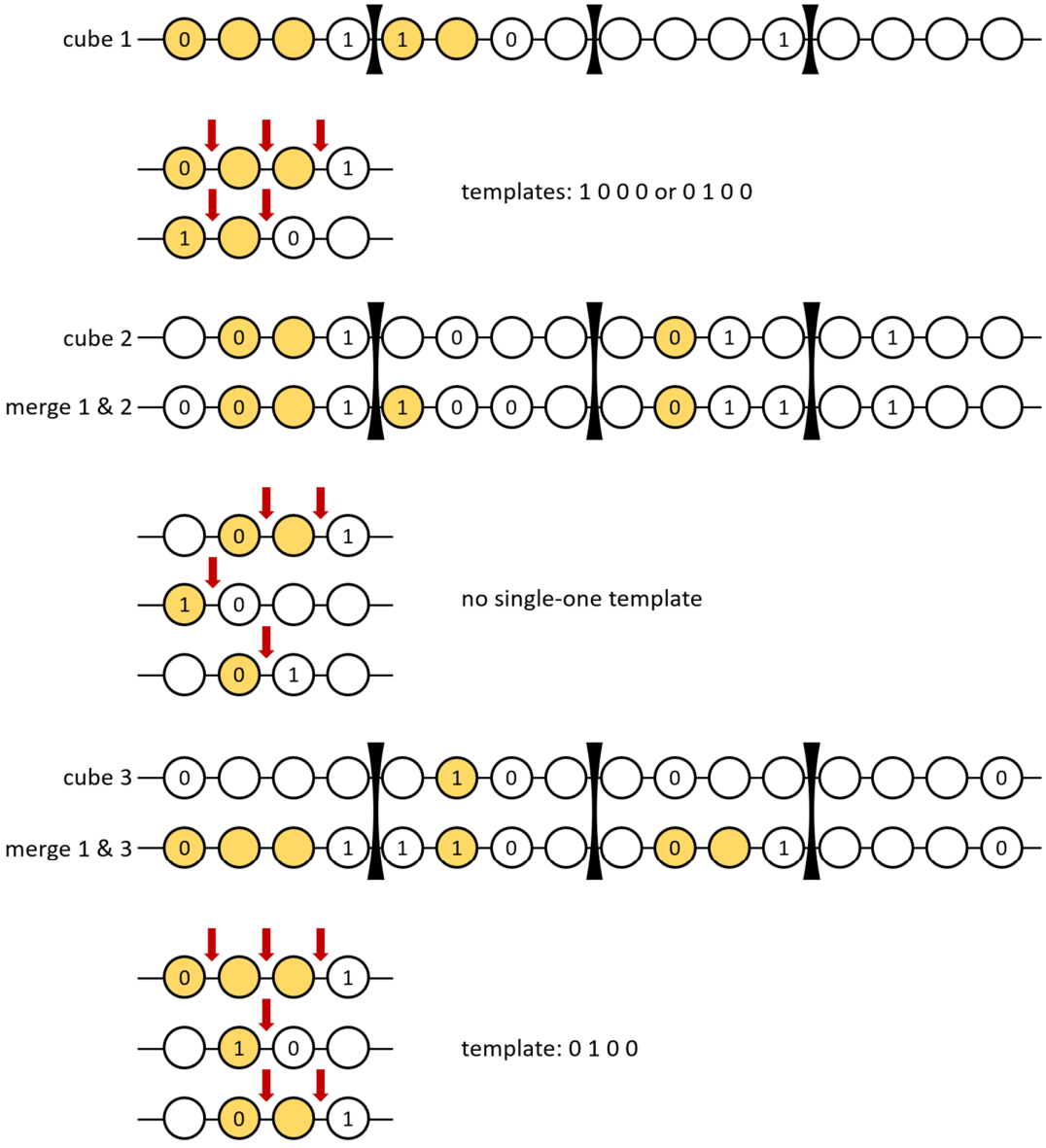

Fig. 6.4. Transformations of TPRs.

Note that the very same rules are used as those presented earlier. If a given list cannot be merged because of the resultant threshold, we try a next list until all lists are examined. Scan chains whose lists have not been merged become full toggle scan chains, where every care bit is set individually. The above procedure works repeatedly by selecting a new essential test cube that could be merged with a cube already processed (see Example 6.3) provided it has the ability on its own to form an acceptable test template. If there are no more essential test cubes that could be merged, a new test template is formed and process continues until all essential test cubes have been checked.

Recall that enabling certain stages of the hold register makes the corresponding scan chains fully toggling. The number of such full toggle taps should not be greater than two, as shown in Fig. 6.1. Identifying at most two taps that feed full toggle scan chains via 3-input XOR gates is somewhat similar to solving a maximum covering problem. To this end, a greedy approach presented in [43] is used. First, it sorts full toggle scan chains in descending order according to the number of test cubes these scan chains are involved. Next, we iterate the process that picks full toggle chains and determines their tap drivers. Assuming a scan chain is driven by a 3-input XOR gate, the highest-ranked chain will have three such taps to choose from. Next, subsequent chains will either share certain taps with the previously selected ones or narrow down the space of available taps. The procedure continues until there are no more than two taps available. At the end, certain test cubes may not be served by the selected taps. They go back to the test cube pool to be merged next time.

6.4 Test compression flow

Fig. 6.5 is the proposed test compression flow. Once a test template becomes available (as shown in the previous sections), ATPG starts with faults that have been used to arrive with essential test cubes and the corresponding templates. Information provided by test templates is passed to ATPG as constraints. However, in order to guide ATPG in a test-template-aware manner, an identification of so-called surrogates and other blocked gates is needed. Here, the same methodology is used as the one presented in Section 5.3.

In addition to short test templates, other mechanisms that allow the presented scheme to foster a close interaction between ATPG and encoding are used. These techniques are primarily aimed at tailoring test cubes towards a template-enforced format and at reducing pattern counts.

They guide ATPG back in such a way that ATPG does not assign certain nodes values that otherwise lead to conflicts, compression aborts, and backtracks. As can be seen in Fig. 6.5, the above objectives are achieved by taking the following additional steps:

- modifying conventional testability measures, i.e., controllability and observability, in such a way that every net in a circuit is assigned five integral metrics; two of them give the number of specified bits necessary to set a give line to 0 and 1, respectively; the third one provides the number of specified bits necessary to make a given net observable; the last two integers count how many faults are blocked if a given line is set to 0 and 1, respectively,
- limiting the encoding capacity of a compression solver during the regular merging of test cubes; typically the process of cube merging terminates once the number of seed variables with assigned logic values of 0 or 1 becomes higher than 70% of the total number of variables injected into a system; the resultant specified values assigned to scan cells are passed

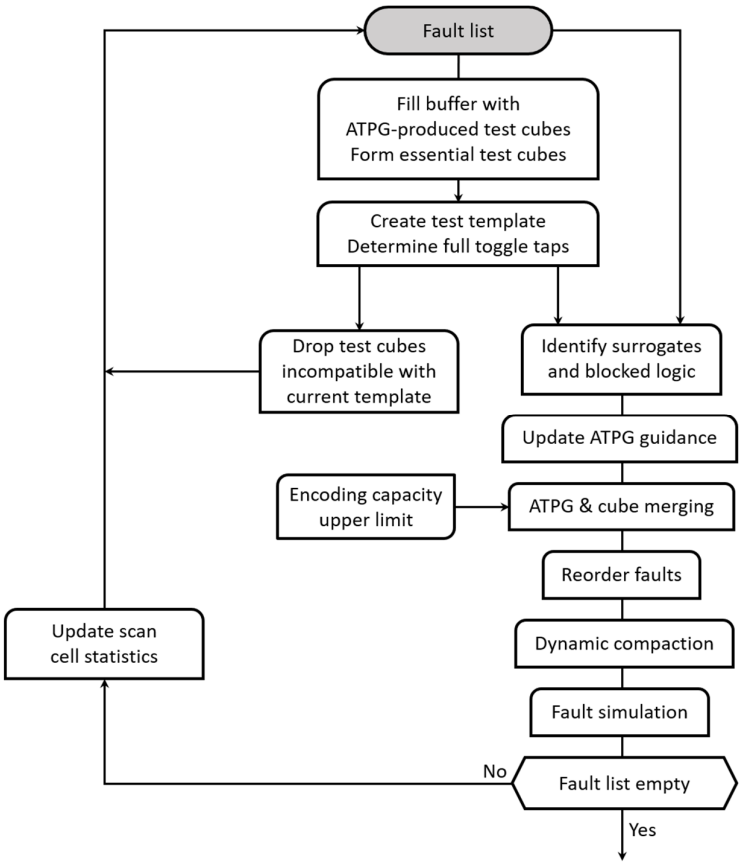

Fig. 6.5. High-level test flow.

to ATPG as its constraints – maintaining a proper fraction of unspecified scan cells leaves more space for dynamic-compaction-made assignments, which, in turn, increases efficiency of the latter technique,

- reordering faults based on simple heuristics – see the following paragraph,
- computing probabilities of having scan cells set to 0 or 1 for a given subset of already produced test cubes; this information is subsequently guiding ATPG so as to use – preferably – the same values for next test cubes (faults) to reduce the number of care bits; clearly, this approach generalizes immediately by taking advantage of probabilistic data obtained in the process of finding essential test cubes; the scan cell statistics are being iteratively updated when processing successive faults, as shown in Fig. 6.5,
- trimming the number of ATPG-produced test cubes by using values that ATPG considers necessary assignments as they become gradually available; details regarding these implied values can be found in [28].

Given test patterns produced by now, the fault reordering is periodically checking whether these patterns have established, for faults still on a list, the following: (1) an excitation path, (2) complete off-paths, (3) off-paths within a FFR hosting a given fault. The fault list is then sorted in such a way that faults with patterns setting their off-paths come first, followed

Table 6.1: Circuits characteristics.

	Gates	Scan cells	Scan chains	Longest chain	EDT channels	EDT size
D1	1.2M	72K	400	181	1	22
D2	3.2M	281K	960	293	2	32
D3	4.8M	287K	960	299	2	32
D4	1.3M	52K	200	260	1	32
D5	3.2M	213K	500	426	2	33
D6	2.4M	182K	1,365	134	6	55
D7	4.0M	421K	1,500	281	10	56

by faults with patterns setting their off-paths within the corresponding FFRs. The tail end consists of faults with patterns forming excitation paths. Within each group, preference is given to faults with the smallest pattern counts. These heuristics help to target, in the first place, faults that might be the most demanding in terms of finding their compressible test cubes.

Experiments run on several large industrial designs indicate that the above heuristics, paving the way for the guided ATPG, can yield better compression results. Nevertheless, it is

worth noting that the situation when a single essential test cube has no acceptable test template is usually extraneous: in such a case all scan chains enter the full toggle mode and a test cube is handled by the conventional EDT-based compression.

6.5 Experimental results

As done before, the low power (LP) hypercompression technology has been verified by conducting a series of experiments with several industrial designs. The basic data regarding these circuits such as the number of gates, the number of scan cells, the number of scan chains, the size of the longest chain, the number of input channels, and the size of decompressor are listed in Table 6.1. All experiments are performed with a 4-bit test template and at most two full toggle taps. It is also assumed that there is only a single toggle point per template. A test session consists of two repeatable steps: (1) shifting in IDs of a test template and two full toggle latches, and (2) applying 64 test patterns corresponding to this template.

Table 6.2: Experimental results – stuck-at faults

	TC [%]	Standard EDT		Hypercompression		LP Hypercompression		TPRE	TPRH
		TP	DV [Mb]	TP	DV [Mb]	TP	DV [Mb]		
D1	96.92	31,246	6.34	19,870	4.03	19,584	3.98	1.60	1.01
D2	96.95	57,037	37.07	24,137	15.69	23,528	15.29	2.42	1.03
D3	94.84	33,013	21.85	21,160	14.01	20,823	13.78	1.59	1.02
D4	91.34	45,523	13.73	11,219	3.38	10,643	3.20	4.28	1.05
D5	97.87	17,153	15.75	10,237	9.40	10,186	9.35	1.68	1.01
D6	99.78	15,551	15.12	8,118	7.89	7,973	7.75	1.95	1.02
D7	98.50	39,226	122.78	15,589	48.80	15,011	46.98	2.61	1.04

Table 6.2 summarizes the results of the experiments for stuck-at faults, obtained for the conventional EDT-based compression, the original hypercompression, and the new version of hypercompression presented in this chapter. The successive columns of the table list the test coverage (TC), the number of test patterns (TP), the corresponding input data volume (DV) in megabits, and the resultant reduction of test pattern counts over the conventional EDT (TPRE) and over the former version of hypercompression (TPRH). Note that the hypercompression-based DV includes test templates data and binary-coded full toggle tap labels, whereas the LP

version of hypercompression-based DV includes binary-coded IDs of test templates and full toggle taps.

As TPRE and TPRH columns of Table 6.2 indicate, the LP hypercompression compares favorably with its predecessors as far as test pattern counts and compression rates are concerned. In all test cases, the compression ratios are higher than those of the standard EDT and similar to the former hypercompression, while test coverage is not compromised. The observed pattern reduction relative to the EDT varies from 1.59x to 4.28x, and its average value computed across the examined designs is equal to 2.3x. Clearly, the proposed scheme raises compression beyond what state-of-the-art sequential compression schemes can achieve.

Table 6.3. Power metrics [%].

	Standard EDT			Hypercompression			LP Hypercompression		
	Load	Capture	Unload	Load	Capture	Unload	Load	Capture	Unload
D1	49.34	10.79	49.94	32.90	10.48	36.36	17.50	10.25	26.35
D2	49.38	11.70	44.37	28.71	11.54	29.68	15.97	11.32	20.21
D3	49.53	10.69	42.01	30.25	10.54	26.09	16.21	10.48	19.96
D4	49.52	14.69	42.04	30.00	14.40	33.48	17.01	14.32	24.54
D5	49.65	15.96	44.78	33.47	15.37	33.16	18.12	15.17	23.98
D6	49.24	36.55	46.75	28.32	31.58	43.40	15.15	31.08	38.76
D7	49.23	15.83	44.77	24.63	10.36	26.20	15.03	10.16	20.01

Table 6.3 reports the switching activity results. In all examined test cases the scan-shift-induced switching activity is measured by the normalized weighted transition metric (WTM) [76]. As can be seen, the hypercompression reduces WTM to approximately 30% compared to the reference value of nearly 50% obtained as the average value over all examined designs for the standard EDT. Furthermore, the new hypercompression lowers WTM down to nearly 17%. The toggling activity in the capture mode is represented by means of a weighted switching activity (WSA) [86]. Since the average test power is proportional to a scan-shift-clock frequency, its almost 3-fold reduction, as observed in the reported experiments, creates a significant margin that, at a cost of complex and time-consuming computations, allows for acceleration of scan shifting (and thus the entire test session) at a rate that corresponds to the difference between the power consumption of conventional test solutions and the new scheme.

Chapter 7
Conclusion

CMOS technology scaling has been a constant since its initial development with the purpose of obtaining integrated circuits that work at higher frequencies. The semiconductor industry is rapidly moving into new technology nodes and offers chips that comprise billions of transistors operating at GHz frequencies. The capabilities that allowed the industry to build this unprecedented trend require a sustained improvement in fabrication technologies, design automation tools, and last, but by no means least, test and verification methods. Clearly, technology scaling into the nanometer regime has a direct impact on contemporary test schemes. In particular, test application time and the amount of test data are progressively increasing because of higher complexity of designs, new types of defects, variance, and noise mechanisms. As a result, the high-quality tests become increasingly expensive to maintain. Furthermore, shaping a circuit power envelop, as required by reliable tests, is becoming more and more demanding. On top of everything, integrated circuits need constant and thorough monitoring in safety-critical applications. The novel concepts, schemes, and detailed solutions presented in the thesis address some of the challenges the contemporary test is facing. The author believes that these methods may play an important role in the quest for more pattern- and time-efficient VLSI testing schemes.

The test compaction scheme, presented in the first part of the thesis, is capable of reducing pattern counts of the state-of-the-art ATPG tools in a noticeable manner. The proposed solution combines a novel fault profiling, fault grouping, and a customized version of the SAT-based ATPG. It begins with a comprehensive analysis of necessary assignments associated with primary inputs, pseudo-primary inputs, and all internal primary fanout stems. Having determined this data, the thesis introduces an automated selection of fault groups, and then a method to produce a single test pattern for the entire group, if possible. As confirmed by experiments on industrial designs, the proposed approach can yield compact test sets, in many cases breaking barriers laid out by former test set compaction algorithms. In principle, the method can work

with any precomputed deterministic test set. As the new approach preserves all benefits of contemporary ATPG tools, it is an important factor in the process of gradual development of new automated test pattern generation techniques.

The hypercompression of test data, presented in Chapter 5, is a step towards a next generation of test data compression schemes outperforming conventional sequential compression techniques. Although it builds on the isometric compression paradigm, the hypercompression substantially limits its silicon overhead, remains non-intrusive to the core logic, and further elevates encoding efficiency and compression ratios. Moreover, it offers a flexible technique to control scan toggling rates by deploying a programmable selection of full-toggle scan chains, which in turn alleviates problems traditionally related to fault coverage drop and pattern count inflation. The use of a small test template makes the proposed scheme very flexible – it is capable of reconfiguring test templates at negligible cost related to test time and data volume overheads, for example, by working with very few external ATE channels. As the hypercompression preserves all benefits of the sequential test compression, its adoption may become an important factor in future scaled DFT technologies.

Finally, Chapter 6 describes a new low-power test data compression scheme. With the hypercompression as its modus operandi, the new technique limits silicon overhead even further, remains non-intrusive to the core logic, and elevates compression ratios. Moreover, high-quality tests are guaranteed as this approach can work with all traditional fault models and any new fault model of the future. More importantly, however, it is inherently a test-power-friendly solution that alleviates several problems typically related to test power constraints. The extremely small test templates it deploys can be attached to every pattern at negligible cost as far as test time and data volume are concerned. However, yielding very compact, high quality, and exceedingly low power tests comes at the expense of additional processing.

The thesis demonstrates that even in such mature areas as test pattern generation and test data compression, there are improvement opportunities. Moreover, the proposed solutions can also be regarded as starting points for future research directions. For example, the test set compaction technique of Chapters 3 and 4 works exclusively with the tail end of a test set. One of the next steps could be to develop rules that help a test set compaction algorithm form a subset of test patterns susceptible to more effective merging based on the initial and complete test set. Furthermore, the use of necessary assignments, as shown in Chapter 3, could be adapted by the hypercompression approach to better adjust templates, and thus to further improve results. In

summary, the solutions proposed in the thesis address some of the key challenges of the modern VLSI test and help to converge toward an ultimate on-chip test solution with a negligible impact on a design and manufacturing realm.

Bibliography

[1] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing & Testable Design*. New York: Wiley, 1994.

[2] C. Acero, D. Feltham, F. Hapke, E. Moghaddam, N. Mukherjee, V. Neerkundar, M. Patyra, J. Rajski, J. Tyszer, and J. Zawada, "On new test points for compact cell-aware tests," *IEEE Des. Test Comput.*, vol. 33, no. 6, pp. 7-14, Dec. 2016.

[3] S. B. Akers, "Partitioning for testability," *J. Des. Autom. Fault-Toler. Comput.*, vol. 1, pp. 121–126, Feb. 1977.

[4] P. H. Bardell, W. H. McAnney, and J. Savir, *Built-In Test for VLSI: Pseudorandom Techniques*. New York: Wiley, 1987.

[5] C. Barnhart, V. Brunkhorst, F. Distler, O. Farnsworth, A. Ferko, B. Keller, D. Scott, B. Koenemann, and T. Onodera, "Extending OPMISR beyond 10x scan test efficiency," *IEEE Des. Test Comput.*, vol. 19, no. 5, pp. 65–73, Sep. 2002.

[6] C. Barnhart, V. Brunkhorst, F. Distler, O. Farnsworth, B. Keller, and B. Koenemann, "OPMISR: the foundation for compressed ATPG vectors," in *Proc. ITC*, 2001, pp. 748–757.

[7] I. Bayraktaroglu and A. Orailoglu, "Test volume and application time reduction through scan chain concealment," in *Proc. DAC*, 2001, pp. 151–155.

[8] I. Bayraktaroglu and A. Orailoglu, "Concurrent application of compaction and compression for test time and data volume reduction in scan designs," *IEEE Trans. Comput.*, vol. 52, no. 11, pp. 1480–1489, Nov. 2003.

[9] P. D. Bisschop and E. Hendrickx, "Stochastic effects in EUV lithography," in *Proc. SPIE Advanced Lithography*, 2018, vol. 10583, p. 105831K.

[10] P. Camurati, D. Medina, P. Prinetto, and M. Sonza Reorda, "A diagnostic test pattern generation algorithm," in *Proc. ITC*, 1990, pp. 52–58.

[11] K. Chakrabarty, B. T. Murray, and V. Iyengar, “Built-in test pattern generation for high-performance circuits using twisted-ring counters,” in *Proc. VTS*, 1999, pp. 22–27.

[12] J.-S. Chang and C.-S. Lin, “Test set compaction for combinational circuits,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 14, no. 11, pp. 1370–1378, Nov. 1995.

[13] R. Chau and R. Arghavani, “A method for making a semiconductor device having a high-k gate dielectric,” U.S. Patent 6 617 210, Sep. 9, 2003.

[14] W.-T. Cheng, “The BACK algorithm for sequential test generation,” in *Proc. ICCD*, 1988, pp. 66–69.

[15] V. Chvatal, “A greedy heuristic for the set-covering problem,” *Math. Oper. Res.*, vol. 4, no. 3, pp. 233–235, Aug. 1979.

[16] F. Corno, P. Prinetto, M. Rebaudengo, and M. Sonza Reorda, “GATTO: a genetic algorithm for automatic test pattern generation for large synchronous sequential circuits,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 15, no. 8, pp. 991–1000, Aug. 1996.

[17] F. Corno, P. Prinetto, M. Rebaudengo, and M. Sonza Reorda, “A test pattern generation methodology for low power consumption,” in *Proc. VTS*, 1998, pp. 453–457.

[18] A. Czutro, I. Polian, P. Engelke, S. M. Reddy, and B. Becker, “Dynamic compaction in SAT-based ATPG,” in *Proc. ATS*, 2009, pp. 187–190.

[19] D. Czysz, M. Kassab, X. Lin, G. Mrugalski, J. Rajski, and J. Tyszer, “Low-Power Scan Operation in Test Compression Environment,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 28, no. 11, pp. 1742–1755, Nov. 2009.

[20] R. Drechsler, S. Eggergluss, G. Fey, A. Glowatz, F. Hapke, J. Schloeffel, and D. Tille, “On acceleration of SAT-Based ATPG for industrial designs,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 27, no. 7, pp. 1329–1333, July 2008.

[21] N. Eén and N. Sörensson, “An extensible SAT-solver,” in *Theory and Applications of Satisfiability Testing*, E. Giunchiglia and A. Tacchella, Eds. Berlin: Springer, 2004, pp. 502–518.

[22] N. Eén and N. Sörensson, “Translating pseudo-Boolean constraints into SAT,” *J. Satisf. Boolean Model. Comput.*, vol. 2, no. 1–4, pp. 1–26, March 2006.

[23] S. Eggersglüß, S. Milewski, J. Rajski, and J. Tyszer, “On reduction of deterministic test patterns sets,” in *Proc. ITC*, 2021.

[24] S. Eggersglüß, K. Schmitz, R. Krenz-Bååth, and R. Drechsler, “On optimization-based ATPG and its application for highly compacted test sets,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 35, no. 12, pp. 2104–2117, April 2016.

[25] E. B. Eichelberger and T. W. Williams, “A logic design structure for LSI testability,” in *Proc. DAC*, 1988, pp. 462–468.

[26] H. Fujiwara and T. Shimono, “On the acceleration of test generation algorithms,” *IEEE Trans. Comput.*, vol. 32, no. 12, pp. 1137-1144, Dec. 1983.

[27] S. Funatsu, N. Wakatsuki, and T. Arima, “Test generation systems in Japan,” in *Proc. DAC*, 1975, pp. 114–122.

[28] M. Gębala, G. Mrugalski, N. Mukherjee, J. Rajski, and J. Tyszer, “On using implied values in EDT-based test compression,” in *Proc. DAC*, 2014, pp. 1–6.

[29] P. Gelsinger, “Discontinuities driven by a billion connected machines,” *IEEE Des. Test Comput.*, vol. 17, no. 1, pp. 7–15, Jan. 2000.

[30] P. Girard, N. Nicolici, and X. Wen, Eds., *Power-Aware Testing and Test Strategies for Low Power Devices*. New York: Springer, 2010.

[31] P. Goel, “An implicit enumeration algorithm to generate tests for combinational logic circuits,” *IEEE Trans. Comput.*, vol. 30, no. 3, pp. 215–222, March 1981.

[32] P. Goel and B. C. Rosales, “Test generation and dynamic compaction of tests,” in *Proc. ITC*, 1979, pp. 189–192.

[33] L. H. Goldstein and E. L. Thigpen, “SCOAP: sandia controllability/observability analysis program,” in *Proc. DAC*, 1980, pp. 190–196.

[34] T. Gruning, U. Mahlstedt, and H. Koopmeiners, “DIATEST: a fast diagnostic test pattern generator for combinational circuits,” in *Proc. ICCAD*, 1991, pp. 194–197.

[35] A.-W. Hakmi, S. Holst, H.-J. Wunderlich, J. Schloffel, F. Hapke, and A. Glowatz, “Restrict encoding for mixed-mode BIST,” in *Proc. VTS*, 2009, pp. 179–184.

[36] I. Hamzaoglu and J. H. Patel, “Reducing test application time for full scan embedded cores,” in *Proc. FTCS*, 1999, pp. 260–267.

[37] I. Hamzaoglu and J. H. Patel, “Test set compaction algorithms for combinational circuits,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 19, no. 8, pp. 957–963, Aug. 2000.

[38] J. P. Hayes and A. D. Friedman, “Test point placement to simplify fault detection,” *IEEE Trans. Comput.*, vol. 23, no. 7, pp. 727–735, July 1974.

[39] S. Hellebrand, H.-G. Liang, and H.-J. Wunderlich, “A mixed mode BIST scheme based on reseeding of folding counters,” in *Proc. ITC*, 2000, pp. 778–784.

[40] S. Hellebrand, J. Rajski, S. Tarnick, S. Venkataraman, and B. Courtois, “Built-in test for circuits with scan based on reseeding of multiple-polynomial linear feedback shift registers,” *IEEE Trans. Comput.*, vol. 44, no. 2, pp. 223-233, Feb. 1995.

[41] S. Hellebrand, B. Reeb, S. Tarnick, and H.-J. Wunderlich, “Pattern generation for a deterministic BIST scheme,” in *Proc. ICCAD*, 1995, pp. 88–941.

[42] M. S. Hsiao, E. M. Rudnick, and J. H. Patel, “Fast static compaction algorithms for sequential circuit test vectors,” *IEEE Trans. Comput.*, vol. 48, no. 3, pp. 311–322, March 1999.

[43] Y. Huang, S. Milewski, J. Rajski, J. Tyszer, and C. Wang, “Low cost hypercompression of test data,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 39, no. 10, pp. 2964–2975, Oct. 2020.

[44] O. H. Ibarra and S. K. Sahni, “Polynomially complete fault detection problems,” *IEEE Trans. Comput.*, vol. C–24, no. 3, pp. 242–249, March 1975.

[45] S. Kajihara, I. Pomeranz, K. Kinoshita, and S. M. Reddy, “Cost-effective generation of minimal test sets for stuck-at faults in combinational logic circuits,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 14, no. 12, pp. 1496–1504, Dec. 1995.

[46] R. Kapur, S. Mitra, and T. W. Williams, “Historical perspective on scan compression,” *IEEE Des. Test Comput.*, vol. 25, no. 2, pp. 114–120, March 2008.

[47] T. P. Kelsey and K. K. Saluja, “Fast test generation for sequential circuits,” in *Proc. ICCAD*, 1989, pp. 345–347.

[48] T. Kirkland and M. R. Mercer, “A topological search algorithm for ATPG,” in *Proc. DAC*, 1987, pp. 502–508.

[49] T. Kobayashi, T. Matsue, and H. Shiba, “Flip-flop circuit with FLT capability,” in *Proc. IECEJ Conf.*, 1968, p. 692.

[50] B. Koenemann, “LFSR-coded test patterns for scan designs,” in *Proc. ETC*, 1991, pp. 237–242.

[51] A. Kumar, M. Kassab, E. Moghaddam, N. Mukherjee, J. Rajski, S. M. Reddy, J. Tyszer, and C. Wang, “Isometric test data compression,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 34, no. 11, pp. 1847–1859, Nov. 2015.

[52] T. Larrabee, “Test pattern generation using Boolean satisfiability,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 11, no. 1, pp. 4–15, Jan. 1992.

[53] J. Lee and N. Touba, “LFSR-reseeding scheme achieving low-power dissipation during test,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 26, no. 2, pp. 396-401, Feb. 2007.

[54] K.-J. Lee, J.-J. Chen, and C.-H. Huang, “Using a single input to support multiple scan chains,” in *Proc. ICCAD*, 1998, pp. 74–78.

[55] X. Lin, J. Rajski, I. Pomeranz, and S. M. Reddy, “On static test compaction and test pattern ordering for scan designs,” in *Proc. ITC*, 2001, pp. 1088–1097.

[56] W. Maly, “Modeling of lithography related yield losses for CAD of VLSI circuits,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 4, no. 3, pp. 166–177, July 1985.

[57] R. A. Marlett, “EBT: a comprehensive test generation technique for highly sequential circuits,” in *Proc. DAC*, 1978, pp. 335–339.

[58] K. Miyase and S. Kajihara, “XID: Don’t care identification of test patterns for combinational circuits,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 23, no. 2, pp. 321–326, Feb. 2004.

[59] G. E. Moore, “Cramming more components onto integrated circuits,” *Electronics*, vol. 38, no. 8, pp. 114-117, April 1965.

[60] N. Mukherjee, J. Rajski, G. Mrugalski, A. Pogiel, and J. Tyszer, “Ring generator: an ultimate linear feedback shift register,” *IEEE Computer*, vol. 44, no. 6, pp. 64-71, June 2011.

[61] P. Muth, “A nine-valued circuit model for test generation,” *IEEE Trans. Comput.*, vol. 25, no. 6, pp. 630–636, June 1976.

[62] S. N. Neophytou and M. K. Michael, “Test set generation with a large number of unspecified bits using static and dynamic techniques,” *IEEE Trans. Comput.*, vol. 59, no. 3, pp. 301–316, March 2010.

[63] S. N. Neophytou and M. K. Michael, “Test pattern generation of relaxed *n*-detect test sets,” *IEEE Trans. VLSI Syst.*, vol. 20, no. 3, pp. 410–423, March 2012.

[64] T. Niermann and J. H. Patel, “HITEC: a test generation package for sequential circuits,” in *Proc. ECDA*, 1991, pp. 214–218.

[65] R. Norman, J. Last, and I. Haas, “Solid-state micrologic elements,” in *Proc. ISSCC*, 1960, pp. 82–83.

[66] I. Pomeranz, L. N. Reddy, and S. M. Reddy, “COMPACTEST: a method to generate compact test sets for combinational circuits,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 12, no. 7, pp. 1040–1049, July 1993.

[67] I. Pomeranz and S. M. Reddy, “A diagnostic test generation procedure based on test elimination by vector omission for synchronous sequential circuits,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 19, no. 5, pp. 589–600, May 2000.

[68] I. Pomeranz and S. M. Reddy, “Forward-looking fault simulation for improved static compaction,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 20, no. 10, pp. 1424–1428, Oct. 2001.

[69] I. Pomeranz and S. M. Reddy, “Forward-looking reverse order fault simulation for n-detection test sets,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 28, no. 9, pp. 1424–1428, Sep. 2009.

[70] J. Rajski and H. Cox, “A method to calculate necessary assignments in algorithmic test pattern generation,” in *Proc. ITC*, 1990, pp. 25–34.

[71] J. Rajski, M. Kassab, N. Mukherjee, N. Tamarapalli, J. Tyszer, and J. Qian, “Embedded deterministic test for low-cost manufacturing,” *IEEE Des. Test Comput.*, vol. 20, no. 5, pp. 58–66, Sep. 2003.

[72] J. Rajski, J. Tyszer, M. Kassab, and N. Mukherjee, “Embedded deterministic test,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 23, no. 5, pp. 776–792, May 2004.

[73] P. M. Rosinger, B. M. Al-Hashimi, and N. Nicolici, “Low power mixed-mode BIST based on mask pattern generation using dual LFSR re-seeding,” in *Proc. ICCD*, 2002, pp. 474–479.

[74] J. P. Roth, “Diagnosis of automata failures: a calculus and a method,” *IBM J. Res. Dev.*, vol. 10, no. 4, pp. 278–291, July 1966.

[75] 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. Electron. Comput.*, vol. 16, no. 5, pp. 567–580, Oct. 1967.

[76] R. Sankaralingam, R. R. Oruganti, and N. A. Touba, “Static compaction techniques to control scan vector power dissipation,” in *Proc. VTS*, 2000, pp. 35–40.

[77] M. H. Schulz and E. Auth, “Improved deterministic test pattern generation with applications to redundancy identification,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 8, no. 7, pp. 811–816, July 1989.

[78] M. H. Schulz, E. Trischler, and T. M. Sarfert, “SOCRATES: a highly efficient automatic test pattern generation system,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 7, no. 1, pp. 126–137, Jan. 1988.

[79] N. Sitchinava, E. Gizdarski, S. Samaranayake, F. Neuveux, R. Kapur, and T. W. Williams, “Changing the scan enable during shift,” in *Proc. VTS*, 2004, pp. 73–78.

[80] C. E. Stroud, *A Designer’s Guide to Built-In Self-Test*. New York: Springer, 2002.

[81] N. A. Touba, “Survey of test vector compression techniques,” *IEEE Des. Test Comput.*, vol. 23, no. 4, pp. 294–303, April 2006.

[82] J. A. Waicukauski, P. A. Shupe, D. J. Giramma, and A. Matin, “ATPG for ultra-large structured designs,” in *Proc. ITC*, 1990, pp. 44–51.

[83] L.-T. Wang, C.-W. Wu, and X. Wen, *VLSI Test Principles and Architectures*. San Francisco: Elsevier, 2006.

[84] S. Wang and S. K. Gupta, “ATPG for heat dissipation minimization during scan testing,” in *Proc. DAC*, 1997, pp. 614–619.

[85] S. Wang and S. K. Gupta, “ATPG for heat dissipation minimization during test application,” *IEEE Trans. Comput.*, vol. 47, no. 2, pp. 256–262, Feb. 1998.

[86] S. Wang and S. K. Gupta, “An automatic test pattern generator for minimizing switching activity during scan testing activity,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 21, no. 8, pp. 954–968, Aug. 2002.

[87] Z. Wang and K. Chakrabarty, “Test data compression for IP embedded cores using selective encoding of scan slices,” in *Proc. ITC*, 2005, pp. 581–590.

[88] M. J. Y. Williams and J. B. Angell, “Enhancing testability of large-scale integrated circuits via test points and additional logic,” *IEEE Trans. Comput.*, vol. 22, no. 1, pp. 46–60, Jan. 1973.

[89] P. Wohl, J. Waicukauski, S. Patel, and M. Amin, “Efficient compression and application of deterministic patterns in a logic BIST architecture,” in *Proc. DAC*, 2003, pp. 566–569.

[90] M.-F. Wu, J.-L. Huang, X. Wen, and K. Miyase, “Reducing power supply noise in linear-decompressor-based test data compression environment for at-speed scan testing,” in *Proc. ITC*, 2008, pp. 1–10.

[91] Q. Wu, Y. Li, Y. Yang, and Y. Zhao, “A photolithography process design for 5 nm logic process flow,” *J. Microelectron. Manuf.*, vol. 2, no. 4, pp. 1–8, Dec. 2019.

[92] A. Yen, H. Meiling, and J. Benschop, “EUV lithography at threshold of high-volume manufacturing,” in *Proc. IEDM*, 2018, pp. 11.6.1-11.6.4.

[93] X. Yu, J. Wu, and E. M. Rudnick, “Diagnostic test generation for sequential circuits,” in *Proc. ITC*, 2000, pp. 225–234.

[94] “Apple unleashes M1,” *Apple Newsroom*. Accessed: May 6, 2021. [Online]. Available: https://www.apple.com/newsroom/2020/11/apple-unleashes-m1/.

[95] “Tessent FastScan,” *Siemens Digital Industries Software*, May 20, 2021. Accessed: May 20, 2021. [Online]. Available: https://eda.sw.siemens.com/en-US/ic/tessent/test/fastscan.

[96] “TestMAX DFT comprehensive, advanced design-for-test (DFT),” *Synopsys, Inc.* Accessed: May 20, 2021. [Online]. Available: https://www.synopsys.com/implementation-and-signoff/test-automation/testmax-dft.html.

[97] “Modus DFT Software Solution,” *Cadence Design Systems, Inc.*, May 20, 2021. Accessed: May 20, 2021. [Online]. Available: https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/test/modus-test.html.