UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

FAKULTÄT FÜR
ELEKTROTECHNIK,
INFORMATIK UND
MATHEMATIK

Testinstrumente und Testdatenanalyse zur Verarbeitung von Unsicherheiten in Logikblöcken hochintegrierter Schaltungen

Von der Fakultät für Elektrotechnik, Informatik und Mathematik
der Universität Paderborn

zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften (Dr.-Ing.)

genehmigte Dissertation
von

Alexander Sprenger, M.Sc.

Erster Gutachter:	Prof. Dr. Sybille Hellebrand
Zweiter Gutachter:	Prof. Dr. Marco Platzner
Tag der mündlichen Prüfung:	13.07.2023

Paderborn 2023

Diss. EIM-E/373

Danksagung

Zunächst danke ich meiner Familie für die vielseitige Unterstützung während dieser fordernden, anstrengenden aber auch sehr interessanten Zeit meines Lebens.

Ich danke meiner Doktormutter, Prof. Dr. Sybille Hellebrand, für die vielfältigen Diskussionen während meiner Arbeit als wissenschaftlicher Mitarbeiter im Fachgebiet Datentechnik des Instituts für Elektrotechnik und Informationstechnik an der Universität Paderborn. Des Weiteren danke ich Prof. Dr. Marco Platzner für die Übernahme der Aufgaben als Zweitgutachter meiner Dissertation.

Außerdem danke ich meinen wissenschaftlichen Kollegen des Fachgebiets Datentechnik, Dr.-Ing. Thomas Indlekofer, Dr.-Ing. Matthias Kampmann, Mohammad Urf Maaz, Jan Dennis Reimer und Dr. Somayeh Sadeghi-Kohan, für die vielen anregenden Diskussionen während unserer gemeinsamen Arbeit. Nicht zuletzt möchte ich mich auch bei dem Laboringenieur Rüdiger Ibers und der Sekretärin unseres Fachgebiets Anita Hüser für die stetige Unterstützung und für die stets offenen Ohren bedanken. Des Weiteren möchte ich mich bei den Kollegen im Institut für technische Informatik an der Universität Stuttgart, Dr. Laura Rodríguez Gómez, Dr. Michael Kochte, Dr. Chang Liu, Dr. Eric Schneider, Dr. Chih-Hao Wang, Dr. Natalia Lylina, Zahra Paria Najafi Haghi, Dr. Alexandra Kourfali, Hanieh Jafarzadeh und Florian Klemme unter der Leitung von Prof. Dr. Hans-Joachim Wunderlich und Jun.-Prof. Dr.-Ing. Hussam Amrouch für die zahlreichen gemeinsamen Diskussion bedanken. Außerdem danke ich Prof. Dr. Stefan Holst vom Kyushu Institut of Technology, Japan für die gemeinsame Arbeit und anregenden Diskussionen.

Warstein - Mülheim im März 2023 *Alexander Sprenger*

Abstract

In recent decades, increasingly high performance requirements have been placed on microchips, leading to increasingly complex semiconductor technologies with ever shrinking structure sizes, till sizes of 3 nm [TSMC2023]. Complex applications with high safety and reliability requirements, such as functional safety during autonomous driving standardized in ISO 26262 [ISO2018], medical technology, or the application of artificial intelligence in large data centers, are simultaneously driving the requirements for testing and diagnosis of very large scale integrated circuits (VLSI). Throughout the entire life cycle of a microchip, starting with manufacturing, uncertainties occur that affect the behavior of the microchip. For example, even smallest particles can disrupt the fabrication process, causing the fabricated structure to deviate from the intended structure. This can lead to weak circuit structures, potentially causing a change in the timing of the fabricated circuit. While this change does not necessarily have to lead to a change in the logical behavior, it may cause Early Life Failures (ELFs), leading to a reliability problem. Similarly, aging effects of the interconnects, such as electromigration, can lead to uncertainties in the timing behavior of the circuit and thus in reliability problems, resulting in a premature failure of a circuit or increased aging.

This work contributes to the treatment of uncertainties throughout the entire life cycle of a VLSI circuit, which can be used in the context of *Silicon Lifecycle Management*. With modular and hybrid compaction two test instruments are presented, which can be used for X-tolerant test response compaction e.g. in the built-in Faster-than-At-Speed Test (FAST). The built-in Faster-than-At-Speed Test is used to detect uncertainties in the fabricated circuit that manifest as small delay faults. A challenge for the test response compaction during FAST is the high and varying X-rate at the outputs of the circuit under test. By dividing the circuit outputs into test groups and processing the test groups separately using stochastic compactors, the presented modular compaction can handle the high and varying X-rates. In the shown experiments the modular compaction achieved an X-reduction ratio (XRR) of up to 11,75, with a fault permeability (FP) of 94,96 %. By utilizing the flexible structures of a stochastic compactor in a deterministic phase after the stochastic phase in hybrid compaction, the fault permeability

(FP) could be further increased, by specifically considering essential faults in the deterministic phase. Thus, it was possible to increase the fault permeability by up to 12,31 % by reapplying 10 % of the test pattern set to the circuit.

Another contribution of this thesis for dealing with uncertainties throughout the life cycle of a highly integrated circuit is a method for distinguishing crosstalk on logic interconnects and process variation. Due to the small structure sizes in current semiconductor technologies, parasitic coupling capacitances between the interconnects of a VLSI circuit occur more frequently. These parasitic coupling capacitances lead to crosstalk. Due to the increased current flow during the crosstalk, the current density in the interconnects increases, which in turn can lead to increased electromigration. To be able to prevent the circuit from this premature aging, a method is presented that uses delay maps describing the timing behavior of an output of a circuit at different operating points to classify the tested circuits into *fault-free* and *faulty* using an artificial neural network. In the case study shown, an accuracy of up to 98,8 % could be achieved with a precision of 99,7 % and a recall of 97,9 %, where precision can be interpreted as a measure of yield and recall as a measure of product quality.

With this work, a significant contribution has been made to addressing uncertainties throughout the entire life cycle of VLSI circuits.

Kurzfassung

In den letzten Jahrzehnten wurden immer größere Anforderungen an die Leistungsfähigkeit von Mikrochips gestellt, was zu immer komplexeren Halbleitertechnologien mit immer kleiner werdenden Strukturgrößen von bis zu 3 nm [TSMC2023] geführt hat. Komplexe Anwendungen mit hohen Ansprüchen an Sicherheit und Zuverlässigkeit, wie z. B. die in der ISO 26262 [ISO2018] standardisierte funktionale Sicherheit während des autonomen Fahrens, die Medizintechnik oder die Anwendung künstlicher Intelligenz in großen Rechenzentren, treiben gleichzeitig die Anforderungen an den Test und die Diagnose hochintegrierter Schaltungen an. Während des gesamten Lebenszyklus eines Mikrochips, beginnend mit der Fertigung, kommt es zu Unsicherheiten, die das Verhalten des Mikrochips beeinflussen. So können z. B. bereits kleinste Partikel den Fertigungsprozess stören, wodurch die gefertigte Struktur von der geplanten Struktur abweichen kann, was zu schwachen Schaltungsstrukturen führen kann. Diese schwachen Schaltungsstrukturen können zu einer Veränderung des Zeitverhaltens der gefertigten Schaltung führen. Während diese Veränderung nicht notwendigerweise zu einer Veränderung des logischen Verhaltens führen muss, kann sie jedoch dazu führen, dass die Schaltung frühzeitig ausfällt und somit zu einem Zuverlässigkeitsproblem führt. Ähnlich können Alterungseffekte der Verbindungsleitungen, wie die Elektromigration, zu Unsicherheiten im Zeitverhalten der Schaltung und somit zu Zuverlässigkeitsproblemen führen. So kann es zu dem verfrühten Ausfall einer Schaltung (engl. Early Life Failure, ELF) oder einer beschleunigten Alterung kommen.

Mit den vorgestellten Verfahren dieser Arbeit, die im Rahmen des *Silicon Lifecycle Managements* eingesetzt werden können, wird ein Beitrag zur Behandlung von Unsicherheiten während des gesamten Lebenszyklus geleistet. Mit der modularen und der hybriden Kompaktierung werden zwei Testinstrumente vorgestellt, die unter anderem für die X-tolerante Testantwortkompaktierung im eingebauten Hochgeschwindigkeitstest verwendet werden können. Der eingebaute Hochgeschwindigkeitstest wird eingesetzt, um Unsicherheiten in der gefertigten Schaltung zu detektieren, die sich als kleine Verzögerungsfehler manifestieren. Eine Herausforderung für die Testantwortkompaktierung während des Hochgeschwindigkeitstests ist die hohe und variierende X-Rate an den Ausgängen der zu

testenden Schaltung. Durch die Einteilung der Schaltungsausgänge in Prüfgruppen und die separierte Kompaktierung der Prüfgruppen mithilfe von stochastischen Kompaktierern, konnte mit der modularen Kompaktierung ein Kompaktierungsverfahren vorgestellt werden, das diese hohen und variierenden X-Raten verarbeiten kann. In den durchgeführten Experimenten erreicht die modulare Kompaktierung einen X-Reduktionsfaktor (XRF) von bis zu 11,75, wobei die Fehlerdurchlässigkeit (FD) auf einem hohen Niveau von 94,96 % gehalten werden konnte. Durch die Ausnutzung der flexiblen Strukturen eines stochastischen Kompaktierers in einer deterministischen Phase nach der stochastischen Phase in der hybriden Kompaktierung konnte die Fehlerdurchlässigkeit (FD) weiter gesteigert werden, indem gezielt essenzielle Fehler in der deterministischen Phase berücksichtigt wurden. So ist es möglich, die Fehlerdurchlässigkeit um bis zu 12,31 % zu erhöhen, indem 10 % der Testmuster erneut an die Schaltung angelegt werden.

Ein weiterer Beitrag dieser Arbeit für den Umgang mit Unsicherheiten im Laufe des Lebenszyklus einer hochintegrierten Schaltung ist ein Verfahren zur Unterscheidung von Übersprechen auf Verbindungsleitungen der Logik-Schaltung und Prozessvariation. Aufgrund der geringen Strukturgrößen in aktuellen Halbleitertechnologien, kommt es vermehrt zu parasitären Koppelkapazitäten zwischen den Verbindungsleitungen einer hochintegrierten Schaltung. Diese parasitären Koppelkapazitäten führen zu Übersprechen. Durch den hierbei entstehenden erhöhten Stromfluss steigt die Stromdichte in den Verbindungsleitungen, was wiederum zu erhöhter Elektromigration führen kann. Um Maßnahmen gegen die verfrühte Alterung vornehmen zu können, wird ein Verfahren zur Unterscheidung von Übersprechen und Prozessvariation vorgestellt. Hierzu werden Verzögerungskarten genutzt, die das Zeitverhalten eines Ausgangs einer Schaltung in unterschiedlichen Arbeitspunkten beschreiben, um mithilfe eines künstlichen neuronalen Netzes die getesteten Schaltungen in *fehlerfrei* und *fehlerhaft* zu klassifizieren. In der gezeigten Fallstudie konnte eine Genauigkeit von bis zu 98,8 % erreicht werden, bei einer Relevanz von 99,7 % und einer Sensitivität von 97,9 %, wobei die Relevanz als Maß für die Ausbeute und die Sensitivität als Maß für Produktqualität interpretiert werden kann.

Durch diese Arbeit konnte ein großer Beitrag geleistet werden, um den Unsicherheiten im gesamten Lebenszyklus von hochintegrierten Schaltungen zu begegnen.

Inhaltsverzeichnis

1 Einleitung **1**

2 Einführung in den Test von hochintegrierten Schaltungen **7**

2.1 Defekte 8

2.2 Alterungseffekte 8

2.2.1 Alterungseffekte in Transistoren 9

2.2.2 Alterungseffekte in Leitungen 9

2.3 Fehlermodelle 11

2.3.1 Haftfehler 12

2.3.2 Verzögerungsfehler 13

2.3.2.1 Pfad-Verzögerungsfehlermodell 13

2.3.2.2 Transitions-Verzögerungsfehlermodell 14

2.3.3 Kleine Verzögerungsfehler 15

2.3.4 Verbindungsfehler 16

2.4 Test 18

2.4.1 Eingebauter Selbsttest 20

2.4.2 Testantwortkompaktierung 23

2.4.2.1 Räumliche Kompaktierung 24

2.4.2.2 Zeitliche Kompaktierung 28

2.4.3 Hochgeschwindigkeitstest 31

3 Einführung in künstliche neuronale Netze **35**

3.1 Merkmalsextraktion 41

3.1.1 Hauptkomponentenanalyse 41

3.2 Ensemble-Methoden 42

4 Testantwortkompaktierung für den Hochgeschwindigkeitstest **45**

4.1 Stand der Technik 46

4.1.1 X-tolerante lineare Kompaktierung (X-Compact) 49

4.1.2 Stochastische Kompaktierung 52

4.2 Modulare Kompaktierung . . . 53
4.2.1 Gruppierungsalgorithmen . . . 56
4.2.1.1 Lösungsansatz mithilfe von linearer Programmierung . . 57
4.2.1.2 Heuristische Lösungen . . . 58
4.2.2 Ausnutzung von X-Pfaden und essenziellen Prüfpfaden . . . 66
4.2.3 Experimente . . . 67
4.2.3.1 Kompaktierungsrate vs. Prüfgruppengröße . . . 68
4.2.3.2 Auswertung der Gruppierungsalgorithmen . . . 71
4.2.3.3 Einfluss der Prüfpfadkonfiguration . . . 74
4.2.3.4 X-Pfad-Blockierung und Pfad-Bypass für essenzielle Prüfpfade . . . 75
4.2.4 Fazit . . . 77
4.3 Hybride Kompaktierung . . . 79
4.3.1 Bestimmung der deterministischen Steuerungsvektoren für die hybride Kompaktierung . . . 81
4.3.2 Experimente . . . 86
4.3.3 Fazit . . . 92

5 Verbindungstest in Logik-Schaltungen 95
5.1 Stand der Technik . . . 97
5.2 Verhalten von Übersprechen und Prozessvariation unter Berücksichtigung des Arbeitspunktes . . . 98
5.3 Konfiguration des künstlichen neuronalen Netzes . . . 104
5.4 Merkmalsextraktion . . . 106
5.5 Experimente . . . 107
5.5.1 Reduktion der Arbeitspunkte . . . 111
5.6 Fazit . . . 114

6 Fazit 117

7 Ausblick 121

Symbolverzeichnis 125

Abbildungsverzeichnis 131

Tabellenverzeichnis 135

Algorithmenverzeichnis 137

Abkürzungsverzeichnis 139

Literaturverzeichnis **141**

Eigene Konferenz- und Zeitschriftenbeiträge **155**

Eigene Workshopbeiträge **157**

Betreute wissenschaftliche Arbeiten **159**

1 Einleitung

Die hohen Anforderungen an die Leistungsfähigkeit moderner System-on-Chips (SoCs) sind treibende Kräfte für die Entwicklung von Prozesstechnologien mit immer kleiner werdenden Strukturgrößen. So sind Halbleiterfertiger z. B. in der Lage Mikrochips mit Strukturgrößen von 3 nm zu produzieren [TSMC2023]. Aktuelle High-End SoCs, wie der am 09. März 2022 vorgestellte *Apple M1 Ultra* Chip, werden bereits serienmäßig in einer 5 nm Technologie gefertigt. Aber auch andere Hersteller von High-End SoCs verwendeten zu diesem Zeitpunkt ähnlich ambitionierte Technologiegrößen wie die 7 nm Alder Lake-Architektur der *Intel Core i* Prozessoren [Intel2022] oder die 7 nm Technologie der *AMD Ryzen Threadripper* Prozessoren [AMD2023].

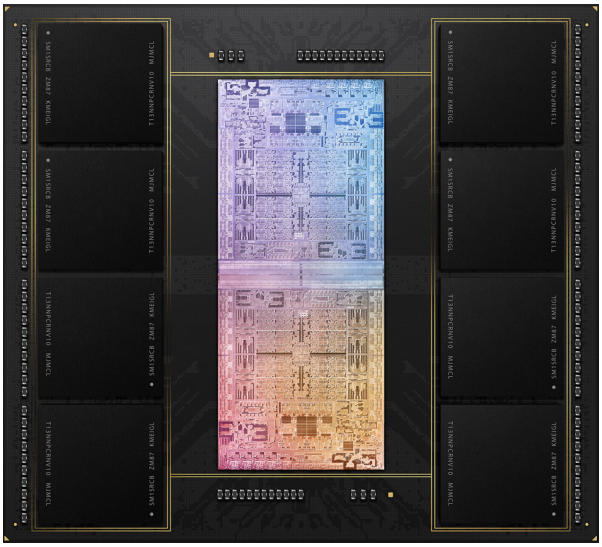


Abbildung 1.1: Aufnahme eines *Apple M1 Ultra* [Apple2022].

Durch die immer kleiner werdenden Strukturgrößen ist es möglich, immer komplexere Systeme auf einem Chip zu integrieren. Der in Abbildung 1.1 gezeigte *Apple M1 Ultra* Chip integriert 114 Milliarden Transistoren, wodurch es möglich ist, 20 CPU-Kerne, 64 GPU-Kerne und 32 Neuronale-Kerne auf einem Chip zu integrieren. Dies ist auch durch die sogenannte *UltraFusion* Architektur möglich, die mithilfe von direkter Durchkontaktierung von 10 000 Signalen auf dem Silizium eine Übertragungsrate von 2,5 TB/s zwischen zwei identischen *Apple M1 Max* SoCs ermöglicht. Diese hohe Anzahl an Ver-

bindungsleitungen ist in der Abbildung 1.1 als regelmäßige Struktur in der Mitte des *Apple M1 Ultra* Chips zu sehen. [Apple2022]

Durch die schrumpfenden Strukturgrößen und die steigende Komplexität der SoCs kommt es jedoch zu Herausforderungen im Herstellungsprozess der SoCs. Es kann daher nicht ausgeschlossen werden, dass es während der Fertigung zu Defekten kommt. So kann es unter anderem dazu kommen, dass durch Schmutzpartikel Strukturen nicht korrekt aufgebracht werden, wodurch es zu Kurzschlüssen, offenen Verbindungen oder schwachen Strukturen kommen kann, die zu Unsicherheiten im Zeitverhalten der gefertigten Schaltung führen. Durch die kleinen Strukturgrößen kommt es außerdem während der Lebensdauer des SoCs vermehrt zu Alterungseffekten, die sich als Verzögerungen der Signalverläufe manifestieren. Gleichzeitig wachsen die Anforderungen an die Zuverlässigkeit der SoCs, wie z. B. für die in der ISO 26262 [ISO2018] standardisierte funktionale Sicherheit von SoCs in der Automobilindustrie. Daher ist der Test und die lebenslange Überwachung der SoCs obligatorisch.

Während des gesamten Lebenszyklus eines SoCs, von der Fertigung bis zur Ausmusterung, werden schon heute eine Vielzahl an Daten über das SoC gesammelt. Bereits während des Fertigungsprozesses können Daten über die Prozessbedingungen gesammelt werden. Der Fertigungstest liefert anschließend wichtige Informationen über den Zustand des gefertigten Chips. Gleichzeitig kann eingebettete Testhardware während des gesamten Lebenszyklus des SoCs wiederverwendet werden, um das Verhalten zu überwachen. Durch die Vereinigung dieser bereits vorhandenen Datensätze und die Wiederverwendung von bereits vorhandenen Testinstrumenten und Testmethoden können verschiedene Ziele, wie in Abbildung 1.2 zusammengefasst, verfolgt werden. Diese Wiederverwendung von Monitoren, Sensoren und anderen Testinstrumenten, die Vereinigung von Datensätzen aus den unterschiedlichen Phasen des Lebenszyklus eines SoCs und die Analyse dieser vereinigten Datensätze ist im Allgemeinen als *Silicon Lifecycle Management* bekannt. Mithilfe des *Silicon Lifecycle Managements* werden dabei verschiedene Ziele verfolgt, wie die vorhersagbare Wartung von Mikrochips, die Steigerung der Leistungsfähigkeit des Systems, die Verbesserung der Testergebnisse und der Qualität des Mikrochips, die Steigerung der Ausbeute und Identifizierung von Ausbeutebegrenzern, sowie die Verbesserung und Beschleunigung der Design-Kalibrierung. [Crosher2022]

Eine große Herausforderung für den Test von hochintegrierten Schaltungen sind Unsicherheiten, die es erschweren zwischen einem fehlerhaften und einem fehlerfreien Chip zu unterscheiden. Zum einen unterliegt die Fertigung von Mikrochips Prozessvariationen, wie z. B. die Variation der gefertigten Gatelänge eines Transistors, was durch kleine Verzögerungen am Ausgang der Schaltung sichtbar wird. Falls der gefertigte Chip nur

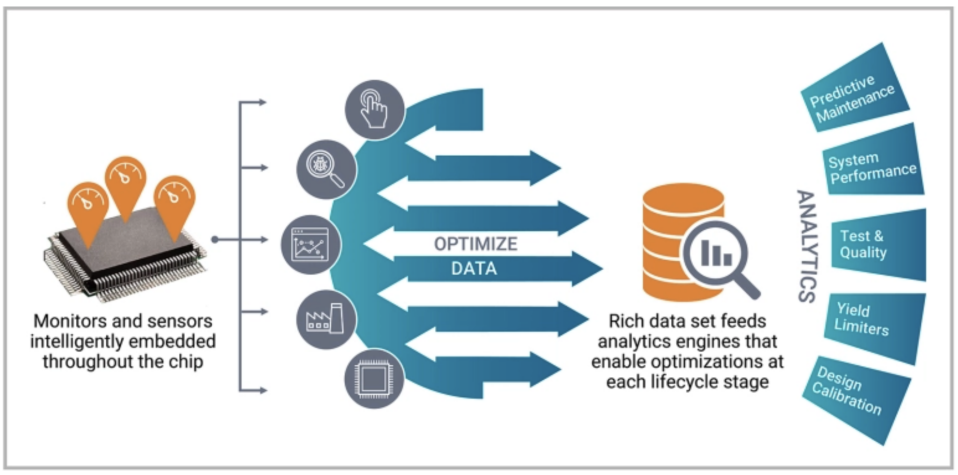


Abbildung 1.2: Überblick über das *Silicon Lifecycle Management* [Crosher2022].

unter dieser Prozessvariation leidet, ist er jedoch in der Regel als fehlerfrei zu betrachten und kann z. B. für Anwendungen mit geringeren Ansprüchen an die Taktperiode des Chips genutzt werden. Zum anderen führen aber auch andere Effekte zu kleinen Verzögerungen am Ausgang der zu testenden Schaltung. Dies können unter anderem resistive Defekte sein, die z. B. durch Fehler während der Lithografie entstehen können oder andere schwache Schaltungsstrukturen, die zu Frühausfällen (ELFs) führen können [Kim2010; Malandruccolo2011]. Aber auch im Laufe des Lebenszyklus eines Mikrochips führen Alterungseffekte in den Transistoren oder in den Verbindungsleitungen zu kleinen Verzögerungen. Alterungseffekte, wie die Bias-Temperaturinstabilität (engl. Bias Temperature Instability, BTI) und die Injektion heißer Ladungsträger (engl. Hot Carrier Injection, HCI) [McPherson2019] führen, abhängig von der Nutzungsdauer, zu einer Erhöhung der Schwellenspannung der Transistoren und damit zu einer Verlangsamung der Schaltgeschwindigkeit des Transistors. Auch das Übersprechen zweier Leitungen wird mit schrumpfenden Strukturgrößen zu einer immer größeren Herausforderung [Geden2011]. Zwei parallel verlaufende Leitungen auf einem Mikrochip sind unter anderem, wie in dem in Kapitel 2.3.4 eingeführten Modell zu sehen, parasitär kapazitiv gekoppelt, was zu Übersprechen führt. Einerseits kann dieses Übersprechen zu kleinen Verzögerungen des Signalverlaufs führen und andererseits zu einem erhöhten Stromfluss in den Verbindungsleitungen, was zu einer beschleunigten Alterung aufgrund von Elektromigration führen kann [Livshits2012; Sadeghi-Kohan2021]. Diese wiederum kann zur Erhöhung der parasitären kapazitiven Kopplung führen, was seinerseits wieder zu einer erhöhten Elektromigration führt. Für den ganzheitlichen Test einer Schaltung ist es daher wichtig, die Verbindungsleitungen der Schaltung zu berücksichtigen. Insbesondere ist es notwendig zwischen den Ursachen für kleine Verzögerungen unterscheiden zu können.

Eine weitere Herausforderung für den Test von Mikrochips sind unbekannte Schaltungszustände, die z. B. durch nicht initialisierte Speicherelemente oder Tri-State Buffer auftreten. Hierbei handelt es sich um Signalpegel, die während der logischen Simulation der Schaltung nicht eindeutig bestimmt werden können. Diese unbekannten Signalpegel werden auch X-Werte genannt. Treten X-Werte in einer Schaltung auf, können die Sollantworten für Testbelegungen nicht eindeutig vorhergesagt werden. Dies ist insbesondere ein Problem beim eingebauten Selbsttest, der für die Überwachung während der Lebenszeit einer hochintegrierten Schaltung immer wichtiger wird. Daher sind X-tolerante Verfahren zur Auswertung der Ausgabe während des Tests der Schaltung notwendig. Zur Veranschaulichung ist in Abbildung 1.3 eine schematische Darstellung eines eingebauten Selbsttests zu sehen. Mithilfe eines Testmustergenerators werden Testmuster an die zu testende Schaltung (engl. Circuit Under Test, CUT) angelegt. Die Antwort der zu testenden Schaltung auf die Testmuster, die sogenannte Testantwort, wird anschließend mithilfe von räumlichen und zeitlichen Kompaktierungsverfahren zu Signaturen komprimiert. Diese Signaturen können daraufhin mit im Vorfeld simulierten Referenzsignaturen verglichen werden. Um eine eindeutige Identifikation von fehlerhaften Signaturen zu gewährleisten müssen die Signaturen X-frei sein.

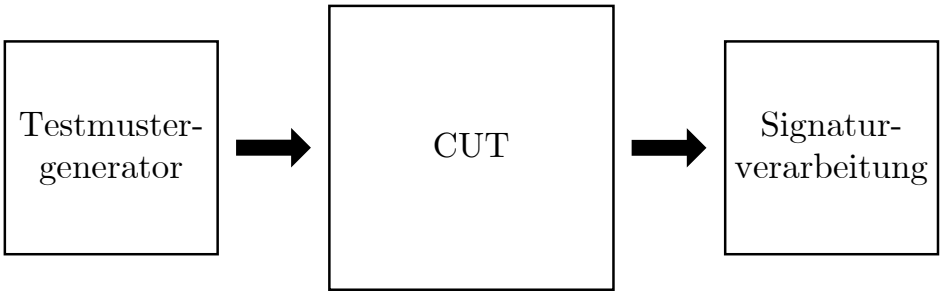


Abbildung 1.3: Schema eines eingebauten Selbsttests.

Während ein Logikfehler am Ausgang der zu testenden Schaltung durch einen Vergleich der tatsächlichen mit der simulierten Ausgangsbelegung für eine gegebene Testbelegung detektierbar ist, kann eine kleine Verzögerung einer Signaländerung am Ausgang der zu testenden Schaltung nur erkannt werden, wenn sie dazu führt, dass die Signaländerung erst nach dem Beobachtungszeitpunkt stattfindet. Um eine Schaltung auf kleine Verzögerungen zu testen, die nicht am Ausgang sichtbar werden, wird ein spezielles Testverfahren benötigt. Hierzu kann der Hochgeschwindigkeitstest [Hellebrand2014; Yan2003] verwendet werden. Während des Hochgeschwindigkeitstests wird die zu testende Schaltung übertaktet, um so kleine Verzögerungen sichtbar zu machen, die während der normalen Taktfrequenz versteckt geblieben wären. Durch Übertaktung der Schaltung kommt es jedoch auch dazu, dass zusätzliche X-Werte generiert werden, da zum verfrühten Beobachtungszeitpunkt noch nicht alle Signalpegel stabil an den Ausgängen der Schaltung anliegen. So entsteht während des Hochgeschwindigkeitstests eine höhere Anzahl

an X-Werten, sowie eine variierende X-Rate im Verlauf des Tests. Beides führt zu neuen Herausforderungen während der Verarbeitung der Ausgangssignale der zu testenden Schaltung, weswegen neue X-tolerante Kompaktierungsverfahren notwendig sind. Der Hochgeschwindigkeitstest wird in Kapitel 2.4.3 näher erläutert.

Mit dieser Arbeit wird ein Beitrag zur Bewältigung der zuvor beschriebenen Herausforderungen aufgrund von Unsicherheiten geleistet. In Kapitel 4 wird am Beispiel des eingebauten Hochgeschwindigkeitstests ein Verfahren zur Testantwortkompaktierung vorgestellt, welches in der Lage ist hohe und variierende X-Raten zu verarbeiten. Die vorgestellte X-tolerante Kompaktierung ist insbesondere für das *Silicon Lifecycle Management* notwendig. Um eine lebenslange Überwachung eines Mikrochips auf kleine Verzögerungen zu ermöglichen, ist es notwendig einen eingebauten Hochgeschwindigkeitstest [Kampmann2020; Hellebrand2014] zu verwenden. Hierbei wird die benötigte Testhardware vollständig auf dem Chip integriert (vgl. Kapitel 2.4.3), um einen regelmäßigen Test des Mikrochips während des gesamten Lebenszyklus zu ermöglichen. Insbesondere ist es bei einem eingebauten Selbsttest nötig, die Ausgaben der zu testenden Schaltung während des Tests zu kompaktieren. Mit dem in dieser Arbeit vorgestellten Verfahren zur Testantwortkompaktierung wird ein wichtiger Beitrag zur Verarbeitung der X-behafteten Ausgaben eines eingebauten Hochgeschwindigkeitstests geleistet.

Neben Unsicherheiten in Logik-Gattern kann es auch zu Unsicherheiten auf den Verbindungsleitungen der Logik-Schaltung kommen. Zur ganzheitlichen Betrachtung einer Logik-Schaltung ist es daher notwendig, auch die Verbindungsleitungen der Logik-Schaltung zu berücksichtigen. In Kapitel 5 wird daher ein Verfahren zur Unterscheidung zwischen kleinen Verzögerungen, die aufgrund von Übersprechen entstehen und kleinen Verzögerungen, die aufgrund von Prozessvariation entstehen, vorgestellt. Durch die Analyse der Testdaten eines Verzögerungstests in unterschiedlichen Arbeitspunkten mithilfe eines künstlichen neuronalen Netzes ist es, dank des vorgestellten Verfahrens, möglich, zwischen den Gründen für die Verzögerung zu unterscheiden. Durch das neue Testdatenanalyseverfahren ist es im Rahmen des *Silicon Lifecycle Managements* möglich, die Ausbeute zu erhöhen, da Mikrochips mit Verzögerungen aufgrund von Prozessvariationen nicht als fehlerhaft deklariert werden müssen. Sie können somit für Anwendungen mit geringeren Leistungsanforderungen verwendet werden. Gleichzeitig wird die Zuverlässigkeit der gefertigten Schaltungen erhöht, da Schaltungen mit Verzögerungen aufgrund von Übersprechen erkannt werden und somit Maßnahmen ergriffen werden können, um eine frühzeitige Alterung aufgrund von Elektromigration zu verhindern.

Bevor die vorgestellten Beiträge dieser Arbeit zum *Silicon Lifecycle Management* in Kapitel 4 und in Kapitel 5 eingeführt werden, wird in Kapitel 2 und Kapitel 3 eine kurze Einführung in den Test von hochintegrierten Schaltungen und in künstliche neuronale Netze gegeben. Abschließend wird in Kapitel 6 ein Fazit der Arbeit gezogen und in Kapitel 7 ein Ausblick auf weiterführende Arbeiten gegeben.

2 Einführung in den Test von hochintegrierten Schaltungen

Wie in der Einleitung erwähnt, werden aktuelle High-End SoCs in Technologien mit minimalen Strukturgrößen von 5 nm bis 7 nm gefertigt. Hierzu sind eine Vielzahl von komplexen Prozessschritten notwendig. Während dieses hochkomplexen Fertigungsprozesses kann es immer wieder zu Abweichungen zwischen der gewünschten Struktur und der gefertigten Struktur auf dem Siliziumwafer, wie z. B. durch Verunreinigungen, kommen. Mit diesem Kapitel wird eine kurze Einführung in den Test von hochintegrierten Schaltungen gegeben. Weiterführende Informationen sind unter anderem in [Wang2006], [Bushnell2000] und [Tehranipoor2011] zu finden.

Zum besseren Verständnis ist es zunächst wichtig, zwischen Defekten, Fehlverhalten und Fehlern zu unterscheiden, die wie folgt definiert sind.

Definition 2.1 (Defekt)**.** *Defekte (engl. defect) in elektronischen Systemen sind ungewollte Unterschiede zwischen der produzierten Hardware und dem geplanten Design [Bushnell2000].*

Definition 2.2 (Fehlverhalten)**.** *Unter einem Fehlverhalten (engl. error) in einem elektronischen System versteht man die Auswirkung eines Defektes am Ausgang der elektronischen Schaltung [Bushnell2000].*

Definition 2.3 (Fehler)**.** *Fehler (engl. fault) modellieren Defekte auf einer höheren Abstraktionsebene [Bushnell2000].*

Nach dieser Definition zur Unterscheidung von Defekten, Fehlverhalten und Fehlern gibt der folgende Abschnitt einen kurzen Überblick über häufig auftretende Defekte während der Fertigung von hochintegrierten Schaltungen.

2.1 Defekte

Die Abbildung 2.1 zeigt zwei Beispiele für die zuvor genannten Abweichungen. In der linken Abbildung ist ein Querschnitt einer hochintegrierten Schaltung zu sehen, in der zwei fehlende Durchverbindungen zwischen der letzten Metallebene (hellgrau) und dem Polysilizium (untere Ebene) mit roten Kreisen markiert sind. In der rechten Abbildung hingegen ist eine Aufsicht einer hochintegrierten Schaltung zu sehen, in der ein Schmutzpartikel zur Unterbrechung einer Verbindungsleitung und zu einer Verjüngung einer zweiten Verbindungsleitung geführt hat [Zivkovic2011].

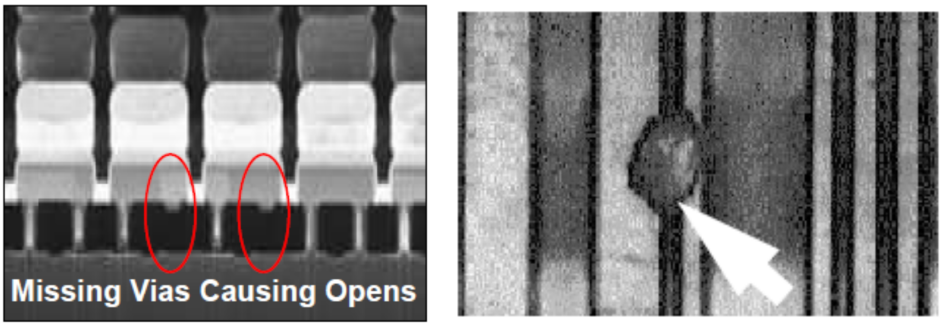


Abbildung 2.1: Mögliche Defekte in hochintegrierten Schaltungen [Zivkovic2011].

Die so entstehenden Defekte können unter anderem zu Kurzschlüssen führen, wodurch ein Signal z. B. dauerhaft mit dem positiven oder negativen Potential der Versorgungsspannung verbunden wird. Aufgrund von fehlenden Durchverbindungen kann es außerdem dazu kommen, dass ein Signal keinen definierten Signalpegel aufweist. Wie rechts in der Abbildung 2.1 zu sehen ist, kann es aber auch dazu kommen, dass eine Struktur nicht die gewünschte Breite aufweist und somit über einen erhöhten Widerstand im Vergleich zur gewünschten Struktur verfügt. Diese resistiven Defekte können unter anderem zu erhöhten Signallaufzeiten oder verfrühter Alterung führen. Der folgende Abschnitt gibt einen kurzen Überblick über häufige Alterungseffekte in hochintegrierten Schaltungen.

2.2 Alterungseffekte

Während des Lebenszyklus einer hochintegrierten Schaltung werden sowohl die Halbleiterbauelemente wie z. B. Transistoren, als auch die Leitungen der integrierten Schaltung beansprucht, was zu Alterungseffekten führen kann. Im Laufe dieses Kapitels wird auf die wichtigsten Alterungseffekte für Halbleiterbauelemente sowie Verbindungsleitungen eingegangen. Wenn nicht anders gekennzeichnet basiert dieses Kapitel auf [McPherson2019].

2.2.1 Alterungseffekte in Transistoren

Unter Alterungseffekten in Transistoren versteht man die Veränderungen der elektrischen Eigenschaften des Transistors wie z. B. der Schwellenspannung U_{TH} aufgrund der elektrischen Belastung des Transistors. Häufig auftretende Effekte in aktuellen Halbleitertechnologien sind die Injektion heißer Ladungsträger und die Bias-Temperaturinstabilität, die im Folgenden kurz erläutert werden.

Injektion heißer Ladungsträger (engl. Hot Carrier Injection, HCI)

Durch die Beschleunigung der Ladungsträger im Kanal eines MOSFETs, aufgrund des anliegenden elektrischen Feldes, können die Ladungsträger (Elektronen oder Löcher) eine so hohe kinetische Energie erhalten, dass sie die Barriere zum Gate-Oxid überschreiten und Löcher in die Siliziumdioxid- (SiO_2-)Strukturen schlagen. Der Effekt tritt hauptsächlich bei NMOS-Transistoren auf und führt zu einer Erhöhung der Schwellenspannung U_{TH} und einer Verringerung des Drain-Stroms I_D. Aufgrund der größeren Gate-Oxid-Barriere für Löcher ist der Effekt in PMOS-Transistoren geringer.

Bias-Temperaturinstabilität (engl. Bias Temperature Instability, BTI)

Durch die Bias-Temperaturinstabilität kommt es während des Lebenszyklus eines Transistors zur Erhöhung der Schwellenspannung U_{TH}. Der Effekt tritt bei PMOS-Transistoren als negative Bias-Temperatur-Instabilität (engl. Negative Bias Temperature Instability, NBTI) und bei NMOS-Transistoren als positive Bias-Temperatur-Instabilität (engl. Positive Bias Temperature Instability, PBTI) auf, wobei der Effekt in PMOS-Transistoren größer ist (NBTI). Wenn der Transistor eingeschaltet ist, können sich in dem Übergang zwischen Kanal und Gate-Oxid des Transistors H-Ionen lösen und vom Übergang wegdriften. Dies führt zu einer Erhöhung der Schwellenspannung U_{TH} des Transistors. Teilweise wird dieser Effekt repariert, wenn der Transistor ausgeschaltet ist. Eine vollständige Erholung des Transistors ist jedoch nicht möglich.

2.2.2 Alterungseffekte in Leitungen

In hochintegrierten Schaltungen treten Alterungseffekte nicht nur innerhalb der Transistoren auf, sondern auch auf den Verbindungsleitungen zwischen den Modulen eines SoCs und zwischen den Logikelementen von kombinatorischen Schaltungen. Im Folgenden wird der häufig auftretende Effekt der Elektromigration kurz erläutert.

Elektromigration

Unter Elektromigration versteht man den Materialmigrationsprozess in metallischen Leiterbahnen elektronischer Schaltung aufgrund eines elektrischen Feldes. Die Hauptursache für die Elektromigration ist die Kraft, die aufgrund von der Impulsübertragung der bewegten Ladungsträger auf die Metallionen der Leiterbahn wirkt. [Lienig2018]

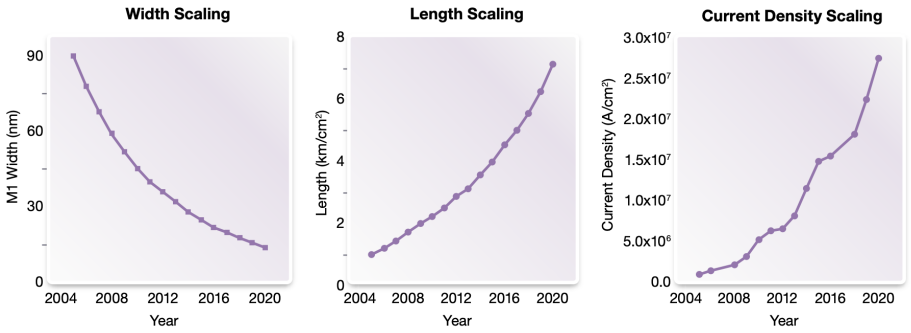


Abbildung 2.2: Veränderung der Dimensionen von Verbindungsleitungen in hochintegrierten Schaltungen [Geden2011].

Mit immer weiter schrumpfenden Strukturgrößen werden auch die Querschnitte von Verbindungsleitungen in integrierten Schaltungen immer kleiner. Bei gleichzeitig steigender Komplexität der hochintegrierten Schaltungen wird die Gesamtleitungslänge hingegen immer größer, da immer mehr und längere Verbindungsleitungen benötigt werden. Da die Stromstärke jedoch nicht im gleichen Maß schrumpft wie der Leitungsquerschnitt, kommt es in den Verbindungsleitungen zu immer größeren Stromdichten, was wiederum zu erhöhter Elektromigration führt. Dies veranschaulichen auch die Diagramme in der Abbildung 2.2. Im linken Diagramm ist die Entwicklung der Leitungsbreite über die Jahre 2004 bis 2020 aufgetragen. Das mittlere Diagramm zeigt die Entwicklung der Länge der Verbindungsleitungen, während im rechten Diagramm die Entwicklung der Stromdichte zu sehen ist. [Geden2011; Lienig2018]

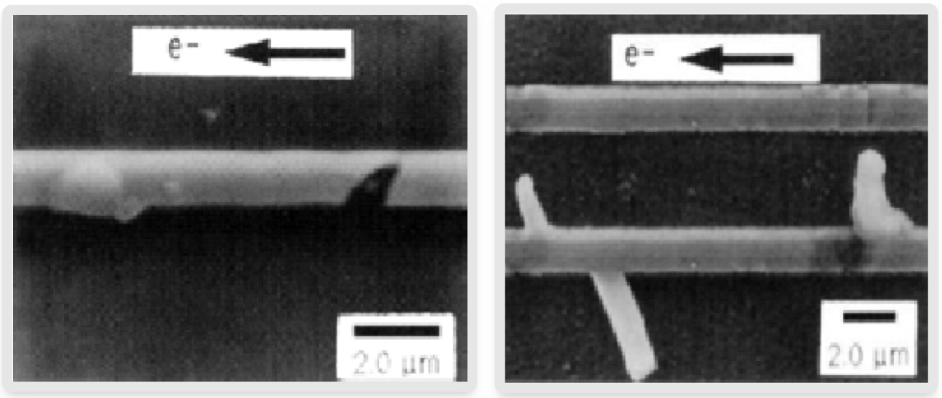


Abbildung 2.3: Beispiele von Elektromigration [Geden2011].

Wie in Abbildung 2.3 zu sehen ist, kann es zu zwei unterschiedlichen Defekten durch Elektromigration kommen. Einerseits kann es zu Fehlstellen (engl. voids) in der Verbindungsleitung oder zu Auswüchsen (engl. hillocks) kommen, welche im schlimmsten Fall zu Unterbrechungen (engl. open circuits), respektive zu Kurzschlüssen führen können.

Mithilfe von Blacks Gleichung [Black1969], kann der Zentralwert der Zeit bis zum Ausfall (engl. Median Time To Failure, MTTF) aufgrund von Elektromigration wie folgt berechnet werden.

$$MTTF = \frac{A}{J^n} e^{\frac{Q}{k_B T}} \tag{2.1}$$

Wobei A eine Materialkonstante ist, J die Stromdichte in $\mathrm{A\,cm^{-2}}$, n ein Materialparameter, Q die Aktivierungsenergie der Elektromigration in eV, k_B die Boltzmann-Konstante in $\mathrm{J\,K^{-1}}$ und T die Temperatur in K.

Wie in [Sadeghi-Kohan2021; Lienig2018; Geden2011] gezeigt, steigt die Stromdichte der Verbindungsleitungen immer weiter mit schrumpfender Strukturgröße, was laut (2.1) zur Verringerung der mittleren Zeit bis zum Ausfall der Schaltung und somit zu einem Zuverlässigkeitsproblem führt.

2.3 Fehlermodelle

Wie bereits zuvor gesehen, können neben Defekten auch Alterungseffekte und Prozessvariationen zu Unsicherheiten führen, die als Fehlverhalten sichtbar werden. Nicht zuletzt die Unterscheidung der Ursachen des Fehlverhaltens stellt den Test von hochintegrierten Schaltungen vor große Herausforderungen. Bevor in den Kapiteln 4 und 5 Lösungsansätze für den Umgang mit diesen Unsicherheiten gegeben werden, werden im folgenden Abschnitt Fehlermodelle zur Modellierung der zuvor genannten Defekte vorgestellt, die zur Entwicklung eines Tests notwendig sind.

Zunächst wird das Haftfehlermodell vorgestellt, mit dem Defekte modelliert werden können, die zu logischem Fehlverhalten der Schaltung führen. Anschließend werden mit dem Pfad-Verzögerungsfehlermodell und dem Transitions-Verzögerungsfehlermodell zwei Fehlermodelle eingeführt, mit denen Verzögerungen von Signaländerungen modelliert

werden können. Zur Modellierung von kleinen Verzögerungen von Signalverläufen wird daraufhin das Modell der kleinen Verzögerungsfehler eingeführt. Abschließend wird das Modell der Verbindungsfehler eingeführt, mit dem parasitäre Effekte zwischen Verbindungsleitungen modelliert werden können.

2.3.1 Haftfehler

Eines der weitverbreitetsten Fehlermodelle ist das Haftfehlermodell (engl. stuck-at fault model). In diesem Fehlermodell gibt es die zwei Fehlertypen Haftfehler-an-1 und Haftfehler-an-0, die an einer Signalleitung der Schaltung entstehen können. Abbildung 2.4 zeigt zwei UND-Gatter an denen jeweils am Eingang A ein Haftfehler-an-1 (links) bzw. -an-0 (rechts) aufgetreten ist. Wie in der Abbildung 2.4 zu sehen ist, wird die Leitung A dauerhaft auf logisch 1 (U_{DD}) oder 0 (GND) gezogen, was zu einem Fehlverhalten am Ausgang Z führt, der dauerhaft logisch 1, respektive 0 ist, unabhängig von der Belegung des Eingangs A.

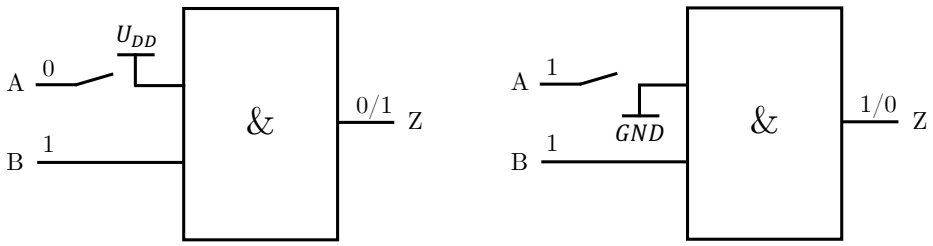


Abbildung 2.4: Beispiel eines Haftfehlers-an-1 und eines Haftfehlers-an-0 am Eingang A eines UND-Gatters.

Wie im linken Beispiel zu sehen, wird das Signal A mit dem logischen Wert 0 belegt und das Signal B mit 1. Aufgrund der logischen 0 am Eingang A müsste der Ausgang Z des UND-Gatters auch den Wert 0 annehmen. Da der Eingang A des UND-Gatters dauerhaft auf 1 gezogen wird, ist auch der Ausgang Z bei der gezeigten Eingangsbelegung 1. Die Belegung des Ausgangs Z wird entsprechend mit 0/1 angegeben, da das Signal Z im fehlerfreien Fall den Wert 0 annimmt und im fehlerhaften Fall den Wert 1. Das Haftfehlermodell deckt eine Vielzahl von Defekten ab, wie z. B. einen Kurzschluss einer Leitung zu U_{DD} oder GND. Aber auch weitere Defekte können mithilfe des Haftfehlermodells abgedeckt werden [Ma1995].

Für das Haftfehlermodell wird häufig angenommen, dass immer nur ein Fehler in einer Schaltung vorhanden ist und nur die Signalleitungen der Schaltung beeinflusst werden.

Das Verhalten der Logik-Gatter bleibt unberührt. Die Anzahl der möglichen Haftfehler in einer Schaltung ergibt sich somit zu # Haftfehler $= 2 \cdot n_s \in \mathcal{O}(n_s)$ wobei n_s die Anzahl der Signalleitungen in der Schaltung angibt.

2.3.2 Verzögerungsfehler

Wie in der Einleitung erwähnt wurde, kommt es während der Fertigung auch zu Defekten, die sich als Verzögerung des Signalverlaufs manifestieren. Für die Modellierung dieses Fehlverhaltens kann das Modell der Verzögerungsfehler verwendet werden, welches im folgenden Abschnitt eingeführt wird. Hierbei wird zwischen zwei möglichen Modellen unterschieden. Im Pfad-Verzögerungsfehlermodell wird ein Defekt als kumulativ auftretender Verzögerungsfehler modelliert, wohingegen im Transitions-Verzögerungsfehlermodell ein Defekt als lokal auftretender Verzögerungsfehler modelliert wird. Weiterführende Informationen sind in [Krstic1998] zu finden.

2.3.2.1 Pfad-Verzögerungsfehlermodell

Zur Modellierung von Defekten, die das Zeitverhalten von Signalen verändern, wie z. B. resistive Defekte, kann das Pfad-Verzögerungsfehlermodell verwendet werden, welches in [Smith1985; Lin1987] eingeführt wurde. In diesem Fehlermodell können an einem Fehlerort zwei verschieden Fehlertypen auftreten, die jeweils das Signal verzögern. Hierbei handelt es sich um den *slow-to-rise*- und den *slow-to-fall*-Fehlertypen. Beim *slow-to-rise* Fehler wird die steigende Flanke am Fehlerort so lange verzögert, dass das Ausgangssignal der Schaltung während des Beobachtungszeitpunktes nicht mehr korrekt bestimmt werden kann. Entsprechend wird beim *slow-to-fall* Fehler die fallende Flanke verzögert.

Im Pfad-Verzögerungsfehlermodell wird angenommen, dass es zu einer kumulativen Verzögerung entlang eines Pfades zwischen den Eingängen und den Ausgängen kommt, welche insgesamt groß genug ist, um das Ausgangssignal über den Beobachtungszeitpunkt t_s zu verzögern.

Der Beobachtungszeitpunkt t_s wird dabei, bei sequenziellen Schaltungen, durch die verwendete Taktperiode T_{CLK} festgelegt, wobei gilt $t_s = t_0 + c \cdot T_{CLK}$ für den aktuellen Takt $c \in \mathbb{N}$ und den Startzeitpunkt $t_0 \in \mathbb{R}$. Zur Vereinfachung der Darstellung wird im Weiteren $t_0 = 0\,\mathrm{s}$ und $c = 1$ angenommen, wodurch $t_s = T_{CLK}$ gilt.

Als Fehlerort kommen alle möglichen Pfade zwischen den Eingängen und Ausgängen der Schaltung infrage. Da die Anzahl der Pfade einer Schaltung exponentiell mit der Schaltungsgröße (Anzahl Gatter) steigt, kann das Pfad-Verzögerungsfehlermodell nur für kleine Bereiche von Schaltungen verwendet werden. Die Anzahl der Pfade kann dabei wie folgt anhand der Anzahl der Gatter n_g abgeschätzt werden.

$$\#\ \text{Pfade} \in \mathcal{O}(2^{n_g}) \tag{2.2}$$

Aufgrund des exponentiellen Wachstums der Anzahl der Fehler im Pfad-Verzögerungsfehlermodell ist es häufig nicht möglich, das Pfad-Verzögerungsfehlermodell auf moderne Schaltungen mit hoher Gatteranzahl effizient anzuwenden. Daher wird häufig das Transitions-Verzögerungsfehlermodell verwendet, welches im folgenden Abschnitt beschrieben wird. [Wang2006]

2.3.2.2 Transitions-Verzögerungsfehlermodell

Im Gegensatz zum Pfad-Verzögerungsfehlermodell, welches annimmt, dass es zu einer kumulativen Verzögerung entlang eines Pfades kommt, wird im Transitions-Verzögerungsfehlermodell angenommen, dass eine Verzögerung immer lokal an einer Signalleitung auftritt und groß genug ist, um das Ausgangssignal über den Beobachtungszeitpunkt t_s hinaus zu verzögern. Genau wie im Pfad-Verzögerungsfehlermodell werden im Transitions-Verzögerungsfehlermodell zwei Fehlertypen unterschieden, einerseits ein Fehler, der eine steigende Flanke verzögert (*slow-to-rise*) und andererseits ein Fehler, der eine fallende Flanke verzögert (*slow-to-fall*).

Im Gegensatz zum Pfad-Verzögerungsfehlermodell kommen im Transitions-Verzögerungsfehlermodell jedoch nicht alle möglichen Pfade als Fehlerort infrage, sondern nur alle Signalleitungen. Die Anzahl der möglichen Transitions-Verzögerungsfehler ergibt sich somit zu

$$\#\ \text{Fehler} = \#\ \text{Fehlertypen} \cdot \#\ \text{Fehlerorte} = 2 \cdot n_s \in \mathcal{O}(n_s), \tag{2.3}$$

wobei n_S der Anzahl der Signalleitungen der zu testenden Schaltung entspricht. Somit entspricht die Anzahl der Transitions-Verzögerungsfehler der Anzahl der Fehler im Haftfehlermodell, wobei mit dem Transitions-Verzögerungsfehler eine andere Menge an Defekten abgedeckt werden kann.

2.3.3 Kleine Verzögerungsfehler

Das Fehlermodell der kleinen Verzögerungsfehler ist vergleichbar mit dem Transitions-Verzögerungsfehlermodell. Während im Transitions-Verzögerungsfehlermodell und im Pfad-Verzögerungsfehlermodell der Fehler als $> T_{CLK}$ angenommen wird, was dazu führt, dass der Fehler zum nächsten Beobachtungszeitpunkt erkannt werden kann, wenn er an mindestens einen Ausgang propagiert wird, wird im Modell der kleinen Verzögerungsfehler ein Fehler mit einer dedizierten Größe $\delta \in \mathbb{R}$ angenommen. So ist es möglich Defekte zu modellieren, die zwar eine Abweichung des Zeitverhaltens eines Signals verursachen aber nicht notwendigerweise zu einem Fehlverhalten am Ausgang der Schaltung während des Beobachtungszeitpunktes führen.

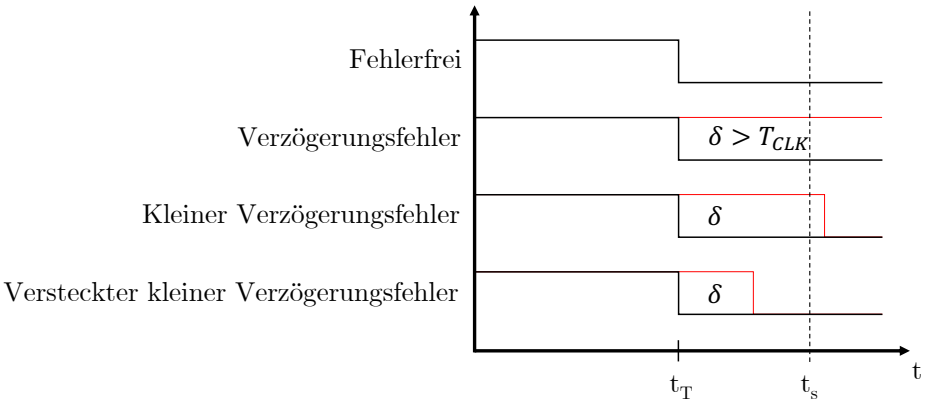


Abbildung 2.5: Unterschied zwischen Verzögerungsfehlern und kleinen Verzögerungsfehlern.

Die Abbildung 2.5 verdeutlicht die Unterschiede zwischen den Fehlermodellen. Während in der ersten Zeile ein fehlerfreier Signalverlauf zu sehen ist, zeigt die zweite Zeile dasselbe Signal, bei dem ein Verzögerungsfehler aufgetreten ist. In der dritten und vierten Zeile wurde die fallende Flanke jeweils durch einen kleinen Verzögerungsfehler mit unterschiedlicher Fehlergröße verzögert. Im letzten Fall ist die Fehlergröße des Fehlers kleiner als der Schlupf (engl. slack) des längsten Pfades, auf dem der Fehler zum Ausgang propagiert werden kann. Wobei der Schlupf durch die Differenz zwischen dem Beobachtungszeitpunkt t_s und dem Zeitpunkt t_T, an dem die Transition stattfindet, definiert ist. Diese kleinen Verzögerungsfehler nennt man versteckte kleine Verzögerungsfehler (engl.

Hidden Delay Faults, HDFs), wenn der Schlupf an allen Ausgängen immer zu groß ist, um die Verzögerung zu erkennen.

Obwohl das Verhalten der digitalen Schaltung durch versteckte kleine Verzögerungsfehler nicht beeinflusst wird, ist die Detektion solcher kleiner Verzögerungsfehler wichtig, da sie einerseits ein Indikator für einen Frühausfall (engl. Early Life Failure, ELF) der Schaltung [Kim2010] und andererseits ein Hinweis auf die Alterung einer hochintegrierten Schaltung sein können. Durch die Detektion dieser kleinen Verzögerungsfehler im Laufe des Lebenszyklus, z. B. mithilfe eines periodisch durchgeführten eingebauten Selbsttests (siehe Kapitel 2.4.1), kann eine Alterung der Schaltung detektiert werden. So ist es möglich, Maßnahmen aufgrund der Alterung zu ergreifen wie z. B. die Reduktion der Leistungsfähigkeit der Schaltung (engl. graceful degradation) oder das Senden einer entsprechenden Wartungsaufforderung an den Nutzer, im Sinne einer vorhersagbaren Wartung.

2.3.4 Verbindungsfehler

Während die bisherigen Fehlermodelle in erster Linie die Defekte innerhalb von Logikbausteinen einer digitalen Schaltung beschreiben, können außerdem parasitäre Effekte, wie z. B. Übersprechen, zwischen Verbindungsleitungen auftreten, die von den bisherigen Fehlermodellen nicht abgedeckt werden. Diese Übersprecheffekte können einerseits die Signalqualität beeinflussen, wodurch sie als Fehler modelliert werden können und anderseits die Alterung der Schaltung aufgrund von Elektromigration fördern. In diesem Abschnitt wird daher ein Fehlermodell für Übersprecheffekte vorgestellt.

Ein Modell zur Modellierung von Übersprechen wurde in [Roy2011] vorgestellt. Unter Übersprechen versteht man dabei, die Beeinflussung des Signalverlaufs einer Leitung durch den Signalverlauf einer zweiten Leitung aufgrund von parasitärer kapazitiver oder induktiver Kopplung der beiden Leitungen. Wie in Abbildung 2.6 zu sehen ist, wird ein Leitungspaar mithilfe von RLC-Gliedern modelliert, wobei die Leitungswiderstände R_1 und R_2, die Leitungskapazitäten zum Massenpotential C_1 und C_2, die Leitungsinduktivitäten L_1 und L_2, sowie die Koppelkapazität C_c berücksichtigt werden.

Mithilfe des Modells lassen sich vier Fehlverhalten durch Übersprechen ableiten. Die vier Fehlverhalten sind in der Abbildung 2.7 anhand eines Beispiels mit einem Leitungspaar dargestellt. Die obere Leitung wird hierbei als Opfer-Leitung angesehen und die untere Leitung als Angreifer-Leitung. Wenn die Opfer-Leitung auf logisch 1 liegt und auf der

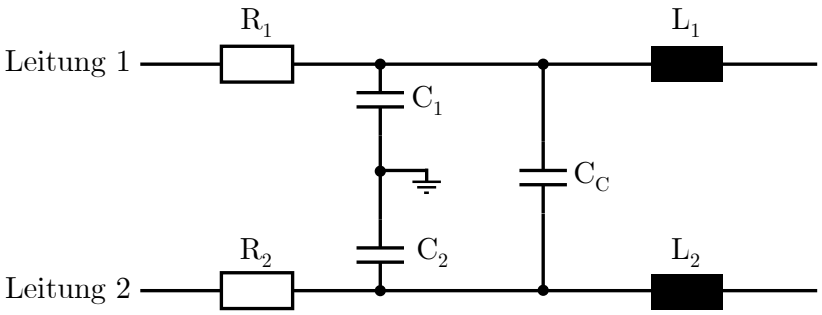


Abbildung 2.6: RLC Fehlermodell zur Modellierung von Übersprechen [Cuviello1999].

Angreifer-Leitung eine Flanke auftritt, kann es zu einer Überhöhung der Signalspannung oder einem negativen Signalimpuls auf der Opfer-Leitung kommen. Liegt die Opfer-Leitung in der gleichen Situation auf logisch 0, kann es zu einem positiven Signalimpuls oder einer Unterschreitung der Signalspannung auf der Opfer-Leitung kommen. Treten sowohl auf der Opfer-Leitung als auch auf der Angreifer-Leitung gleichzeitig Flanken in derselben Richtung auf, kommt es zu einer Beschleunigung des Signalverlaufs auf der Opfer-Leitung. Treten die Flanken in entgegengesetzte Richtung auf, wird der Signalverlauf auf der Opfer-Leitung verzögert. Insbesondere die Verzögerung des Signalverlaufs kann mithilfe des Fehlermodells der kleinen Verzögerungsfehler modelliert werden.

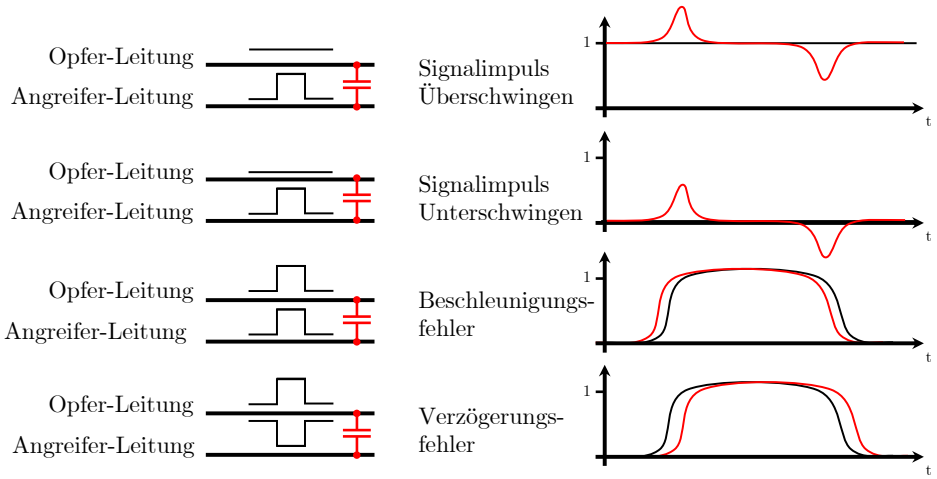


Abbildung 2.7: Fehlverhalten aufgrund von Übersprechen [Chen1998].

Nachdem in diesem Abschnitt auf die am weitverbreitetsten Fehlermodelle eingegangen wurde, wird im folgenden Abschnitt erläutert, wie diese Fehlermodelle genutzt werden können, um einen Test für hochintegrierte Schaltungen zu entwickeln.

2.4 Test

Wie bereits in der Einleitung erwähnt, kommt es während der Fertigung von hochintegrierten Schaltungen aufgrund der hohen Komplexität der Fertigungsverfahren immer wieder zu Defekten in der Schaltung. Um defekte Schaltungen nicht auszuliefern oder sogar in sicherheitskritische Anwendungen wie ein selbstfahrendes Auto oder medizinische Geräte einzubauen, werden hochintegrierte Schaltungen einem Fertigungstest unterzogen. Hierzu werden an die Eingänge der zu testenden Schaltung (engl. Circuit Under Test, CUT) unterschiedliche Signalpegel angelegt und die Ausgaben der Schaltung beobachtet. Eine Eingangsbelegung wird dabei als Testmuster bezeichnet und die zugehörige Ausgabe der zu testenden Schaltung Testantwort genannt.

Die für den Test notwendige Menge an Testmustern wird als Testmustermenge $\mathscr{T}_M$ bezeichnet. In Abbildung 2.8 ist eine zu testende Schaltung mit einem Testmuster und der zugehörigen Testantwort zu sehen. Das Testmuster und die fehlerfreie Testantwort lassen sich dabei als Zeilenvektoren darstellen.

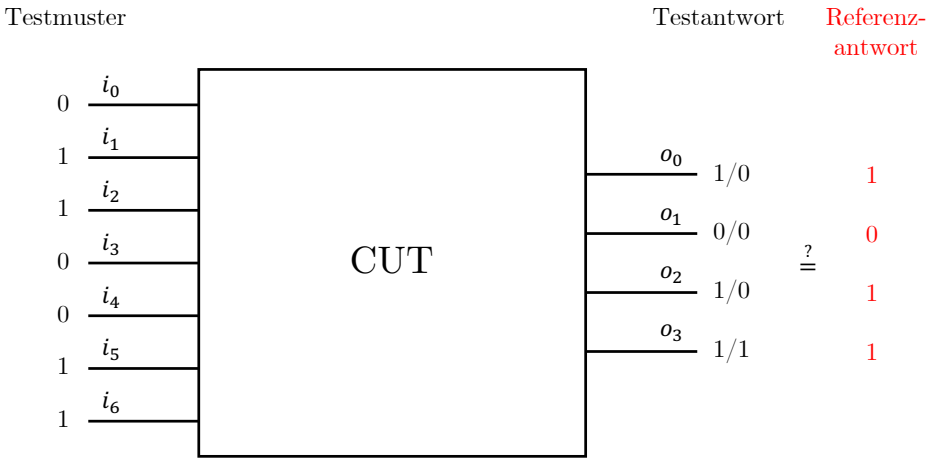


Abbildung 2.8: Beispiel einer zu testenden Schaltung (CUT) mit Testmustern und Testantworten.

Ein naiver Ansatz zum Test einer hochintegrierten Schaltung ist, alle möglichen Kombinationen der Eingangsbelegungen an die zu testende Schaltung anzulegen und die Testantworten mit zuvor berechneten Referenzantworten (engl. golden reference) zu vergleichen. Wie im Beispiel in Abbildung 2.8 zu sehen ist, entspricht die fehlerhafte Testantwort $(o_1, o_2, o_3, o_4) = (0, 0, 0, 1)$ nicht der Referenzantwort $(1, 0, 1, 1)$. Mithilfe des Testmusters $(i_0, i_1, i_2, i_3, i_4, i_5, i_6) = (0, 1, 1, 0, 0, 1, 1)$ ist es somit möglich, den gegebenen Fehler zu entdecken. Die getestete Schaltung kann so als fehlerhaft klassifiziert und aussortiert werden, bevor sie ausgeliefert wird.

Dieser naive Ansatz ist jedoch nicht praktikabel, was am Beispiel des im Jahr 2022 veröffentlichen Intel® Core™ i9-12900T Prozessors schnell deutlich wird. Der genannte Prozessor wird in einem Flip Chip Land Grid Array (FCLGA) Sockel vertrieben, welcher über 1700 Ein- und Ausgabe Pins verfügt [Intel2022]. Unter der Annahme, dass die Hälfte der Pins als Eingang verwendet wird, müssten für einen vollständigen (engl. exhaustive) Test $2^{850} \approx 10^{255}$ Testmuster an die zu testende Schaltung angelegt werden. Bei der Betrachtung von Verzögerungsfehlern würde sich die Anzahl der Testmuster sogar noch weiter auf $2^{2 \cdot 850} \approx 10^{511}$ erhöhen, da pro Test zwei Testmuster benötigt werden, um eine Flanke am Eingang der Schaltung zu erzeugen. Wenn der Test mit der maximal möglichen Taktfrequenz f_{CLK} von 4,9 GHz [Intel2022] durchgeführt wird, ergibt sich eine Testzeit von $\approx 2{,}59 \cdot 10^{278}$ Jahren pro Chip.

Um einen Test in angemessener Zeit und mit möglichst geringen Kosten durchführen zu können, müssen die Testmuster so ausgewählt werden, dass möglichst wenig Testmuster benötigt werden und gleichzeitig eine möglichst hohe Fehlerabdeckung erreicht wird. Unter Fehlerabdeckung versteht man dabei, wie in (2.4) definiert, den Anteil der detektierten Fehler an der Anzahl aller Fehler im verwendeten Fehlermodell.

$$\text{Fehlerabdeckung} = \frac{\#\text{ detektierter Fehler}}{\#\text{ Fehler im Fehlermodell}} \tag{2.4}$$

Zur Erzeugung einer Testmustermenge kann eine Vielzahl an Algorithmen verwendet werden. Die Testmustererzeugung mithilfe dieser Algorithmen wird als automatisierte Testmustererzeugung (engl. Automatic Test Pattern Generation, ATPG) bezeichnet. Der Stand der Technik verwendet hierzu häufig ein zweistufiges Verfahren. In der ersten Phase werden zufällig Testmuster generiert und mithilfe von Fehlersimulation die Fehlerabdeckung dieser zufälligen Testmustermenge berechnet. Mithilfe dieser zufälligen Testmustermenge kann in der Regel bereits eine hohe Anzahl „leicht detektierbarer“ Fehler erkannt werden. Für die verbleibenden „schwer detektierbaren“ Fehler werden in einer zweiten deterministischen Phase konkrete Testmuster für die einzelnen Fehler mithilfe komplexer Algorithmen erzeugt. Es wurde gezeigt, dass die Testmustererzeugung für einen dedizierten Fehler innerhalb einer beliebigen kombinatorischen Schaltung NP-vollständig ist [Fujiwara1982]. Daher ist insbesondere die zweite Phase der automatischen Testmustererzeugung häufig sehr zeitaufwendig. Da sie pro Schaltungsentwurf jedoch lediglich einmal durchgeführt werden muss, ist dieser Aufwand in der Regel vertretbar. Weiterführende Informationen sind in [Eggersglüß2012] zu finden.

2.4.1 Eingebauter Selbsttest

Noch einen Schritt weiter geht der eingebaute Selbsttest (engl. Built-In Self-Test, BIST). Hierunter versteht man die vollständige Integration der notwendigen Module für die Durchführung des Tests in die zu testende Schaltung. Der eingebaute Selbsttest kann z. B. von außen mithilfe eines Startsignals gestartet werden. Nach Ablauf des Tests wird dann mithilfe eines Ausgangssignals gezeigt, ob der Test erfolgreich war und somit die Schaltung fehlerfrei ist oder nicht.

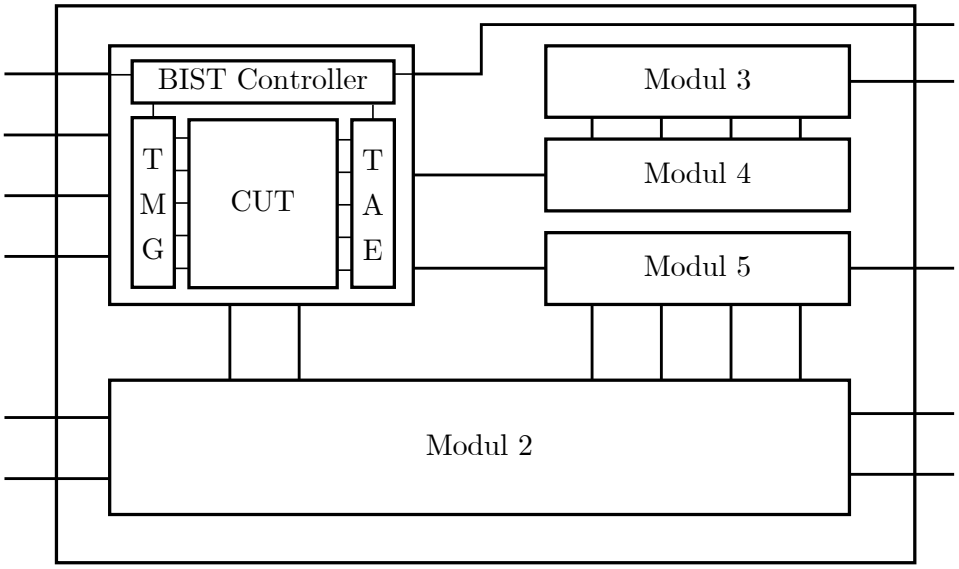
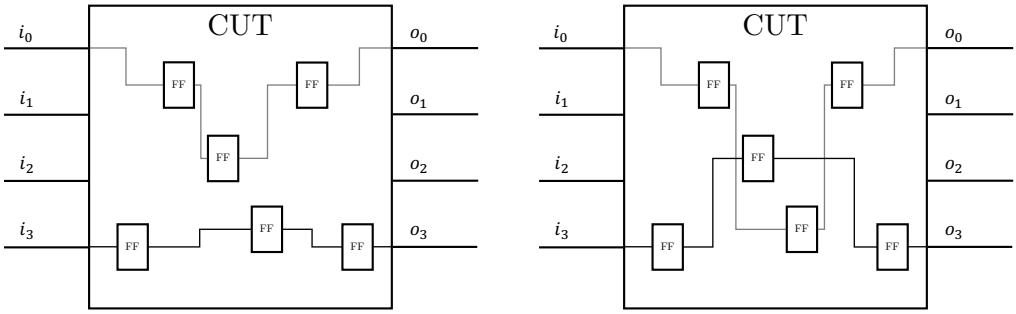


Abbildung 2.9: System-on-Chip mit eingebautem Selbsttest.

Mithilfe des eingebauten Selbsttests ist es möglich, einzelne Module eines System-on-Chips mit individuellen Testmethoden zu testen. In Abbildung 2.9 ist z. B. eine schematische Darstellung eines System-on-Chips zu sehen. Oben links in der Abbildung ist ein Logikmodul zu sehen, welches um einen eingebauten Selbsttest erweitert wurde. Hierzu wurde zusätzlich ein Testmustergenerator (TMG), eine Testantwortevaluierung (TAE) und ein BIST Controller integriert.

Mithilfe des TMG Moduls werden die benötigten Testmuster direkt auf dem Chip generiert und an die zu testende Schaltung angelegt. Die Testantwort der Schaltung wird ebenfalls direkt auf dem Chip mithilfe des TAE Moduls evaluiert. Die Steuerung des Tests übernimmt hierzu der integrierte BIST Controller. Ein Beispiel für einen Testmustergenerator ist z. B. ein linear rückgekoppeltes Schieberegister, welches genutzt wird, um pseudozufällige Testmuster zu generieren. Der Aufbau eines linear rückgekoppelten Schieberegisters ist [Wang2006] zu entnehmen. Beispiele für die Testantwortevaluierung werden in Kapitel 2.4.2 gegeben. Neben dem Testmustergenerator und der Testantwortevaluierung werden in dieser als STUMPS (engl. Self Test Using MISR and Parallel Shift

register sequence generator) [Bardell1982] bekannten Architektur parallel angeordnete Prüfpfade verwendet. Bei den Prüfpfaden handelt es sich um Speicherelemente der zu testenden Schaltung, die im Testbetrieb zu Schieberegistern verschaltet sind. Durch die Verwendung von Prüfpfaden ist es möglich, sequenzielle Schaltungen während des Tests als rein kombinatorische Schaltungen zu betrachten. Für die Bildung der Prüfpfade können verschiedene Optimierungsziele wie die Minimierung der Verdrahtungskosten oder Verteilung der Schaltaktivität während des Tests verwendet werden. Je nach Optimierungsziel kommt es zu unterschiedlichen Prüfpfadkonfigurationen. Die Abbildung 2.10 zeigt zwei Beispiele für unterschiedliche Prüfpfadkonfigurationen, die im Weiteren als parallele Prüfpfade dargestellt werden.



(a) Prüfpfadkonfiguration A (b) Prüfpfadkonfiguration B

Abbildung 2.10: Beispiel zweier Prüfpfadkonfigurationen.

Diese parallel angeordneten Prüfpfade werden im eingebauten Selbsttest, wie in Abbildung 2.11 gezeigt, mithilfe des Testmustergenerators mit Testmustern gefüllt und die Testantworten mithilfe eines Signaturregisters mit mehreren Eingängen (engl. Multiple-Input Signature Register, MISR) zu Signaturen komprimiert.

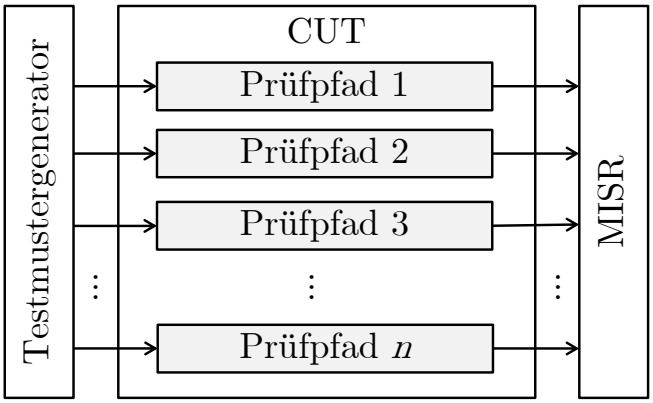


Abbildung 2.11: STUMPS Architektur [Bardell1982].

Für den Test auf Verzögerungsfehler muss an den Eingängen der zu testenden Schaltung eine Flanke erzeugt werden, die durch den Fehlerort zu mindestens einem Ausgang der Schaltung propagiert werden muss. Hierzu muss die Schaltung zunächst mit einem

ersten Testmuster initialisiert und im darauffolgenden Takt eine Flanke mithilfe eines zweiten Testmusters erzeugt werden. In der ursprünglichen STUMPS Architektur ist es jedoch nicht möglich, ein zweites Testmuster innerhalb eines Taktes an die Schaltung anzulegen, da zunächst das gesamte Testmuster über mehrere Taktzyklen in die Prüfpfade eingeschoben werden muss, bevor es an die kombinatorische Logik angelegt werden kann.

Um den Test auf Verzögerungsfehler auch im eingebauten Selbsttest zu ermöglichen, muss daher die STUMPS Architektur erweitert werden. Dies ist z. B. durch die Verwendung von Launch-on-Capture [Savir1994], Launch-on-Shift [Savir1993] oder Enhanced Scan [DasGupta1995] möglich.

Bei der Launch-on-Capture Methode wird zunächst ein Testmuster in die Prüfpfade eingeschoben und an die zu testende Schaltung angelegt. Die resultierende Testantwort wird nun im nächsten Takt wiederum als Eingabe für die zu testende Schaltung verwendet, um so die benötigten Flanken an den Eingängen der Schaltung zu erzeugen. Die möglichen Testmusterpaare werden so jedoch durch die Struktur der zu testenden Schaltung eingeschränkt.

Bei der Verwendung von Launch-on-Shift wird genau wie beim Launch-on-Capture zunächst ein Testmuster in die Prüfpfade eingeschoben und so die Schaltung initialisiert. Nun wird jedoch nach einem Taktzyklus das Testmuster um eine Stelle innerhalb der Prüfpfade verschoben und so das benötigte zweite Testmuster erzeugt. Auch hier ist die Menge der möglichen Testmusterpaare durch die Struktur der Prüfpfade eingeschränkt.

Eine uneingeschränkte Menge an möglichen Testmusterpaaren erhält man bei der Verwendung des sogenannten Enhanced Scans. Hier werden die Prüfpfadelemente durch ein zusätzliches Latch erweitert, in dem ein weiteres Bit gespeichert werden kann. So ist es möglich, zunächst das erste Testmuster in die Prüfpfade einzuschieben und in den Latches der Prüfzellen abzulegen. Anschließend kann ein zweites Testmuster in die Prüfpfade geladen werden. Beide Testmuster können so innerhalb von zwei Taktzyklen an die zu testende Schaltung angelegt werden.

Für unterschiedliche Module innerhalb des Systems können mithilfe des eingebauten Selbsttests unterschiedliche Testmethoden angewandt werden. Für den Zugriff auf die einzelnen eingebauten Testmethoden der unterschiedlichen Module kann, wie im Standard IEEE 1149 [IEEE1994] beschrieben, der standardisierte Test Access Port (TAP) verwendet werden, der mithilfe einer Boundary Scan Architektur zu einem Schiebere-

gister verschaltet werden kann. Auch der Zugriff auf einzelne Instrumente innerhalb der Teststrukturen kann mithilfe von rekonfigurierbaren Prüfpfaden, wie im Standard IEEE 1687 [IEEE2014] beschrieben, realisiert werden.

Ein weiterer großer Vorteil des eingebauten Selbsttests ist, dass er auch im Laufe des Lebenszyklus der hochintegrierten Schaltung wieder verwendet werden kann und so auch die Detektion von Alterungseffekten ermöglicht.

2.4.2 Testantwortkompaktierung

Um während des eingebauten Selbsttests die Testantworten mit den Referenzantworten vergleichen zu können, müssen letztere auf dem Mikrochip gespeichert werden. Zur Verringerung des hierzu benötigten Speicheraufwands, werden Verfahren zur Testantwortkompaktierung verwendet, die im Folgenden näher erläutert werden.

Hierzu wird zunächst die Testantwortmatrix $\boldsymbol{T} = (t_{ij})$ definiert, die die Ausgangsbelegung der zur testenden Schaltung für eine gegebene Testmustermenge darstellt.

Definition 2.4 (Testantwortmatrix)**.** *Gegeben sei eine Testmustermenge $\mathscr{T}_M$ mit k Testmustern und eine Schaltung mit m Prüfpfaden der Länge l. Eine Testantwort $\boldsymbol{T}_i$ sei eine $m \times l$ Matrix, welche die Antwort der Schaltung auf das i-te Testmuster repräsentiert. Die $m \times n$ Testantwortmatrix $\boldsymbol{T} = (\boldsymbol{T}_{k-1}, \ldots, , \boldsymbol{T}_i, \ldots, \boldsymbol{T}_0)$ ergibt sich durch die Konkatenation der k Testantworten. Die Anzahl der Spalten ergibt sich somit zu $n = k \cdot l$.*

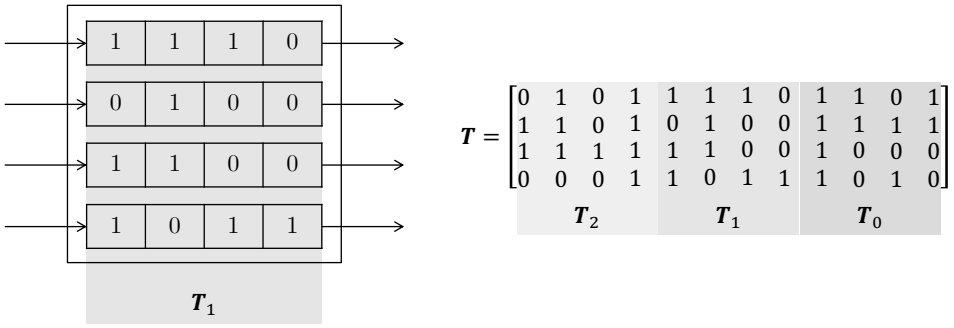


Abbildung 2.12: Beispiel einer Testantwortmatrix

Zur Veranschaulichung des Aufbaus der Testantwortmatrix ist in Abbildung 2.12 ein Beispiel für eine Testantwortmatrix mit drei Testantworten gegeben. Links in der Abbildung ist eine Schaltung mit vier Prüfpfaden mit je vier Elementen zu sehen. In dem gezeigten Beispiel enthalten die Prüfpfade der Schaltung die Testantwort auf das zweite

Testmuster $\boldsymbol{T}_1$. Um die Testantwort verarbeiten zu können, muss sie aus der Schaltung geschoben werden. Spaltenweise wird so jede Testantwort $\boldsymbol{T}_i$ aus der Schaltung in die Testantwortmatrix hineingeschoben. Wie rechts in der Abbildung zu sehen ist, wird die Testantwortmatrix so von rechts nach links mit den Testantworten gefüllt, beginnend mit der Ältesten.

Mithilfe der Testantwortmatrix $\boldsymbol{T}$ lässt sich die Testantwortkompaktierung wie folgt definieren.

Definition 2.5 (Testantwortkompaktierung)**.** *Unter einer Testantwortkompaktierung versteht man die Transformation der als $m \times n$ Matrix dargestellten Testantworten $\boldsymbol{T} = (t_{ij})$ in eine $p \times q$ Matrix der Testsignaturen $\boldsymbol{S} = (s_{ij})$ mithilfe der Transformationsfunktion Ψ, sodass $\boldsymbol{S} = \Psi(\boldsymbol{T})$ gilt, mit $t_{ij}, s_{ij} \in \{0, 1, X\}$.*

Die in der Definition 2.5 eingeführten Dimensionen der Testantwortmatrix $\boldsymbol{T}$ und der Signaturmatrix $\boldsymbol{S}$ werden entlang der Spaltenindizes $1, \ldots, n$ als zeitliche Dimension interpretiert und entlang der Zeilenindizes $1, \ldots, m$ als räumliche Dimension, welche die Ausgänge der zu testenden Schaltung, respektive die Ausgänge des Kompaktierers, repräsentieren. Durch die Konkatenation aller Testantworten, respektive Signaturen, kann mithilfe der Testantwortmatrix oder der Signaturmatrix die Ausgabe eines gesamten Tests dargestellt werden.

Insbesondere kann die Testantwortkompaktierung in räumliche und zeitliche Kompaktierungsverfahren eingeteilt werden. Wenn nach der Transformation $p < m$ gilt, spricht man von einer räumlichen Kompaktierung, da die Anzahl der Zeilen reduziert wurde. Gilt $q < n$, spricht man von einer zeitlichen Kompaktierung, da die Anzahl der Spalten reduziert wurde. Gilt sowohl $p < m$ als auch $q < n$, fanden beide Kompaktierungsarten simultan statt. In den folgenden beiden Abschnitten wird auf die Grundlagen der räumlichen und zeitlichen Kompaktierung eingegangen. Wenn nicht anders gekennzeichnet, basieren die Angaben dieses Kapitels auf [Wang2006], [Wang2008a] und [Bushnell2000].

2.4.2.1 Räumliche Kompaktierung

Betrachten wir zunächst ein Beispiel für eine räumliche Kompaktierung. In Abbildung 2.13 ist ein räumlicher Kompaktierer zu sehen, der aus zwei XOR-Bäumen besteht. Der Kompaktierer verfügt über acht Eingänge und zwei Ausgänge, was einer Kompaktierungsrate von vier entspricht. Die Kompaktierungsrate ist dabei wie folgt definiert.

Definition 2.6 (Kompaktierungsrate KR)**.** *Sei $\boldsymbol{T}$ eine $m \times n$ Testantwortmatrix und $\boldsymbol{S}$ eine $p \times q$ Testsignaturenmatrix, dann ist die Kompaktierungsrate KR wie folgt definiert.*

$$KR = \frac{m \cdot n}{p \cdot q} \tag{2.5}$$

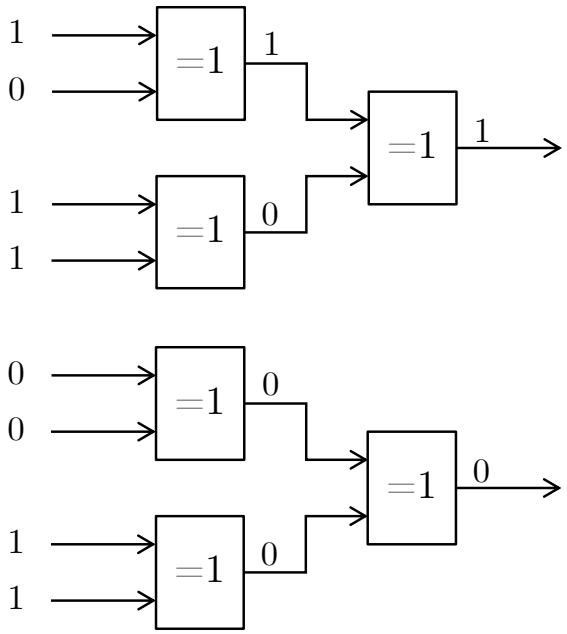


Abbildung 2.13: Beispiel einer linearen Kompaktierung mithilfe zweier XOR-Bäume.

In dem gezeigten Beispiel werden die oberen vier Eingänge des Kompaktierers vom ersten XOR-Baum zusammengefasst und die unteren vier Eingänge vom zweiten. Dies lässt sich mithilfe der Kompaktormatrix $\boldsymbol{C} = (c_{ij})$ wie folgt beschreiben: jeder Eintrag in c_{ij} der Kompaktormatrix ist genau dann 1, wenn der Ausgang j vom Eingang i abhängt. Ansonsten ist der Eintrag $c_{ij} = 0$. Wenn, wie in dem Beispiel in Abbildung 2.13 zu sehen ist, der zweite Ausgang unter anderem vom fünften Eingang abhängt, dann muss der Eintrag c_{52} in der zweiten Spalte und der fünften Zeile gleich 1 sein. Die Kompaktormatrix des Beispiels ergibt sich somit zu

$$\boldsymbol{C} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}. \tag{2.6}$$

Mithilfe der Kompaktormatrix $\boldsymbol{C}$ lässt sich die in Definition 2.5 eingeführte Transformationsfunktion Ψ mithilfe einer Matrixmultiplikation darstellen. Die Signaturmatrix $\boldsymbol{S}$ lässt sich für lineare Kompaktierer unter Verwendung von Modulo-2-Arithmetik [Nocker2004], wie folgt bestimmen.

$$\boldsymbol{S} = \boldsymbol{C}^T \cdot \boldsymbol{T} \tag{2.7}$$

Für das Beispiel in Abbildung 2.13 ergibt sich die Signaturmatrix $\boldsymbol{S}$ zu

$$\begin{aligned} \boldsymbol{S} &= \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}^T \\ &= \begin{bmatrix} 1 \cdot 1 \oplus 1 \cdot 0 \oplus 1 \cdot 1 \oplus 1 \cdot 1 \oplus 0 \cdot 0 \oplus 0 \cdot 0 \oplus 0 \cdot 1 \oplus 0 \cdot 1 \\ 0 \cdot 1 \oplus 0 \cdot 0 \oplus 0 \cdot 1 \oplus 0 \cdot 1 \oplus 1 \cdot 0 \oplus 1 \cdot 0 \oplus 1 \cdot 1 \oplus 1 \cdot 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{aligned} \tag{2.8}$$

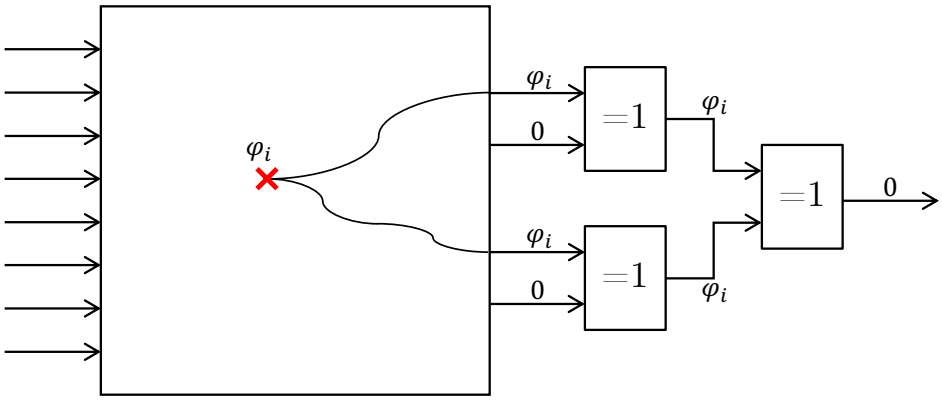


Abbildung 2.14: Beispiel einer Fehlermaskierung.

Während der Kompaktierung kann es zur sogenannten Fehlermaskierung kommen. Diese tritt auf, wenn während der Kompaktierung an beiden Eingängen eines XOR-Gatters ein Fehlverhalten auftritt. Das XOR-Gatter liefert somit im fehlerhaften und im fehlerfreien Fall dieselbe Ausgabe, wodurch der Fehler nach der Kompaktierung nicht mehr sichtbar ist.

Das Beispiel in Abbildung 2.14 zeigt eine zu testende Schaltung mit acht Eingängen und vier Ausgängen, in welcher der Fehler φ_i aufgetreten ist. In diesem Beispiel wurde der

Fehler φ_i mithilfe eines Testmusters aktiviert und zum ersten und dritten Ausgang der Schaltung propagiert. Die Testantwort der Schaltung wird mithilfe eines XOR-Baumes aus drei XOR-Gattern zu einem Signatur-Bit kompaktiert. Die XOR-Verknüpfung des ersten und des zweiten Ausgangs hat, genau wie die Verknüpfung des dritten und vierten Ausgangs, unabhängig von der Ausgangsbelegung des zweiten und vierten Ausgangs, das Fehlverhalten des Fehlers φ_i als Ergebnis. Das Ergebnis des folgenden XOR-Gatters ist damit sowohl im fehlerhaften als auch im fehlerfreien Fall 0. Der Fehler kann somit nicht mit dem gegebenen Testmuster erkannt werden. Um eine möglichst hohe Fehlerabdeckung zu erreichen, ist es daher wichtig, beim Entwurf des Kompaktierers darauf zu achten, die Fehlermaskierung zu minimieren.

Eine weitere große Herausforderung bei der Testantwortkompaktierung ist die Verarbeitung von unbekannten Werten, den sogenannten X-Werten, die während der Simulation der Schaltung auftreten, wenn der tatsächliche Signalpegel während der Simulation noch nicht bekannt ist. Diese können aufgrund von verschiedenen Ursachen auftreten, wie z. B. durch potentialfreien Bussignalen, uninitialisierten Speicherelementen oder durch die Übertaktung der zu testenden Schaltung während des Hochgeschwindigkeitstests.

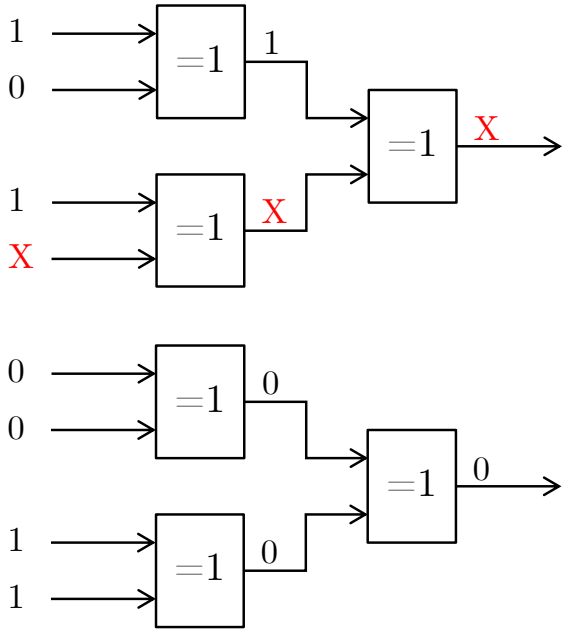


Abbildung 2.15: Beispiel einer linearen Kompaktierung mit einem unbekannten Wert.

Die Abbildung 2.15 zeigt den aus dem obigen Beispiel bekannten Kompaktierer. Die Testantwort weist jedoch diesmal einen unbekannten Wert an dem vierten Eingang des Kompaktierers auf und ist mit einem roten X markiert. Dieser einzelne unbekannte Wert führt dazu, dass auch der erste Ausgang des Kompaktierers einen unbekannten Wert annimmt, was dazu führt, dass ein Fehlverhalten, das an den ersten drei Eingängen des Kompaktierers auftritt, nicht detektiert werden kann.

2.4.2.2 Zeitliche Kompaktierung

Bei der zeitlichen Kompaktierung werden mithilfe von sequenzieller Logik aufeinanderfolgende Testantworten zu einer geringeren Anzahl an Signaturen kompaktiert. Wie in Definition 2.5 eingeführt, wird hier mithilfe der Transformationsfunktion Ψ die Anzahl der Spalten q in der Signaturmatrix $\boldsymbol{S}$ reduziert. Hierzu arbeitet die Transformationsfunktion Ψ die Testantwortmatrix spaltenweise ab.

Im folgenden Abschnitt wird das Multiple Input Signature Register (MISR) als eines der weitverbreitetsten Verfahren zur zeitlichen Kompaktierung vorgestellt.

Multiple Input Signature Register (MISR) Das Signaturregister mit mehreren Eingängen (engl. Multiple-Input Signature Register, MISR) ist ein linear rückgekoppeltes Schieberegister mit n_c Eingängen und n_c Flipflops, welches durch das charakteristische Polynom $f(X)$ (2.9) mit dem Grad n_c dargestellt werden kann.

$$f(X) = X^{n_c} + h_{n_c-1}X^{n_c-1} + \cdots + h_2X^2 + h_1X + 1 \tag{2.9}$$

Die Koeffizienten des charakteristischen Polynoms $h_1, \ldots, h_{n_c-1}$ können die Werte 0 und 1 annehmen und stellen die Rückkopplung des Schieberegisters dar. Wie in Abbildung 2.16 mithilfe von UND-Gattern dargestellt wird, besteht eine Rückkopplung zum i-ten Bit des Schieberegisters, wenn der Koeffizient $h_i = 1$ ist. Der Zustand des Schieberegisters wird durch den Inhalt der Flipflops bestimmt und kann mithilfe des Zustandsvektors $\boldsymbol{Z}$ repräsentiert werden. Wird das MISR für die zeitliche Kompaktierung genutzt, werden die Eingänge des Schieberegisters mit den Ausgängen der zu testenden Schaltung verbunden. Mit jedem Takt wird somit eine Spalte der Testantwortmatrix $\boldsymbol{T}$, beginnend mit der rechten, in das MISR eingeschoben. Zum Zeitpunkt t liegt somit die $(n-t)$-te Spalte der Testantwortmatrix $\boldsymbol{T}^{[n-t]}$ am MISR an, wobei n die Anzahl der Spalten der Testantwortmatrix angibt. Die einzelnen Bits des Eingangsvektors $\boldsymbol{T}^{[n-t]}$ werden mithilfe von XOR-Gattern mit den entsprechenden Flipflops des MISRs verknüpft.

Zur Berechnung des Zustandsvektors $\boldsymbol{Z}(t)$ zum Zeitpunkt t kann das MISR alternativ mithilfe einer Zustandsübergangsmatrix $\boldsymbol{R}$ beschrieben werden, welche die Zustandsübergänge eines Schiebezyklus beschreibt.

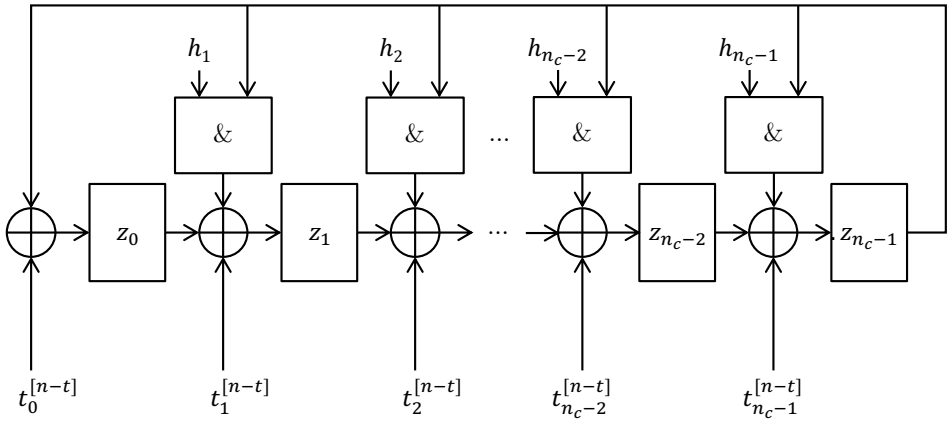


Abbildung 2.16: Signaturregister mit mehreren Eingängen (MISR).

Der Zustand des MISRs zum Zeitpunkt $t+1$ lässt sich mithilfe der Gleichung (2.10) berechnen.

$$\boldsymbol{Z}(t+1) = \boldsymbol{R} \cdot \boldsymbol{Z}(t) + \boldsymbol{T}^{[n-t]} \tag{2.10}$$

Wie in (2.11) gezeigt wird, repräsentiert $\boldsymbol{R}$ die Übergänge des MISRs. Die erste Zeile der Zustandsübergangsmatrix repräsentiert die Rückkopplung des letzten MISR-Bits zum ersten MISR-Bit und ist daher an allen Stellen, außer an der letzten Stelle, 0. Die restlichen $n_c - 1$ Zeilen der letzten Spalte werden entsprechend der linearen Rückkopplungen $h_1, ..., h_{n_c-1}$ aufgefüllt. Besteht eine Rückkopplung, gilt $h_i = 1$, sonst $h_i = 0$. Um das Schieben innerhalb des Schieberegisters zu beschreiben, wird die untere Nebendiagonale der Zustandsübergangsmatrix mit Einsen gefüllt und die verbleibenden Einträge auf 0 gesetzt.

$$\begin{bmatrix} z_0(t+1) \\ z_1(t+1) \\ z_2(t+1) \\ \vdots \\ z_{n_c-2}(t+1) \\ z_{n_c-1}(t+1) \end{bmatrix} = \begin{bmatrix} 0 & 0 & \dots & \dots & 0 & 1 \\ 1 & 0 & \dots & \dots & 0 & h_1 \\ 0 & 1 & \dots & \dots & 0 & h_2 \\ \vdots & \dots & \ddots & \dots & \vdots & \vdots \\ 0 & 0 & \dots & \ddots & 0 & h_{n_c-2} \\ 0 & 0 & \dots & \dots & 1 & h_{n_c-1} \end{bmatrix} \cdot \begin{bmatrix} z_0(t) \\ z_1(t) \\ z_2(t) \\ \vdots \\ z_{n_c-2}(t) \\ z_{n_c-1}(t) \end{bmatrix} + \begin{bmatrix} t_0^{[n-t]} \\ t_1^{[n-t]} \\ t_2^{[n-t]} \\ \vdots \\ t_{n_c-2}^{[n-t]} \\ t_{n_c-1}^{[n-t]} \end{bmatrix} \tag{2.11}$$

Mit dieser Gleichung lässt sich die Ausbreitung des Datenwortes $\boldsymbol{T}^{[n-t]}$ berechnen, wenn der anfängliche Statusvektor $\boldsymbol{Z}(0)$ bekannt ist. Dies lässt sich in einem Beispiel zeigen, indem der Zustandsvektor nach zwei Verschiebungen berechnet wird.

Für dieses Beispiel gelte ohne Einschränkung der Allgemeinheit $\boldsymbol{Z}(0) = \mathbf{0}$. Die Zustandsvektoren $\boldsymbol{Z}(1)$ und $\boldsymbol{Z}(2)$ nach einem, respektive zwei Taktzyklen können wie folgt bestimmt werden.

$$\boldsymbol{Z}(1) = \boldsymbol{T}^{[n]} \tag{2.12}$$

$$\begin{aligned} \boldsymbol{Z}(2) &= \boldsymbol{R} \cdot \boldsymbol{Z}(1) + \boldsymbol{T}^{[n-1]} \\ &= \boldsymbol{R} \cdot \boldsymbol{T}^{[n]} + \boldsymbol{T}^{[n-1]} \end{aligned} \tag{2.13}$$

Bereits nach zwei Verschiebungen zeigt sich, dass die Propagierung der Eingabedaten nur vom Anfangszustand des MISRs und den Eingabedaten abhängt. Mithilfe von vollständiger Induktion kann gezeigt werden, dass der Zustandsvektor nach $t+1$ Taktzyklen wie folgt berechnet werden kann.

$$\boldsymbol{Z}(t) = \boldsymbol{R}^{t-1} \cdot \boldsymbol{T}^{[n]} + \boldsymbol{R}^{t-2} \cdot \boldsymbol{T}^{[n-1]} + \cdots + \boldsymbol{T}^{[n-t-1]} \tag{2.14}$$

Diese Gleichung lässt sich in (2.15) zusammenfassen.

$$\boldsymbol{Z}(t+1) = \sum_{i=0}^{t} \boldsymbol{R}^{t-i} \cdot \boldsymbol{T}^{[n-i]} \tag{2.15}$$

Mit (2.15) haben wir dann eine vereinfachte Gleichung zur Berechnung der Ausbreitung der Daten im MISR.

Mithilfe eines solchen MISRs ist es möglich, sehr hohe Kompaktierungsraten zu erreichen, solange keine unbekannten Werte das MISR erreichen. Wie in (2.15) zu erkennen ist, werden bereits einzelne X-Werte im Eingangsvektor $\boldsymbol{T}^{[n-t]}$ mit der Zeit über die Flipflops des MISRs verteilt, was dazu führt, dass der gesamte Zustand des MISRs unbekannt wird und somit nicht mit der Referenzantwort verglichen werden kann.

Auch für die zeitliche Kompaktierung von X-behafteten Testantworten finden sich Lösungsansätze in der Literatur, wie zum Beispiel das in [Touba2007] vorgestellte X-Canceling MISR. Beim X-Canceling MISR wird ausgenutzt, dass die X-Werte unbekannt

aber fest sind und symbolisch verarbeitet werden können. Zwei gleiche X-Symbole heben sich somit auf, wenn sie während der Berechnung der Signatur miteinander XOR-verknüpft werden. Durch die lineare Kombination der MISR-Bits ist es im Allgemeinen möglich, unbekannte Werte so zu kombinieren, dass deren Effekt auf die Signatur aufgehoben wird. Wenn q X-freie Linearkombinationen der MISR-Bits gebildet werden, können $(1 - 2^{-q}) \cdot 100\%$ der Fehler, die das X-Canceling MISR erreichen, auch nach der Kompaktierung noch erkannt werden. Werden z. B. $q = 8$ X-freie Kombinationen betrachtet, kann die Fehlerdurchlässigkeit mit 99,61 % abgeschätzt werden.

Erreichen zu viele X-Werte das X-Canceling MISR, ist es nicht mehr möglich, genügend X-freie Kombinationen der MISR-Bits zu bilden und die Fehlerinformation in der Signatur geht verloren. Um dies zu verhindern, muss das X-Canceling MISR zurückgesetzt und die Zwischensignatur gespeichert werden, sobald nicht mehr genügend X-freie Kombinationen gebildet werden können. Die Anzahl der nötigen Zwischensignaturen n_z, die gespeichert werden müssen, wenn ein m-Bit MISR verwendet wird und mindesten q X-freie Kombinationen der MISR-Bits als Signatur verwendet werden sollen, kann mit (2.16) abgeschätzt werden. Dabei ist X_{Total} die Anzahl der X-Werte, die das X-Canceling MISR während des gesamten Tests erreichen. [Touba2007]

$$n_z = \frac{X_{Total}}{m - q} \tag{2.16}$$

Auch im Hochgeschwindigkeitstest, der für den Test von kleinen Verzögerungsfehlern eingesetzt wird, wird ein X-Canceling MISR zur zeitlichen Kompaktierung verwendet. Der Hochgeschwindigkeitstest wird im folgenden Abschnitt erläutert.

2.4.3 Hochgeschwindigkeitstest

Während für den Test von kleinen Verzögerungsfehlern ATPG-Algorithmen existieren, die versuchen die kleinen Verzögerungsfehler über möglichst lange Pfade an die Ausgänge zu propagieren [Sauer2013; Eggersglüß2012], ist es für versteckte kleine Verzögerungsfehler, wie in Kapitel 2.3.3 gezeigt, nicht möglich, das Fehlverhalten über einen langen Pfad an einen Ausgang der Schaltung zu propagieren. Zur Detektion von versteckten kleinen Verzögerungsfehlern muss daher der konventionelle Verzögerungstest zum Hochgeschwindigkeitstest erweitert werden. [Yan2003; Ahmed2006; Lee2008; Hellebrand2014; Kampmann2020]

Während des Hochgeschwindigkeitstests wird die Schaltung unter erhöhter Taktfrequenz getestet. Dadurch kann die Schaltung zu einem früheren Zeitpunkt beobachtet werden, als es mit der nominellen Taktfrequenz möglich wäre.

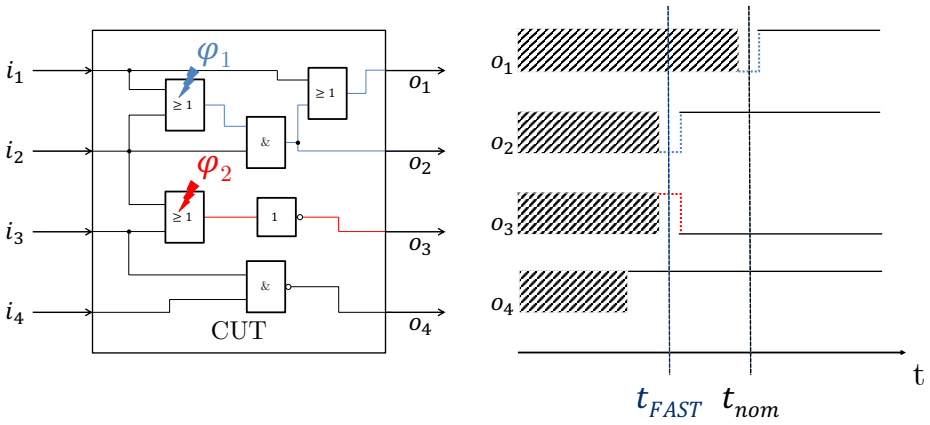


Abbildung 2.17: Detektion eines versteckten kleinen Verzögerungsfehlers (HDF) mithilfe eines Hochgeschwindigkeitstests [Sprenger2019].

In Abbildung 2.17 ist ein Beispiel für die Anwendung eines Hochgeschwindigkeitstests gegeben. In der Abbildung ist eine kombinatorische Schaltung (links) mit vier Ein- und Ausgängen gezeigt. Rechts sind die Signalverläufe der vier Ausgänge zu sehen. Wenn der kleine Verzögerungsfehler φ_1 auftritt, kann das Fehlverhalten an die Ausgänge o_1 und o_2 propagiert werden. In diesem Beispiel wird der Fehler über einen langen Pfad an den Ausgang o_1 propagiert, sodass der Fehler während der nominellen Beobachtungszeit $t_{nom} = 1/f_{nom}$ sichtbar ist und somit in einem normalen Verzögerungstest detektiert werden kann. Der Fehler φ_2 hingegen kann nur über einen kurzen Pfad an den Ausgang o_3 propagiert werden, wodurch der Fehler zur nominellen Beobachtungszeit nicht erkannt werden kann. Der versteckte kleine Verzögerungsfehler kann im Hochgeschwindigkeitstest erkannt werden, indem eine erhöhte Taktfrequenz f_{FAST} verwendet wird. Zur Beobachtungszeit $t_{FAST} = 1/f_{FAST}$ wird der Fehler φ_2 sichtbar. Aufgrund der erhöhten Taktfrequenz muss jedoch der Ausgang o_1 als unbekannt angenommen werden, da das Signal noch keinen stabilen Wert angenommen hat, was durch die schraffierten Flächen angedeutet wird. Um die Testdauer möglichst gering zu halten, ist es wichtig möglichst wenige Taktfrequenzen zu verwenden und gleichzeitig viele versteckte kleine Verzögerungsfehler abzudecken. Arbeiten zur optimalen Frequenzauswahl sind in [Hellebrand2015; Kampmann2015; Liu2018; Kampmann2020] zu finden.

Wie in [Hellebrand2014] vorgestellt, wird zur Verarbeitung der Testantworten die in Abbildung 2.18 gezeigte Architektur verwendet. Mithilfe eines Testmustergenerators (TMG) werden Testmuster an die zu testende Schaltung angelegt. Die Testantworten durchlaufen anschließend ein zweistufiges Kompaktierungsverfahren. Dabei werden die Testantworten zunächst räumlich kompaktiert, um anschließend eine zeitliche Kom-

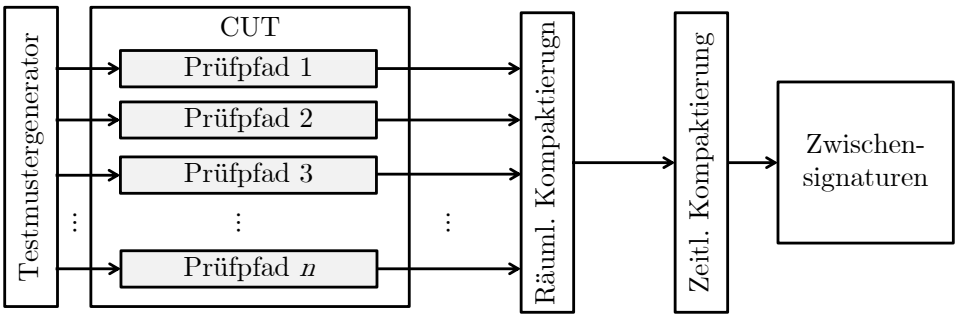


Abbildung 2.18: Testarchitektur für den Hochgeschwindigkeitstest [Sprenger2019].

paktierung mithilfe eines X-Canceling MISRs zu durchlaufen. Die bei der zeitlichen Kompaktierung generierten Zwischensignaturen werden in einem kleinen Speicher zur weiteren Verarbeitung gespeichert. Zur Verarbeitung von variierenden X-Raten, wird in Kapitel 4 ein Verfahren zur X-toleranten räumlichen Kompaktierung entwickelt.

3 Einführung in künstliche neuronale Netze

Künstliche neuronale Netze sind eine bekannte Methode des maschinellen Lernens und erhalten aufgrund von ihrer Vergleichbarkeit mit dem menschlichen Gehirn eine große Aufmerksamkeit. Insbesondere durch die immense Steigerung der Rechenleistung von Computersystemen in den letzten Jahrzehnten finden künstliche neuronale Netze Anwendung in nahezu allen Forschungsbereichen und in einer Vielzahl kommerzieller Produkte. Das erste künstliche neuronale Netz wurde dabei schon 1943 von McCulloch und Pitts [McCulloch1943] vorgestellt.

In diesem Kapitel wird eine kurze Einführung in die in dieser Arbeit verwendeten künstlichen neuronalen Netze gegeben. Eine umfassende Einführung in das Thema des maschinellen Lernens ist unter anderem in [Bishop2016; Duda2012; Géron2020; Abu-Mostafa2012] zu finden, auf denen dieses Kapitel basiert, falls nicht anders gekennzeichnet.

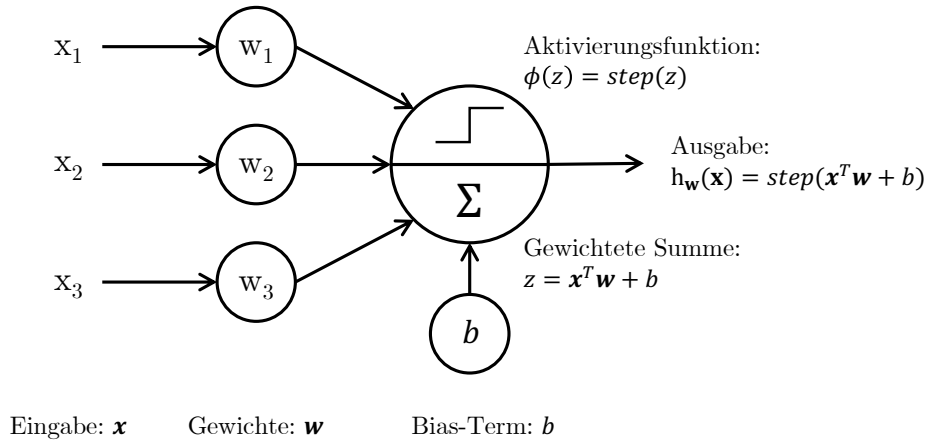


Abbildung 3.1: Beispiel eines künstlichen Neurons.

Ein künstliches neuronales Netz ist aus einer Vielzahl einzelner künstlicher Neuronen aufgebaut. Ein Beispiel eines künstlichen Neurons mit drei Eingängen ist in der Abbildung 3.1 dargestellt. Um die Ausgabe des künstlichen Neurons zu bestimmen, wird

zunächst die gewichtete Summe der Eingänge gebildet. Mithilfe des Eingangsvektors $\boldsymbol{x}$, des Vektors $\boldsymbol{w}$, der die einzelnen Gewichte des Neurons repräsentiert und des Bias-Terms b lässt sich die gewichtete Summe wie folgt berechnen.

$$z = \boldsymbol{x}^T \boldsymbol{w} + b \tag{3.1}$$

Jeder Eingangswert x_i wird hierzu mit dem zugehörigen Gewicht w_i multipliziert und die Ergebnisse werden aufsummiert. Auf die gewichtete Summe der Eingänge wendet das künstliche Neuron anschließend die Aktivierungsfunktion $\phi(z)$ an. In dem hier gezeigten Beispiel wird die Stufenfunktion als Aktivierungsfunktion verwendet $\phi(z) = \mathit{step}(z)$. Wobei die Stufenfunktion $\mathit{step}(x)$ wie folgt definiert ist.

$$\mathit{step}(x) = \begin{cases} 0, \text{ wenn } x < 0 \\ 1, \text{ wenn } x \geq 0 \end{cases} \tag{3.2}$$

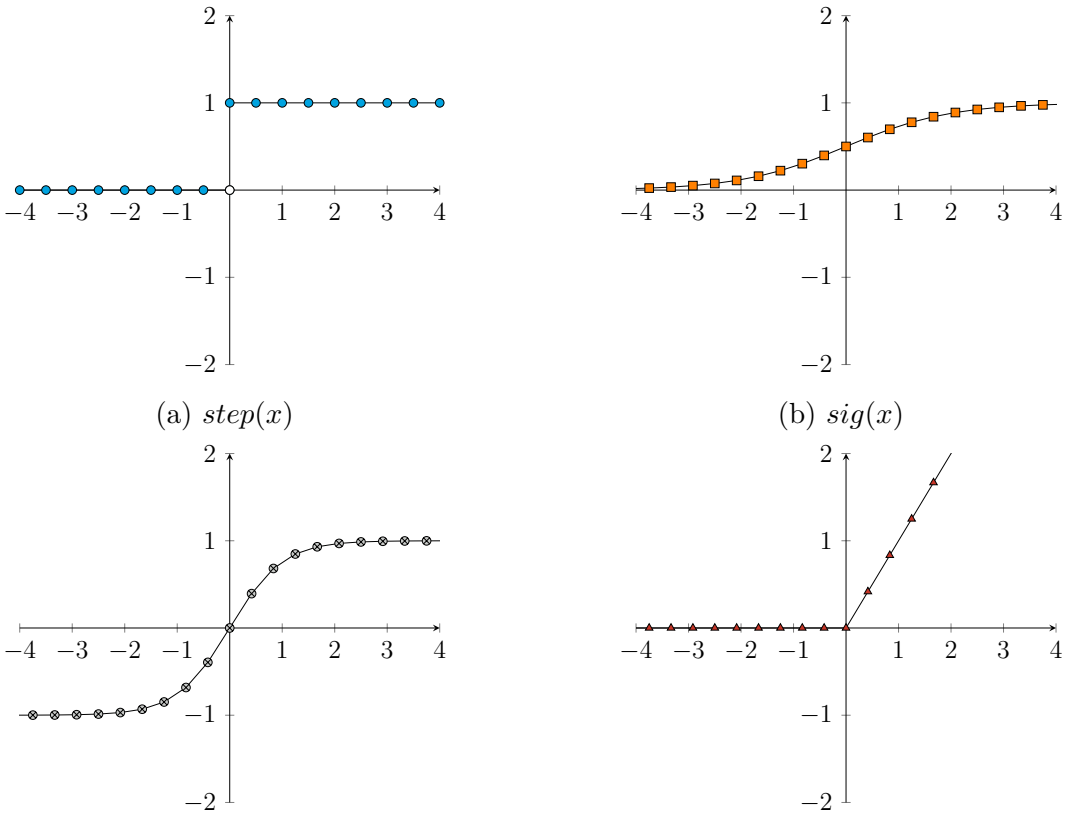
Als Aktivierungsfunktion können jedoch beliebige Funktionen verwendet werden. In Anlehnung an das biologische Vorbild, das Gehirn, wurden häufig Funktionen verwendet, die das Verhalten von biologischen Neuronen imitieren, wie z. B. die Sigmoidfunktion $\mathit{sig}(x)$ (vgl. (3.3)) oder der Tangens hyperbolicus $\mathit{tanh}(x)$ (vgl. (3.4)). Aktuelle Forschungsarbeiten zeigen jedoch, dass diese Funktionen nicht zwangsläufig zu den besten Ergebnissen führen. Daher finden vermehrt Funktionen wie die ReLU-Funktion (engl. Rectified Linear Unit, ReLU) (vgl. (3.5)) Verwendung als Aktivierungsfunktion.

$$\mathit{sig}(x) = \frac{1}{1 + e^{-x}} \tag{3.3}$$

$$\mathit{tanh}(x) = 1 - \frac{2}{e^{2x} + 1} \tag{3.4}$$

$$\mathit{relu}(x) = \mathit{max}(0, x) \tag{3.5}$$

Die Abbildung 3.2 zeigt den Verlauf der Stufenfunktion $step(x)$, der Sigmoidfunktion $sig(x)$, des Tangens hyperbolicus $tanh(x)$ und der ReLU-Funktion $relu(x)$ für $x \in [-4, 4]$. Wie in der Abbildung zu sehen ist, sind die Wertebereiche der Sigmoidfunktion und des Tangens hyperbolicus auf $[0, 1]$, respektive $[1, 1]$ beschränkt und die Funktionen sind stetig differenzierbar. Während der Wertebereich der Stufenfunktion ebenfalls beschränkt ist, ist die Funktion nicht stetig differenzierbar. Der Wertebereich der ReLU-Funktion ist hingegen nur einseitig beschränkt und die Funktion ist ebenfalls nicht stetig differenzierbar.



(a) $step(x)$ (b) $sig(x)$

(c) $tanh(x)$ (d) $relu(x)$

Abbildung 3.2: Verlauf der Aktivierungsfunktionen $step(x)$, $sig(x)$, $tanh(x)$ und $relu(x)$.

Die Ausgabe $h_{\boldsymbol{w}}(\boldsymbol{x})$ des in Abbildung 3.1 gezeigten künstlichen Neurons in Abhängigkeit von $\boldsymbol{w}$ und $\boldsymbol{x}$ lässt sich mithilfe der Aktivierungsfunktion $\phi(z)$ wie folgt bestimmen.

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \phi(z) = step(\boldsymbol{x}^T \boldsymbol{w} + b) \tag{3.6}$$

Mithilfe der vorgestellten künstlichen Neuronen lassen sich, wie in Abbildung 3.3 gezeigt, einfache künstliche neuronale Netze aufbauen. Das hier gezeigte einschichtige künstliche neuronale Netz wird auch Perzeptron genannt und besteht im Allgemeinen aus einem oder mehreren künstlichen Neuronen, wobei alle Eingänge mit allen Neuronen verbunden sind. Zur besseren Veranschaulichung wurde auf die Darstellung des Bias-Terms b verzichtet. Mithilfe eines solchen Perzeptrons ist es bereits möglich linear separierbare Klassifizierungsprobleme, wie z. B. das des logischen UND-Operators, zu lösen.

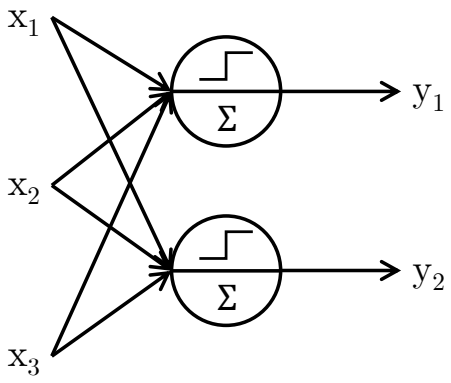


Abbildung 3.3: Beispiel eines Perzeptrons.

Ein Klassifizierungsproblem wird linear separierbar genannt, wenn es mithilfe einer linearen Hyperebene möglich ist, die Punkte eines Raumes in ihre zugehörigen Klassen einzuteilen. Im Beispiel des in Abbildung 3.4 zu sehenden zweidimensionalen Raumes ist es beispielsweise möglich, die Klasse der orangenen Punkte und die Klasse der blauen Rauten mithilfe einer Geraden zu separieren.

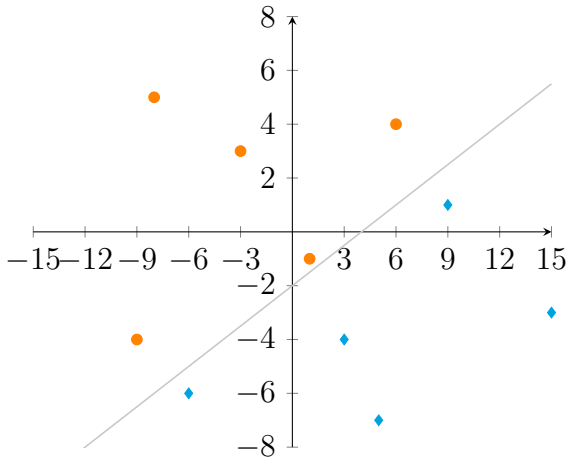


Abbildung 3.4: Linear separierbares Klassifizierungsproblem.

Um auch nicht linear separierbare Klassifizierungsprobleme lösen zu können, muss das Perzeptron zu einem mehrschichtigen Perzeptron (MLP) erweitert werden. Bei einem mehrschichtigen Perzeptron handelt es sich um ein vollständig verbundenes, vorwärts

gerichtetes, künstliches neuronales Netz mit mindestens einer versteckten Ebene. Als versteckte Ebenen werden dabei die Ebenen bezeichnet, die zwischen Eingabe- und Ausgabe-Ebene liegen. Abbildung 3.5 zeigt ein mehrschichtiges Perzeptron mit einer Eingabe-Ebene, die die Eingabewerte aufnimmt, zwei versteckten Ebenen und einer Ausgabe-Ebene. Häufig wird bei der Verwendung eines mehrschichtigen Perzeptrons mit mehr als zwei versteckten Ebenen auch von einem tiefen künstlichen neuronalen Netz (engl. Deep Neural Network, DNN) gesprochen.

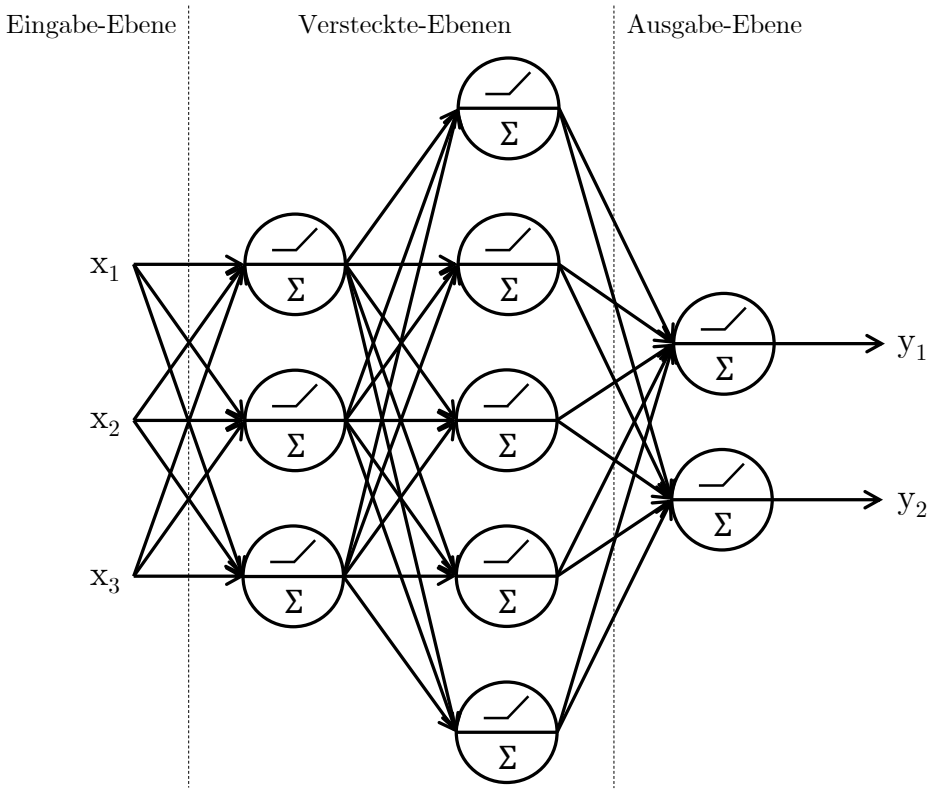


Abbildung 3.5: Künstliches neuronales Netz mit zwei versteckten Ebenen.

Die Herausforderung ist nun die einzelnen Gewichte so auszuwählen, dass das künstliche neuronale Netz das gewünschte Verhalten aufweist, wie z. B. die Klassifizierung der Eingabedaten in unterschiedliche Klassen. Hierzu muss das mehrschichtige Perzeptron zunächst mithilfe eines überwachten Lernverfahrens trainiert werden. Ein überwachtes Lernverfahren zeichnet sich dadurch aus, dass ein Datensatz zur Verfügung steht, in dem jedem Datenpunkt ein sogenanntes Label zugewiesen ist, welches der gewünschten Antwort des künstlichen neuronalen Netzes entspricht, wenn der Datenpunkt an die Eingänge des Netzes angelegt wird. Hierzu wird in der Regel eine Variante des Backpropagation-Algorithmus [Rumelhart1987] verwendet, der auf einem Gradientenverfahren basiert. Zu Beginn des Backpropagation-Algorithmus werden zunächst alle Gewichte des mehrschichtigen Perzeptrons mit zufälligen Werten initialisiert. Anschließend wird in einem Vorwärtsdurchlauf die Vorhersage des mehrschichtigen Perzeptrons für einen Teil des Trainingsdatensatzes bestimmt.

Mithilfe der erwarteten Antworten (Label) kann daraufhin eine Fehlerfunktion bestimmt werden. Als Fehlerfunktion $E(\boldsymbol{w})$ kann z. B. die Summe der quadratischen Fehler zwischen den Labeln $\boldsymbol{l}$ und der tatsächlichen Antworten $\boldsymbol{y}$ auf n Eingabevektoren eines Trainingsdatensatzes, wie in (3.7) definiert, verwendet werden. Der Vektor $\boldsymbol{l}_i$ gibt dabei die aus dem Trainingsdatensatz bekannte Antwort auf den i-ten Eingabevektor an, während der Vektor $\boldsymbol{y}_i$ die Antwort des neuronalen Netzes auf den i-ten Eingabevektor für die im Vektor $\boldsymbol{w}$ zusammengefassten verwendeten Gewichte darstellt.

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{n}(\boldsymbol{l}_i - \boldsymbol{y}_i) = \frac{1}{2}||\boldsymbol{l} - \boldsymbol{y}||^2 \tag{3.7}$$

Im folgenden Schritt wird für jede Schicht, angefangen mit der Ausgabeschicht, der Beitrag jeder Verbindung zu dem zuvor berechneten Fehler berechnet und das Gewicht äquivalent zu einem Gradientenverfahren aktualisiert. So wird der Trainigsdatensatz mehrfach durchlaufen, wobei jeder Durchlauf als Epoche bezeichnet wird.

Soll ein mehrschichtiges Perzeptron zur Klassifizierung eingesetzt werden, kann die Ausgabe des künstlichen neuronalen Netzes als Einordnung in eine bestimmte Klasse interpretiert werden. So ist es z. B. mithilfe eines einzelnen künstlichen Neurons in der Ausgabeschicht möglich, eine binäre Klassifikation durchzuführen. Für eine Einteilung in eine von mehreren Klassen, die sogenannten Multiclass-Klassifizierung, wird ein Neuron je Klasse in der Ausgabeschicht benötigt. Als Aktivierungsfunktion der Ausgabeschicht sollte hier die *softmax*-Funktion verwendet werden. Diese sorgt dafür, dass die Ausgabewerte des mehrschichtigen Perzeptrons zwischen 0 und 1 liegen und die Summe der Ausgabewerte 1 ergibt. Somit lassen sich die Ausgabewerte als Wahrscheinlichkeiten interpretieren, mit der sich der aktuelle Datenpunkt in der entsprechenden Klasse befindet.

$$\begin{aligned} softmax &: \mathbb{R}^K \to [0,1]^K \\ softmax(\boldsymbol{x})_j &= \frac{e^{x_j}}{\Sigma_{k=1}^{K} e^{x_k}}, j = 1, \cdots, K \end{aligned} \tag{3.8}$$

3.1 Merkmalsextraktion

Eine Herausforderung für das maschinelle Lernen ist der sogenannte *Fluch der Dimensionen* [Bellman1961]. Hierunter versteht man das exponentielle Wachstum des Merkmalraums in Abhängigkeit der Anzahl der Merkmale (engl. features) des verwendeten Datensatzes. Unter einem Merkmal versteht man dabei eine Dimension des Eingabevektors, der die Lage des Datenpunktes in einem mehrdimensionalen Raum beschreibt. Die Anzahl der möglichen Punkte innerhalb dieses mehrdimensionalen Raums wächst exponentiell mit der Anzahl der Dimensionen.

Da für das Training eines künstlichen neuronalen Netzes in einem großen Merkmalsraum entsprechend viele Trainingsdatenpunkte notwendig sind, werden häufig Verfahren zur Reduktion der Dimensionalität oder Merkmalsextraktion eingesetzt.

Einfache Verfahren wählen die Merkmale entweder rein zufällig oder anhand von statistischen Eigenschaften des Datensatzes aus. Eine Möglichkeit zur Merkmalsextraktion ist z. B. die Auswahl anhand der Varianz der einzelnen Merkmale. Dabei werden nur die Merkmale ausgewählt, die eine Varianz oberhalb eines gegebenen Grenzwertes aufweisen. Somit werden Merkmale mit einer geringen Varianz vernachlässigt, da sie nur wenig zur Klassifizierung des Datensatzes beitragen. Eine weitere Möglichkeit ist die Auswahl der Merkmale anhand von Hypothesentests wie z. B. dem χ^2-Test. Hierbei wird je ein Merkmal und die zugehörige Klasse als Zufallsvariable modelliert und ihre statistische Unabhängigkeit geprüft. So können die Merkmale, die statistisch unabhängig von dem Ergebnis der Klassifikation sind, vernachlässigt werden.

Ein häufig verwendetes Verfahren der Merkmalsextraktion ist das Verfahren der Hauptkomponentenanalyse (engl. Principal Component Analysis, PCA), welches im Folgenden kurz erläutert wird.

3.1.1 Hauptkomponentenanalyse

Die Hauptkomponentenanalyse (engl. Principal Component Analysis, PCA) ist ein Verfahren, das häufig zur Merkmalsextraktion oder zur Visualisierung von mehrdimensionalen Datensätzen angewandt wird. Mithilfe der PCA ist es möglich, einen m-dimensionalen Raum auf einen n-dimensionalen Raum abzubilden, wobei typischerweise $n < m$ gilt. Dabei wird der m-dimensionale Raum mithilfe einer orthogonalen Projektion so auf den

n-dimensionalen Raum abgebildet, dass die Varianz der abgebildeten Daten maximal ist. Entsprechend [Bishop2016] lassen sich die n Hauptkomponenten eines m-dimensionalen Datensatzes $\boldsymbol{x}_i \in \mathbb{R}^m$ mit k Datenpunkten bestimmen, indem man die n Eigenvektoren der n größten Eigenwerte der Kovarianzmatrix der Daten $\boldsymbol{KV}$ bestimmt, wobei die Kovarianzmatrix wie folgt definiert ist.

$$\boldsymbol{KV} = \frac{1}{k}\sum_{i=1}^{k}(\boldsymbol{x}_i - \bar{\boldsymbol{x}})(\boldsymbol{x}_i - \bar{\boldsymbol{x}})^T \tag{3.9}$$

$\bar{\boldsymbol{x}}$ entspricht dabei dem Mittelwert des Datensatzes. Durch die n Eigenvektoren wird so ein neuer Merkmalraum mit n Dimensionen aufgespannt, was einer Reduktion auf n Merkmale entspricht.

3.2 Ensemble-Methoden

Zur Lösung eines Klassifizierungsproblems können durch die Variation der Architektur eines künstlichen neuronalen Netzes oder durch Variation des Lernprozesses des Netzes unterschiedliche Prädiktoren gebildet werden. Eine Menge von Prädiktoren für dasselbe Klassifizierungsproblem wird Ensemble genannt. Mithilfe von verschiedenen Ensemble-Methoden können die Prädiktoren des Ensembles zu einem neuen Prädiktor zusammengefasst werden, der eine bessere Vorhersagegenauigkeit als die einzelnen Prädiktoren des Ensembles aufweisen kann. In der Regel werden hierzu die Ergebnisse der einzelnen Prädiktoren mithilfe eines Mehrheitsentscheids zu einem neuen Ergebnis zusammengefasst.

Eine häufig verwendete Ensemble-Methode ist das sogenannte Bagging [Breiman1996], das im Folgenden kurz erläutert wird. Für die Bagging-Methode werden zunächst mehrere zufällige Stichproben aus dem Trainingsdatensatz gezogen. Wie in Abbildung 3.6 zu sehen ist, werden mithilfe dieser zufälligen Stichproben mehrere künstliche neuronale Netze trainiert. Für die Klassifizierung eines neuen Datenpunktes treffen zunächst alle trainierten künstlichen neuronalen Netze eine separate Entscheidung. Mithilfe eines Mehrheitsentscheids wird anschließend die Entscheidung für das gesamte Ensemble gebildet.

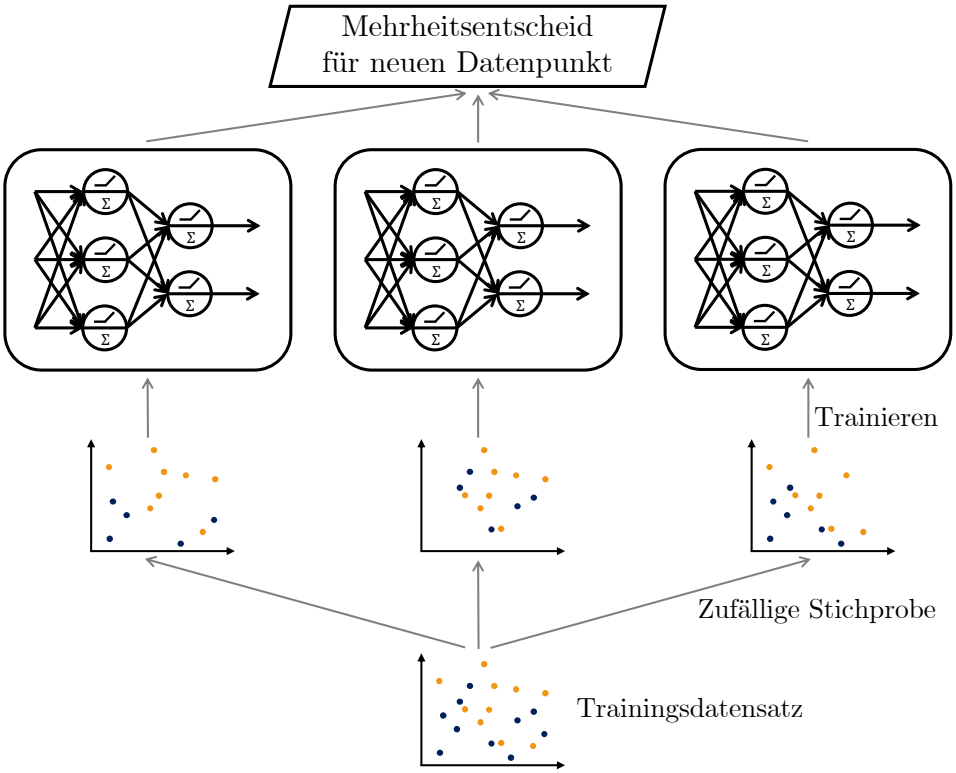
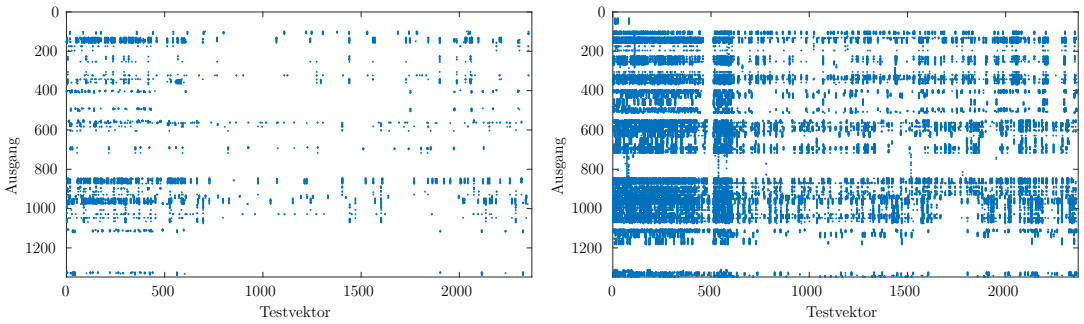


Abbildung 3.6: Schema der Bagging Ensemble-Methode [Géron2020].

Mithilfe der in den letzten beiden Kapiteln eingeführten Grundlagen, werden die Beiträge dieser Arbeit zur Bewältigung von Unsicherheiten während der gesamten Lebenszeit einer hochintegrierten Schaltung in den folgenden beiden Kapiteln eingeführt.

4 Testantwortkompaktierung für den Hochgeschwindigkeitstest

Eine der Herausforderungen für den Test auf versteckte kleine Verzögerungsfehler (engl. Hidden Delay Faults, HDFs) mithilfe des eingebauten Hochgeschwindigkeitstests stellt die variierende X-Rate im Verlauf des Tests dar. Wie in Kapitel 2.4.3 erläutert wurde, kommt es durch die Verwendung unterschiedlicher Testfrequenzen zu variierenden X-Raten während des Tests. Die variierenden X-Raten führen dazu, dass Kompaktierungsverfahren, die dem Stand der Technik entsprechen, nicht effizient verwendet werden können, da sie immer für den schlechtesten Fall ausgelegt werden müssen. In diesem Kapitel wird ein effizientes Verfahren zur räumlichen Testantwortkompaktierung für variierende X-Raten vorgestellt.



(a) Doppelte Taktfrequenz ($f_1 = 2 \cdot f_{nom}$) (b) Dreifache Taktfrequenz ($f_2 = 3 \cdot f_{nom}$)

Abbildung 4.1: X-Verteilung für b17_1 [Sprenger2019].

Abbildung 4.1 zeigt die Verteilung der X-Werte für die Schaltung *b17_1* aus der ITC'99 Benchmark-Suite [Corno2000] für zwei verkürzte Taktperioden. In Abbildung 4.1a wurde mit $T_{FAST} = 1794\,\text{ps}$ die Hälfte der nominellen Taktperiode verwendet und in Abbildung 4.1b wurde mit $T_{FAST} = 1196\,\text{ps}$ ein Drittel der nominalen Taktperiode verwendet. Auf der X-Achse wurden dabei die einzelnen Testantworten auf eine Testmenge für Verzögerungsfehler aufgeführt. Die verwendete Testmenge wurde mithilfe einer kommerziellen Software generiert. Auf der Y-Achse sind die Ausgänge der Schaltung aufgetragen. Jeder blaue Punkt in der Abbildung stellt dabei einen unbekannten Wert (X-Wert) dar.

Dabei fällt auf, dass mit sinkender Taktperiode die Anzahl der X-Werte steigt. Aber auch während der Nutzung einer einzelnen Taktperiode variiert die Anzahl der X-Werte pro Testantwort. [Sprenger2018b; Sprenger2019]

Daraus ergeben sich zwei Anforderungen an das Verfahren zur Testantwortkompaktierung im Hochgeschwindigkeitstest. Einerseits muss das Kompaktierungsverfahren flexibel auf die variierenden X-Raten bei unterschiedlichen Testfrequenzen reagieren können. Andererseits muss innerhalb einer Testfrequenz auf die schwankenden X-Raten reagiert werden können. Neben den Anforderungen zur Flexibilität des Kompaktierungsverfahrens soll die Fehlerabdeckung durch das Kompaktierungsverfahren nicht signifikant verringert werden. Zur Unterstützung des zur zeitlichen Kompaktierung verwendeten X-Canceling MISRs im eingebauten Hochgeschwindigkeitstests ist es außerdem notwendig, die X-Rate am X-Canceling MISR zu reduzieren, um die Anzahl der benötigten Zwischensignaturen zu reduzieren.

Im Laufe dieses Kapitels wird ein Verfahren zur räumlichen Testantwortkompaktierung vorgestellt, welches die oben genannten Anforderungen erfüllt. Hierzu wird in Kapitel 4.1 zunächst der Stand der Technik vorgestellt, bevor in Kapitel 4.2 die modulare Kompaktierung eingeführt wird. Im Anschluss wird in Kapitel 4.3 der modulare Kompaktierer zum hybriden Kompaktierer erweitert.

4.1 Stand der Technik

In der Literatur sind eine Vielzahl an Verfahren zur X-toleranten Kompaktierung von Testantworten zu finden, wie zum Beispiel die sogenannte X-Blockierung [Naruse2003; Wang2008b]. Hier werden die unbekannten Werte auf dem Weg vom Entstehungsort zum Eingang der Kompaktierung blockiert, und somit wird verhindert, dass unbekannte Werte in der Testantwort sichtbar werden.

In der Abbildung 4.2 ist ein Beispiel einer X-Blockierung zu sehen. Ein Teil der zu testenden Schaltung enthält eine X-Quelle (graue Markierung). Diese X-Quelle kann mithilfe des gezeigten UND-Gatters blockiert werden. Der unbekannte Wert erreicht somit die Kompaktierung nicht. Ein Nachteil der X-Blockierung ist der hohe Hardwareaufwand in der zu testenden Schaltung. Ein weiteres Problem ist, dass Fehler, die nur über einen der blockierten Pfade propagiert werden können, nicht detektiert werden können und somit die Fehlerabdeckung verringert wird.

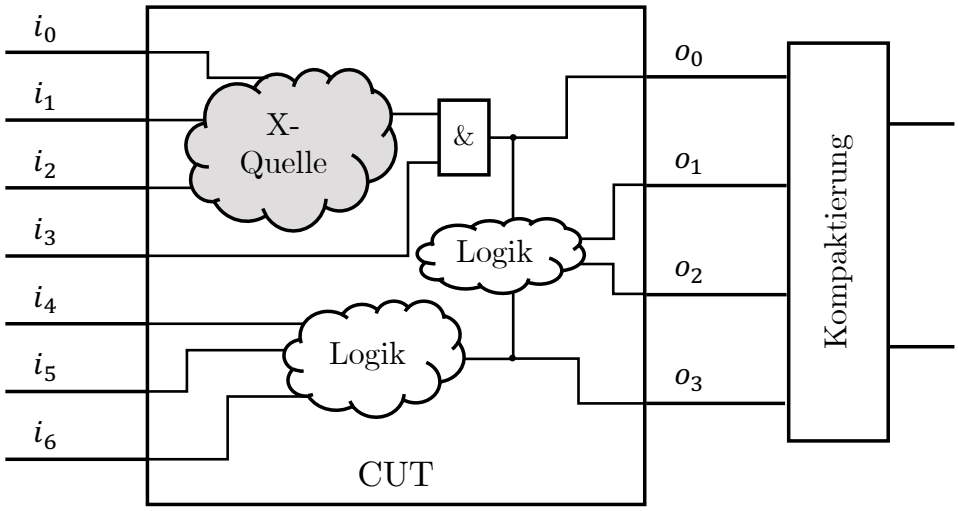


Abbildung 4.2: Beispiel einer X-Blockierung.

Ein weiteres Verfahren ist die X-Maskierung [Pomeranz2002; Wohl2004; Volkerink2005; Rajski2005; Tang2006; Wohl2007; Wohl2008; Czysz2010; Mrugalski2022b]. Bei der X-Maskierung werden die unbekannten Werte mithilfe von zusätzlichen UND- oder ODER-Gattern maskiert, um so einen deterministischen Wert am Eingang des Kompaktierers zu erzeugen.

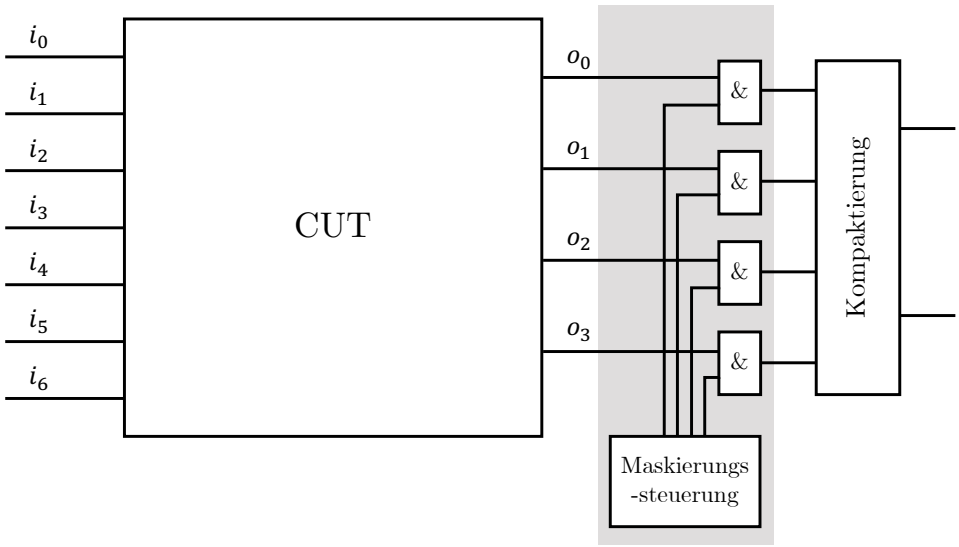


Abbildung 4.3: Beispiel einer X-Maskierung.

Die Abbildung 4.3 zeigt eine zu testende Schaltung mit einer X-Maskierung und einer Kompaktierung. Die grau markierte X-Maskierung wurde hier mithilfe von UND-Gattern und einer Maskierungssteuerung implementiert. Mithilfe von Steuerungsvektoren, die von der Maskierungssteuerung zur Verfügung gestellt werden, ist es so möglich, entweder die Ausgänge an die Kompaktierung weiterzuleiten oder die Ausgänge zu maskieren. Um einen Ausgang zu maskieren, wird von der Maskierungssteuerung eine 0 an das zugehörige UND-Gatter angelegt und somit eine logische 0 an die Kompaktierung

weitergeleitet. So können die von der zu testenden Schaltung erzeugten X-Werte maskiert werden. Ein Nachteil der X-Maskierung ist die große Menge an benötigten Steuerungsvektoren. Für jede Ausgabe der zu testenden Schaltung muss auch ein Steuerungsvektor an die Maskierungslogik angelegt werden, was zu einem hohen Hardwareaufwand führen kann.

Zur Reduktion des Hardwareaufwands ist es möglich hierarchische Verfahren zu verwenden, bei denen die Maskierung über mehrere Takte konstant gehalten wird. Hierdurch kann es jedoch dazu kommen, dass erkennbare Fehler maskiert werden und dadurch die Fehlerabdeckung des Tests verringert wird. In [Rabenalt2009] wird hierzu ein hierarchisches Maskierungsregister vorgestellt, das es ermöglicht die Maskierung nur auf eine Teilmenge der Kompaktierereingänge anzuwenden. Für diese Teilmenge kann das Maskierungsregister anschließend mit Steuerungsvektoren geladen werden, die für jedes Testmuster optimiert sind. Mithilfe eines einzelnen Steuerungssignal kann der aktuelle Steuerungsvektor dann taktweise aktiviert bzw. deaktiviert werden. Dieser hierarchische Ansatz findet aktuell auch in kommerziellen Kompaktierern Anwendung [Rajski2008; Liu2021; Mrugalski2022a]. Hier werden ebenfalls Konfigurationsregister verwendet, die es einerseits ermöglichen, Kompaktierereingänge gruppenweise über mehrere Testmuster hinweg zu maskieren und andererseits ermöglichen, einzelne Kompaktierereingänge taktgenau zu maskieren. Zur weiteren Reduktion des benötigten Speicheraufwands werden die Steuerungsvektoren in diesen Ansätzen komprimiert auf dem Chip gespeichert oder an den Chip übertragen. Hierzu können z. B. Verfahren, wie das *LFSR reseeding* [Naruse2003] verwendet werden.

Während des Entwurfs der hochintegrierten Schaltung kann die X-Maskierung unterstützt werden, indem die X-Rate der einzelnen Speicherelemente berücksichtigt wird. So wird in [Kampmann2018] ein graphbasiertes Verfahren zur Bildung von Prüfpfaden vorgestellt, welches die Speicherelemente anhand der zu erwartenden Anzahl an X-Werten während des Tests gruppiert. Prüfpfade, die aus den Speicherelementen gebildet werden, welche die meisten X-Werte enthalten, werden auch X-Pfade [Wohl2008] genannt. Während der X-Maskierung können diese X-Pfade für einzelne oder mehrere Testantworten blockiert werden, wodurch der Speicheraufwand der Steuerungsvektoren geringer wird.

Zur Einführung in das in diesem Kapitel vorgestellte Kompaktierungsverfahren wird im folgenden Abschnitt näher auf codebasierte Kompaktierungsverfahren [Patel2003; Sharma2005] eingegangen, die fehlerkorrigierende Codes ausnutzen, um X-Werte in Testantworten zu tolerieren. Zunächst wird auf das X-Compact [Mitra2004a] Verfahren eingegangen und anschließend die stochastische Kompaktierung [Mitra2004b] erläutert.

4.1.1 X-tolerante lineare Kompaktierung (X-Compact)

Bei dem in [Mitra2004a] vorgestellten X-toleranten Kompaktierungsverfahren handelt es sich um eine lineare Kompaktierung mithilfe von XOR-Bäumen. Wie in dem Beispiel in Abbildung 4.4 zu sehen ist, werden die acht Ausgänge der zu testenden Schaltung an den X-Kompaktierer angeschlossen und so auf vier Kompaktiererausgänge abgebildet. Der X-Kompaktierer kann dabei durch die Kompaktormatrix $\boldsymbol{C} = (c_{ij})$, wie rechts in Abbildung 4.4 zu sehen, repräsentiert werden. Die Spalten der Matrix stellen hierzu die Ausgänge des Kompaktierers dar und die Zeilen die Eingänge.

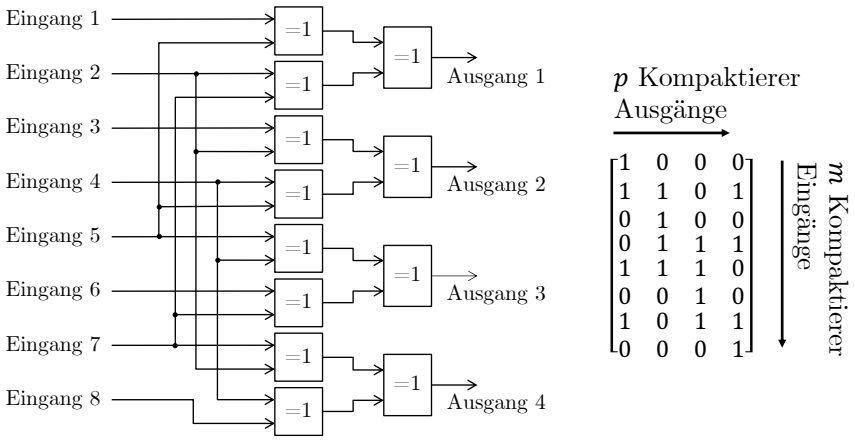


Abbildung 4.4: Aufbau eines X-Kompaktierers und Matrixrepräsentation.

In der linearen Kompaktierung kann es, wie in Kapitel 2.4.2.1 erläutert wurde, zu einer Fehlermaskierung kommen. Diese tritt auf, wenn eine gerade Anzahl an Fehlereffekten in einem Kompaktiererausgang zusammengefasst wird.

Um die Fehlermaskierung im X-Kompaktierer zu verhindern, können aus der Kodierungstheorie bekannte Theoreme zur Bildung von Checkmatrizen linearer Codes [Huffman2003] angewandt werden. Im Folgenden werden die in [Mitra2004a] beschriebenen Theoreme zum Aufbau linearer Kompaktierer eingeführt.

Theorem 4.1. *Wenn in einem Taktzyklus nur ein Ausgang der zu testenden Schaltung fehlerhaft ist, dann ist der Fehler genau dann am Ausgang des X-Kompaktierers zu erkennen, wenn keine Zeile der Kompaktormatrix nur* 0*en enthält.*

Theorem 4.2. *Wenn in einem Taktzyklus ein, zwei oder eine ungerade Anzahl an Ausgängen fehlerhaft sind, dann wird der Fehler genau dann am Ausgang des Kompaktierers erkannt, wenn jede Zeile der Kompaktormatrix mindestens einen Wert* $\neq 0$ *enthält, eindeutig ist und eine ungerade Anzahl* 1*en enthält.*

Durch das Theorem 4.2 wird garantiert, dass, wenn ein Fehlverhalten an einem, zwei oder einer ungeraden Anzahl an Ausgängen sichtbar wird, der Fehler mindestens an einem Ausgang des Kompaktierers erkennbar wird. Beim Auftreten von Fehlern, die nicht durch die Theoreme 4.1 oder 4.2 abgedeckt sind, kann keine Garantie zur Detektion des Fehlers gegeben werden.

Die Fehlererkennung bei einem unbekannten Wert wird durch folgendes Theorem charakterisiert.

Theorem 4.3. *Wenn in einem Taktzyklus ein oder zwei Ausgänge der Schaltung fehlerhaft sind und zusätzlich ein unbekannter Wert auftritt, dann ist das Fehlverhalten genau dann am Ausgang des Kompaktierers sichtbar, wenn:*

1. *keine Zeile der Kompaktormatrix nur* 0*en enthält.*
2. *für jede Zeile der Kompaktormatrix gilt, dass in der Untermatrix, die dadurch entsteht, wenn die jeweilige Zeile und alle Spalten, die eine* 1 *in dieser Zeile enthalten, entfernt werden, alle Zeilen eindeutig sind und keine Zeile nur* 0*en enthält.*

Mithilfe des in [Mitra2004a] vorgestellten Algorithmus erhält man für einen linearen Kompaktierer mit acht Eingängen die folgende Kompaktormatrix $\boldsymbol{C}$, die das Theorem 4.3 erfüllt.

$$\boldsymbol{C} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \tag{4.1}$$

An einem Beispiel mit einem X-Wert am ersten Eingang des Kompaktierers lässt sich das Theorem 4.3 nachvollziehen. Tritt ein X-Wert am ersten Eingang des Kompaktierers auf, dann sind laut Kompaktormatrix $\boldsymbol{C}$ auch die ersten drei Ausgänge des Kompaktierers unbekannt, da sie vom ersten Eingang abhängen. Die verbleibenden drei Ausgänge des Kompaktierers sind unabhängig vom ersten Eingang und in diesem Beispiel somit X-

frei. Die Submatrix $\boldsymbol{C_{sub}}$, die durch das Entfernen der ersten Zeile und der ersten drei Spalten entsteht, sieht wie folgt aus.

$$\boldsymbol{C_{sub}} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{4.2}$$

Tritt an einem oder zwei der verbleibenden Eingänge ein Fehlverhalten auf, so bleibt es an mindestens einem der verbleibenden Ausgänge sichtbar.

Um auch mehrere X-Werte und gleichzeitig mehrere Fehlverhalten zu tolerieren, muss folgendes gelten.

Theorem 4.4. *Wenn in einem Taktzyklus maximal k_1 Ausgänge fehlerhaft sind und an maximal k_2 Ausgängen ein unbekannter Wert auftritt, dann ist das Fehlverhalten am Ausgang des Kompaktierers genau dann sichtbar, wenn*

1. *keine Zeile der Kompaktormatrix nur* 0*en enthält.*
2. *für jede Menge $\mathscr{M}$ aus k_1 Zeilen der Kompaktormatrix, jede Menge aus k_2 Zeilen der Untermatrix, die dadurch entsteht, wenn die Zeilen der Menge $\mathscr{M}$ und alle Spalten, die eine* 1 *in einer dieser Zeilen enthalten, aus der Kompaktormatrix entfernt werden, linear unabhängig sind.*

Lineare Kompaktierer, die die oben genannten Theoreme erfüllen, werden während der Entwurfsphase der zu testenden Schaltung entwickelt und können nicht flexibel auf variierende X-Raten reagieren. Daher müssen sie für die höchste auftretende X-Rate entworfen werden, was zu einer sehr großen Kompaktormatrix und somit zu einem hohen Hardwareaufwand führt.

Um die Erzeugung der Kompaktormatrizen zu vereinfachen, wurde in [Mitra2004b] eine stochastische Methode zur Generierung der Kompaktormatrizen vorgestellt. Diese wird im folgenden Abschnitt kurz erläutert.

4.1.2 Stochastische Kompaktierung

In [Mitra2004b] erweitern *Mitra et al.* die Idee des X-Kompaktierers zu einem stochastischen Kompaktierer. Während bei dem X-Compact Verfahren die Kompaktorkonfiguration einmalig deterministisch festgelegt wird, wird bei einem stochastischen Kompaktierer die Kompaktorkonfiguration während des Betriebs zufällig geändert. In jedem Taktzyklus werden zufällig neue Linearkombinationen der Kompaktoreingänge gebildet, was sich durch die Verwendung von Wahrscheinlichkeiten in der Kompaktormatrix beschreiben lässt. Die gebildeten Linearkombinationen lassen sich mithilfe der Kompaktormatrix $\boldsymbol{C} = (c_{ij})$ beschreiben. Jeder Eintrag c_{ij} nimmt mit der Wahrscheinlichkeit p den Wert 1 an. Wie im vorherigen Abschnitt beschrieben wurde, ist der Ausgang j vom Eingang i abhängig, falls $c_{ij} = 1$ ist. Der Ausgang j ist unabhängig vom Eingang i falls $c_{ij} = 0$. Wie in [Mitra2004b] gezeigt wurde, ist die Wahrscheinlichkeit, dass die Fehlerinformation am Ausgang sichtbar ist, am höchsten, wenn $p = 1/(k_x + 1)$ gilt, wobei k_x die Anzahl der X-Werte an den Eingängen des Kompaktierers angibt.

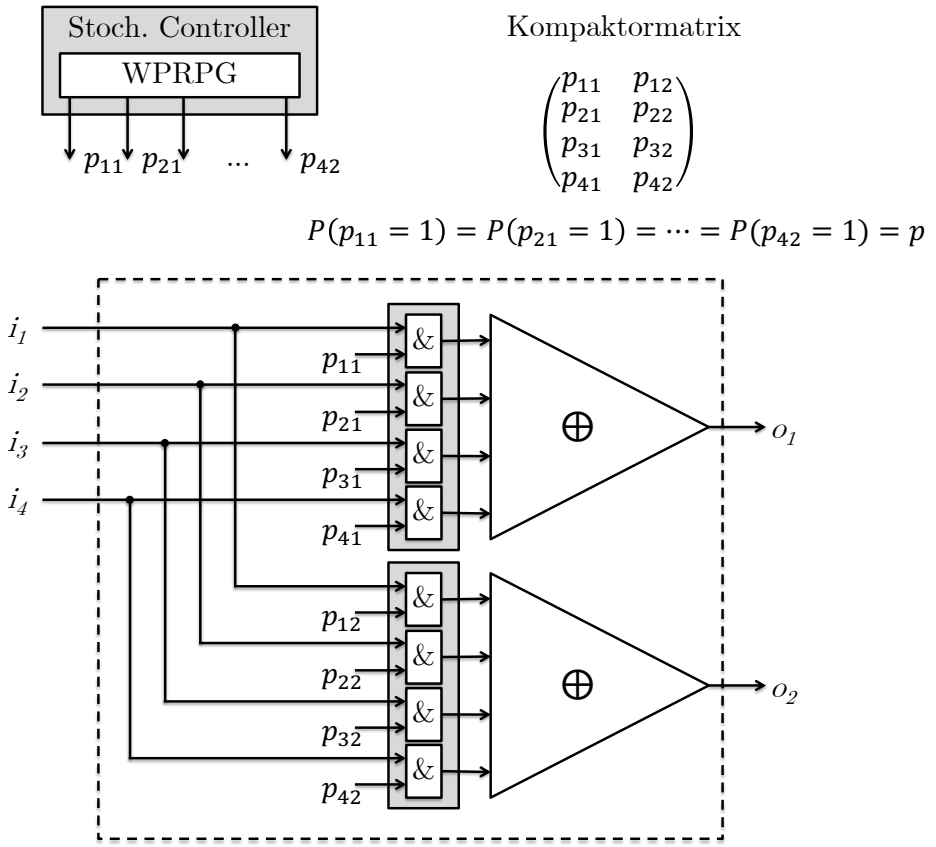


Abbildung 4.5: Implementierung eines stochastischen Kompaktierers [Sprenger2019].

Wie in Abbildung 4.5 zu sehen ist, kann ein stochastischer Kompaktierer mithilfe von XOR-Bäumen, UND-Gatter Netzwerken und einem Modul zur gewichteten pseudozufälligen Mustererzeugung (engl. Weighted Pseudo Random Pattern Generator, WPRPG)

[Strole1991] implementiert werden. Mithilfe eines WPRPG ist es möglich eine Sequenz von Steuerungsvektoren zu erzeugen, in der jedes Bit mit einer Wahrscheinlichkeit von p den Wert 1 annimmt. Je Ausgang werden zunächst alle Eingänge des stochastischen Kompaktierers mithilfe eines UND-Gatters mit dem zugehörigen pseudozufälligen Signal p_{ij} verbunden, welches durch den WPRPG erzeugt wird. Anschließend werden diese UND-Gatter mit einem XOR-Baum verbunden. So ist es möglich, dass der stochastische Kompaktierer in Abhängigkeit der pseudozufälligen Signale p_{ij} eine beliebige Konfiguration annimmt.

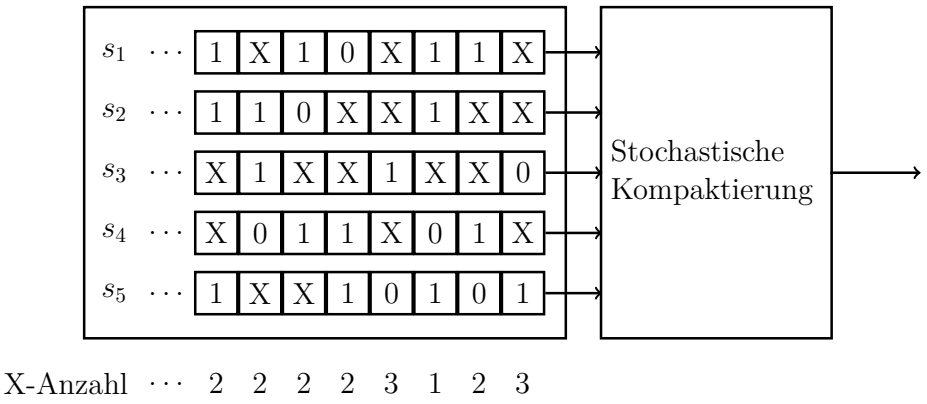


Abbildung 4.6: Beispiel stochastischer Kompaktierer.

Zur Veranschaulichung ist in Abbildung 4.6 ein Beispiel einer stochastischen Kompaktierung für eine Schaltung mit fünf Prüfpfaden gegeben. Die Belegung der Prüfpfade zeigt eine Testantwort auf eine Testbelegung mit unbekannten Werten, die mit X markiert wurden. Unter der Schaltung ist die Anzahl der X-Werte pro Taktzyklus k_x gegeben. Dabei ist zu erkennen, dass die Anzahl der X-Werte pro Taktzyklus variiert und daher in nahezu jedem Takt die Wahrscheinlichkeit p angepasst werden muss. Durch die Variation der Wahrscheinlichkeit p ist es dem stochastischem Kompaktierer zwar möglich auf die variierende X-Rate zu reagieren, für einen eingebauten Selbsttest muss jedoch für jeden Takt eine Wahrscheinlichkeit auf dem Chip gespeichert werden, was zu einem sehr hohen Speicheraufwand führt. Im folgenden Abschnitt wird ein Verfahren vorgestellt, welches die Flexibilität der stochastischen Kompaktierung ausnutzt und gleichzeitig nur einen kleinen Speicherbedarf aufweist.

4.2 Modulare Kompaktierung

Wie bereits in Kapitel 2.4.2 erläutert wurde, sind X-tolerante Verfahren zur Testantwortkompaktierung unter anderem im eingebauten Selbsttest [Bardell1982] unumgänglich. Insbesondere bei der Erweiterung zu einem eingebauten Selbsttest für den Hoch-

geschwindigkeitstest [Hellebrand2014; Kampmann2020] kommt es, wie in Kapitel 2.4.3 gezeigt wurde, durch die Verwendung von erhöhten Taktfrequenzen zu vermehrten unbekannten Werten in den Testantworten der Schaltung, den sogenannten X-Werten.

Um das Problem der variierenden X-Raten zu lösen, wurde die in Kapitel 2.4.3 vorgestellte Architektur für den Hochgeschwindigkeitstest, wie bereits in [Sprenger2018a] und [Sprenger2019] gezeigt, erweitert. Wie in Abbildung 4.7 zu sehen ist, wird der räumliche Kompaktierer durch mehrere kleinere stochastische Kompaktierer ersetzt. Hierzu werden die Ausgänge der zu testenden Schaltung gruppiert und auf die einzelnen stochastischen Kompaktierer aufgeteilt. Die Ausgänge der stochastischen Kompaktierer werden anschließend an ein X-Canceling MISR [Touba2007] weitergegeben. Die vom X-Canceling MISR erzeugten Zwischensignaturen werden in einem kleinen Speicher abgelegt. Durch die Gruppierung der Prüfpfade ist es möglich, die Variation der X-Rate während einer Testfrequenz im Hochgeschwindigkeitstest zu reduzieren, wodurch die Wahrscheinlichkeit p des stochastischen Kompaktierers während einer Testfrequenz konstant gehalten werden kann, ohne die Fehlerdurchlässigkeit des Kompaktierers zu verringern.

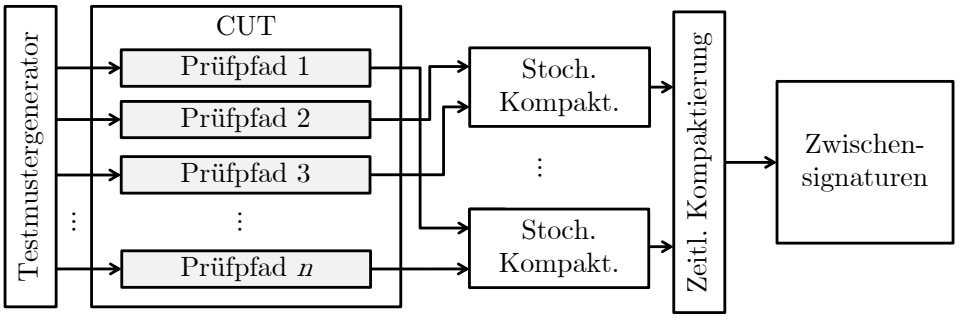


Abbildung 4.7: Testarchitektur für den Hochgeschwindigkeitstest mit modularer Kompaktierung [Sprenger2019].

Durch die Rekonfiguration der Wahrscheinlichkeit p des Kompaktierers kann, wie in Kapitel 4.1.2 gezeigt wurde, auf die variierenden X-Raten, aufgrund der wechselnden Taktfrequenzen während eines Hochgeschwindigkeitstests, reagiert werden. Für eine hohe Fehlerdurchlässigkeit des stochastischen Kompaktierers muss jedoch zusätzlich eine konstante Anzahl an X-Werten pro Takt vorliegen.

In Abbildung 4.8 ist das aus der Abbildung 4.6 bekannte Beispiel für die Verwendung der modularen Kompaktierung mithilfe von zwei stochastischen Kompaktierern zu sehen. Um die Varianz der X-Rate pro Taktzyklus festzustellen, wird zunächst der Vektor $X_{Count}(\mathscr{G}_{Total})$ gebildet. Hierzu wird die Anzahl der X-Werte je Takt (im Beispiel die Anzahl X-Werte pro Spalte) berechnet. Anschließend kann die Varianz $\sigma^2(X_{Count}(\mathscr{G}_{Total})) \approx 0{,}41$ über den Vektor $X_{Count}(\mathscr{G}_{Total})$ berechnet werden, wobei $\mathscr{G}_{Total}$ die Menge aller Prüfpfade der zu testenden Schaltung repräsentiert.

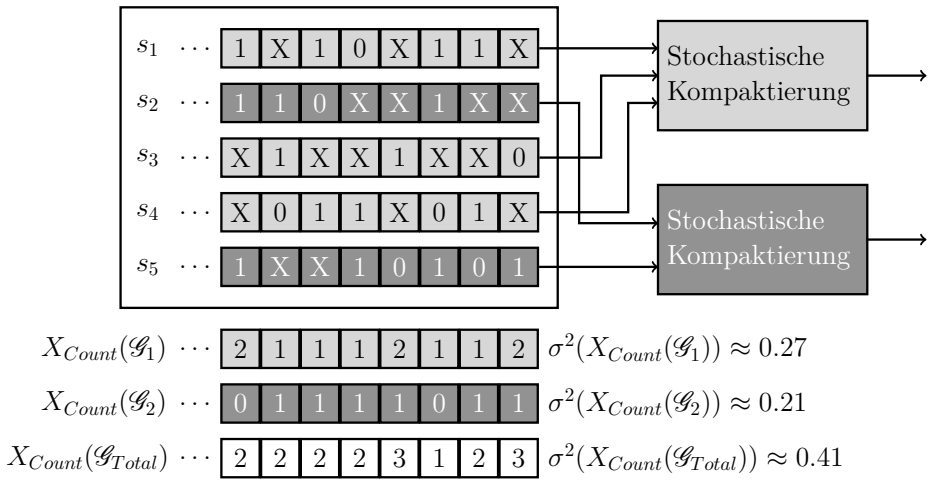


Abbildung 4.8: Beispielkonfiguration der modularen Kompaktierer [Sprenger2019].

Eine Varianz $\sigma^2(X_{Count}(\mathscr{G}_i)) = 0$ würde bedeuten, dass die Anzahl der X-Werte pro Taktzyklus in der Prüfgruppe $\mathscr{G}_i$ konstant ist. Eine Prüfgruppe stellt dabei eine Gruppe von Prüfpfaden (Zeilen der Testantwortmatrix $\boldsymbol{T}$) dar. Mit einer Varianz von $\sigma^2(X_{Count}(\mathscr{G}_{Total})) \approx 0{,}41$ variiert die Anzahl der X-Werte jedoch relativ stark.

Nach der Bestimmung der Varianzen $\sigma^2(X_{Count}(\mathscr{G}_1)) \approx 0{,}27$ und $\sigma^2(X_{Count}(\mathscr{G}_2)) \approx 0{,}21$ für die beiden Prüfgruppen $\mathscr{G}_1$ und $\mathscr{G}_2$ fällt auf, dass die einzelnen Varianzen geringer ausfallen als zuvor. Die geringeren Varianzen bedeuten, dass die Anzahl der X-Werte in den Prüfgruppen $\mathscr{G}_1$ und $\mathscr{G}_2$ weniger variiert als bei der Verwendung aller Prüfpfade in einer Prüfgruppe $\mathscr{G}_{Total}$.

Die Ausgänge jeder Prüfgruppe können nun mit einem stochastischen Kompaktierer verbunden werden. Für jede Taktfrequenz während des Hochgeschwindigkeitstests und jeden stochastischen Kompaktierer kann nun, wie in [Mitra2004b] gezeigt wurde, eine optimale Wahrscheinlichkeit p berechnet werden. Der benötigte Speicherplatz M hängt dabei von der Anzahl der verwendeten Frequenzen f, der Anzahl der benötigten Prüfgruppen $|\mathscr{G}|$ und der Anzahl w der auswählbaren Wahrscheinlichkeiten des WPRPG ab. Werden die auswählbaren Wahrscheinlichkeiten binär codiert, dann lässt sich der Speicherplatz M in Bit wie folgt berechnen.

$$M = f \cdot |\mathscr{G}| \cdot \lceil log_2(w) \rceil \tag{4.3}$$

Zur besseren Anpassung an die variierenden X-Verteilungen kann die Prüfgruppeneinteilung auch je Taktfrequenz durchgeführt werden. Allerdings ist hierzu ein rekonfigurier-

bares Verbindungsnetzwerk zwischen den Ausgängen der zu testenden Schaltung und den Eingängen der stochastischen Kompaktierer notwendig. Wird anstatt dessen eine feste Konfiguration des Verbindungsnetzwerkes gewählt, wird zwar die Anpassung an die X-Verteilungen eingeschränkt, dafür ist jedoch eine einfache Implementierung des Verbindungsnetzwerkes möglich. Die in dem folgenden Abschnitt vorgestellten Gruppierungsalgorithmen können für beide Anwendungsfälle verwendet werden.

4.2.1 Gruppierungsalgorithmen

Im Folgenden werden verschiedene Gruppierungsalgorithmen zur Einteilung der Prüfgruppen eines gegebenen Tests vorgestellt. Alle Testantworten des Tests werden hierzu, wie in Definition 2.4 eingeführt, zu einer Testantwortmatrix zusammengefasst. Das Problem der optimalen Prüfgruppeneinteilung lässt sich für eine gegebene Testantwortmatrix $\boldsymbol{T}$ wie folgt definieren.

Problem 4.1 (Optimale Prüfgruppeneinteilung (OPE))**.** *Gegeben sei eine Schaltung mit einer Menge $\mathscr{S} = \{s_1, ..., s_n\}$ von n Prüfpfaden und einer Testantwortmatrix $\boldsymbol{T}$ über $\{0, 1, X\}$. Finde eine Prüfgruppeneinteilung der Prüfpfade in m disjunkte Prüfgruppen $\mathscr{G} = \{\mathscr{G}_1, \ldots, \mathscr{G}_m\}$, so dass*

$$\sum_{j=1}^{m} \sigma^2(X_{Count}(\mathscr{G}_j)) \tag{4.4}$$

minimal ist. [Sprenger2018b]

Das Problem ist vergleichbar mit dem Partitionsproblem (engl. Set-Partitioning-Problem) [Balas1976], das wie folgt definiert ist.

Problem 4.2 (Partitionsproblem)**.** *Gegeben sei eine Menge $\mathscr{S} = \{s_1, ..., s_n\}$ und die Teilmengen $\mathscr{G}_j \subset \mathscr{S}$ mit den Kosten c_j, $j \in \mathscr{J} = \{1, ..., l\}$. Finde eine Überdeckung an disjunkten Teilmengen $\mathscr{G}_j$, $j \in \mathscr{J}^* \subset \mathscr{J}$, so dass*

$$\sum_{j \in \mathscr{J}^*} c_j \tag{4.5}$$

minimal ist.

Während im Partitionsproblem die Kosten der Teilmengen c_j frei wählbar und unabhängig voneinander sind, sind die Kosten der Prüfgruppen $\sigma^2(X_{Count}(\mathscr{G}_j))$ in der optimalen Prüfgruppeneinteilung voneinander abhängig. Das Problem der optimalen Prüfgruppeneinteilung ist somit als Spezialfall des Partitionsproblems anzusehen.

4.2.1.1 Lösungsansatz mithilfe von linearer Programmierung

Mithilfe von gemischter ganzzahliger Optimierung (engl. Mixed Integer Linear Programming) kann für das Problem 4.1 eine optimale Lösung gefunden werden [Floudas1995]. Um die beste mögliche Auswahl an Prüfgruppen aus allen möglichen Prüfgruppen $\mathscr{G}_1, ..., \mathscr{G}_L$ zu treffen, wird die Kostenfunktion

$$\sum_{j=1}^{L} x_j \cdot \sigma^2(X_{Count}(\mathscr{G}_j)) \tag{4.6}$$

minimiert unter den Randbedingungen

$$\sum_{j \in \Gamma(i)} x_j = 1; i = 1, \ldots, n. \tag{4.7}$$

Mithilfe der binären Variablen x_j wird angegeben, ob die Prüfgruppe $\mathscr{G}_j$ Teil der Lösung ist. Die Randbedingung (4.7) stellt sicher, dass jeder Prüfpfad s_i genau einmal ausgewählt wurde, wobei $\Gamma(i)$ die Menge aller Prüfgruppen ist, die den Prüfpfad s_i beinhalten. Durch die Reduktion der möglichen Prüfgruppen $\mathscr{G}_1, ..., \mathscr{G}_L$ kann der Suchraum weiter eingeschränkt werden und damit auch die benötigte Laufzeit zur Berechnung der optimalen Lösung. Die Anzahl der möglichen Prüfgruppen $\mathscr{G}_1, ..., \mathscr{G}_L$ steigt jedoch exponentiell mit der Anzahl der verwendeten Prüfpfade. Da die Anzahl der verwendeten Prüfpfade typischerweise mit der Größe der Schaltung steigt, kann nur für kleine Schaltungen eine optimale Lösung in vertretbarer Zeit gefunden werden. Mit der Anpassung der Kostenfunktion könnten weitere Ziele verfolgt werden. Um die Homogenität der X-Verteilungen zu steigern, könnte z. B. die Differenz zwischen zwei Varianzen minimiert werden.

4.2.1.2 Heuristische Lösungen

Um eine Lösung des Problems 4.1 in akzeptabler Laufzeit zu erhalten, müssen heuristische Verfahren angewandt werden. Dies ermöglicht es auch, benutzerorientierte Randbedingungen zu berücksichtigen oder praxisorientierte Ansätze zu verfolgen. So könnte z. B. die Anzahl der verwendeten stochastischen Kompaktierer eine Entwurfsentscheidung sein oder die Anzahl der Eingänge je stochastischem Kompaktierer.

Testunabhängige Prüfgruppeneinteilung In dem ersten heuristischen Ansatz werden zunächst keine Testmuster berücksichtigt. Dies hat den Vorteil, dass der Entwurf der Schaltung schon abgeschlossen werden kann, bevor die Testmustererzeugung abgeschlossen ist. Jedoch kann nicht auf die tatsächliche X-Verteilung der Schaltung eingegangen werden. In diesem einfachen Ansatz wird nur die Anzahl an verwendeten Prüfpfaden berücksichtigt. Der Algorithmus 4.1 teilt hierzu die Prüfpfade in Prüfgruppen der Größe N ein. Die Anzahl der Prüfgruppen ergibt sich damit zu $m = \lceil n/N \rceil$, wobei n der Anzahl an Prüfpfaden $|\mathscr{S}|$ entspricht.

Algorithmus 4.1 Testunabhängige Prüfgruppeneinteilung [Sprenger2019]

```
1: function TestunabhängigeEinteilung(𝒮, N)
2:     n ← |𝒮|
3:     m ← ⌈n/N⌉
4:     𝒢 ← {𝒢_j ← ∅|∀j ∈ {1, ..., m}}          ▷ Generiere leere Prüfgruppen
5:     for i ← 1 to n do          ▷ Auffüllen der Prüfgruppen anhand der Prüfpfadindizes
6:         𝒢_⌈i/N⌉ ← 𝒢_⌈i/N⌉ ∪ {s_i}
7:     end for
8:     return 𝒢
9: end function
```

Nachdem die Anzahl der benötigten Prüfgruppen berechnet wurde, werden zunächst m leere Prüfgruppen erzeugt. In den Zeilen 5-6 iteriert der Algorithmus 4.1 dann über die Prüfpfade in aufsteigender Reihenfolge und fügt die ersten N Prüfpfade in die erste Prüfgruppe. Anschließend werden die restlichen $m - 1$ Prüfgruppen mit Prüfpfaden gefüllt. Falls die Anzahl an Prüfpfaden nicht ganzzahlig durch die Anzahl der Prüfgruppen teilbar ist, weicht die Größe der m-ten Prüfgruppe von der Zielgröße N ab und die letzte Prüfgruppe enthält die verbleibenden Prüfpfade. Für eine ausgeglichene Prüfgruppengröße können in einem weiteren Schritt die Prüfpfade der letzten Prüfgruppe auf die ersten $m - 1$ Prüfgruppen aufgeteilt werden.

In der Abbildung 4.9 ist das Ergebnis eines Durchlaufs des Algorithmus 4.1 mit fünf Prüfpfaden und einer Prüfgruppengröße von $N = 2$ gegeben. Dementsprechend wurden

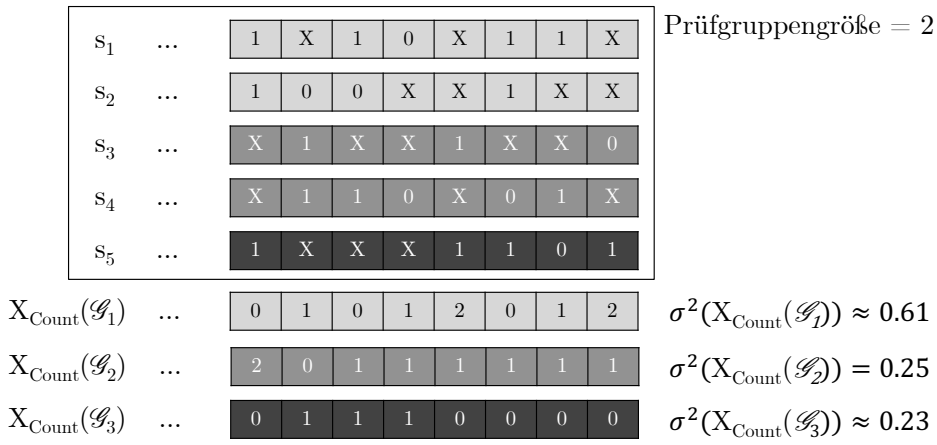


Abbildung 4.9: Beispiel für die testunabhängige Prüfgruppeneinteilung [Sprenger2019].

drei Prüfgruppen gebildet. Die erste Prüfgruppe enthält die ersten beiden Prüfpfade, die zweite Prüfgruppe die nächsten beiden und der letzte Prüfpfad wurde der dritten Prüfgruppe zugeordnet.

Bin-Packing Prüfgruppeneinteilung Bei diesem Algorithmus wird nach einer festen Anzahl m Prüfgruppen gesucht. Jede Prüfgruppe wird dabei als Behälter angesehen mit einer gegebenen X-Kapazität γ, welche die Anzahl an X-Werten vorgibt, die der zugehörige stochastische Kompaktierer verarbeiten kann. Jeder Behälter wird dabei so gefüllt, dass die X-Kapazität nicht überschritten wird. Dies entspricht einem klassischen Behälterproblem, für welches eine Vielzahl an Heuristiken bekannt sind. Der in Algorithmus 4.2 gezeigte Bin-Packing Algorithmus ist eine Variante des klassischen First-Fit Algorithmus. [Korte2018]

Algorithmus 4.2 besitzt drei Eingabeparameter, die Menge der Prüfpfade $\mathscr{S}$, die Anzahl der Behälter m und die Anzahl der X-Werte innerhalb der Schaltung unter Verwendung der gegebenen Testmustermenge X_{Total}. Die Anzahl der X-Werte innerhalb einer Prüfgruppe gibt $X_{Total}(\mathscr{G}_j)$ an.

In Zeile 2 des Algorithmus 4.2 wird zunächst die X-Kapazität γ der einzelnen Behälter berechnet mithilfe von (4.8).

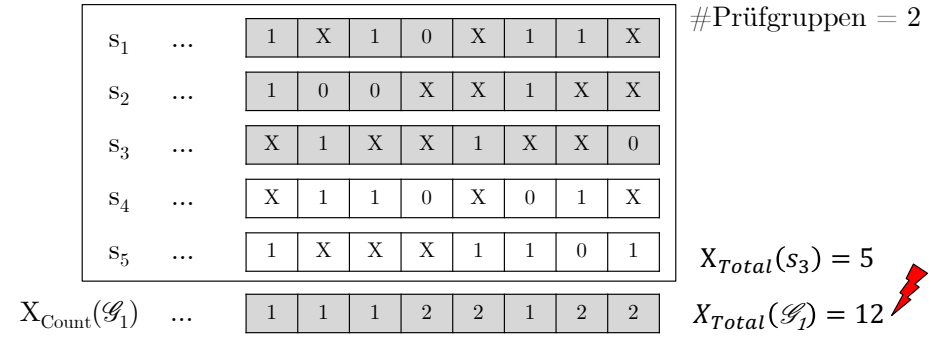
$$\gamma = \left\lceil \frac{X_{Total}}{m} \right\rceil \tag{4.8}$$

Algorithmus 4.2 Bin-Packing Prüfgruppeneinteilung [Sprenger2019]

function BinPacking($\mathscr{S}, m, X_{Total}$)
  $\gamma \leftarrow \lceil X_{Total}/m \rceil$ ▷ Berechne die X-Kapazität γ der Behälter
  $\mathscr{G} \leftarrow \{\mathscr{G}_j \leftarrow \emptyset | \forall j \in \{1, ..., m\}\}$ ▷ Generiere leere Prüfgruppen
  $\mathscr{NC} \leftarrow \emptyset$
  for $i \leftarrow 1$ **to** n **do** ▷ Auffüllen der Prüfgruppen
    $added \leftarrow false$
    $j \leftarrow 1$
    while $(j < m)$ **and** (**not** *added*) **do**
      if $X_{Total}(\mathscr{G}_j \cup \{s_i\}) \leq \gamma$ **then**
        $\mathscr{G}_j \leftarrow \mathscr{G}_j \cup \{s_i\}$ ▷ Füge Prüfpfad s_i zur Prüfgruppe $\mathscr{G}_j$ hinzu
        $added \leftarrow true$
      end if
      $j \leftarrow j + 1$
    end while
    if not added **then**
      $\mathscr{NC} \leftarrow \mathscr{NC} \cup \{s_i\}$ ▷ Füge Prüfpfad s_i zur Prüfgruppe $\mathscr{NC}$ hinzu
    end if
  end for
  for $i \leftarrow 1$ **to** $|\mathscr{NC}|$ **do** ▷ $\mathscr{NC} = \{s_{NC(1)}, s_{NC(2)}, ...\}$
    $\mathscr{G}_{min} \leftarrow \arg\min_{\mathscr{G}_j \in \mathscr{G}} \{X_{Total}(\mathscr{G}_j)\}$ ▷ $X_{Total}(\mathscr{G}_{min})$ ist minimal
    $\mathscr{G}_{min} \leftarrow \mathscr{G}_{min} \cup \{s_{NC(i)}\}$
  end for
  return $\mathscr{G}$
end function

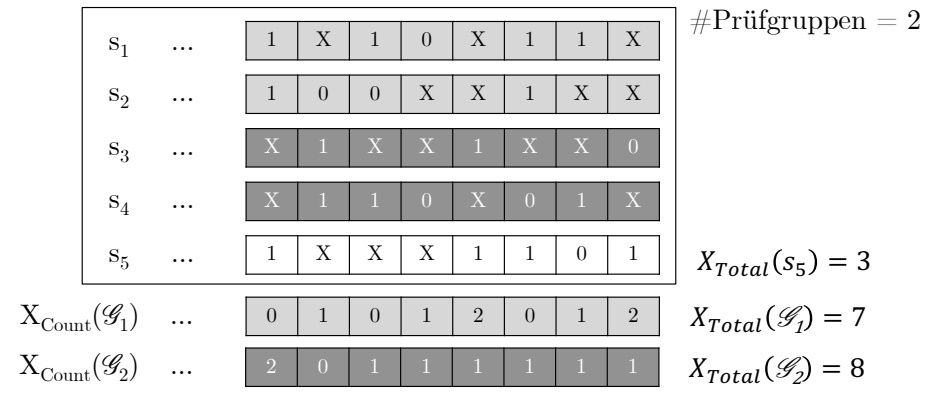
Anschließend werden m leere Behälter erzeugt, sowie ein Behälter $\mathscr{NC}$ für nicht eingeordnete Prüfpfade. Für alle Prüfpfade in $\mathscr{S}$ iteriert der Bin-Packing Algorithmus nun über die Behälter $\mathscr{G}_j$ und fügt den Prüfpfad zu dem ersten Behälter hinzu, in dem, nach dem Hinzufügen des aktuellen berücksichtigten Prüfpfades s_i, die X-Kapazität nicht überschritten wird. Kann der Prüfpfad zu keinem Behälter $\mathscr{G}_j$ hinzugefügt werden, ohne die X-Kapazität zu überschreiten, wird der Prüfpfad dem Behälter $\mathscr{NC}$ hinzugefügt. Nachdem alle Prüfpfade einem der $m+1$ Behälter zugeordnet wurden, werden die Prüfpfade die dem Behälter $\mathscr{NC}$ zugeordnet sind, so auf die restlichen m Behälter verteilt, dass die X-Kapazität möglichst wenig überschritten wird. Hierzu wird über die Prüfpfade $s_{NC(i)}$ des Behälters $\mathscr{NC}$ iteriert und der Prüfpfad dem Behälter zugeteilt, der bisher die geringste Zahl an X-Werten enthält.

In Abbildung 4.10 ist ein Beispiel zur Anwendung des Bin-Packing Algorithmus gegeben. Wie in Abbildung 4.10a zu sehen ist, werden die ersten Prüfpfade solange dem ersten Behälter $\mathscr{G}_1$ zugeordnet, bis die X-Kapazität überschritten wird. Dies ist der Fall sobald versucht wird, den Prüfpfad s_3 hinzuzufügen. Da dies nicht möglich ist, wird der Prüfpfad dem nächsten Behälter $\mathscr{G}_2$ zugeteilt. Nachdem alle Prüfpfade abgearbeitet wurden, befinden sich die ersten beiden Prüfpfade in dem Behälter $\mathscr{G}_1$, die folgenden



$$\gamma = \left\lceil \frac{X_{Total}}{m} \right\rceil = \frac{18}{2} = 9$$

(a) X-Kapazität überschritten



$$\gamma = \left\lceil \frac{X_{Total}}{m} \right\rceil = \frac{18}{2} = 9$$

(b) Prüfgruppeneinteilung ohne Nachbearbeitungsschritt

Abbildung 4.10: Beispiel für die Bin-Packing Prüfgruppeneinteilung [Sprenger2019].

beiden Prüfpfade in dem Behälter $\mathscr{G}_2$, und der Prüfpfad s_5 befindet sich im Behälter $\mathscr{NC}$, da er keinem Behälter hinzugefügt werden konnte (Siehe Abbildung 4.10b). Im Nachbearbeitungsschritt wird anschließend der Prüfpfad s_5 dem Behälter $\mathscr{G}_1$ zugeteilt, da dieser die geringste Auslastung der X-Kapazität aufweist.

Varianzorientierte Prüfgruppeneinteilung Bei der dritten Heuristik, die in dieser Arbeit vorgestellt wird, handelt es sich um einen gierigen (engl. greedy) Algorithmus, der die aktuelle Prüfgruppe anhand der Varianz der X-Verteilung $\sigma^2(X_{Count}(\mathscr{G}_j))$ innerhalb der Prüfgruppe $\mathscr{G}_j$ aufbaut. Hierzu wird, wie in Kapitel 4.2 erläutert, die X-Verteilung $X_{Count}(\mathscr{G}_j)$ und die Varianz $\sigma^2(X_{Count}(\mathscr{G}_j))$ für die aktuelle Prüfgruppe $\mathscr{G}_j$ berechnet, um iterativ Prüfpfade zu der Prüfgruppe hinzuzufügen, so dass die Varianz $\sigma^2(X_{Count}(\mathscr{G}_j))$ sinkt. Hierzu wird mithilfe von (4.9) die X-Verteilung in der bisherigen Prüfgruppe $X^{old}_{Count}(\mathscr{G}_j)$ mit der X-Verteilung des zu bewertenden Prüfpfades $X_{Count}(s_i)$ addiert und so die X-Verteilung in der aktuellen Prüfgruppe $X^{new}_{Count}(\mathscr{G}_j)$ berechnet.

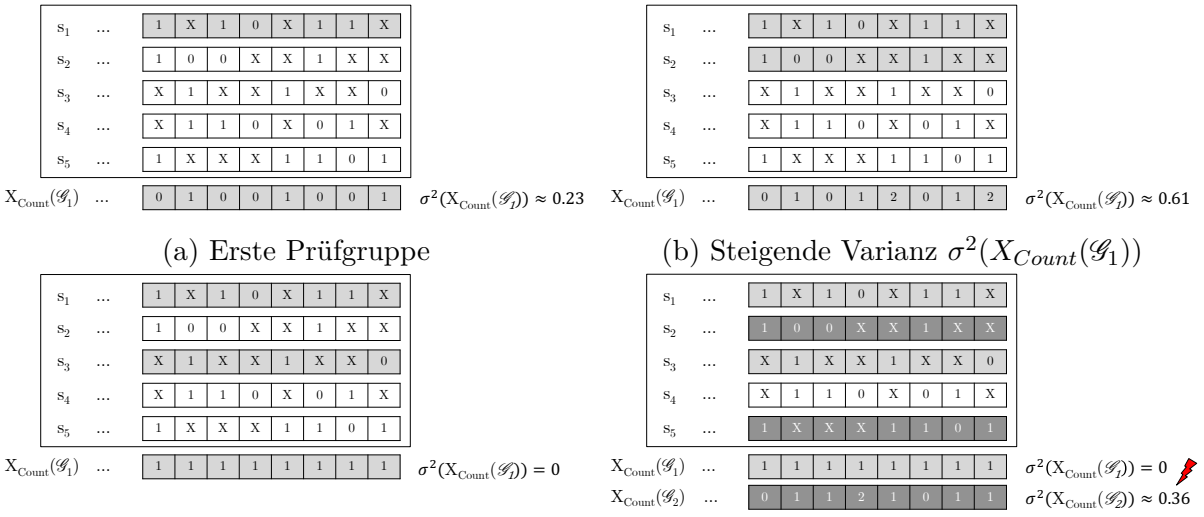
$$X_{Count}^{new}(\mathscr{G}_j) = X_{Count}^{old}(\mathscr{G}_j) + X_{Count}(s_i) \tag{4.9}$$

Algorithmus 4.3 Varianzorientierte Prüfgruppeneinteilung [Sprenger2019]

```
1: function VarianzorientierteEinteilung(𝒮, numRuns)
2:     𝒩𝒴𝒞 ← 𝒮                          ▷ Menge der noch nicht eingeteilten Prüfpfade 𝒩𝒴𝒞
3:     j ← 1
4:     𝒢 ← ∅
5:     while 𝒩𝒴𝒞 ≠ ∅ do
6:         Wähle Prüfpfad s ∈ 𝒩𝒴𝒞
7:         𝒢_j ← {s}                                  ▷ Initialisiere Prüfgruppe 𝒢_j
8:         𝒩𝒴𝒞 ← 𝒩𝒴𝒞 \ {s}
9:         success ← false
10:        runs ← numRuns
11:        for each s ∈ 𝒩𝒴𝒞 do
12:            if σ²(X_Count(𝒢_j ∪ {s})) < σ²(X_Count(𝒢_j)) then   ▷ Prüfgruppe iterativ befüllen
13:                𝒢_j ← 𝒢_j ∪ {s}
14:                success ← true
15:            end if
16:        end for
17:        if success then
18:            𝒩𝒴𝒞 ← 𝒩𝒴𝒞 \ {𝒢_j}                          ▷ Aktualisiere 𝒩𝒴𝒞
19:        else
20:            while (runs > 0) and 𝒩𝒴𝒞 ≠ ∅ do
21:                s_min ← arg min_{s∈𝒩𝒴𝒞} {σ²(X_Count(𝒢_j ∪ {s})) − σ²(X_Count(𝒢_j))}
22:                                                   ▷ s mit kleinster Steigerung von σ²
23:                𝒢_j ← 𝒢_j ∪ {s_min}
24:                𝒩𝒴𝒞 ← 𝒩𝒴𝒞 \ {s_min}
25:                runs ← runs − 1
26:            end while
27:        end if
28:        𝒢 ← 𝒢 ∪ 𝒢_j
29:        j ← j + 1
30:    end while
31:    return 𝒢
32: end function
```

Wie in Algorithmus 4.3 gezeigt, hat der varianzorientierte Algorithmus die Menge der Prüfpfade $\mathscr{S}$ und die Anzahl der Prüfpfade *numRuns*, welche die Varianz einer Prüfgruppe erhöhen dürfen, als Eingabeparameter. Zunächst werden alle Prüfpfade in die Menge der noch nicht zugeordneten Prüfpfade $\mathscr{NYC}$ hinzugefügt. Solange noch Prüfpfade existieren, die noch nicht zugeordnet wurden, wird in Zeile 7 des Algorithmus 4.3 eine neue Prüfgruppe $\mathscr{G}_j$ erstellt und der aktuelle Prüfpfad $s \in \mathscr{NYC}$ hinzugefügt. Anschließend wird der Prüfpfad s aus der Menge der noch nicht zugeordneten Prüfpfade $\mathscr{NYC}$ entfernt. In den Zeilen 11 - 16 wird nun für alle Prüfpfade, die noch keiner Prüf-

gruppe angehören, geprüft, ob sich die Varianz $\sigma^2(X_{Count}(\mathscr{G}_j))$ der aktuellen Prüfgruppe verringert, wenn der Prüfpfad zur Prüfgruppe hinzugefügt wird. Falls sich die Varianz verringert, wird der Prüfpfad zur Prüfgruppe hinzugefügt. Falls mindestens ein Prüfpfad in die aktuelle Prüfgruppe hinzugefügt werden konnte, werden die Prüfpfade der aktuellen Prüfgruppe aus der Menge, der noch nicht zugeordneten Prüfpfade entfernt und eine neue Prüfgruppe wird aufgebaut, falls noch Prüfpfade zuzuordnen sind. Sollte es keinen Prüfpfad geben, der die Varianz $\sigma^2(X_{Count}(\mathscr{G}_j))$ verringern kann, werden in den Zeilen 20 - 26 des Algorithmus 4.3 die $numRuns$ Prüfpfade, welche die Varianz am wenigsten erhöhen, zu der aktuellen Prüfgruppe $\mathscr{G}_j$ hinzugefügt. Mithilfe des Parameters $numRuns$ ist es einerseits möglich, die Größe der Prüfgruppen zu beeinflussen, anderseits ist es so möglich lokalen Minima in der Kostenfunktion des Algorithmus zu entkommen.



(a) Erste Prüfgruppe

(b) Steigende Varianz $\sigma^2(X_{Count}(\mathscr{G}_1))$

(c) Sinkende Varianz $\sigma^2(X_{Count}(\mathscr{G}_1))$

(d) Erlaubte steigende Varianz $\sigma^2(X_{Count}(\mathscr{G}_1))$

Abbildung 4.11: Beispiel für die varianzorientierte Prüfgruppeneinteilung [Sprenger2019].

Ein Beispiel für die Anwendung des varianzorientierten Algorithmus ist in Abbildung 4.11 zu sehen. Zunächst wird die Prüfgruppe $\mathscr{G}_1$ erstellt und, wie in Abbildung 4.11a gezeigt, mit dem Prüfpfad s_1 gefüllt. Anschließend wird versucht, den Prüfpfad s_2 zur Prüfgruppe $\mathscr{G}_1$ hinzuzufügen. Hierzu wird, wie in Abbildung 4.11b gezeigt, zunächst $X_{Count}(\mathscr{G}_1)$ aktualisiert und die Varianz $\sigma^2(X_{Count}(\mathscr{G}_1)) \approx 0{,}61$ berechnet. Da die Varianz gestiegen ist, wird der Prüfpfad s_3 nicht zur Prüfgruppe hinzugefügt und der nächste Prüfpfad wird untersucht. Wie in Abbildung 4.11c zu sehen ist, reduziert sich die Varianz der Prüfgruppe $\mathscr{G}_1$ auf 0. Die Anzahl der X-Werte je Taktzyklus ist somit konstant. Da sich die Varianz der Prüfgruppe verringert hat, wird der Prüfpfad s_3 der Prüfgruppe hinzugefügt. Da die Varianz nicht weiter reduziert werden kann, kann in diesem Fall kein weiterer Prüfpfad zur Prüfgruppe hinzugefügt werden, wenn der Parameter $numRuns = 1$ ist. Nur Prüfpfade, die keine X-Werte beinhalten oder an jeder Stelle ein X tragen, können zu einer Prüfgruppe hinzugefügt werden, ohne die Varianz zu erhöhen.

Daher wird in diesem Beispiel eine weitere Prüfgruppe aufgebaut, in die zunächst der Prüfpfad s_2 hinzugefügt wird. Die Abbildung 4.11d zeigt den Zustand der Prüfgruppeneinteilung, nachdem anschließend versucht wurde die Prüfpfade s_4 und s_5 hinzuzufügen. Beide Prüfpfade würden die Varianz $\sigma^2(X_{Count}(\mathscr{G}_2))$ erhöhen und wurden somit nicht zur Prüfgruppe hinzugefügt. Daher wird der Prüfpfad zur Prüfgruppe hinzugefügt, der die Varianz am wenigsten erhöht.

In der Abbildung 4.12 sind die Prüfgruppeneinteilungen für die drei Heuristiken, bei der Verwendung von 5 äquidistanten Testfrequenzen zwischen der nominellen Taktfrequenz und der dreifachen nominellen Taktfrequenz, dargestellt. Für die Taktperioden gilt $T_{nom} = T_{FAST_1} > T_{FAST_2} > T_{FAST_3} > T_{FAST_4} > T_{FAST_5} = T_{nom}/3$. Die verwendete Testmustermenge für Verzögerungsfehler wurde mithilfe eines kommerziellen Testmustergenerators erzeugt. Als Schaltung wurde eine Industrieschaltung mit 128 Prüfpfaden verwendet. Auf der X-Achse sind die ersten sieben Prüfgruppen aufgetragen. Die Y-Achse zeigt den Prüfpfadindex. Wie zu erwarten, gibt es bei der Verwendung des Algorithmus zur testunabhängigen Prüfgruppeneinteilung (Abbildung 4.12a) keinen Unterschied bei unterschiedlichen Testfrequenzen, da das Verfahren unabhängig von der Testmustermenge und von der Testfrequenz ist. Bei der Anwendung der Bin-Packing und der varianzorientierten Prüfgruppeneinteilung in Abbildung 4.12b und Abbildung 4.12c hingegen fällt auf, dass die Verteilung der Prüfpfade auf die Prüfgruppen für unterschiedliche Testfrequenzen stark variieren kann. Um die Rekonfiguration der Prüfgruppen nach einem Testfrequenzwechsel zu ermöglichen, ist jedoch ein erheblicher Hardwareaufwand notwendig, da ein rekonfigurierbares Verbindungsnetzwerk zwischen zu testender Schaltung und den stochastischen Kompaktierern notwendig ist. Daher wird im Rest dieser Arbeit die Prüfgruppeneinteilung für den gesamten Test durchgeführt.

Wie in Kapitel 2.4.3 gezeigt wurde, können mithilfe des Hochgeschwindigkeitstests kleine Verzögerungsfehler detektiert werden, die bei einem gewöhnlichen Verzögerungstest nicht detektiert werden können. Mithilfe des hier vorgestellten Kompaktierungsverfahrens soll die Architektur des eingebauten Hochgeschwindigkeitstests [Hellebrand2014; Kampmann2020] unterstützt werden. In dieser Architektur folgt auf den modularen Kompaktierer, wie in Abbildung 4.7 gezeigt wurde, eine zeitliche Kompaktierung. Hierzu wird ein X-Canceling MISR [Touba2007] verwendet, welches durch die Reduzierung der X-Werte am MISR-Eingang unterstützt werden soll. Des Weiteren ist es wichtig so viele Fehlerinformationen wie möglich zu dem X-Canceling MISR zu propagieren. Mit Hinblick auf diese Eigenschaften werden der X-Reduktionsfaktor (XRF) und die Fehlerdurchlässigkeit (FD) als Metriken für die Bewertung des Kompaktierungsverfahrens verwendet. Bevor die Metriken definiert werden können, wird zunächst die Fehlermenge $\mathscr{F}(s_{ij})$ wie folgt definiert.

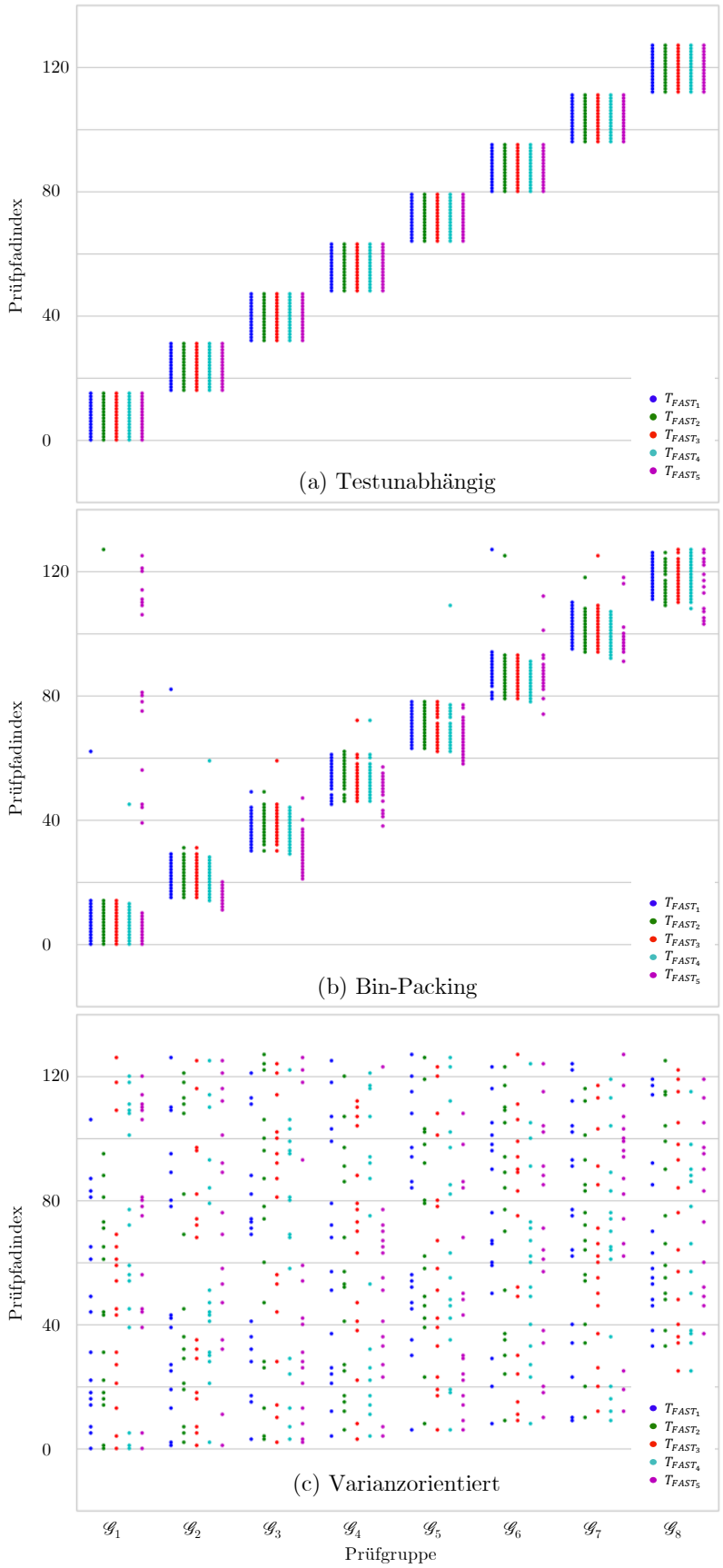


Abbildung 4.12: Prüfgruppeneinteilung mit unterschiedlichen Algorithmen für fünf Testfrequenzen.

Definition 4.1 (Fehlermengen)**.** *Die Fehlermenge $\mathscr{F}$ sei die Menge aller Fehler in einer Schaltung bei einem gegebenen Fehlermodell. Die Fehlermenge $\mathscr{F}(t_{ij}) \subseteq \mathscr{F}$ sei die Fehlermenge, die in t_{ij} der Testantwortmatrix $\boldsymbol{T} = (t_{ij})$ sichtbar wird und $\mathscr{F}(s_{ij}) \subseteq \mathscr{F}$ sei die Fehlermenge, die in s_{ij} der Signaturmatrix $\boldsymbol{S} = (s_{ij})$ sichtbar wird.*

Der X-Reduktionsfaktor (XRF) kann mit

$$XRF = \frac{\#\text{ X-Werte vor der Kompaktierung}}{\#\text{ X-Werte nach der Kompaktierung}} \tag{4.10}$$

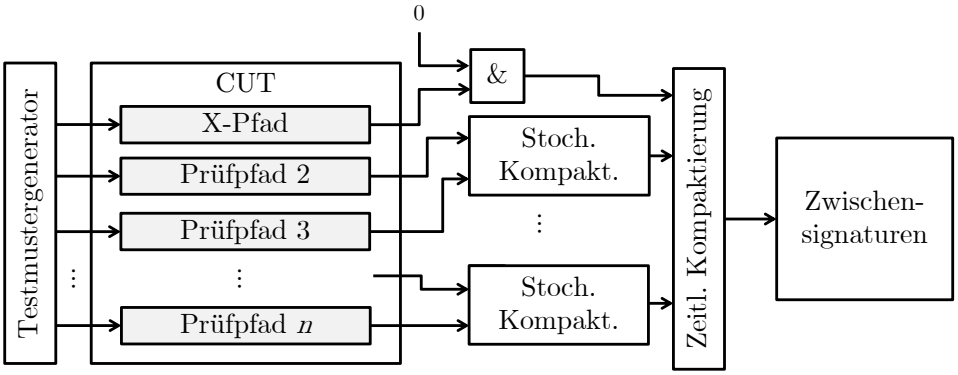
berechnet werden und die Fehlerdurchlässigkeit (FD) mit

$$FD = \frac{\bigcup_{s_{ij} \in \boldsymbol{S}} |\mathscr{F}(s_{ij})|}{\bigcup_{t_{ij} \in \boldsymbol{T}} |\mathscr{F}(t_{ij})|}. \tag{4.11}$$

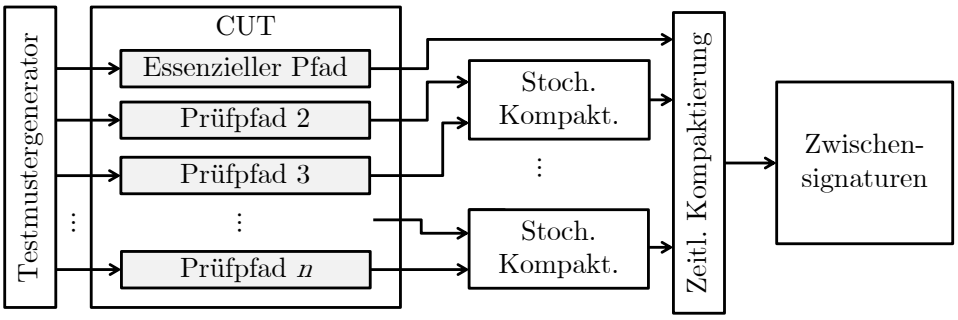
4.2.2 Ausnutzung von X-Pfaden und essenziellen Prüfpfaden

Steht weitere Information über die Prüfpfadkonfiguration zur Verfügung, kann diese genutzt werden, um den X-Reduktionsfaktor (XRF) oder die Fehlerdurchlässigkeit (FD) zu erhöhen. Wie in Kapitel 4.1 eingeführt wurde, können z. B. die Speicherelemente mit hohen X-Raten zu X-Pfaden gruppiert werden. Soll der X-Reduktionsfaktor erhöht werden, können diese X-Pfade, wie in Abbildung 4.13a gezeigt wird, während der räumlichen Kompaktierung vollständig blockiert werden. Dies wird als X-Pfad-Blockierung bezeichnet. Kein X-Wert des X-Pfades erreicht somit die Phase der zeitlichen Kompaktierung, wodurch der X-Reduktionsfaktor erhöht wird.

Prüfpfade mit Speicherelementen, in denen viele Fehler sichtbar sind, können hingegen zu essenziellen Prüfpfaden zusammengefasst werden. Die essenziellen Prüfpfade tragen nur wenige, oder keine X-Werte und können daher direkt an der räumlichen Kompaktierung vorbeigeführt werden, wie in Abbildung 4.13b gezeigt. Diese wenigen X-Werte können mithilfe eines X-toleranten Verfahrens zur zeitlichen Kompaktierung verarbeitet werden. Mithilfe dieses sogenannten Pfad-Bypasses ist es möglich die Fehlerdurchlässigkeit zu erhöhen, da keine Fehler während der räumlichen Kompaktierung maskiert werden können.



(a) X-Pfad-Blockierung



(b) Pfad-Bypass essenzieller Prüfpfade

Abbildung 4.13: Stochastische Kompaktieurng mit X-Pfad-Blockierung und Pfad-Bypass für essenzielle Prüfpfade.

Der Nachteil bei dem jeweiligen Verfahren ist, dass entweder die Fehlerdurchlässigkeit weiter abgesenkt wird, wenn Fehlerinformationen blockiert werden, oder der X-Reduktionsfaktor abgesenkt wird, wenn sich X-Werte in den essenziellen Prüfpfaden befinden.

4.2.3 Experimente

Um das vorgestellte Verfahren zur Testantwortkompaktierung zu bewerten, wurden verschiedene Experimente mit unterschiedlichen Benchmark-Schaltungen durchgeführt. Die Ergebnisse für zwei Schaltungen der ITC'99 Benchmark-Suite [Davidson1999; Corno2000] und vier Industrieschaltungen werden hier gezeigt.

Die Eigenschaften der Schaltungen werden in der Tabelle 4.1 zusammengefasst. In der ersten Spalte werden die Bezeichnungen der Schaltungen aufgeführt. Die nominelle Taktperiode T_{nom} der Schaltung in ns wird in der zweiten Spalte angegeben. Die Anzahl der Logik-Gatter in den Schaltungen wird in der dritten Spalte gezeigt. Die vierte und fünfte Spalte geben die Anzahl der primären und pseudo-primären Eingänge, respektive Ausgänge an. In der sechsten Spalte ist die Anzahl der verwendeten Prüfpfade $|\mathscr{S}|$ gegeben.

In der Spalte $|\mathscr{T}_M|$ ist die Anzahl der verwendeten Testmuster gegeben. Die Anzahl der X-Werte und der detektierbaren Fehler in den zugehörigen Testantworten sind in der achten und neunten Spalte gegeben. [Sprenger2018b; Sprenger2019]

Tabelle 4.1: Eigenschaften der verwendeten Benchmark-Schaltungen [Sprenger2019]

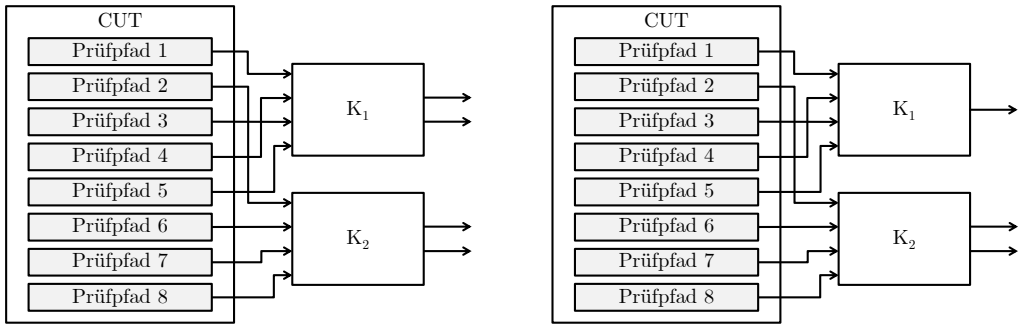
Name	T_{nom} [ns]	# Gatter	# Eingänge	# Ausgänge	$\lvert\mathscr{S}\rvert$	$\lvert\mathscr{T}_M\rvert$	X_{Total}	$\lvert\mathscr{F}(\boldsymbol{T})\rvert$
b17_1	3,588	21 858	1827	1348	32	1935	72 526	48 777
b18_1	4,533	75 618	4116	3085	64	3507	163 536	116 231
bench1	3,191	22 414	3739	2550	64	3596	668 783	64 840
bench2	1,604	78 665	4029	3952	64	3541	1 049 623	363 317
bench3	2,240	56 662	4627	4557	64	6601	511 116	186 959
bench4	3,040	53 836	5902	5829	128	6601	511 116	186 959
bench5	1,511	46 504	3148	3484	64	929	681 995	208 282

Als Testeingabe wurde eine Testmenge für Verzögerungsfehler verwendet, die mithilfe eines kommerziellen Testmustergenerators erzeugt wurde. Für die gezeigten Experimente wurde der Hochgeschwindigkeitstest mit fünf äquidistanten Testfrequenzen zwischen der nominellen Taktfrequenz und der dreifachen Taktfrequenz durchgeführt. Für die zugehörigen Taktperioden gilt $T_{nom} = T_{FAST_1} > T_{FAST_2} > T_{FAST_3} > T_{FAST_4} > T_{FAST_5} = T_{nom}/3$. Für die Prüfgruppeneinteilung wurden alle Testantworten des Tests zu einer Testantwortmatrix $\boldsymbol{T}$ konkateniert.

4.2.3.1 Kompaktierungsrate vs. Prüfgruppengröße

Selbst bei der Verwendung derselben Prüfgruppeneinteilung kann es bei unterschiedlichen Kompaktierungsraten KR zu unterschiedlichen Ergebnissen kommen. Zur Erläuterung des Einflusses der Kompaktierungsrate ist in Abbildung 4.14 ein kleines Beispiel gegeben. In der Abbildung sind zwei Konfigurationen zu sehen, die dieselbe Prüfgruppeneinteilung verwenden. Der einzige Unterschied ist die Kompaktierungsrate des stochastischen Kompaktierers K_1. In der linken Konfiguration wird eine Kompaktierungsrate $KR = \text{Anzahl Eingänge}/\text{Anzahl Ausgänge} = 2$ verwendet, während in der rechten Konfiguration eine Kompaktierungsrate $KR = 4$ für den Kompaktierer K_1 verwendet wird. Die Kompaktierungsrate der rechten Konfiguration steigt damit von 2 auf 2,67.

Zur Untersuchung des Einflusses der Kompaktierungsrate auf den X-Reduktionsfaktor und die Fehlerdurchlässigkeit (FD) wurden Experimente mit variierenden Kompaktierungsraten durchgeführt. Die Ergebnisse sind in den Diagrammen in Abbildung 4.15 und Abbildung 4.16 für die verwendeten Benchmark-Schaltungen zu sehen. Für die gezeigten



(a) Kompaktierungsrate $8/4 = 2$ (b) Kompaktierungsrate $8/3 \approx 2{,}67$

Abbildung 4.14: Kompaktierungsrate vs. Prüfgruppengröße [Sprenger2019].

Ergebnisse wurde die testunabhängige Prüfgruppeneinteilung mit einer Prüfgruppengröße von 32 verwendet. Auf der X-Achse ist die Kompaktierungsrate aufgetragen.

Es konnte gezeigt werden, dass im Allgemeinen der X-Reduktionsfaktor mit der Kompaktierungsrate steigt, wohingegen die Fehlerdurchlässigkeit (FD) mit steigender Kompaktierungsrate fällt. Um vergleichbare Ergebnisse zu erhalten, werden die gezeigten Ansätze nur bei Verwendung der gleichen Kompaktierungsrate verglichen.

Es konnte ebenfalls gezeigt werden, dass die Prüfgruppengröße einen Einfluss auf den X-Reduktionsfaktor (XRF) und die Fehlerdurchlässigkeit (FD) hat. Die Diagramme in der Abbildung 4.17 und Abbildung 4.18 zeigen den Einfluss der Prüfgruppengröße bei der Verwendung des Algorithmus zur testunabhängigen Prüfgruppeneinteilung bei einer Kompaktierungsrate von 2. Diesmal ist auf der X-Achse die Prüfgruppengröße aufgetragen. Während der X-Reduktionsfaktor mit der Prüfgruppengröße sinkt, steigt die Fehlerdurchlässigkeit mit der Prüfgruppengröße.

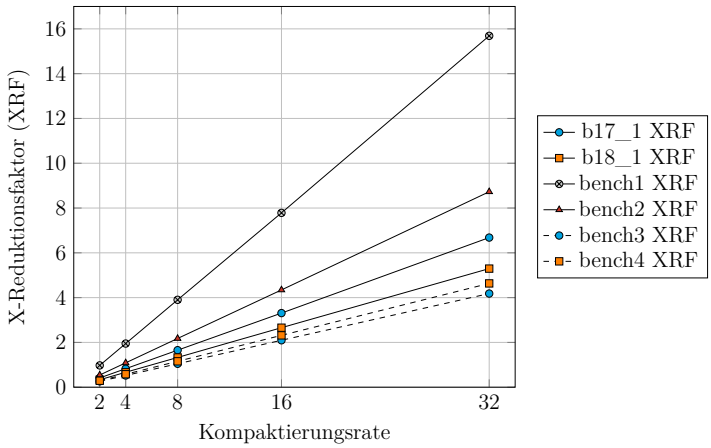


Abbildung 4.15: Einfluss der Kompaktierungsrate auf den X-Reduktionsfaktor (XRF) für die testunabhängige Prüfgruppeneinteilung bei einer Prüfgruppengröße von 32 [Sprenger2019].

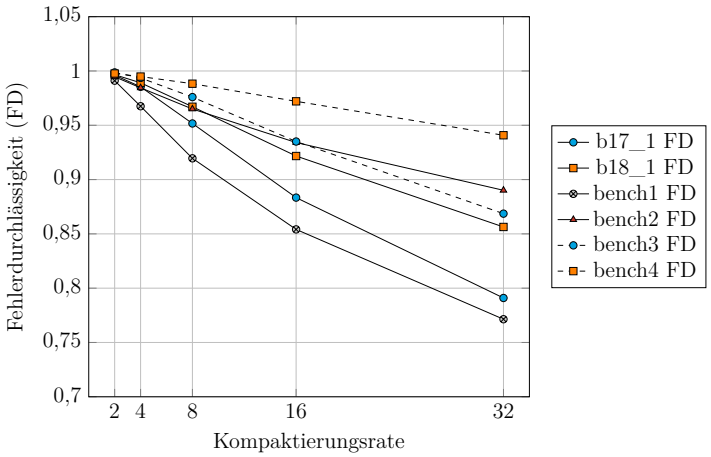


Abbildung 4.16: Einfluss der Kompaktierungsrate auf die Fehlerdurchlässigkeit für die testunabhängige Prüfgruppeneinteilung bei einer Prüfgruppengröße von 32 [Sprenger2019].

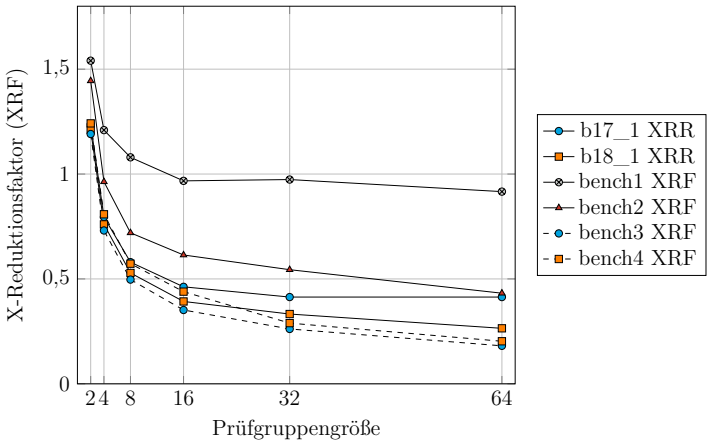


Abbildung 4.17: Einfluss der Prüfgruppengröße auf den X-Reduktionsfaktor (XRF) für die testunabhängige Prüfgruppeneinteilung bei einer Kompaktierungsrate von 2 [Sprenger2019].

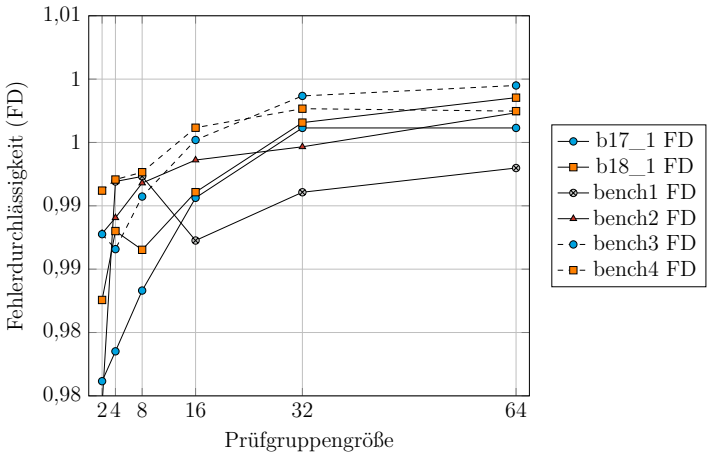


Abbildung 4.18: Einfluss der Prüfgruppengröße auf die Fehlerdurchlässigkeit (FD) für die testunabhängige Prüfgruppeneinteilung bei einer Kompaktierungsrate von 2 [Sprenger2019].

Die in den Abbildungen 4.15 - Abbildung 4.18 dargestellten Ergebnisse sind ebenfalls in der Tabelle 4.2, respektive Tabelle 4.3 zu finden. Die erste Spalte der beiden Tabellen gibt dabei die Kompaktierungsrate KR bzw. die Prüfgruppengröße $|\mathscr{G}_j|$ an, während in den folgenden 12 Spalten abwechselnd der X-Reduktionsfaktor (XRF) und die Fehlerdurchlässigkeit (FD) der Benchmark-Schaltungen angegeben wird.

Tabelle 4.2: Einfluss der Kompaktierungsrate (KR) auf X-Reduktionsfaktor (XRF) und Fehlerdurchlässigkeit (FD) bei einer Prüfgruppengröße von 32

KR	XRF	FD [%]	XRF	FD [%]	XRF	FD [%]	XRF	FD [%]	XRF	FD [%]	XRF	FD [%]
	b17_1		b18_1		bench1		bench2		bench3		bench4	
2	0,41	99,61	0,33	99,66	0,97	99,11	0,54	99,47	0,26	99,87	0,29	99,77
4	0,83	98,54	0,67	98,91	1,95	96,75	1,09	98,45	0,52	99,36	0,58	99,49
8	1,65	95,17	1,33	96,71	3,90	91,96	2,18	96,50	1,05	97,60	1,16	98,89
16	3,31	88,34	2,66	92,17	7,78	85,42	4,34	93,41	2,10	93,50	2,31	97,20
32	6,68	79,10	5,30	85,64	15,69	77,14	8,73	89,01	4,18	0,87	4,63	94,08

Tabelle 4.3: Einfluss der Prüfgruppengröße ($|\mathscr{G}_j|$) auf X-Reduktionsfaktor (XRF) und Fehlerdurchlässigkeit (FD) bei einer Kompaktierungsrate von 2

$\|\mathscr{G}_j\|$	XRF	FD [%]	XRF	FD [%]	XRF	FD [%]	XRF	FD [%]	XRF	FD [%]	XRF	FD [%]
	b17_1		b18_1		bench1		bench2		bench3		bench4	
2	1,24	97,62	1,21	98,27	1,54	97,42	1,44	98,78	1,19	98,78	1,24	99,12
4	0,80	97,85	0,76	98,80	1,21	99,19	0,96	98,91	0,73	98,66	0,81	99,21
8	0,58	98,33	0,53	98,65	1,08	99,23	0,72	99,18	0,50	99,07	0,57	99,27
16	0,46	99,06	0,39	99,11	0,97	98,73	0,61	99,36	0,35	99,52	0,44	99,62
32	0,41	99,61	0,33	99,66	0,97	99,11	0,54	99,47	0,26	99,87	0,29	99,77
64	-	-	0,26	99,85	0,92	99,30	0,43	99,73	0,18	99,95	0,20	99,75

4.2.3.2 Auswertung der Gruppierungsalgorithmen

In den folgenden Experimenten wird untersucht, wie gut das X-Canceling MISR, das in der vorgestellten Architektur auf die räumliche Kompaktierung folgt, mithilfe der modularen Kompaktierung unterstützt werden kann. Aufgrund der in Kapitel 4.2.3.1 dargestellten Experimente wurden die Parameter für den Bin-Packing Algorithmus und für den Algorithmus zur testunabhängigen Prüfgruppeneinteilung so gewählt, dass die größte Prüfgruppengröße erreicht wird, mit der die geforderte Kompaktierungsrate noch eingehalten werden kann. Durch die Maximierung der Prüfgruppengröße wird, wie in Abbildung 4.17 gezeigt, auch der X-Reduktionsfaktor maximiert. Der Benutzerparameter *numRuns* des varianzorientierten Algorithmus wurde über 2^i für $i \in \{0, 1, 2, 3, 4, 5\}$ iteriert, um anschließend das beste Ergebnis auszuwählen.

Zunächst wurde ein Experiment zum Vergleich der heuristischen Algorithmen mit der optimalen Lösung des Problems 4.1 durchgeführt. Eine optimale Lösung kann nur für Schaltungen mit einer geringen Anzahl an Prüfpfaden in angemessener Zeit gefunden werden. Daher wurden für dieses Experiment die Benchmark-Schaltung b17_1 mit 8 (b17_1_8) und 16 (b17_1_16) Prüfpfaden genutzt und für Kompaktierungsraten von 2, 4 und 8 untersucht. Die Gruppierungsalgorithmen wurden für alle Testfrequenzen individuell angewandt, bis auf den Algorithmus zur testunabhängigen Prüfgruppeneinteilung, da dieser für jede Testfrequenz dieselbe Prüfgruppeneinteilung liefern würde.

Tabelle 4.4 und Tabelle 4.5 zeigen die Ergebnisse für den X-Reduktionsfaktor (XRF) und die Fehlerdurchlässigkeit (FD). In beiden Tabellen gibt die erste Spalte die verwendete Schaltung an. In der zweiten Spalte wird dann die Kompaktierungsrate *KR* angegeben. In den folgenden Spalten werden die Ergebnisse für den X-Reduktionsfaktor (XRF), respektive die Fehlerdurchlässigkeit (FD) der verschiedenen Gruppierungsalgorithmen in der folgenden Reihenfolge angegeben. Zunächst sind die Ergebnisse für die Verwendung eines einzelnen stochastischen Kompaktierers, wie in [Mitra2004b] vorgestellt, angegeben. Mit *MILP* sind die Spalten mit den optimalen Ergebnissen markiert, die mithilfe von gemischter ganzzahliger Optimierung (engl. Mixed Integer Linear Programming, MILP) gefunden wurden. Die folgenden drei Spalten geben die Ergebnisse für die heuristischen Methoden der varianzorientierten Prüfgruppeneinteilung (Varianzorientiert), der Bin-Packing Prüfgruppeneinteilung (Bin-Packing) und der testunabhängigen Prüfgruppeneinteilung (Testunabhängig) an.

Es zeigt sich, dass durch die Prüfgruppeneinteilung immer eine Verbesserung des X-Reduktionsfaktors (XRFs) erreicht werden konnte, wobei die Fehlerdurchlässigkeit (FD) in den meisten Fällen nur um 1 % bis 2 % reduziert wurde. Je nach Konfiguration können die besten Ergebnisse mit unterschiedlichen Gruppierungsalgorithmen erreicht werden.

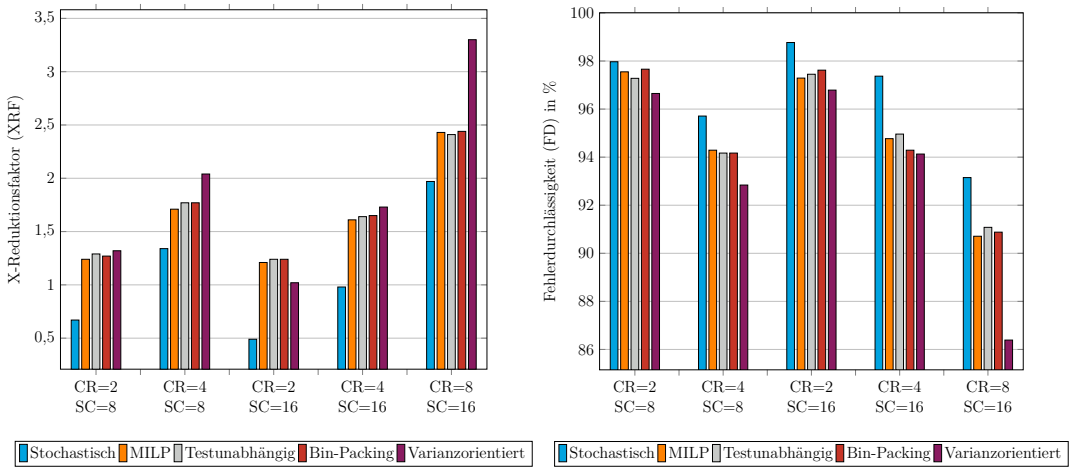
Tabelle 4.4: X-Reduktionsfaktor (XRF) [Sprenger2018b]

Name	KR	XRF Stochastisch	XRF MILP	XRF Varianzorientiert	XRF Bin-Packing	XRF Testunabhängig
b17_1_8	2	0,67	1,24	1,32	1,27	1,29
	4	1,34	1,71	2,04	1,77	1,77
b17_1_16	2	0,49	1,21	1,02	1,24	1,24
	4	0,98	1,61	1,73	1,65	1,64
	8	1,97	2,43	3,30	2,44	2,41

Tabelle 4.5: Fehlerdurchlässigkeit (FD) [Sprenger2018b]

Name	KR	FD Stochastisch	FD MILP	FD Varianzorientiert	FD Bin-Packing	FD Testunabhängig
b17_1_8	2	97,97 %	97,55 %	96,65 %	97,66 %	97,28 %
	4	95,71 %	94,29 %	92,84 %	94,17 %	94,17 %
b17_1_16	2	98,77 %	97,29 %	96,79 %	97,62 %	97,45 %
	4	97,37 %	94,77 %	94,13 %	94,29 %	94,96 %
	8	93,15 %	90,71 %	86,39 %	90,88 %	91,08 %

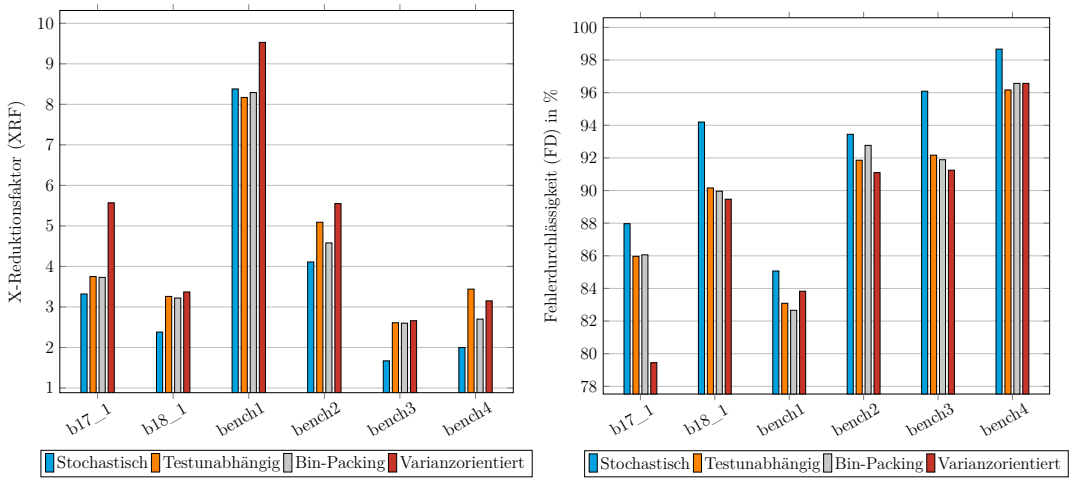
Die Abbildung 4.19 zeigt die Ergebnisse aus den Tabellen 4.4 und 4.5 als Balkendiagramm. In der linken Grafik (Abbildung 4.19a) wird der X-Reduktionsfaktor (XRF) für die Kompaktierungsraten 2, 4 und 8 bei der Verwendung von 8 und 16 Prüfpfaden angegeben. In der rechten Grafik (Abbildung 4.19b) wird die Fehlerdurchlässigkeit (FD) für dieselben Konfigurationen gezeigt. Es zeigt sich, dass der varianzorientierte Algorithmus in den meisten Fällen den besten Kompromiss zwischen X-Reduktionsfaktor (XRF) und Fehlerdurchlässigkeit (FD) erreicht. Nur für eine Kompaktierungsrate $KR = 2$ und bei der Verwendung von 16 Prüfpfaden ($|\mathscr{S}| = 16$) ist der X-Reduktionsfaktor des varianzorientierten Algorithmus geringer als bei alle anderen Heuristiken und als bei der MILP-Lösung.



(a) X-Reduktionsfaktor (b) Fehlerdurchlässigkeit

Abbildung 4.19: Vergleich zwischen optimaler Lösung und Heuristiken für die Schaltung b17_1 [Sprenger2018a; Sprenger2018b; Sprenger2019].

In einem weiteren Experiment wurden größere Schaltungen untersucht. Da in diesen Schaltungen eine größere Anzahl an Prüfpfaden verwendet wurde, wurden nur die Heuristiken zur Prüfgruppeneinteilung bei einer Kompaktierungsrate von 16 verwendet. Wie in der Abbildung 4.20 zu sehen ist, sind die Ergebnisse vergleichbar mit den Ergebnissen des vorherigen Experiments. Die genauen Ergebnisse sind in der Tabelle 4.6 aufgeführt.



(a) X-Reduktionsfaktor (b) Fehlerdurchlässigkeit

Abbildung 4.20: X-Reduktionsfaktor (XRF) und Fehlerdurchlässigkeit (FD) bei zufälliger Prüfpfadkonfiguration [Sprenger2019].

Um die bestmögliche Konfiguration zu erhalten, können in der Praxis alle Heuristiken simuliert werden, um so den bestmöglichen X-Reduktionsfaktor (XRF) und die bestmögliche Fehlerdurchlässigkeit (FD) zu erreichen. Erwähnenswert ist dabei, dass durch die Reduktion der Fehlerdurchlässigkeit (FD) nicht die Fehlerabdeckung des normalen Verzögerungstests vermindert wird, da aufgrund von niedrigeren X-Raten herkömmliche Verfahren zur Testantwortkompaktierung verwendet werden können. Der erhöhte X-Reduktionsfaktor (XRF) führt jedoch zur Reduktion der X-Werte, die das folgende X-Canceling MISR erreichen (vgl. Abbildung 4.7). Dies führt wiederum, wie in Kapitel 2.4.2.2 gezeigt wurde, zu einer Reduktion der zu speichernden Zwischensignaturen.

4.2.3.3 Einfluss der Prüfpfadkonfiguration

Während der Testerzeugung kann häufig kein Einfluss auf die Prüfpfadkonfiguration genommen werden, da diese bereits während der Entwurfsphase unter Berücksichtigung unterschiedlicher Optimierungsziele, wie z. B. Verdrahtungskosten oder Randbedingungen zur Signallaufzeit, festgelegt wurden. Daher wurden die Experimente mit einer weiteren Prüfpfadkonfiguration wiederholt, um den Einfluss der Prüfpfadkonfiguration auf den X-Reduktionsfaktor und die Fehlerdurchlässigkeit zu untersuchen. Während in den bisherigen Experimenten eine zufällige Anordnung der Speicherelemente innerhalb der Prüfpfade verwendet wurde, wurde für die zweite Durchführung eine graphbasierte Prüfpfadkonfiguration verwendet [Kampmann2016; Kampmann2017; Kampmann2018].

In der Tabelle 4.6 werden die Ergebnisse unter Verwendung der zufälligen Prüfpfadkonfiguration aus dem Kapitel 4.2.3.2 mit den Ergebnissen der graphbasierten Prüfpfadkonfiguration verglichen. Die Tabelle 4.6 ist wie folgt aufgebaut. Die erste Spalte gibt die zugrunde liegende Prüfpfadkonfiguration an und die zweite Spalte den verwendeten Gruppierungsalgorithmus. Die folgenden zwölf Spalten geben abwechselnd den X-Reduktionsfaktor (XRF) und die Fehlerdurchlässigkeit (FD) in % für die sechs verwendeten Benchmark-Schaltungen an.

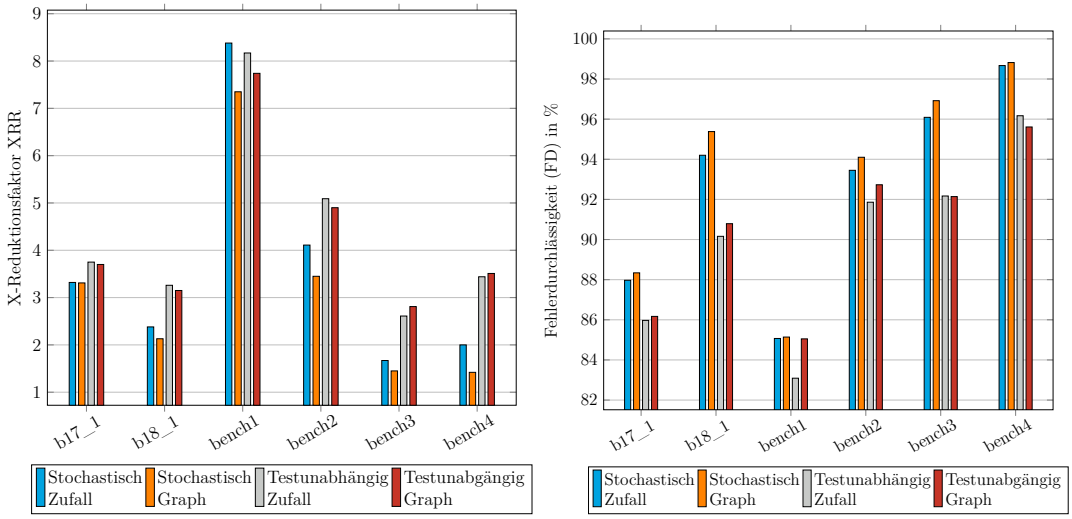
Tabelle 4.6: Vergleich zwischen zufälliger und graphbasierter Prüfpfadkonfiguration [Sprenger2019]

Prüfpfad		XRF	FD	XRF	FD	XRF	FD	XRF	FD	XRF	FD	XRF	FD
Konfig.	Algorithmus	b17_1		b18_1		bench1		bench2		bench3		bench4	
Zufall	Stochastisch	3,32	87,97	2,38	94,20	8,38	85,07	4,11	93,45	1,67	96,09	2,00	98,67
	Testunabhängig	3,75	85,97	3,26	90,16	8,17	83,09	5,09	91,86	2,61	92,17	3,44	96,17
	Bin-Packing	3,73	86,06	3,22	89,96	8,29	82,66	4,58	92,77	2,60	91,89	2,70	96,57
	Varianzorientiert	5,57	79,45	3,37	89,47	9,53	83,83	5,55	91,10	2,66	91,25	3,15	96,57
Graph	Stochastisch	3,31	88,34	2,13	95,38	7,35	85,14	3,45	94,10	1,45	96,92	1,42	98,82
	Testunabhängig	3,70	86,17	3,15	90,79	7,74	85,05	4,90	92,73	2,81	92,14	3,51	95,61
	Bin-Packing	3,52	86,07	2,59	92,85	7,35	80,32	4,28	93,44	1,85	93,56	1,72	96,26
	Varianzorientiert	5,33	80,54	3,96	86,79	11,75	94,96	5,70	89,86	4,11	90,03	2,87	96,83
	Blockierung	3,77	81,93	3,23	86,36	7,55	83,95	5,08	92,69	3,07	91,57	3,71	92,63
	Bypass	2,95	87,19	2,77	91,38	7,13	85,59	3,77	93,04	2,71	93,32	2,58	96,22

Es zeigt sich, dass die Prüfpfadkonfiguration nur einen geringen Einfluss auf den X-Reduktionsfaktor (XRF) und die Fehlerdurchlässigkeit (FD) hat, wobei die zufällige Prüfpfadkonfiguration leicht bessere Ergebnisse für den X-Reduktionsfaktor erzeugt. Auch in diesem Experiment zeigt der varianzorientierte Algorithmus bessere Ergebnisse als die normale Anwendung des stochastischen Kompaktierers. Wie in Abbildung 4.21 zu sehen ist, zeigt aber auch der Algorithmus zur testunabhängigen Prüfgruppeneinteilung bereits gute Ergebnisse.

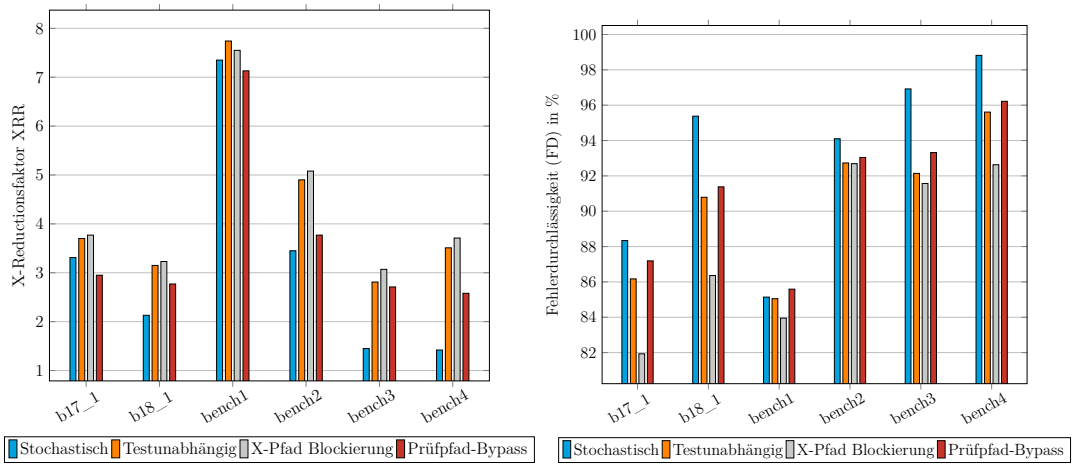
4.2.3.4 X-Pfad-Blockierung und Pfad-Bypass für essenzielle Prüfpfade

Wie im vorherigen Abschnitt gezeigt wurde, hat die Prüfpfadkonfiguration nur einen geringen Einfluss auf die Qualität des gezeigten Kompaktierungsverfahrens. In einem weiteren Experiment wurde untersucht, wie weitere Information über die Prüfpfadkonfiguration, wie in Kapitel 4.2.2 beschrieben, genutzt werden kann, um die Qualität der Kompaktierung weiter zu verbessern.



(a) X-Reduktionsfaktor (XRF) (b) Fehlerdurchlässigkeit (FD)

Abbildung 4.21: Vergleich von X-Reduktionsfaktor (XRF) und Fehlerdurchlässigkeit (FD) für zufällige und graphbasierte Prüfpfadkonfigurationen [Sprenger2019].



(a) X-Reduktionsfaktor (XRF) (b) Fehlerdurchlässigkeit (FD)

Abbildung 4.22: Einfluss von X-Pfad-Blockierung und Pfad-Bypass auf X-Reduktionsfaktor (XRF) und Fehlerdurchlässigkeit (FD) bei einer Kompaktierungsrate von 16 [Sprenger2019].

Für dieses Experiment wurde zunächst der Prüfpfad mit der höchsten X-Rate (X-Pfad) und der Prüfpfad mit der größten Fehlermenge (essenzieller Prüfpfad) in der graphbasierten Prüfpfadkonfiguration bestimmt und während der Phase der räumlichen Kompaktierung blockiert (X-Pfad-Blockierung) oder an der räumlichen Kompaktierung vorbeigeführt (Pfad-Bypass).

In der Abbildung 4.22 werden der X-Reduktionsfaktor (Abbildung 4.22a) und die Fehlerdurchlässigkeit (Abbildung 4.22b) für die Verwendung eines einzelnen stochastischen Kompaktierers [Mitra2004b], der modularen Kompaktierung mit einer Prüfgruppeneinteilung nach dem Algorithmus zur testunabhängigen Prüfgruppeneinteilung (Testunabhängig), dem zusätzlichen Blockieren von X-führenden Prüfpfaden (X-Pfad-Blockierung) und der Umgehung für essenzielle Prüfpfade (Pfad-Bypass) miteinander verglichen. Die genauen Ergebnisse sind in der Tabelle 4.6 zu finden.

Wie der Tabelle 4.6 zu entnehmen ist, wird durch die X-Pfad-Blockierung der X-Reduktionsfaktor für alle Schaltungen im Vergleich zur stochastischen Kompaktierung erhöht, wobei die Fehlerdurchlässigkeit reduziert wird. Entsprechend wird durch die Verwendung eines Prüfpfad-Bypasses die Fehlerdurchlässigkeit für alle gezeigten Schaltungen im Vergleich zur modularen Kompaktierung ohne Prüfpfad-Bypass erhöht. Wie zu erwarten war, wird jedoch der X-Reduktionsfaktor durch den Prüfpfad-Bypass verringert. Je nach Anwendung kann so auf Anforderungen an den X-Reduktionsfaktor oder die Fehlerdurchlässigkeit reagiert werden.

4.2.4 Fazit

In diesem Kapitel wurde mit der modularen Kompaktierung ein stochastisches Kompaktierungsverfahren vorgestellt, welches es ermöglicht mithilfe von programmierbaren gewichteten Zufallssignalen auf variierende X-Raten, wie sie unter anderem während des Hochgeschwindigkeitstests entstehen, zu reagieren. Dabei ist die Anwendung nicht auf die variierenden X-Raten während des Hochgeschwindigkeitstests beschränkt. Andere Quellen von hohen oder variierenden X-Raten wie Alterungseffekte oder *IR-drop* [Ahmed2006; Ahmed2010] können ebenfalls gehandhabt werden. Hierzu wurde der Kompaktierer in mehrere kleine Kompaktierer aufgeteilt, welche disjunkte Prüfgruppen der Schaltungsausgänge kompaktieren. So ist es möglich den X-Reduktionsfaktor zu erhöhen und die Fehlerdurchlässigkeit aufrecht zu erhalten.

Die Gruppierungsalgorithmen können an benutzerdefinierte Bedingungen angepasst werden, um so bestimmte Prüfgruppengrößen oder eine bestimmte Anzahl an Prüfgruppen zu erreichen. Mit dem Algorithmus zur testunabhängigen Prüfgruppeneinteilung ist außerdem ein testunabhängiger Gruppierungsalgorithmus gegeben. Durch die X-Pfad-Blockierung oder den Pfad-Bypass kann außerdem auf benutzerspezifische Anforderungen an den X-Reduktionsfaktor oder die Fehlerdurchlässigkeit eingegangen werden.

4.3 Hybride Kompaktierung

Wie in Kapitel 4.2 gezeigt wurde, kann mithilfe der modularen Kompaktierung auf die variierenden X-Raten des Hochgeschwindigkeitstest eingegangen werden. Während der X-Reduktionsfaktor (XRF) mithilfe des vorgestellten Kompaktierungsverfahrens erhöht werden konnte, konnte die Maskierung von Fehlerinformation während des Verfahrens nicht ausgeschlossen werden. Um diesen Nachteil auszugleichen, wurde in [Maaz2019b] und [Maaz2019a] ein hybrides Kompaktierungsverfahren vorgestellt, das die stochastische Phase durch eine deterministische Phase ergänzt.

Zur Untersuchung der verlorenen Fehlerinformation wurde zunächst ein Experiment durchgeführt mit demselben Versuchsaufbau wie in Kapitel 4.2.3. Für jedes Testmuster wurde die Fehlerinformation vor und nach der stochastischen Kompaktierung bestimmt, um festzustellen, welche Fehlerinformation pro Testmuster verloren gegangen ist. Sollte die Fehlerinformation in einer anderen Testfrequenz am Ausgang des stochastischen Kompaktierers sichtbar sein, wird die Fehlerinformation aus der Liste der verlorenen Fehler gestrichen. So konnte für jedes Muster innerhalb der Testmustermenge eine Liste der *tatsächlich* verlorenen Fehler erzeugt werden. In der Abbildung 4.23 sind die verlorenen Fehler für die ersten 100 Testmuster für die höchste Testfrequenz für eine Industrieschaltung aufgetragen. Während auf der X-Achse der Testmusterindex aufgeführt ist, ist auf der Y-Achse die Anzahl der verlorenen Fehler angegeben.

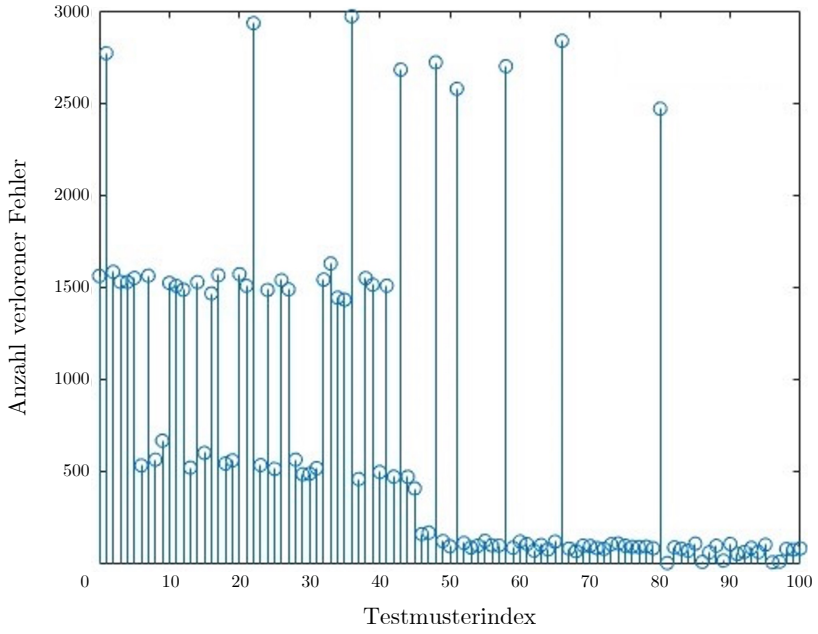


Abbildung 4.23: Verlorene Fehler für die Benchmark-Schaltung *bench5* (KR=8) [Maaz2019a].

Es fällt auf, dass für einzelne Testmuster sehr viel Fehlerinformation verloren geht. Durch die Rückgewinnung der Fehlerinformation in diesen wenigen Testmustern könnte die Fehlerdurchlässigkeit (FD) bereits signifikant erhöht werden. Um dies zu erreichen, wurde das in Kapitel 4.2 erläuterte stochastische Verfahren zur modularen Kompaktierung um eine deterministische Phase zu einem hybriden Kompaktierer erweitert. Aus Gründen der Übersichtlichkeit wird das Verfahren hier an einem Kompaktiererblock erläutert. Das Verfahren kann jedoch auch auf kleinere Kompaktiererblöcke angewendet werden.

Die Idee des hybriden Kompaktierers ist es, einige wenige Testmuster nach der stochastischen Phase der Kompaktierung noch einmal mithilfe einer spezifischen Konfiguration der Kompaktormatrix zu kompaktieren. Hierzu wurde die Architektur der modularen Kompaktierung, wie in Abbildung 4.24 gezeigt, um einen Speicher für deterministische Steuerungsvektoren, einen Multiplexer und einen kleinen endlichen Zustandsautomat erweitert. Mithilfe des Zustandsautomaten wird zwischen der stochastischen und der deterministischen Phase umgeschaltet, um mithilfe des Multiplexers die deterministischen Steuerungsvektoren an die UND-Netzwerke des stochastischen Kompaktierers weiterleiten zu können. So ist es möglich, in Abhängigkeit der Testantwoten, spezifisch X-Werte zu blockieren und Fehlermaskierung zu vermeiden.

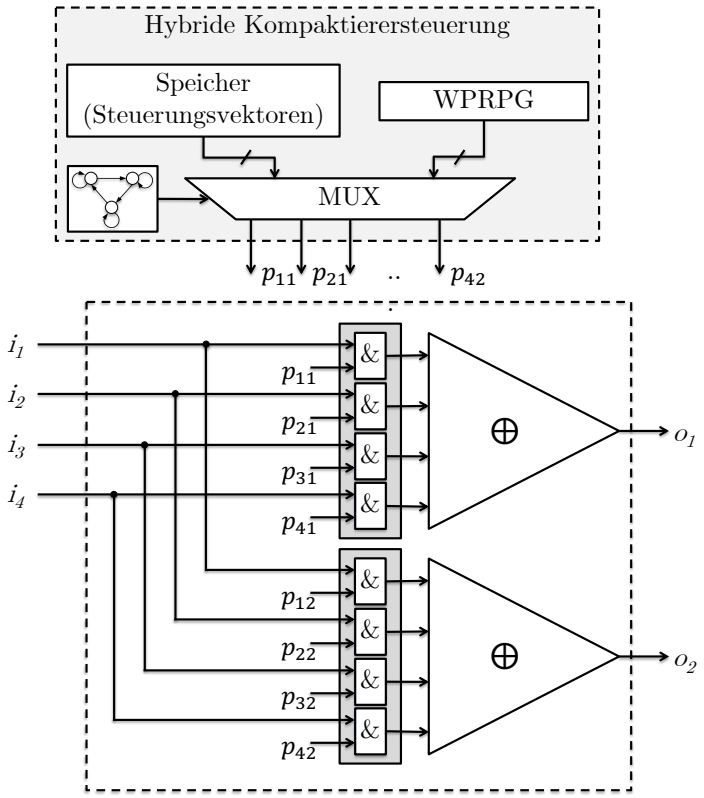


Abbildung 4.24: Hybride Kompaktierung [Maaz2019b; Maaz2019a].

Zum Entwurf eines hybriden Kompaktierers sind zwei Schritte notwendig.

1. Identifikation der n kritischen Testmuster
2. Bestimmung der deterministischen Steuerungsvektoren

Während der erste Schritt bereits in diesem Kapitel erläutert wurde, wird im folgenden Abschnitt auf den zweiten Schritt eingegangen.

4.3.1 Bestimmung der deterministischen Steuerungsvektoren für die hybride Kompaktierung

Häufig ist es das Ziel, Kompaktierer zu entwerfen, die unabhängig von der Testmustermenge bestimmte Eigenschaften erfüllen wie z. B. bei den bereits erwähnten codebasierten linearen Kompaktierern [Reddy1988; Mitra2004a]. Bisher gibt es nur wenige Ansätze, welche die Testmustermenge beim Entwurf des Kompaktierers berücksichtigen [Chakrabarty1998; Morosov2001]. Während bei der Berücksichtigung der Testmustermenge direkt die Fehlermaskierung verhindert werden kann, hat es den Nachteil, dass bei der Veränderung oder Ergänzung der Testmustermenge auch die Konfiguration des Kompaktierers angepasst werden muss. Wie in Kapitel 4.1.2 gezeigt wurde, erlaubt die Struktur des stochastischen Kompaktierers bereits die Rekonfiguration des Kompaktierers, was es ermöglicht Testmuster-spezifische Steuerungsvektoren zur Konfiguration zu verwenden. Um den benötigten Speicher für die Steuerungsvektoren zu beschränken, kann die Granularität der Rekonfigurationen angepasst werden. Im Weiteren wird eine Rekonfiguration pro Testmuster angenommen.

Die Testantworten der kritischen Testmuster, d. h. der Testantworten mit verlorener Fehlerinformation, sollen so kompaktiert werden, dass die vorhandenen X-Werte nicht propagiert werden und die Fehlerinformation nicht maskiert wird. Das Problem zur Definition des besten Steuerungsvektors für eine gegebene Testantwortmatrix kritischer Testmuster kann wie folgt definiert werden.

Problem 4.3 (Optimale Kompaktierer Konfiguration (OKK))**.** *Gegeben sei eine Testantwortmatrix* $\boldsymbol{T} = (t_{il})$ *und eine Kompaktormatrix* $\boldsymbol{C} = (c_{ij})$*. Finde eine Belegung der Einträge* $c_{ij} \in \{0, 1\}$*, sodass die Fehlerdurchlässigkeit (FD) und der X-Reduktionsfaktor (XRF) nach der Kompaktierung von* $\boldsymbol{T}$ *maximal ist.*

Mithilfe der Testantwortmatrix $\boldsymbol{T}$ und der Kompaktormatrix $\boldsymbol{C}$ lässt sich die Signaturmatrix $\boldsymbol{S}$ mit (2.7) bestimmen. Die Fehlerdurchlässigkeit und der X-Reduktionsfaktor wird, wie in Kapitel 4.2.1.2 erläutert, berechnet.

Wie bereits in Kapitel 4.1.1 gezeigt wurde, beschreibt die Kompaktormatrix $\boldsymbol{C}$, wie die Ausgänge der zu testenden Schaltung mit den XOR-Bäumen des Kompaktierers verbunden werden. Um das Problem 4.3 zu lösen, müssen die Eingänge des Kompaktierers in Eingangsgruppen eingeteilt werden, die dann den einzelnen XOR-Bäumen eines stochastischen Kompaktierers zugewiesen werden.

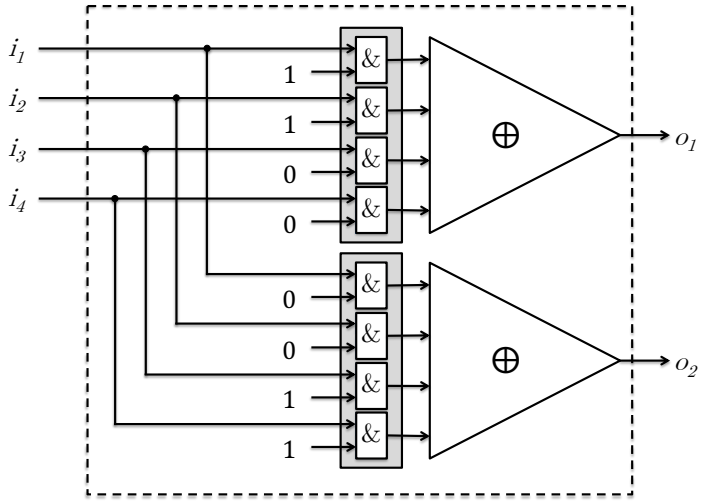


Abbildung 4.25: Beispiel einer Eingangsgruppeneinteilung für die hybride Kompaktierung.

In Abbildung 4.25 ist ein Beispiel mit vier Prüfpfaden und einem stochastischen Kompaktierer mit einer Kompaktierungsrate von 2 gegeben. Die Prüfpfade in diesem Beispiel sind in zwei Eingagnsgruppen eingeteilt. Die erste Eingangsgruppe $\mathscr{G}_1 = \{i_1, i_2\}$, bestehend aus dem ersten und zweiten Eingang des Kompkatierers, soll mithilfe des ersten XOR-Baumes kompaktiert werden und die zweite Eingangsgruppe $\mathscr{G}_2 = \{i_3, i_4\}$ mithilfe des zweiten. Die Kompaktormatrix ergibt sich somit zu

$$\boldsymbol{C} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}. \tag{4.12}$$

Zur Reduktion des Suchraums wurden zunächst nur disjunkte Teilmengen der Menge der Prüfpfade $\mathscr{S}$ berücksichtigt.

Die vorgestellte Eingangsgruppeneinteilung ist ein heuristisches Verfahren zur Lösung des OKK Problems 4.3 und basiert auf dem Ähnlichkeits-Index η, der wie folgt definiert ist.

Definition 4.2 (Ähnlichkeitsindex)**.** *Sei $\boldsymbol{T} = (t_{il})$ eine $m \times n$ Testantwortmatrix. t_i und $t_{i'}$ seien zwei Zeilen der Matrix $\boldsymbol{T}$. Der Ähnlichkeits-Index $\eta(t_i, t_{i'})$ ist dann definiert als*

$$\eta(t_i, t_{i'}) = \sum_{l=1}^{n} \eta(t_{il}, t_{i'l}), \tag{4.13}$$

mit

$$\eta(t_{il}, t_{i'l}) = \begin{cases} 1 & , \text{ wenn } t_{il} = t_{i'l} = X \\ -1 - 0{,}5 \cdot |\mathscr{F}(t_{il})| & , \text{ wenn } t_{il} \neq X, t_{i'l} = X \\ -1 - 0{,}5 \cdot |\mathscr{F}(t_{i'l})| & , \text{ wenn } t_{il} = X, t_{i'l} \neq X \\ -0{,}5 \cdot |\mathscr{F}(t_{il}) \cap \mathscr{F}(t_{i'l})|, & \text{sonst.} \end{cases}$$

Während der Kompaktierung werden alle Eingänge des Kompaktierers, die sich in einer Eingangsgruppe befinden XOR-verknüpft. Mithilfe des Ähnlichkeits-Index soll bewertet werden, wie sehr die XOR-Verknüpfung zweier Prüfpfade den X-Reduktionsfaktor (XRF) und die Fehlerdurchlässigkeit (FD) erhöht. Werden zwei X-Werte zusammengefasst ($t_{il} = t_{i'l} = X$) erhöht sich der X-Reduktionsfaktor. Daher wird der Ähnlichkeits-Index um 1 erhöht. Wird ein deterministisches Bit mit einem X-Wert verknüpft ($t_{il} \neq X, t_{i'l} = X$ oder $t_{il} = X, t_{i'l} \neq X$), wird der X-Wert propagiert und etwaige Fehlerinformation geht verloren. Für diesen ungewollten Fall wird der Ähnlichkeits-Index reduziert. Für das Propagieren des X-Wertes wird der Index um 1 reduziert. Zusätzlich wird für jeden Fehler, der verloren geht, der Index um 0,5 verringert. Im letzten Fall werden zwei deterministische Bits miteinander verknüpft. Handelt es sich hierbei um D-Bits, welche dieselben Fehler transportieren, kommt es zur Fehlermaskierung, die vermieden werden soll. Daher wird für jeden maskierten Fehler der Index ebenfalls um 0,5 reduziert.

Zur Erläuterung ist in Abbildung 4.26 ein Beispiel mit zwei Prüfpfaden und fünf Taktzyklen gegeben. Die X-Werte sind mit X markiert. Deterministische Bits werden mithilfe der Fehlermenge $\mathscr{F}(t_{il})$ dargestellt. Im ersten Taktzyklus werden zwei X-Werte miteinander verknüpft, daher ergibt sich ein Ähnlichkeits-Wert $\eta(t_{il}, t_{i'l})$ von 1. Die zweiten Bits

der Prüfpfade tragen keine Fehlerinformation. Bei der Kombination der beiden Bits geht daher auch keine Fehlerinformation verloren und nach Definition 4.2 ergibt sich der Ähnlichkeits-Wert zu 0. Im dritten Taktzyklus wird ein X-Wert propagiert und es geht die Information eines Fehlers verloren. Somit wird 1,5 vom Ähnlichkeits-Index abgezogen. Bei der Verknüpfung der vierten Bits werden die Fehler φ_1 und φ_2 maskiert, der Ähnlichkeits-Index muss daher um 1 reduziert werden. Der Ähnlichkeits-Wert für den fünften Taktzyklus ist $-1{,}5$, da wie im Taktzyklus 3 ein X-Wert propagiert wird und ein Fehler verloren geht. Der Ähnlichkeits-Index ergibt sich somit zu $1 + 0 - 1{,}5 - 1 - 1{,}5 = -3$.

	Taktzyklus 1	Taktzyklus 2	Taktzyklus 3	Taktzyklus 4	Taktzyklus 5
Prüfpfad 1	X	{}	X	$\{\varphi_1, \varphi_2, \varphi_4\}$	$\{\varphi_3\}$
Prüfpfad 2	X	{}	$\{\varphi_6\}$	$\{\varphi_1, \varphi_2, \varphi_5\}$	X
Ähnlichkeits-Wert	1	0	-1,5	-1	-1,5

Abbildung 4.26: Beispiel für den Ähnlichkeits-Index η [Maaz2019a].

Definition 4.3 (Essenzielles D-Bit)**.** *Sei $\boldsymbol{T} = (t_{il})$ eine $m \times n$ Testantwortmatrix und $t_{i'l'} \neq X$ sei ein D-Bit. Dann ist $t_{i'l'}$ ein essenzielles D-Bit in Bezug auf $\boldsymbol{T}$, wenn $\mathscr{F}(t_{i'l'})$ einen Fehler enthält, welcher in keiner anderen Fehlermenge $\mathscr{F}(t_{il})$ enthalten ist für $1 \leq i \neq i' < m$ und $1 \leq l \neq l' < n$.*

Mithilfe der so definierten, essenziellen D-Bits lässt sich der Ähnlichkeits-Index anpassen, um den Verlust von essenzieller Fehlerinformation zu verhindern. Durch die Testantwortmatrix können mithilfe der Definition 4.3 die essenziellen D-Bits für die hybride Kompaktierung bestimmt werden.

In Algorithmus 4.4 ist ein gieriger (engl. greedy) Algorithmus zur Eingangsgruppeneinteilung aufgrund des Ähnlichkeits-Index $\eta(t_i, t_{i'})$ angegeben. Die Eingabeparameter des Gruppierungsalgorithmus sind die Anzahl der Eingänge und Ausgänge des Kompaktierers m und n, die Testantwortmatrix $\boldsymbol{T}$, sowie die Menge der Prüfpfade $\mathscr{S}$. Zunächst wird die Eingangsgruppengröße $size = \lceil m/n \rceil$ bestimmt. Für die ersten $n-1$ Ausgänge, die je einen XOR-Baum repräsentieren, wird in der Zeile 6 eine Eingangsgruppe ($\mathscr{G}_j^{In}$) mithilfe der Funktion *newCluster* gebildet. Die in $\mathscr{G}_j^{In}$ verwendeten Prüfpfade werden anschließend aus der Menge der Prüfpfade $\mathscr{S}$ und die entsprechenden Zeilen aus der Testantwortmatrix $\boldsymbol{T}$ entfernt. Aus den verbleibenden Prüfpfaden wird die Eingangsgruppe für den n-ten Ausgang $\mathscr{G}_n^{In}$ gebildet.

Algorithmus 4.4 Eingangsgruppeneinteilung [Maaz2019a]

function EINGANGSGRUPPENEINTEILUNG(m, n, $\boldsymbol{T}$, $\mathscr{S}$)
    size $\leftarrow \lceil m/n \rceil$
    $\mathscr{G}^{In} \leftarrow \emptyset$
    for $j \leftarrow 1$ to (n-1) **do**
        $\mathscr{G}_j^{In} \leftarrow newCluster(size, \mathscr{S}, \boldsymbol{T})$
        $\mathscr{S} \leftarrow \mathscr{S} \setminus \mathscr{G}_j^{In}$
        Löschen der Prüfpfade aus $\mathscr{G}_j^{In}$ in $\boldsymbol{T}$
        $\mathscr{G}^{In} \leftarrow \mathscr{G}^{In} \cup \mathscr{G}_j^{In}$
    end for
    $\mathscr{G}_n^{In} \leftarrow \mathscr{S}$
    $\mathscr{G}^{In} \leftarrow \mathscr{G}^{In} \cup \mathscr{G}_n^{In}$
end function

Algorithmus 4.5 Algorithmus zur Bildung neuer Eingangsgruppen [Maaz2019a]

function NEWCLUSTER($size$, $\mathscr{S}$, $\boldsymbol{T}$)
    $\mathscr{G}_{new}^{In} \leftarrow \emptyset$
    calculate $\eta(t_i, t_{i'}) \forall i \neq i' \in \{1, \ldots, m\}$
    $s_i, s_{i'} \leftarrow \arg\max_{i \neq i' \in \{1,\ldots,m\}} \eta(t_i, t_{i'})$
    $\mathscr{G}_{new}^{In} \leftarrow \mathscr{G}_{new}^{In} \cup \{s_i, s_{i'}\}$
    delete rows corresponding to $\mathscr{G}_{new}^{In}$ in $\boldsymbol{T}$
    $t = t_i \oplus t_{i'}$
    while $|\mathscr{G}_{new}^{In}| < size$ **do**
        calculate $\eta(t_i, t) \forall i \in \{1, \ldots, numRows(\boldsymbol{T})\}$
        $s_i, \leftarrow \arg\max_{i \in \{1,\ldots,numRows(\boldsymbol{T})\}} \eta(t_i, t)$
        $\mathscr{G}_{new}^{In} \leftarrow \mathscr{G}_{new}^{In} \cup \{s_i\}$
        delete rows corresponding to s_i in $\boldsymbol{T}$
        $t \leftarrow t \oplus t_i$
    end while
end function

Die essenzielle Funktion im Algorithmus 4.4 ist die Funktion *newCluster*, welche eine neue Eingangsgruppe der Größe $size$ aus der Menge der Prüfpfade $\mathscr{S}$ und der Testantwortmatrix $\boldsymbol{T}$ bildet. Wie in Algorithmus 4.5 gezeigt wurde, wird hierzu zunächst für alle Prüfpfadkombinationen der Ähnlichkeits-Index $\eta(t_i, t_{i'})$ berechnet. Anschließend werden die beiden Prüfpfade s_i und $s_{i'}$, die den Ähnlichkeits-Index maximieren, ausgewählt und der neuen Eingangsgruppe $\mathscr{G}_{new}^{In}$ hinzugefügt und die entsprechenden Reihen aus $\boldsymbol{T}$ entfernt. Daraufhin wird die XOR-Verknüpfung der beiden Zeilen t_i und $t_{i'}$ der Testantwortmatrix gebildet. Der Zeilenvektor t repräsentiert damit die kompaktierte Testantwort der bisherigen Eingangsgruppe $\mathscr{G}_{new}^{In}$. Bis die gewünschte Größe der Eingangsgruppe erreicht wird, wird in den Zeilen 7-13 der Ähnlichkeits-Index der aktuellen Eingangsgruppe mit allen verbleibenden Prüfpfaden berechnet ($\eta(t_i, t) \forall i \in \{1, \ldots, numRows(\boldsymbol{T})\}$) und der Prüfpfad mit dem höchsten Ähnlichkeits-Index der aktuellen Eingangsgruppe hinzugefügt.

Da als Granularität in dieser Arbeit ein Testmuster verwendet wurde, wird der Steuerungsvektor nur einmal pro Testmuster verändert. Daher ist es in der deterministischen Phase des hybriden Kompaktierers nicht immer möglich, 100 % Fehlerdurchlässigkeit zu erreichen. Falls 100 % Fehlerdurchlässigkeit erreicht werden soll, kann für jeden Taktzyklus ein optimaler Steuerungsvektor bestimmt werden. Dies hat jedoch den Nachteil, dass die benötigte Speichergröße zum Speichern der deterministischen Steuerungsvektoren ansteigt. Sollte die Speichergröße für die Granularität eines Testmusters nicht ausreichen, ist es außerdem möglich, mehrere Testmuster zusammenzufassen.

4.3.2 Experimente

Zur Evaluierung des vorgestellten hybriden Kompaktierungsverfahrens wurden Experimente für eine ITC'99 Benchmark-Schaltung [Davidson1999; Corno2000] und fünf Industrieschaltungen durchgeführt. In der Tabelle 4.7 sind die Schaltungseigenschaften zusammengefasst. In der ersten Spalte sind die Bezeichnungen der Benchmark-Schaltungen zu sehen. Die Anzahl der Ein- und Ausgänge sind in der zweiten und dritten Spalte zu finden. Die vierte Spalte zeigt die Anzahl der implementierten Prüfpfade. Die Kardinalität der Testmustermenge ist in der fünften Spalte dargestellt. Die Anzahl der detektierbaren versteckten kleinen Verzögerungsfehler und die Anzahl der X-Werte in den Testantworten sind in den letzten beiden Spalten angegeben.

Für alle Versuche wurde ein Hochgeschwindigkeitstest mit fünf Testfrequenzen und einer stochastischen Kompaktierung durchgeführt. Die Testfrequenzen wurden äquidistant im

Tabelle 4.7: Schaltungseigenschaften

CUT	# Eingänge	# Ausgänge	$\lvert\mathscr{S}\rvert$	$\lvert\mathscr{T}_M\rvert$	$\lvert\mathscr{F}(\boldsymbol{T})\rvert$	X_{Total}
b18_1	4116	3085	64	3507	116 231	163 536
bench1	3739	2550	64	3596	64 840	668 783
bench2	4029	3952	64	3541	363 317	1 049 623
bench3	4627	4557	64	6600	186 959	511 116
bench4	5902	5829	128	5506	145 221	339 691
bench5	3148	3484	64	929	208 282	681 995

Intervall $[f_{nom}, 3 \cdot f_{nom}] \in \mathbb{R}$ verteilt. Der stochastische Kompaktierer wurde entsprechend den Vorgaben in [Mitra2004b] konfiguriert. Nach der stochastischen Phase wurde während der deterministischen Phase eine Teilmenge der Testmustermenge noch einmal auf die Testschaltung angewandt. Zur Eingangsgruppeneinteilung wurde der Algorithmus 4.4 verwendet. In einem weiteren Experiment wurde außerdem ein Referenzversuch mit einem Steuerungsvektor pro Taktzyklus durchgeführt (Takt-Granularität).

Zunächst wurde die Fehlerdurchlässigkeit (FD) untersucht. In der Tabelle 4.8 werden die Fehlerdurchlässigkeit für die stochastische Phase der Kompaktierung (Stochastisch) mit der deterministischen Phase verglichen bei einer Granularität eines Testmusters (Hybrid) und eines Taktes (Hybrid pro Takt). Als Kompaktierungsrate wurde 16 und 32 verwendet. Für die Experimente wurden 10 % der Testmustermenge ein weiteres Mal angewendet.

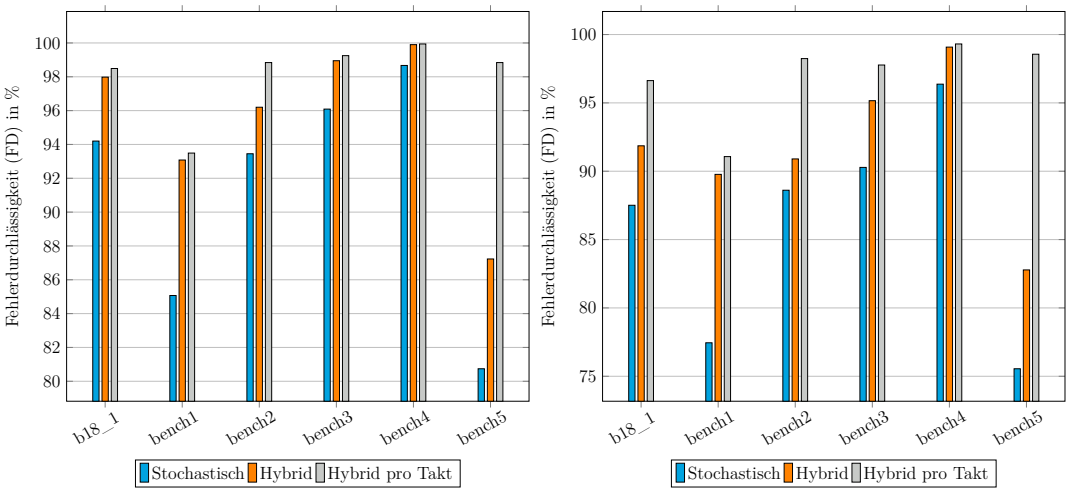
Tabelle 4.8: Fehlerdurchlässigkeit (FD) für $KR = 16$ und $KR = 32$ bei der Wiederanwendung von 10 % der Testmuster [Maaz2019a]

CUT	Kompaktierungsrate 16			Kompaktierungsrate 32		
	Stochastisch	Hybrid	Hybrid pro Takt	Stochastisch	Hybrid	Hybrid pro Takt
b18_1	94,20 %	97,98 %	98,49 %	87,51 %	91,86 %	96,63 %
bench1	85,07 %	93,08 %	93,49 %	77,45 %	89,77 %	91,07 %
bench2	93,45 %	96,20 %	98,84 %	88,61 %	90,90 %	98,24 %
bench3	96,09 %	98,95 %	99,25 %	90,28 %	95,16 %	97,77 %
bench4	98,67 %	99,90 %	99,94 %	96,37 %	99,08 %	99,31 %
bench5	80,74 %	87,23 %	98,84 %	75,55 %	82,78 %	98,56 %

Es zeigt sich, dass durch die Wiederanwendung der kritischen Testmuster in einer deterministischen Phase die Fehlerdurchlässigkeit, bei der Verwendung eines Steuerungsvektors pro Testmuster, gesteigert werden konnte. Die kleinste Verbesserung von 1,23 % wurde für die bench4 Benchmark-Schaltung erzielt bei einer Kompaktierungsrate von 16. Mit 12,31 % konnte die größte Verbesserung für die Schaltung bench1 bei einer

Kompaktierungsrate von 32 erreicht werden. Bei einer Verwendung von individuellen Steuerungsvektoren pro Taktzyklus kann die Fehlerdurchlässigkeit sogar bis zu 23,01 % verbessert werden.

In der Abbildung 4.27 ist eine graphische Repräsentation der Ergebnisse bei einer Kompaktierungsrate von 16 zu finden. Auf der Y-Achse ist die Fehlerdurchlässigkeit (FD) in % aufgetragen, während auf der X-Achse die Benchmark-Schaltungen aufgeführt sind.



(a) Kompaktierungsrate = 16 (b) Kompaktierungsrate = 32

Abbildung 4.27: Stochastische Kompaktierung vs. hybride Kompaktierung - Vergleich der Fehlerdurchlässigkeit (FD) bei einem Kompaktierungsrate (KR) von 16 und 32.

In einem weiteren Experiment wurde der Einfluss der Testmusteranzahl, in der deterministischen Phase, auf die Fehlerdurchlässigkeit untersucht. Hierzu wurde das vorherige Experiment für eine variierende Anzahl an kritischen Testmustern wiederholt. Die Anzahl der Testmuster wurde zwischen 1 % und 10 % der ursprünglichen Testmustermenge variiert. In der Abbildung 4.28 sind beispielhaft die Ergebnisse für die Schaltung bench1 bei einer Kompaktierungsrate von 16 gezeigt. Es zeigt sich, dass mit steigender Anzahl zusätzlicher Testmuster auch die Fehlerdurchlässigkeit steigt. Auffällig ist außerdem, dass die Fehlerdurchlässigkeit zunächst stark ansteigt, bevor sie asymptotisch auf ein Maximum zuläuft. Dies lässt sich durch die Anzahl der verlorenen Fehler pro Testmuster erklären. Wie in Abbildung 4.23 zu sehen ist, gibt es nur wenige Testmuster, die viele verlorene Testmuster tragen. Diese werden in dem vorgestellten Ansatz als *kritisch* angesehen und werden daher als erstes verwendet.

Als nächstes wurde der X-Reduktionsfaktor (XRF) untersucht. Durch die Wiederverwendung von Testmustern kann es dazu kommen, dass zusätzliche X-Werte verarbeitet

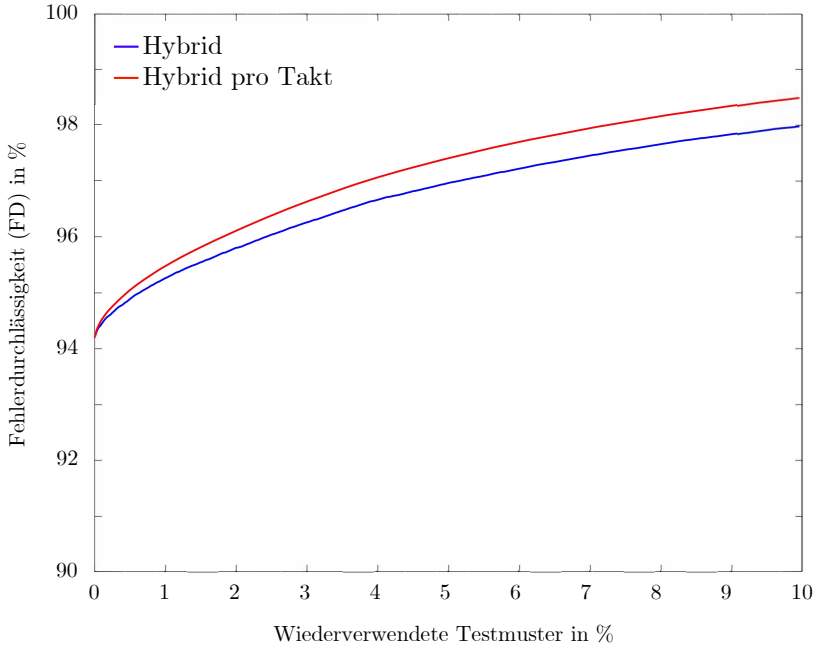


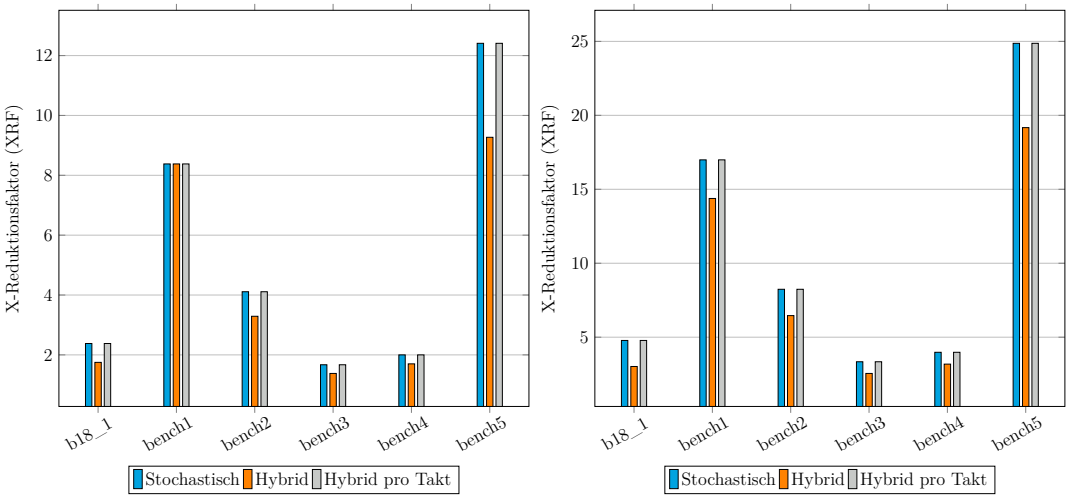
Abbildung 4.28: Entwicklung der Fehlerdurchlässigkeit bei steigender Anzahl der wiederverwendeten Testmuster [Maaz2019a].

werden müssen. Dies beeinflusst den X-Reduktionsfaktor. Wie in Tabelle 4.9 gezeigt, kann sich der X-Reduktionsfaktor (XRF) bei der Verwendung eines Steuerungsvektors pro Testmuster verringern, während bei der Verwendung von individuellen Steuerungsvektoren pro Takt keine X-Werte den Kompaktierer verlassen und somit der X-Reduktionsfaktor erhalten bleibt. Der Aufbau der Tabelle 4.9 ist analog zur Tabelle 4.8. Eine graphische Repräsentation der Ergebnisse für eine Kompaktierungsrate von 16 ist in Abbildung 4.29 dargestellt.

Tabelle 4.9: X-Reduktionsfaktor für $KR = 16$ und $KR = 32$ bei der Wiederverwendung von 10 % der Testmuster [Maaz2019a]

CUT	**Kompaktierungsrate 16**			**Kompaktierungsrate 32**		
	Stochastisch	**Hybrid**	**Hybrid pro Takt**	**Stochastisch**	**Hybrid**	**Hybrid pro Takt**
b18_1	2,38	1,75	2,38	4,78	3,02	4,78
bench1	8,38	8,38	8,38	16,99	14,38	16,99
bench2	4,11	3,29	4,11	8,24	6,46	8,24
bench3	1,67	1,38	1,67	3,34	2,55	3,34
bench4	2,00	1,70	2,00	3,98	3,18	3,98
bench5	12,41	9,27	12,41	24,87	19,17	24,87

Für einen besseren Überblick ist in Tabelle 4.10 die absolute Anzahl der X-Werte in derselben Weise wie in den vorherigen beiden Tabellen dargestellt.



(a) Kompaktierungsrate = 16 (b) Kompaktierungsrate = 32

Abbildung 4.29: Stochastische Kompaktierung vs. hybride Kompaktierung - Vergleich des XRFs bei einer Kompaktierungsrate von 16 und 32.

Tabelle 4.10: Anzahl der X-Werte an den Ausgängen des Kompaktierers bei $KR = 16$ und $KR = 32$ bei der Wiederverwendung von 10 % der Testmuster [Maaz2019a]

CUT	**Kompaktierungsrate 16**			**Kompaktierungsrate 32**		
	Stochastisch	**Hybrid**	Δ_x	**Stochastisch**	**Hybrid**	Δ_x
b18_1	68 706	93 685	24 979	34 200	54 137	19 937
bench1	79 799	79 799	0	39 362	46 513	7151
bench2	255 287	318 710	63 423	127 420	162 505	35 085
bench3	305 456	370 616	65 160	152 850	200 604	47 754
bench4	170 271	200 307	30 036	85 272	106 958	21 686
bench5	54 964	73 565	18 601	27 427	35 577	8150

Es zeigt sich, dass durch die hybride Kompaktierung der X-Reduktionsfaktor (XRF) im Vergleich zur stochastischen Kompaktierung leicht verringert wird, da wie bereits erklärt durch die Wiederverwendung der kritischen Testmuster zusätzliche X-Werte verarbeitet werden müssen. Zur Bewertung der Qualität des Gruppierungsalgorithmus wurde als nächstes nur der X-Reduktionsfaktor während der deterministischen Phase untersucht, ohne die vorangegangenen stochastischen Phase zu berücksichtigen. Der X-Reduktionsfaktor für die deterministische Phase ist in Tabelle 4.11 dargestellt. In der ersten Spalte sind wieder die Bezeichnungen der Schaltungen zu finden, während in der zweiten und dritten Spalte der X-Reduktionsfaktor der deterministischen Phase bei einer Kompaktierungsrate von 16 und 32 dargestellt ist.

Tabelle 4.11: X-Reduktionsfaktor (XRF) während der deterministischen Phase für $KR = 16$ und $KR = 32$ bei einer Wiederanwendung von 10 % der Testmuster [Maaz2019a]

CUT	**Kompaktierungsrate 16**	**Kompaktierungsrate 32**
b18_1	2,19	3,00
bench1	3,24	3,58
bench2	3,69	5,66
bench3	2,16	2,95
bench4	1,00	3,47
bench5	7,45	8,08

Die erreichten X-Reduktionsfaktoren sind niedriger als die der stochastischen Phase. Dies ist anhand der verwendeten Ähnlichkeits-Werte zu erklären, die versuchen die Fehlerdurchlässigkeit zu erhöhen. Durch die Anpassung des Ähnlichkeits-Wertes in der Definition 4.2 lässt sich der Gruppierungsalgorithmus an weitere Optimierungsziele anpassen.

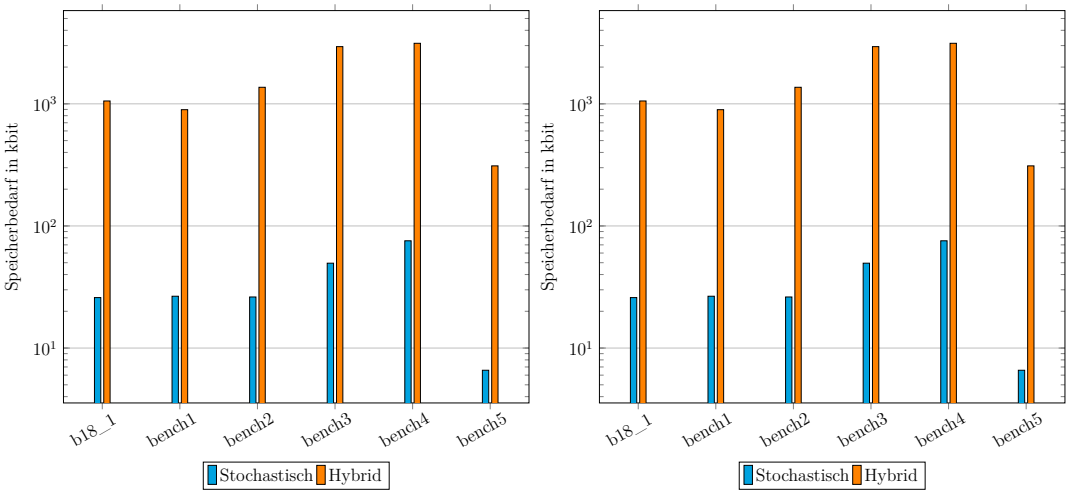
Zuletzt wurde der benötigte Hardwareaufwand für die deterministische Phase der hybriden Kompaktierung untersucht. Hierzu wurde der Speicheraufwand für die Steuerungsvektoren pro Testmuster und pro Taktzyklus untersucht und die Ergebnisse in der Tabelle 4.12 dargestellt. In der ersten Spalte sind die Bezeichnungen der Schaltungen aufgeführt. Die zweite und dritte, sowie die vierte und fünfte Spalte zeigen den benötigten Speicheraufwand pro Testmuster und pro Takt bei einer Kompaktierungsrate von 16 und 32 in kbit.

Wie erwartet, ist der benötigte Speicheraufwand des feingranularen Gruppierungsalgorithmus (pro Takt) deutlich höher als der Speicheraufwand des grobgranularen Gruppierungsalgorithmus (pro Testmuster). Die benötigte Speichergröße ist bei dem feingranularen Algorithmus bis zu zwei Größenordnungen größer. In Abbildung 4.30 sind die Speichergrößen auf einer logarithmischen Skala aufgeführt.

Tabelle 4.12: Speichergröße für $KR = 16$ und $KR = 32$ bei der Wiederverwendung von 10 % der Testmuster (kbit) [Maaz2019a]

CUT	Kompaktierungsrate 16		Kompaktierungsrate 32	
	Hybrid	Hybrid pro Takt	Hybrid	Hybrid pro Takt
b18_1	25,90	1055,52	25,90	1055,52
bench1	26,57	895,70	26,57	895,70
bench2	26,20	1366,50	26,20	1366,50
bench3	49,55	2941,05	49,55	2941,05
bench4	75,59	3132,09	75,59	3132,09
bench5	6,58	310,50	6,58	310,5

Da die Fehlerdurchlässigkeit nicht im gleichen Ausmaß gesteigert werden kann, ist es in der Praxis in der Regel nicht vertretbar die großen Kosten für den feingranularen Gruppierungsalgorithmus aufzubringen. Ein guter Kompromiss könnte jedoch sein, den feingranularen Gruppierungsalgorithmus für einige wenige *kritische* Testmuster anzuwenden, um so eine hohe Fehlerdurchlässigkeit bei einem vertretbaren Speicheraufwand zu erhalten.



(a) Kompaktierungsrate = 16 (b) Kompaktierungsrate = 32

Abbildung 4.30: Speicheraufwand bei einer Kompaktierungsrate (KR) von 16 und 32.

4.3.3 Fazit

Mithilfe des gezeigten hybriden Kompaktierungsverfahrens wird die flexible Architektur des stochastischen Kompaktierers effizient genutzt, um die Fehlerdurchlässigkeit des stochastischen Kompaktierungsverfahrens zu steigern. Hierzu wird in einer deterministischen Phase eine begrenzte Anzahl an Testmustern ein weiteres Mal an die zu testende

Schaltung angelegt und die zugehörigen Testantworten mithilfe von deterministischen Kompaktormatrizen kompaktiert. Die durchgeführten Experimente haben gezeigt, dass die Fehlerdurchlässigkeit durch die deterministische Phase gesteigert werden konnte, wobei der X-Reduktionsfaktor nur geringfügig verringert wurde. Durch die Verwendung eines feingranularen Gruppierungsalgorithmus kann die Fehlerdurchlässigkeit auf Kosten des benötigten Speicheraufwands weiter gesteigert werden. Außerdem können alle X-Werte der deterministischen Phase bei der Verwendung eines Steuervektors pro Takt maskiert werden. Somit erreichen keine weiteren X-Werte die folgende zeitliche Kompaktierung und es müssen keine zusätzlichen Zwischensignaturen des X-Canceling MISRs gespeichert werden [Touba2007]. Durch die Variation der Anzahl der verwendeten Testmuster während der deterministischen Phase kann ein Kompromiss zwischen Fehlerdurchlässigkeit und Speicheraufwand gefunden werden. Zusätzlich zur deterministischen Phase mithilfe der grobgranularen Gruppierungsmethode ist es außerdem möglich, eine zweite deterministische Phase für einige wenige essenzielle D-Bits mit feingranularen Steuerungsvektoren durchzuführen.

5 Verbindungstest in Logik-Schaltungen

Wie in der Einleitung eingeführt wurde, ist der gesamte Lebenszyklus einer hochintegrierten Schaltung von Unsicherheiten geprägt. Im vorherigen Kapitel wurde zunächst angenommen, dass Fehler innerhalb der Gatter einer Logik-Schaltung auftreten. Durch Übersprechen kommt es jedoch auch auf Verbindungsleitungen innerhalb der Logik-Schaltung, zu Unsicherheiten, welche sich als kleine Verzögerungsfehler manifestieren. Um die komplette Logik-Schaltung auf Unsicherheiten überprüfen zu können, werden in diesem Kapitel zusätzlich die Verbindungsleitungen innerhalb der Logik-Schaltung berücksichtigt.

Wie in Kapitel 2.2.2 gezeigt wurde, kommt es bei hochintegrierten Schaltungen, die in immer kleiner werdenden Strukturgrößen gefertigt werden, immer häufiger zu Übersprechen zwischen Verbindungsleitungen. Dieses Übersprechen kann unter anderem, wie in Kapitel 2.3.4 gezeigt, als Verzögerung am Ausgang der Schaltung sichtbar werden. Aber auch Prozessvariationen, wie z. B. die Variation der Gatelänge eines Transistors, können zu Verzögerungen des Signalverlaufs am Ausgang der Schaltung führen.

Wie in Abbildung 5.1 gezeigt wird, ist es jedoch schwer, am Ausgang der Schaltung zwischen den Ursachen einer Verzögerung zu unterscheiden. In der Abbildung 5.1 ist ein Histogramm der Verzögerungen der ISCAS'89 Benchmark-Schaltung s27 [Brglez1989] zu sehen. Für dieses Histogramm wurden zwei Zufallsexperimente mit über 7000 Wiederholungen durchgeführt. Im ersten Experiment wurde das Zeitverhalten fehlerfreier Instanzen der Schaltung unter Prozessvariation simuliert. Hierzu wurde die Prozessvariation als globale Veränderung der Transistor-Gatelänge modelliert. Die Gatelänge wurde hierzu als normalverteilte Zufallsvariable modelliert mit einem Mittelwert $\mu = L_{nom}$ und einer Standardabweichung $\sigma = 0{,}2 \cdot L_{nom}/3$, wobei L_{nom} der nominellen Gatelänge entspricht. In 99,73 % der Fälle liegt die Gatelänge somit im Intervall $[0{,}8 \cdot L_{nom}, 1{,}2 \cdot L_{nom}] \in \mathbb{R}$. Die Abweichung von der nominellen Gatelänge entspricht somit 3σ. Die simulierten Verzögerungen wurden in der Abbildung 5.1 als grüne Balken dargestellt. Im zweiten Experi-

ment wurde das Zeitverhalten fehlerhafter Schaltungen simuliert. Hierzu wurden zufällig Koppelkapazitäten zwischen die Verbindungsleitungen innerhalb der Logik-Schaltungen injiziert. Die Größe der Kapazität wurde dabei durch eine gleichverteilte Zufallsvariable im Intervall $[0\,\mathrm{fF}, 100\,\mathrm{fF}] \in \mathbb{R}$ modelliert. Die Verzögerungen sind als blaue Balken in dem Histogramm aufgetragen.

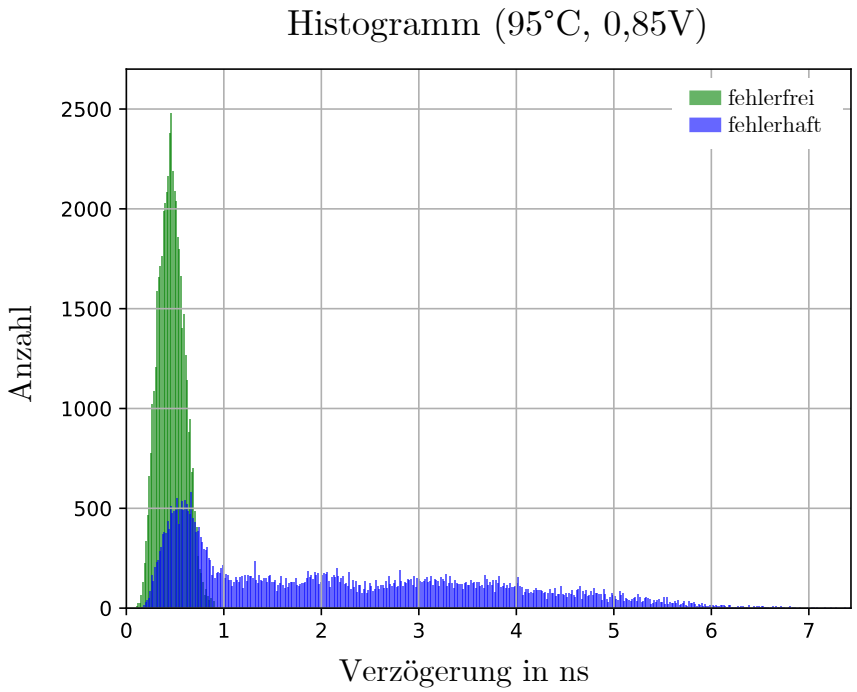


Abbildung 5.1: Verteilung der Laufzeitverzögerungen für Schaltungen bei denen Prozessvariation und Übersprechen auftritt [Sprenger2020].

In der Abbildung 5.1 ist deutlich eine Überlappung der Verzögerungen von fehlerfreien und fehlerhaften Schaltungsinstanzen zu erkennen. Üblicherweise wird ein Verzögerungstest anhand eines Schwellwertes durchgeführt. Liegt die Verzögerung oberhalb des Schwellwertes wird die Schaltung als fehlerhaft betrachtet, liegt sie darunter als fehlerfrei. Aufgrund der Überlappung der Verzögerungszeiten kann dies zu zwei Problemen führen. Einerseits können fehlerhafte Schaltungen als fehlerfrei klassifiziert werden, da sie sich genauso verhalten wie fehlerfreie Schaltungen und somit, wie in Kapitel 2.2.2 erläutert wurde, zu einem Zuverlässigkeitsproblem führen. Anderseits können bei einem aggressiveren Test mit geringerem Schwellwert fehlerfreie Schaltungen als fehlerhaft klassifiziert werden, was zu einer geringeren Ausbeute führt. Daher ist eine Methode zur Unterscheidung von Prozessvariation und Übersprechen notwendig, welche in diesem Kapitel vorgestellt wird.

Zunächst wird in Kapitel 5.1 der Stand der Technik eingeführt, bevor in Kapitel 5.2 das Verhalten von Verzögerungsfehlern aufgrund von Übersprechen und Prozessvariation in

unterschiedlichen Arbeitspunkten untersucht wird. In Kapitel 5.3 und 5.4 wird daraufhin eine Methode zur Unterscheidung der Ursachen von Verzögerungsfehlern mithilfe von künstlichen neuronalen Netzen beschrieben. Experimentelle Ergebnisse werden in Kapitel 5.5 vorgestellt, bevor in Kapitel 5.6 ein Fazit gezogen wird.

5.1 Stand der Technik

Die Detektion von kleinen Verzögerungsfehlern unter der Berücksichtigung von Prozessvariationen oder die Unterscheidung zwischen Verzögerungen, die durch Prozessvariation hervorgerufen wurden und Verzögerungen, die durch Logikdefekte hervorgerufen wurden, ist bereits seit einiger Zeit Teil von Forschungsarbeiten. Oft werden hierzu die Verzögerungen der zu testenden Schaltung in unterschiedlichen Arbeitspunkten oder zusätzliche Eigenschaften der Schaltung untersucht. So wurde z. B. in [Qian2010] und [Qian2012] das Verzögerungsverhalten der Schaltung bei unterschiedlichen Versorgungsspannungen ausgewertet, um dieses Problem zu lösen. In [Najafi-Haghi2020] wurden, auf der Ebene von Logikzellen, Verfahren des maschinellen Lernens zur Lösung dieser Problemstellung angewandt. In eine Menge von NAND-Gatterinstanzen mit Prozessvariation wurden hierzu resistive Defekte in die Gatterinstanzen injiziert und das Zeitverhalten für unterschiedliche Versorgungsspannungen simuliert. Durch den so generierten Datensatz konnten mit dem Verfahren der k-nächsten-Nachbarn (engl. k-nearest-neighbor), einer Support-Vektor-Maschine (engl. support vector machine) und einem mehrschichtigen Perzeptron (MLP) drei Verfahren des überwachten Lernens erfolgreich angewandt werden, um zwischen fehlerhaften und fehlerfreien Gatterinstanzen zu unterscheiden.

Wie in [Najafi-Haghi2020] wurden in vielen bisherigen Arbeiten nur interne Defekte von Logikzellen berücksichtigt, wie z. B. Gateoxiddefekte, die zu resistiven Defekten führen. So wurde in [Aitken2008] z. B. gezeigt, dass sich die Leckströme, welche aufgrund von Prozessvariation entstehen, anders verhalten als die Leckströme, die von resistiven Defekten hervorgerufen werden. Somit kann das Beobachten der Leckströme dabei helfen, zwischen resistiven Defekten und Prozessvariation zu unterscheiden. In [Moreno2016] wurde gezeigt, dass die Effekte von Defekten, die zu unterbrochenen Leitungen führen bei niedrigen Versorgungsspannungen größer ausgeprägt sind als bei normalen Versorgungsspannungen. Für den Test von kleinen Verzögerungsdefekten ist es notwendig, hochwertige Testmuster zu verwenden, die auch unter Prozessvariation die entsprechenden Verzögerungsfehler aktivieren. In [Peng2013] werden hierzu statistische Informationen über die Prozessvariationen und weitere Unsicherheiten (wie Übersprechen) genutzt, um robuste Testmuster zu erzeugen.

Neue Testqualitätsmetriken, wie in [Hasib2019] vorgestellt, nehmen ebenfalls Rücksicht auf die Prozessvariation, indem mehrere Prozess-, Spannungs- und Temperatur- (engl. Process, Voltage, Temperature (PVT)) Arbeitspunkte berücksichtigt werden.

Verbindungsleitungen auf System-Ebene sind in der Regel leicht zugänglich und können daher relativ einfach auf Übersprechen getestet werden [Chen1998]. Innerhalb der kombinatorischen Logik der Schaltung sind die Verbindungsleitungen jedoch oft nur schwer steuerbar und beobachtbar, wodurch die Fehler nur schwer aktiviert und zu einem Ausgang propagiert werden können. Außerdem ist es wichtig, dass die Flanken auf den beiden Signalleitungen des Fehlerortes, möglichst zeitnah erscheinen, um ein möglichst großes Fehlverhalten zu erzeugen [Tehranipoor2011]. Zur Aktivierung von Übersprechen gibt es bereits eine Vielzahl an Testmustererzeugungsalgorithmen in der Literatur [Chen1999; Eggersglüß2010; Bai2003; Chun2009; Ganeshpure2010; Asokan2015], die in dieser Arbeit nicht erneut eingeführt werden.

Wie zuvor erwähnt wurde, versuchen die oben genannten Methoden Prozessvariationen von Defekten innerhalb der Logik zu unterscheiden, die als kleine Verzögerungsfehler modelliert werden können, wohingegen das in diesem Kapitel vorgestellte Verfahren zur Unterscheidung von kleinen Verzögerungsfehlern aufgrund von Übersprechen und Prozessvariation dient. Hierzu wird ein künstliches neuronales Netz verwendet. Künstliche Intelligenz und Methoden des maschinellen Lernens werden jedoch nicht nur im Test und der Diagnose eingesetzt, sondern finden in nahezu allen Phasen des computergestützten Entwurfs von digitalen Schaltungen Anwendung. Ein ausführlicher Überblick ist in [Amuru2022] zu finden.

Im folgenden Abschnitt wird das Zeitverhalten von Logik-Schaltungen in unterschiedlichen Arbeitspunkten untersucht.

5.2 Verhalten von Übersprechen und Prozessvariation unter Berücksichtigung des Arbeitspunktes

Wie in Kapitel 2.3.4 gezeigt wurde, kann es aufgrund von parasitären Koppelkapazitäten, -induktivitäten und -widerständen zu Übersprechen kommen, das durch veränderte Signalverläufe auf der Opfer-Leitung modelliert wird. Um zwischen den Ursachen für kleine Verzögerungsfehler (Übersprechen und Prozessvariation) unterscheiden zu können, wird das Verhalten der hochintegrierten Schaltung in unterschiedlichen Arbeitspunk-

ten $op \in \mathscr{OP} := \mathscr{T} \times \mathscr{U}$ untersucht, wobei $\mathscr{T}$ der Menge der Betriebstemperaturen entspricht und $\mathscr{U}$ der Menge der Versorgungsspannungen. Zur Analyse des Zeitverhaltens wurden Experimente unter Prozessvariation und Übersprechen durchgeführt. Hierzu wurde die Benchmark-Schaltung s27 aus der ISCAS'89 Benchmark-Suite [Brglez1989] in der *FreePDK45 Generic Open Cell Library* Technology [SII2023] synthetisiert. Zur Aktivierung der Übersprecheffekte wurde ein hausinterner Testmustererzeugungsalgorithmus verwendet, der auf dem Booleschen Erfüllbarkeitsproblem basiert [Reimer2019]. Der Testmustererzeugungsalgorithmus arbeitet mit einer diskreten Repräsentation der Signalkurven, wie in [Sauer2012] vorgestellt wurde. Die Größe der Verzögerung aufgrund des Übersprechens hängt von der Differenz zwischen den beiden Ankunftszeiten der Flanken auf der Angreifer-Leitung und der Opfer-Leitung ab [Tehranipoor2011]. Hierzu wurden die Ankunftszeiten ähnlich wie in [Chun2009] in das Boolesche Erfüllbarkeitsproblem kodiert. Zur Simulation des Zeitverhaltens wurden SPICE-Simulationen durchgeführt.

Eine Herausforderung des Verbindungstests in Logik-Schaltungen ist die hohe Anzahl an möglichen Fehlerorten, an denen eine parasitäre Koppelkapazität auftreten kann. Als Fehlerort wird dabei eine Kombination aus einer Angreifer-Leitung und Opfer-Leitung bezeichnet, zwischen denen eine parasitäre Koppelkapazität $\varphi_{CC} \in \mathbb{R}$ auftreten kann. Die Menge der möglichen Fehlerorte bilden dabei alle möglichen Kombinationen an Verbindungsleitungen der Logik-Schaltung. Für eine realistische Reduktion der infrage kommenden Fehlerorte können, mithilfe einer Layout-Analyse, die Fehlerorte so ausgewählt werden, dass die Distanz zwischen der Opfer-Leitung und Angreifer-Leitung klein genug ist [Piel2021]. Aufgrund der geringen Größe der verwendeten Schaltung wurden, in dem hier gezeigten Experiment, alle Kombinationen an Verbindungsleitungen in der aktuellen topologischen Nachbarschaft als mögliche Fehlerorte ausgewählt. In der aktuellen topologischen Nachbarschaft befinden sich alle Verbindungsleitungen, die mit einem Logik-Gatter verbunden sind, das sich auf derselben Logikebene oder auf einer der angrenzenden Logikebenen wie das aktuell betrachtete Logik-Gatter befindet. Für große Schaltungen kann durch diese Annahme die Anzahl der Fehlerorte überschätzt werden. Ein Beispiel für einen möglichen Fehlerort ist in der Abbildung 5.2 zu sehen. Die betrachtete Verbindungsleitung ist mit einem Gatter auf der Logikebene n verbunden. In der aktuellen topologischen Nachbarschaft befinden sich somit alle Verbindungsleitungen, die mit einem Logik-Gatter auf den Logikebenen $n-1$, n, oder $n+1$ verbunden sind. Zur Analyse des Zeitverhaltens wurden drei zufällige Fehlerorte ausgewählt. An jedem der zufällig ausgewählten Fehlerorte wurde ein *slow-to-rise* und *slow-to-fall* Fehler angenommen, wobei die Größe der Koppelkapazität φ_{CC} im halboffenen Intervall $(0\,\mathrm{fF}, 100\,\mathrm{fF}]$ variiert wurde [Peng2013].

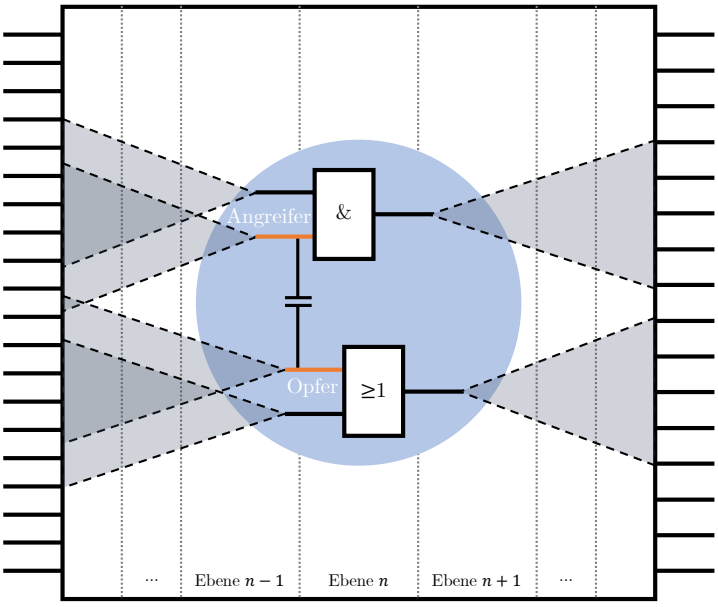


Abbildung 5.2: Beispiel eines Fehlerortes für mögliches Übersprechen.

Neben den Übersprecheffekten wurde außerdem die Prozessvariation modelliert, indem eine veränderliche Gatelänge der Transistoren innerhalb der Logikelemente angenommen wurde. Hierzu wurde eine prozentuale Änderung p_{add} der nominellen Gatelänge L_{nom} zwischen $-20\,\%$ und $20\,\%$ angenommen. Mit $p_{add} \in [-0{,}2, 0{,}2] \subset \mathbb{R}$ ergibt sich die variierte Gatelänge L_{var} zu $L_{var} = (1 + p_{add}) \cdot L_{nom}$. Für die Variation der Gatelänge wurde eine globale Prozessvariation angenommen. Das heißt, dass alle Transistoren der zu testenden Schaltung dieselbe Variation der Gatelänge erfahren.

Für die Auswertung des Zeitverhaltens wurde die relative Veränderung der Verzögerung $\Delta(o, op, \varphi_{CC}, p_{add})$ verwendet, die, wie in (5.1) gezeigt, definiert ist.

$$\Delta(o, op, \varphi_{CC}, p_{add}) := \frac{d_{pd}(o, op, \varphi_{CC}, p_{add})}{d_{pd}(o, op)} \tag{5.1}$$

$\Delta(o, op, \varphi_{CC}, p_{add})$ gibt dabei die Veränderung der Laufzeitverzögerung (engl. propagation delay) $d_{pd}(o, op, \varphi_{CC}, p_{add}))$ am Ausgang $o \in \mathscr{A}$ im Arbeitspunkt $op \in \mathscr{OP}$ bei der Koppelkapazität $\varphi_{CC} \in \mathbb{R}$ und der prozentualen Änderung der Gatelänge $p_{add} \in [-0{,}2, 0{,}2]$ normalisiert auf die Laufzeitverzögerung $d_{pd}(o, op)$ im fehlerfreien Fall und mit nomineller Gatelänge an.

Zur Analyse der Laufzeitverzögerung aufgrund von Übersprechen wurde zunächst ein Experiment durchgeführt, in dem nur die parasitäre Koppelkapazität φ_{CC} an den zufällig

ausgewählten Fehlerorten variiert wurde. Die Größe der Koppelkapazität wurde dabei durch $\varphi_{CC} \in \{10\,\text{fF},\ 30\,\text{fF},\ 50\,\text{fF},\ 70\,\text{fF},\ 90\,\text{fF}\}$ definiert. Während des Experiments wurden Simulationen des Zeitverhaltens für alle Arbeitspunkte $op \in \mathscr{OP} := \mathscr{T} \times \mathscr{U}$ durchgeführt. Die Betriebstemperatur T wurde zwischen 45 °C und 125 °C mit einer Schrittweite von 10 °C variiert und die Versorgungsspannung U_{DD} zwischen 0,6 V und 1,6 V mit einer Schrittweite von 0,025 V, wobei die nominelle Versorgungsspannung bei $U_{nom} = 1{,}1\,\text{V}$ liegt. Die Ergebnisse für den Ausgang $o = o_0$ der Benchmark-Schaltung *s27* aus der Benchmark-Suite ISCAS'89 [Brglez1989] sind in der Abbildung 5.3 zu sehen.

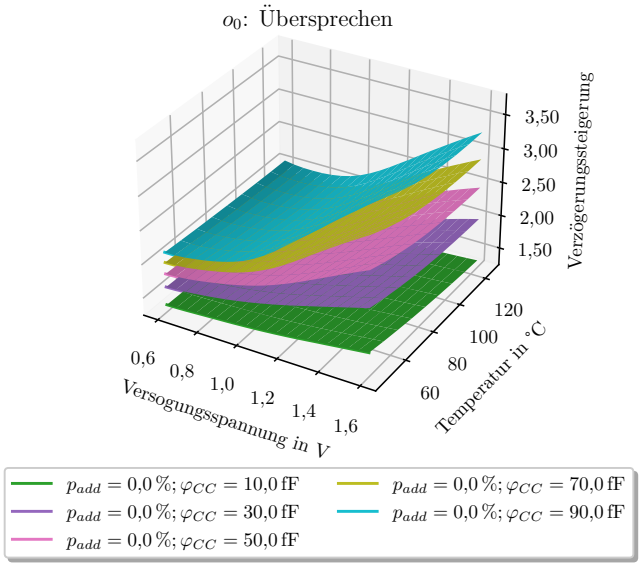


Abbildung 5.3: Veränderung der Laufzeitverzögerungen aufgrund von Übersprechen [Sprenger2020].

In einem zweiten Experiment wurde der fehlerfreie Fall betrachtet und nur die Gatelänge variiert. Für dieselben Arbeitspunkte wie im ersten Experiment wurde hierzu die Laufzeitverzögerung bei unterschiedlichen Gatelängen L_{var} simuliert. In Abbildung 5.4 sind die Ergebnisse für eine prozentuale Änderung der nominellen Gatelänge von $p_{add} \in \{-20\,\%,\ -12\,\%,\ -4\,\%,\ 4\,\%,\ 12\,\%,\ 20\,\%\}$ dargestellt.

Bei dem Vergleich der Abbildung 5.3 mit der Abbildung 5.4 fällt auf, dass sich die relative Änderung der Laufzeitverzögerung $\Delta(o, op, \varphi_{CC}, L_{nom})$ bei Prozessvariation anders verhält als beim Auftreten von Übersprechen. So ist unter anderem in Abbildung 5.3 zu sehen, dass die Laufzeitverzögerung im fehlerhaften Fall bei steigender Versorgungsspannung stärker steigt als im fehlerfreien Fall. Außerdem ist zu sehen, dass dieser Effekt sich mit steigender Kapazitätsgröße weiter verstärkt. Bei der Betrachtung der Prozessvariati-

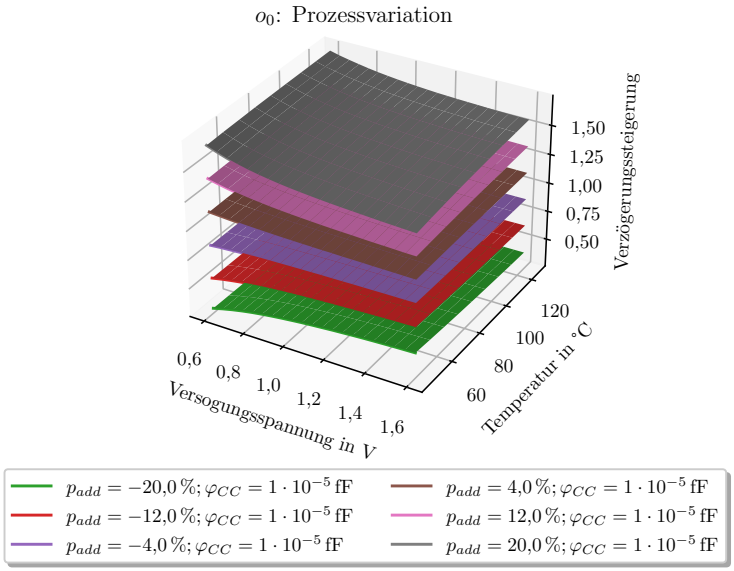


Abbildung 5.4: Veränderung der Laufzeitverzögerungen aufgrund von Prozessvariationen [Sprenger2020].

on in Abbildung 5.4 zeigt sich, dass die Änderung der Laufzeitverzögerung aufgrund von positiver Prozessvariation bei steigender Versorgungsspannung sinkt. Bei Verringerung der Gatelänge kehrt sich dieser Effekt um.

In realen Systemen treten die beiden Effekte simultan auf. Daher wurde in einem dritten Experiment die relative Änderung der Laufzeitverzögerung bei gleichzeitigem Auftreten von Übersprechen und Prozessvariation untersucht. Abbildung 5.5 zeigt die Verzögerungssteigerung $\Delta(o, op, \varphi_{CC}, p_{add}), \forall op \in OP$ bei einer Koppelkapazität φ_{CC} von 100 fF und variierender prozentualer Änderung der nominellen Gatelänge $p_{add} \in \{-20\,\%,\ -12\,\%,\ -4\,\%,\ 4\,\%,\ 12\,\%,\ 20\,\%\}$. In der Abbildung 5.5 ist zu sehen, dass für niedrige Versorgungsspannungen ($\approx 0{,}6\,\mathrm{V}$ bis $\approx 0{,}8\,\mathrm{V}$) die Effekte der Prozessvariation dominieren, während für Versorgungsspannungen über $\approx 0{,}8\,\mathrm{V}$ die Effekte aufgrund von Übersprechen dominieren.

Für die weitere Analyse werden die in den Abbildungen 5.3, 5.4 und 5.5 dreidimensional dargestellten Oberflächen als zweidimensionale Verzögerungskarten dargestellt. Im zweidimensionalen Raum der Arbeitspunkte wird hierzu die relative Änderung der Laufzeitverzögerung $\Delta(o, op, \varphi_{CC}, p_{add})$ durch die Intensität des entsprechenden Pixels in der Verzögerungskarte dargestellt. Die Abbildung 5.6 zeigt ein Beispiel einer Verzögerungskarte für den Ausgang $o = o_0$, für eine prozentuale Änderung der nominellen Gatelänge von $p_{add} = 6\,\%$ und einer Koppelkapazität φ_{CC} von 20,00 fF.

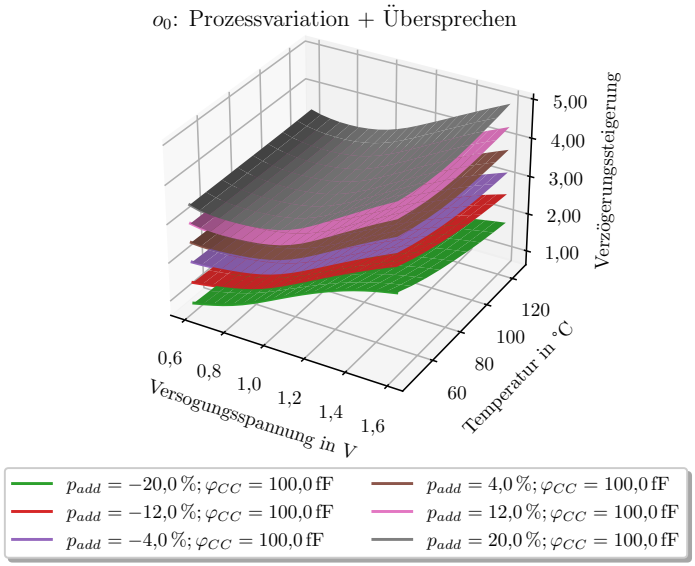


Abbildung 5.5: Veränderungen der Laufzeitverzögerungen aufgrund von Prozessvariationen bei einer Koppelkapazität von 100 fF [Sprenger2020].

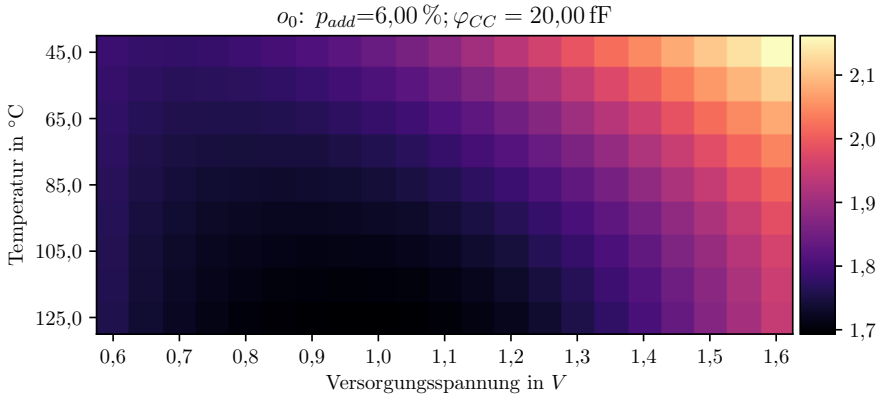


Abbildung 5.6: Verzögerungskarte für den Ausgang o_0 bei $p_{add} = 6{,}00\,\%$ und $\varphi_{CC} = 20{,}00\,\mathrm{fF}$.

Mithilfe dieser Verzögerungskarten kann das Problem der Unterscheidung zwischen Prozessvariation und Übersprechen als Bildklassifizierungsproblem angesehen werden, für welches bereits erfolgreich künstliche neuronale Netze eingesetzt wurden [Bhatnagar2017]. So konnten z. B. mehrschichtige Perzeptren erfolgreich verwendet werden, um handgeschriebene Zahlen mit einer Genauigkeit von bis zu 99,65 % zu erkennen [Cireşan2010; LeCun1998].

Im folgenden Kapitel wird näher darauf eingegangen, wie mithilfe eines künstlichen neuronalen Netzes in Form des in Kapitel 3 eingeführten mehrschichtigen Perzeptrons (engl. Multi-Layer Perceptrons, MLPs) die erzeugten Verzögerungskarten in fehlerfreie und fehlerhafte Verzögerungskarten eingeteilt werden können.

5.3 Konfiguration des künstlichen neuronalen Netzes

Es gibt eine Vielzahl von künstlichen neuronalen Netzen, die sich anhand ihrer Architektur unterscheiden lassen. So werden unter anderem gedächtnislose und gedächtnisbehaftete neuronale Netze unterschieden. Die Klasse der gedächtnislosen künstlichen neuronalen Netze zeichnet sich dadurch aus, dass die Struktur nur einen Datenfluss von den Eingängen zu den Ausgängen zulässt und somit das neuronale Netz keine Information von bereits verarbeiteten Daten speichern kann. Bei gedächtnisbehafteten neuronalen Netzen hingegen ist auch eine Kommunikation in Richtung der Eingänge des Netzes möglich, wodurch Informationen von früheren Daten gespeichert werden können. In dieser Arbeit wurde, wie in Kapitel 3 eingeführt, ein mehrschichtiges Perzeptron (MLP) verwendet, bei dem es sich um ein gedächtnisfreies künstliches neuronales Netz handelt. Abbildung 5.7 zeigt ein mehrschichtiges Perzeptron mit einer Eingangsebene, einer Ausgangsebene und zwei versteckten Ebenen.

Um die Architektur eines künstlichen neuronalen Netzes festzulegen, müssen eine Vielzahl von Parametern festgelegt werden, wie z. B. die Anzahl an versteckten Ebenen, die Anzahl der Neuronen pro Ebene und die Aktivierungsfunktion der Neuronen. Diese Parameter werden auch Hyperparameter genannt und müssen vom Benutzer festgelegt werden, bevor mit dem Training des neuronalen Netzes begonnen werden kann.

Die Anzahl der Neuronen in der Eingangsebene wird durch die Dimension des Eingangsvektors (engl. feature vector) festgelegt. In unserer Anwendung besteht der Eingangsvektor aus den 189 Pixeln der Verzögerungskarte, welche die 189 Arbeitspunkte (21 Versorgungsspannungen und neun Betriebstemperaturen), in denen der Verzöge-

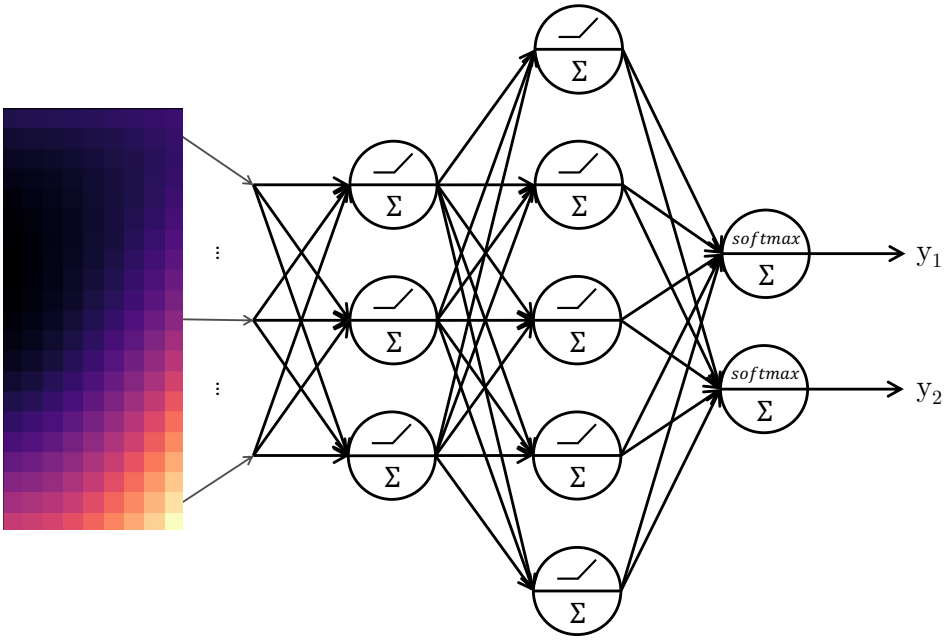


Abbildung 5.7: Konfiguration des künstlichen neuronalen Netzes.

rungstest durchgeführt wurde, repräsentieren. Entsprechend besteht die Eingangsebene aus 189 Neuronen. Die Anzahl der Neuronen der Ausgangsebene wird ebenfalls durch die Anwendung des neuronalen Netzes festgelegt. Die Verzögerungskarten werden in zwei Klassen, fehlerfrei und fehlerhaft, eingeteilt. Daher werden zwei Neuronen in der Ausgangsebene verwendet. Als Aktivierungsfunktion wird in den versteckten Ebenen die in (3.5) definierte Rectified Linear Unit (ReLU) Funktion $relu(x)$ [Nair2010] verwendet. In der Ausgangsebene wird die *softmax* Funktion [Bishop2016], wie in (3.8) definiert, verwendet. Wie in Kapitel 3 erläutert wurde, wird mithilfe der *softmax* Funktion der K-dimensionale Ausgangsvektor des künstlichen neuronalen Netzes auf einen K-dimensionalen Vektor abgebildet, dessen Elemente sich zu 1 summieren. So ermöglicht die *softmax* Funktion die Interpretation der Ausgabe des neuronalen Netzes als Konfidenz für die Vorhersage, dass sich die Eingabe tatsächlich in der vorhergesagten Klasse befindet.

Die Anzahl der verwendeten versteckten Ebenen L und die Anzahl der Neuronen pro Ebene n_l wurde mithilfe einer Suche im Hyperparameterraum durchgeführt. Dieser Hyperparameterraum wurde dabei eingeschränkt, indem als Anzahl versteckter Ebenen $L \in \{1, 2, 3\}$ und als Anzahl Neuronen pro Ebene $n_l \in \{32, 64, 128, 256\}$ zugelassen wurden. Innerhalb dieses Suchraums wurden zwei Suchmethoden zur Festlegung der Hyperparameter der Software-Bibliothek *scikit-learn* [Pedregosa2011] verwendet. Bei der ersten Methode *GridSearchCV* handelt es sich um eine vollständige Suche innerhalb des Suchraums. Die zweite Methode *RandomizedSearchCV* hingegen führt eine zufällige Suche innerhalb des spezifizierten Suchraums durch.

5.4 Merkmalsextraktion

In der Praxis ist es nur schwer möglich einen Verzögerungstest in 189 Arbeitspunkten durchzuführen. Daher ist es notwendig, die Anzahl der benötigten Arbeitspunkte zu reduzieren. Im Bereich des maschinellen Lernens ist dieses Problem auch als Merkmalextraktion (engl. feature extraction) oder Merkmalauswahl (engl. feature selection) bekannt. Für Standardmethoden der Merkmalauswahl, wie z. B. die in Kapitel 3.1.1 vorgestellte Hauptkomponentenanalyse, müssen zunächst alle Merkmale bestimmt werden, bevor diese anhand einer Metrik reduziert oder mathematisch in neue Merkmale transformiert werden können. In dieser Arbeit wurde die Interpretation der Merkmale als Arbeitspunkte ausgenutzt, um eine Merkmalauswahl durchzuführen, die im Folgenden näher erläutert wird.

In modernen hochintegrierten Schaltungen werden oft Methoden wie die adaptive Anpassung der Versorgungsspannung und der Taktfrequenz (engl. Adaptive Voltage and Frequency Scaling, AVFS) [Borkar2011] zur Reduktion der Leistungsaufnahme verwendet. Durch den Einsatz dieser Methoden während des Verzögerungstests ist es daher einfach möglich, die Versorgungsspannung während des Verzögerungstests zu steuern. Im Vergleich dazu ist es schwierig, die Betriebstemperatur der hochintegrierten Schaltung zu kontrollieren, da diese unter anderem von der Umgebungstemperatur und dem bisherigen Arbeitsaufwand der Schaltung abhängt.

Eine an den praktischen Einschränkungen orientierte Methode zur Reduktion der benötigten Merkmale ist die Verwendung aller Versorgungsspannungen bei der kältesten und der wärmsten Betriebstemperatur. Die Versorgungsspannungen können mithilfe des AVFS-Moduls konfiguriert werden. Zur Einstellung der Temperatur werden zwei Testläufe durchgeführt. Während des ersten Testlaufs kann die Betriebstemperatur als kalt angesehen werden, da noch keine Tests durchgeführt wurden. Im zweiten Testlauf befindet sich die zu testende Schaltung bereits auf hoher Betriebstemperatur und kann daher als warm angesehen werden. Alternativ kann mit fortgeschrittenen Methoden zum Testablauf die Betriebstemperatur kontrolliert werden, indem die Schaltung gezielt mit der Anwendung von Testmustern erhitzt wird [Aghaee2014]. Bei dem gezeigten Merkmalraum mit $21 \cdot 9 = 189$ Merkmalen kann mithilfe dieser Methode die Anzahl der benötigten Arbeitspunkte bereits um 78 % auf 42 Arbeitspunkte reduziert werden.

Die Durchführung eines Verzögerungstests in 42 Arbeitspunkten verursacht jedoch immer noch erheblich Kosten während des Tests. Eine weitere Reduktion der Arbeitspunkte ist daher notwendig. Zur Reduktion der verwendeten Versorgungsspannungen wer-

den daher zufällig drei Teilmengen mit k Versorgungsspannugen aus den 21 möglichen Versorgungsspannungen je Temperatur ausgewählt. Für jede Teilmenge wird daraufhin ein mehrschichtiges Perzeptron trainiert und anschließend der beste Klassifizierer ausgewählt. Wie in Abbildung 5.8 gezeigt wird, reduziert sich die Anzahl der benötigten Merkmale pro Teilmenge somit auf $2 \cdot k$. Hier konnte die Anzahl der Merkmale mit $k = 3$ auf sechs Merkmale pro Teilmenge reduziert werden. Bei der Verwendung aller drei Teilmengen, werden noch 18 Merkmale benötigt, was einer zusätzlichen Reduktion von 57,7 % entspricht. Im Vergleich zu der Ausgangssituation mit 189 Arbeitspunkten, konnte die Anzahl der benötigten Arbeitspunkte somit um 90,5 % reduziert werden.

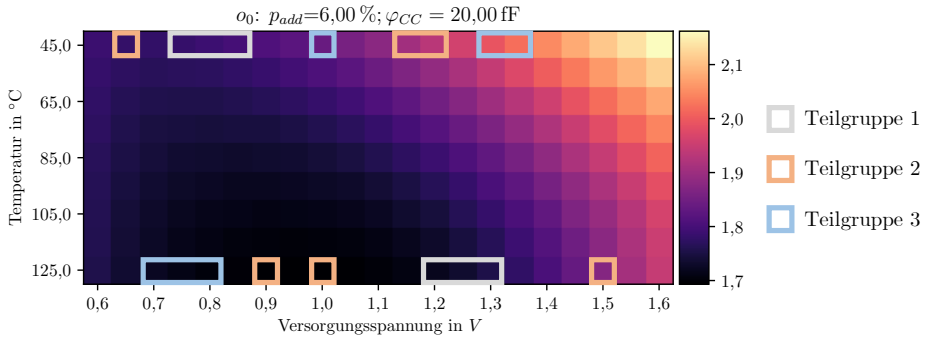


Abbildung 5.8: Merkmalreduktion mithilfe von *Feature Bagging*.

Durch die Reduktion der Anzahl der benötigten Merkmale kann sich die Qualität der Klassifizierer reduzieren. Um diesem Problem entgegenzuwirken, können die Vorhersagen der drei Klassifizierer, die auf den einzelnen Teilmengen trainiert wurden, ähnlich wie in der in Kapitel 3.2 gezeigten Ensemble-Methode der *Bagging Predictors* [Breiman1996], mithilfe eines gewichteten Mehrheitsentscheids ausgewertet werden. Die Methode wird als *Feature Bagging* bezeichnet.

5.5 Experimente

Für das Training des künstlichen neuronalen Netzes wurde eine Menge von insgesamt 95 880 Verzögerungskarten für die Benchmark-Schaltung s27 der ISCAS'89 Benchmark-Suite [Brglez1989] erzeugt. Für den Datensatz wurden, wie in Kapitel 5.2 erläutert, zwischen sechs Verbindungsleiterpaaren parasitäre Koppelkapazitäten injiziert und mit den zugehörigen Testmustern aktiviert. Die Größe der Koppelkapazität und die Prozessvariation wurden hierzu durch zwei Zufallsvariablen modelliert. Um die Koppelkapazität φ_{CC} zu modellieren, wurde die Zufallsvariable X verwendet, wobei X eine gleichverteilte Zufallsvariable im Intervall $[10\,\text{fF}, 100\,\text{fF}] \in \mathbb{R}$ ist. Die Variation der Gatelänge wurde

mithilfe der normalverteilten Zufallsvariable Y modelliert, wobei die Verteilung durch einen Mittelwert $\mu = 0$ und einer Standardabweichung $3\sigma = 0{,}2$ definiert ist. Die zusätzliche Gatelänge L_{add} ergibt sich somit mithilfe der Zufallsvariable Y zu $L_{add} = Y \cdot L_{nom}$, wobei L_{nom} die nominelle Länge des Gates des verwendeten Transistors angibt. Die verwendete Gatelänge ergibt sich somit zu $L_{Gate} = (1 + Y) \cdot L_{nom}$.

Um den Datensatz zu generieren wurden zunächst 1198 fehlerfreie Schaltungsinstanzen unter Berücksichtigung von globaler Variation der Gatelänge L_{Gate} erzeugt. Zusätzlich wurde eine fehlerfreie Schaltungsinstanz ohne Prozessvariation erzeugt. Jede dieser fehlerfreien Schaltungsinstanzen wurde unter Verwendung der sechs Testmuster, mit denen das Übersprechen aktiviert werden kann, in allen Arbeitspunkten simuliert, was $(1198 + 1) \cdot 6 = 7194$ Konfigurationen ergibt. Zur Erzeugung der fehlerhaften Schaltungen, wurde für jede der sechs Koppelkapazitäten eine zufällige Kapazitätsgröße in die zuvor erzeugten 1198 Schaltungsinstanzen injiziert. Die 7188 fehlerhaften Schaltungsinstanzen wurden ebenfalls in allen 189 Arbeitspunkten simuliert. Für jeden Ausgang, an dem eine Veränderung der Verzögerung sichtbar ist, wurde daraufhin eine Verzögerungskarte erzeugt. Falls ein Fehler oder eine Prozessvariation an verschiedenen Ausgängen erkennbar ist, wurde für jeden Ausgang, an dem die Verzögerung sichtbar ist, eine Verzögerungskarte erzeugt. Im Mittel waren die Verzögerungen an 6,67 Ausgängen zu sehen, wodurch sich die Gesamtzahl von $(7194 + 7188) \cdot 6{,}67 = 95\,880$ Verzögerungskarten ergibt. Wie in Tabelle 5.1 gezeigt wird, wurde der Datensatz anschließend zufällig in einen Trainingsdatensatz und einen Testdatensatz aufgeteilt. 90 % der Daten wurden hierbei für den Trainingsdatensatz verwendet und 10 % für den Testdatensatz, wobei darauf geachtet wurde, dass $\approx 50\,\%$ der Daten den fehlerfreien Fall repräsentieren. Anschließend wurde der Trainingsdatensatz zum Training der künstlichen neuronalen Netze verwendet und der Testdatensatz zur Bestimmung der Qualität der trainierten Klassifizierer.

Tabelle 5.1: Datensatzaufteilung [Sprenger2020]

	Gesamt	**Fehlerhaft**	**Fehlerfrei**
Datensatz	95 880	47 960	47 920
	100 %	50,02 %	49,98 %
Trainingsdaten	86 292	43 190	43 102
	90 %	50,05 %	49,95 %
Testdaten	9588	4770	4818
	10 %	49,75 %	50,25 %

Wie bereits in Kapitel 5.3 erläutert wurde, wurde für die Konfiguration der künstlichen neuronalen Netze eine vollständige und eine zufällige Hyperparametersuche durchgeführt. Die mehrschichtigen Perzeptronen wurden hierzu mithilfe der Methode der 5-fachen Kreuzvalidierung (engl. 5-fold-cross-validation) [Bishop2016] trainiert. In jeder Epoche werden dazu zufällig 10 % des Trainingsdatensatzes ausgewählt und zur Validierung der Trainingsepoche verwendet. Als Trainingsalgorithmus wurde der *adam* Algorithmus mit der *sparse_categorical_crossentropy* als Kostenfunktion [Kingma2015] verwendet. Die Qualität der Klassifizierer wurde anschließend anhand des Testdatensatzes bewertet und das beste künstliche neuronale Netz verwendet. Die Ergebnisse sind anhand von Konfusionsmatrizen [Sammut2017] in den Tabellen 5.2 und 5.3 gezeigt. Die Zeilen der Tabelle repräsentieren dabei die tatsächlichen Klassen der Verzögerungskarten, wobei die Spalten die Vorhersagen des Klassifizierers repräsentieren.

Tabelle 5.2: Konfusionsmatrix *GridSearchCV* [Sprenger2020]

Klassifizierung / **Reale Klasse**		**Fehlerfrei Negativ**	**Fehlerhaft Positiv**
Fehlerfrei	**Negativ**	4770 (RN)	0 (FP)
Fehlerhaft	**Positiv**	125 (FN)	4693 (RP)

Tabelle 5.3: Konfusionsmatrix *RandomizedSearchCV* [Sprenger2020]

Klassifizierung / **Reale Klasse**		**Fehlerfrei Negativ**	**Fehlerhaft Positiv**
Fehlerfrei	**Negativ**	4770 (RN)	0 (FP)
Fehlerhaft	**Positiv**	133 (FN)	4685 (RP)

In dieser Arbeit wurde die Einteilung einer Verzögerungskarte in die fehlerhafte Klasse als positiv und die Einteilung in die fehlerfreie Klasse als negativ interpretiert. Die in Tabelle 5.2 und Tabelle 5.3 gezeigten Konfusionsmatrizen zeigen die *richtig negativen (RN)*, *falsch positiven (FP)*, *falsch negativen (FN)* und die *richtig positiven (RP)* Vorhersagen. Bei den *richtig positiven* Vorhersagen handelt es sich z. B. um die richtig als positiv eingeschätzten Verzögerungskarten, also Verzögerungskarten, die mithilfe einer fehlerhaften Schaltungsinstanz erzeugt wurden und auch als fehlerhaft klassifiziert wurden. Mithilfe dieser Einteilung lässt sich die Qualität des Klassifizierers mithilfe verschiedener Metriken bewerten. Hierzu wurden die Metriken Genauigkeit, Relevanz (engl. precision), Sensitivität (engl. recall), F1-Score, Spezifität und der negative Vorhersagewert (NVW) verwendet, die wie folgt definiert sind [Sammut2017; Géron2020].

$$\begin{aligned} \text{Genauigkeit} &= \frac{|RN| + |RP|}{|RN| + |FP| + |FN| + |RP|} \\ &= \frac{\#\text{ richtig klassifiziert}}{\#\text{ Verzögerungskarten}} \end{aligned} \tag{5.2}$$

$$\begin{aligned} \text{Spezifität} &= \frac{|RN|}{|RN| + |FP|} \\ &= \frac{\#\text{ richtig fehlerfrei klassifiziert}}{\#\text{ fehlerfreie Verzögerungskarten}} \end{aligned} \tag{5.3}$$

$$\begin{aligned} \text{NVW} &= \frac{|RN|}{|RN| + |FN|} \\ &= \frac{\#\text{ richtig fehlerfrei klassifiziert}}{\#\text{ fehlerfrei klassifiziert}} \end{aligned} \tag{5.4}$$

$$\begin{aligned} \text{Relevanz} &= \frac{|RP|}{|FP| + |RP|} \\ &= \frac{\#\text{ richtig fehlerhaft klassifiziert}}{\#\text{ fehlerhaft klassifiziert}} \end{aligned} \tag{5.5}$$

$$\begin{aligned} \text{Sensitivität} &= \frac{|RP|}{|FN| + |RP|} \\ &= \frac{\#\text{ richtig fehlerhaft klassifiziert}}{\#\text{ fehlerhafte Verzögerungskarten}} \end{aligned} \tag{5.6}$$

$$\text{F1-Score} = 2 \cdot \frac{\text{Relevanz} \cdot \text{Sensitivität}}{\text{Relevanz} + \text{Sensitivität}} \tag{5.7}$$

Alle als fehlerfrei klassifizierten Schaltungen ($RN + FN$) werden nach dem Test ausgeliefert, während die als fehlerhaft klassifizierten ($FP + RP$) vernichtet werden. Für den Test ist es daher wichtig, eine hohe Spezifität und Relevanz zu erreichen, damit möglichst wenig fehlerfreie Schaltungen aussortiert werden, obwohl sie ausgeliefert werden könnten, um so Ausbeuteverluste zu vermeiden. Außerdem sollen möglichst wenig defekte Schaltungen ausgeliefert werden, um eine hohe Produktqualität zu erreichen. Hierzu muss der Klassifizierer einen hohen negativen Vorhersagewert (NVW) und eine

hohe Sensitivität aufweisen. Der F1-Score kann dabei als Maß dafür verwendet werden, wie gut beide Ziele mithilfe des Klassifizierers erreicht werden.

In Tabelle 5.4 sind die Metriken für die vollständige Hyperparametersuche in der Zeile *vollständig* und der zufälligen Hyperparametersuche in der Zeile *zufällig* wie folgt aufgeführt: Die zweite Spalte der Tabelle zeigt die Genauigkeit (5.2), die dritte die Spezifität (5.3), die vierte den negativen Vorhersagewert (5.4), die fünfte die Relevanz (5.5), die sechste Spalte die Sensitivität (5.6) und die letzte Spalte den F1-Score (5.7). Da die Ergebnisse für die vollständige *GridSearchCV* Suche nur leichte Verbesserungen gegenüber der zufälligen *RandomizedSearchCV* aufweist, wurde für die weiteren Experimente auf eine vollständige Hyperparametersuche verzichtet.

Tabelle 5.4: Evaluierung der Hyperparametersuche [Sprenger2020]

Algorithmus	**Genauigkeit**	**Spezifität**	**NVW**	**Relevanz**	**Sensitivität**	**F1**
vollständig	0,987	1	0,974	1	0,974	0,987
zufällig	0,986	1	0,973	1	0,972	0,986

5.5.1 Reduktion der Arbeitspunkte

In diesem Kapitel werden die in Kapitel 5.4 erläuterten Methoden zur Merkmalauswahl und somit zur Reduktion der Arbeitspunkte untersucht. In der Tabelle 5.5 sind die Ergebnisse der Merkmalauswahl zusammengefasst. In den Zeilen der Tabelle sind die Werte für die *richtig negativen (RN)*, *falsch positiven (FP)*, *falsch negativen (FN)* und die *richtig positiven (RP)* Vorhersagen, sowie für die in (5.2) - (5.7) eingeführten Metriken aufgeführt. In der vorletzten Zeile sind die ausgewählten Hyperparameter in Form eines Vektors $(n_1, ..., n_l)$ angegeben. Dabei gibt $n_1, ..., n_l$ die Anzahl Neuronen in der jeweiligen versteckten Ebene l an. In der letzten Zeile sind die ausgewählten Arbeitspunkte angegeben. In der zweiten Spalte sind die Ergebnisse für das *Kalt-Warm-Test* Verfahren angegeben, in dem alle Versorgungsspannungen bei der kältesten und bei der wärmsten Temperatur ausgewählt wurden. In den Spalten drei bis vier sind die Ergebnisse für die drei Klassifizierer aufgeführt, die jeweils mit einem Datensatz trainiert wurden, der nur eine Teilmenge von $k = 3$ Versorgungsspannungen je wärmster und kältester Temperatur verwendet. Die letzte Spalte zeigt die Ergebnisse, wenn die drei Klassifizierer, der vorherigen drei Spalten mithilfe der Methode des gewichteten Mehrheitsentscheids zusammengefasst werden. Die Ergebnisse für die Benchmark-Schaltung der Benchmark-Suite ISCAS'89 sind in der Tabelle 5.6 zu finden.

Es zeigt sich, dass wenn alle Versorgungsspannungen bei nur zwei Temperaturen verwendet werden, was einer Reduktion des Merkmalraums um 78 % entspricht, die Qualität des Klassifizierers vergleichbar mit dem ursprünglichen Klassifizierer ist. Bei einer weiteren Reduktion des Merkmalraums durch die Auswahl von Teilmengen der Versorgungsspannungen steigt die Anzahl der *falsch negativen* und der *falsch positiven* Vorhersagen, was darauf hindeutet, dass eine Klassifizierung mithilfe des reduzierten Merkmalraums anspruchsvoller wird. Jedoch ist die Qualität des mehrschichtigen Perzeptrons immer noch gut. Dabei weist das mehrschichtige Perzeptron der Teilmenge 0 die beste Qualität auf, da es für die höchste Produktqualität (NVW, Sensitivität) und den geringsten Ausbeuteverlust (Spezifität, Relevanz) sorgt.

Je nach Anwendung können aber auch anderer Klassifizierer eine bessere Wahl sein. Für eine Anwendung, in der die Ausbeute eine größere Rolle spielt als die Produktqualität, wäre z. B. der Klassifizierer, der mithilfe der Teilmenge 2 trainiert wurde, ebenfalls interessant, da dieser mit 100 % die beste Ausbeute erreicht. Für sicherheitskritische Anwendungen hingegen ist es erstrebenswert, einen negativen Vorhersagewert und eine Sensitivität von 100 % zu erreichen.

Durch das *Feature Bagging*, das einen neuen Klassifizierer durch den gewichteten Mehrheitsentscheid aus den drei Klassifizierern erzeugt, die mithilfe der drei Teilmengen der Versorgungsspannungen trainiert wurden, kann die Qualität des schlechtesten Teilmengen-Klassifizierers zwar gesteigert werden, jedoch gibt es auch einen Teilmengenklassifizierer (Teilmenge 0), der eine bessere Qualität als der *Feature Bagging* Klassifizierer aufweist. Um den besten Klassifizierer zu verwenden, sollten daher immer alle Einzelergebnisse berücksichtigt werden.

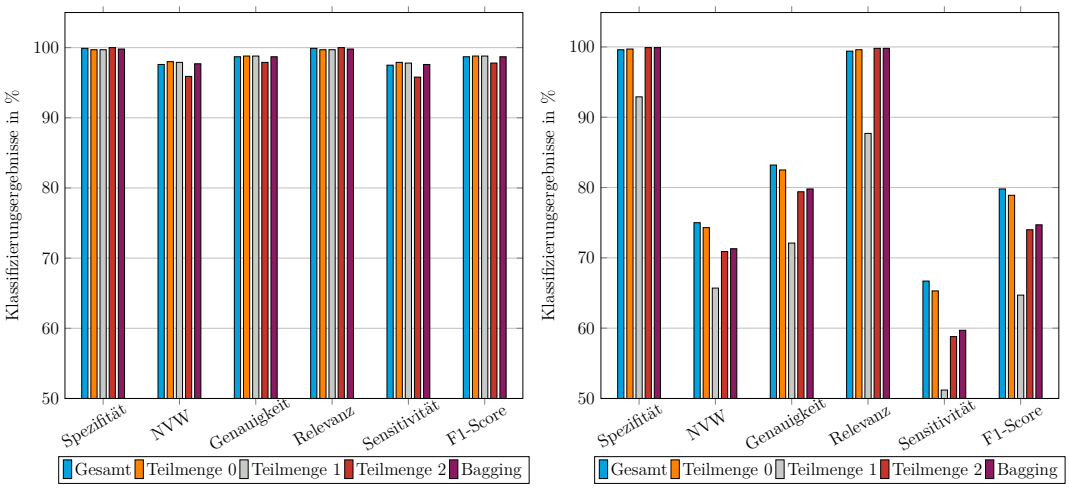
Das Experiment wurde für die ISCAS'89 Benchmark-Schaltung s298 wiederholt. Die Ergebnisse sind in der Tabelle 5.6 äquivalent zur Tabelle 5.5 aufgeführt. Im Vergleich der Ergebnisse in Abbildung 5.9 fällt auf, dass der negative Vorhersagewert und die Sensitivität im Vergleich zur Schaltung s27 geringer ausfällt und somit die Produktqualität nicht so stark verbessert werden kann. Die Ausbeute hingegen kann auch für die Schaltung s298 gesteigert werden, was an der hohen Spezifität und Relevanz erkennbar ist. Aufgrund der Größe der Schaltung ist es schwieriger zwischen Übersprechen und Prozessvariation zu unterscheiden.

Tabelle 5.5: Klassifizierungsergebnisse nach der Merkmalauswahl für die Benchmark-Schaltung s27 [Sprenger2020]

	Kalt-Warm-Test	**Kalt-Warm-Test - Teilmengen und *Bagging* (3 Teilmengen)**			
	42 **(2 x alle Spannungen)**	**6 (2 x 3 Spannungen)**			
		Teilmenge 0	Teilmenge 1	Teilmenge 2	*Bagging*
RN	4789	4782	4783	4796	4787
FP	7	14	13	0	9
FN	119	99	104	203	115
RP	4673	4693	4688	4589	4677
Spezifität	0,999	0,997	0,997	1,000	0,998
NVW	0,976	0,980	0,979	0,959	0,977
Genauigkeit	0,987	0,988	0,988	0,979	0,987
Relevanz	0,999	0,997	0,997	1,000	0,998
Sensitivität	0,975	0,979	0,978	0,958	0,976
F1-Score	0,987	0,988	0,988	0,978	0,987
Hyperparameter	(128, 256)	(64, 32, 32)	(128, 32, 32)	(128, 32, 32)	
Merkmalauswahl (T, U)	2 Temperaturen mit allen Spannungen	((45 °C,1,0 V), (45 °C,0,8 V), (45 °C,0,95 V), (125 °C,1,0 V), (125 °C,0,8 V), (125 °C,0,95 V))	((45 °C,0,6 V), (45 °C,0,7 V), (45 °C,0,65 V), (125 °C,0,6 V), (125 °C,0,7 V), (125 °C,0,65 V))	((45 °C,0,7 V), (45 °C,0,65 V), (45 °C,0,6 V), (125 °C,0,7 V), (125 °C,0,65 V), (125 °C,0,6 V))	

Tabelle 5.6: Klassifizierungsergebnisse nach der Merkmalauswahl für die Benchmark-Schaltung s298

	Kalt-Warm-Test	**Kalt-Warm-Test - Teilmengen und *Bagging* (3 Teilmengen)**			
	42 **(2 x alle Spannungen)**	**6 (2 x 3 Spannungen)**			
		Teilmenge 0	Teilmenge 1	Teilmenge 2	*Bagging*
RN	20 091	20 116	18 737	20 156	20 150
FP	84	59	1438	19	25
FN	6686	6971	9797	8274	8100
RP	13 410	13 125	10 299	11 822	11 996
Spezifität	0,996	0,997	0,929	0,999	0,999
NVW	0,750	0,743	0,657	0,709	0,713
Genauigkeit	0,832	0,825	0,721	0,794	0,798
Relevanz	0,994	0,996	0,877	0,998	0,998
Sensitivität	0,667	0,653	0,512	0,588	0,597
F1-Score	0,798	0,789	0,647	0,740	0,747
Hyperparameter	(256, 128, 64)	(256, 128)	(256, 256, 64)	(32, 256)	
Merkmalauswahl (T, U)	2 Temperaturen mit allen Spannungen	((45 °C,0,9 V), (45 °C,1,1 V), (45 °C,1,3 V), (125 °C,0,9 V), (125 °C,1,1 V), (125 °C,1,3 V))	((45 °C,0,6 V), (45 °C,0,7 V), (45 °C,0,65 V), (125 °C,0,6 V), (125 °C,0,7 V), (125 °C,0,65 V))	((45 °C,0,65 V), (45 °C,0,6 V), (45 °C,0,7 V), (125 °C,0,65 V), (125 °C,0,6 V), (125 °C,0,7 V))	



(a) s27 (b) s298

Abbildung 5.9: Klassifizierungsergebnisse.

5.6 Fazit

Übersprechen innerhalb der Logik-Schaltungen von hochintegrierten Schaltungen, kann zur Verringerung der Signalqualität führen. Durch die erhöhte Stromaufnahme kann dies, wie in Kapitel 2.2.2 erläutert wurde, zusätzlich zu vermehrter Elektromigration und zu einem Zuverlässigkeitsproblem führen. Dabei kann die Elektromigration ihrerseits wiederum zu vermehrtem Übersprechen führen. Während der gesamten Lebensdauer einer hochintegrierten Schaltung ist es daher notwendig, einen Test auf Übersprechen durchzuführen. In diesem Kapitel wurde gezeigt, dass eine große Herausforderung für den Test das Unterscheiden von Verzögerungen aufgrund von Übersprechen und von Prozessvariationen ist. Des Weiteren wurde anhand einer Fallstudie gezeigt, dass dieses Problem mithilfe von künstlichen neuronalen Netzen gelöst werden kann.

Je nach Anwendung kann die Klassifizierung auf unterschiedliche Ziele angepasst werden. Für Anwendungen mit hohen Anforderungen an die Zuverlässigkeit ist ein Klassifizierer mit einem hohen negativen Vorhersagewert (NVW) und einer hohen Sensitivtät notwendig, um zu verhindern, dass fehlerhafte Schaltungen fälschlicherweise als fehlerfrei klassifiziert werden und so zu einem Zuverlässigkeitsproblem führen. Eine Garantie zur korrekten Klassifizierung gibt es bei der Verwendung von künstlichen neuronalen Netzen jedoch nicht. Auch bei einer Sensitivität von 100 % kann es dazu kommen, dass eine Schaltung, die nicht zum Trainieren des künstlichen neuronalen Netzes verwendet wurde, fälschlicherweise als fehlerhaft oder im kritischeren Fall als fehlerfrei klassifiziert wird.

Für das Training der neuronalen Netze ist es nötig, aufwendige und zeitintensive Simulationen für die Erzeugung der Verzögerungskarten durchzuführen. Für weiterführende Arbeiten ist es daher notwendig, hybride Simulationstechniken auf Gatter- und Transistorebene durchzuführen.

6 Fazit

Neue Halbleitertechnologien mit minimalen Strukturgrößen von bis zu 3 nm [TSMC2023] und immer weiter steigende Anforderungen an die Leistungsfähigkeit von hochintegrierten Schaltungen erfordern es, immer komplexere Systeme auf einem Chip zu integrieren. Gleichzeitig wachsen die Anforderungen an den Test und die Diagnose von hochintegrierten Schaltungen. Neben dem steigenden Aufwand aufgrund immer weiter steigender Schaltungsgrößen müssen der Test und die Diagnose von hochintegrierten Schaltungen immer mehr Unsicherheiten berücksichtigen, die sich im Zeitverhalten der zu testenden Schaltung bemerkbar machen. Um diesen Anforderungen gerecht zu werden, ist es notwendig Instrumente zur Beobachtung oder Steuerung der hochintegrierten Schaltungen auf dem Chip zu integrieren. So werden z. B. Testinstrumente in die Schaltung integriert, um einen Test mit ausreichender Fehlerabdeckung zu erreichen oder um die Funktionsfähigkeit während der gesamten Lebenszeit der Schaltung zu überwachen. Daten, die während des gesamten Lebenszyklus einer hochintegrierten Schaltung gesammelt werden, können im Rahmen des *Silicon Lifecycle Managements* analysiert werden, um so z. B. die Zuverlässigkeit einer Schaltung zu erhöhen, einen katastrophalen Ausfall während des Betriebs zu verhindern oder die Alterung der Schaltung zu erkennen.

Für den wiederkehrenden Test auf versteckte kleine Verzögerungsfehler ist ein eingebauter Hochgeschwindigkeitstest notwendig. Während des eingebauten Hochgeschwindigkeitstests wird die hochintegrierte Schaltung mit unterschiedlichen Taktfrequenzen übertaktet, um versteckte kleine Verzögerungsfehler sichtbar zu machen. Aufgrund der erhöhten Taktfrequenz kommt es zu hohen und variierenden X-Raten. Die X-Raten variieren sowohl beim Wechsel der Taktfrequenz, als auch während des Tests mit einer Taktfrequenz. Um den Hochgeschwindigkeitstest als eingebauten Selbsttest zu realisieren, ist ein effizientes X-tolerantes Kompaktierungsverfahren für variierende X-Raten notwendig.

Neben kleinen Verzögerungsfehlern, die in Gattern von Logik-Schaltungen auftreten, können auch Verbindungsleitungen in Logik-Schaltungen Opfer von Veränderungen des Zeitverhaltens werden. Durch die immer weiter schrumpfenden Strukturgrößen und der

steigenden Komplexität hochintegrierter Schaltungen, rücken Verbindungsleitungen in der Logik-Schaltung immer näher zusammen, wodurch es zu unerwünschten parasitären Kopplungen zwischen den Verbindungsleitungen kommen kann. Aufgrund der parasitären Kopplung kommt es zu Übersprechen zwischen den Verbindungsleitungen einer Logik-Schaltung, was einerseits einen kleinen Verzögerungsfehler hervorrufen kann und andererseits zu einem erhöhten Stromfluss in den Verbindungsleitungen führt und Alterungseffekte wie Elektromigration begünstigt. Die verstärkte Elektromigration kann wiederum zu erhöhter parasitärer Kopplung führen. Gleichzeitig werden die Verzögerungen aufgrund von Übersprechen von Verzögerungen aufgrund von Prozessvariation überlagert. Ein Verfahren zur Unterscheidung der Verzögerungen aufgrund von Übersprechen und Prozessvariation ist daher notwendig.

Mit dieser Arbeit wurde ein Beitrag geleistet, den Test für das Auftreten der beschriebenen Unsicherheiten zu optimieren. Mit der in Kapitel 4 vorgestellten modularen Kompaktierung wurde ein räumliches Kompaktierungsverfahren vorgestellt, das flexibel an variierende X-Raten angepasst werden kann. Mithilfe des Verfahrens ist es möglich, hohe und variierende X-Raten während der Testantwortkompaktierung im eingebauten Selbsttest zu verarbeiten. Um auf die variierenden X-Raten bei unterschiedlichen Testfrequenzen reagieren zu können, wurde die flexible Struktur der stochastischen Kompaktierung ausgenutzt. Indem der Parameter p, der zur Bildung der Kompaktormatrix genutzt wird, je Testfrequenz angepasst wird, kann der Kompaktierer auf die aktuelle X-Rate eingestellt werden. Durch die Gruppierung der Prüfpfade in Prüfgruppen und die Aufteilung auf mehrere kleine stochastische Kompaktierer, kann die Variation der X-Rate während einer Testfrequenz reduziert werden. So konnte eine X-Reduktion erzielt werden, während die Fehlerdurchlässigkeit des Kompaktierungsverfahrens im Vergleich zur stochastischen Kompaktierung aufrecht erhalten werden konnte. Durch die Erweiterung des vorgestellten Verfahrens zur hybriden Kompaktierung ist es außerdem möglich, die Fehlerdurchlässigkeit weiter zu erhöhen und zu garantieren, dass essenzielle Fehlerinformation nicht verloren geht. Hierzu wird die flexible Struktur der stochastischen Kompaktierung ein weiteres Mal ausgenutzt. Nach der stochastischen Phase der Kompaktierung werden auf dem Chip gespeicherte deterministische Steuerungsvektoren genutzt, um gezielt verlorengegangene Fehlerinformation wiederzuerlangen. Durch das erneute Anlegen von 10 % der Testmuster konnte eine Steigerung der Fehlerdurchlässigkeit von bis zu 12,31 % im Vergleich zur stochastischen Kompaktierung erreicht werden.

Mithilfe des in Kapitel 5 gezeigten Verfahrens zur Unterscheidung von Übersprechen auf Verbindungsleitungen in Logik-Schaltungen und Prozessvariation ist es möglich die Zuverlässigkeit einer Schaltung zu gewährleisten, ohne gleichzeitig einen Ausbeuteverlust hinnehmen zu müssen. Hierzu wurde ein künstliches neuronales Netz trainiert, um Schal-

tungen anhand von Verzögerungskarten in Schaltungen mit Übersprechen (fehlerhaft) und ohne Übersprechen (fehlerfrei) einzuteilen. Eine Verzögerungskarte repräsentiert dabei das Zeitverhalten eines Ausgangs der zu testenden Schaltung in unterschiedlichen Arbeitspunkten, welche durch die Umgebungstemperatur und die Versorgungsspannung der Schaltung definiert werden. Die Anzahl der benötigten Arbeitspunkte konnte durch eine praxisorientierte Merkmalsextraktion auf sechs Arbeitspunkte reduziert werden. In einer Fallstudie für die ISCAS'89 Benchmark-Schaltung s27 konnte gezeigt werden, dass bei der Verwendung dieser sechs Arbeitspunkte ein mehrschichtiges Perzeptron mit einer Genauigkeit von 98,8 %, einer Spezifität von 99,7 % und einer Sensitivität von 97,9 % trainiert werden kann. Die Sensitivität kann dabei als Maß für die Produktqualität und die Spezifität für die Ausbeute interpretiert werden.

Durch die in dieser Arbeit gezeigten Beiträge zu Bewältigung von Unsicherheiten in aktuellen Halbleitertechnologien wurde ein großer Beitrag zum *Silicon Lifecycle Management* von komplexen hochintegrierten Schaltungen geleistet und somit zur Zuverlässigkeit moderner System-on-Chips und ihren anspruchsvollen Anwendungen in Bereichen des autonomen Fahrens, in der Medizintechnik, im Bereich der künstlichen Intelligenz oder im Bereich großer Rechenzentren.

7 Ausblick

Wie im vorherigen Kapitel erwähnt wurde, stellen die in dieser Arbeit vorgestellten Verfahren einen Beitrag zur Bewältigung von Unsicherheiten während des Lebenszyklus einer hochintegrierten Schaltung durch die Detektion von kleinen Verzögerungsfehlern dar. Durchaus sind aber auch Erweiterungen der vorgestellten Verfahren oder Anpassungen zur Verwendung im *Silicon Lifecycle Management* denkbar.

Die Verwendung der Fehlerberichte aus dem Hochgeschwindigkeitstest ermöglicht die in [Holst2020] vorgestellte Hochgeschwindigkeitsdiagnose. Mithilfe der Hochgeschwindigkeitsdiagnose ist es durch die Verwendung eines zweistufigen Verfahrens möglich, mit einer Erfolgsrate von 94 % den Verursacher eines Fehlverhaltens zu identifizieren. Hierzu wird in der ersten Phase der Diagnose die Anzahl der möglichen Verursacher durch eine Rückverfolgung des Fehlerverhaltens reduziert, um anschließend, in der zweiten Phase der Diagnose, durch eine effiziente GPU-unterstützte Simulation des Zeitverhaltens der fehlerhaften Schaltungen den Verursacher zu identifizieren. Durch die Verwendung der Hochgeschwindigkeitsdiagnose während des Fertigungstests ist es möglich, frühzeitig auf kleine Verzögerungsfehler in der gefertigten Schaltung zu reagieren, um so die fehlerhaften Teile pro Million (engl. Defective Parts Per Million, DPPM) zu verringern und den *Ramp-up* Prozess zu beschleunigen.

Eine wichtige Erweiterung für den Verbindungstest in Logik-Schaltungen ist es, die Klassifizierung anhand von typischen Testdaten durchzuführen. So ist es sowohl während des Fertigungstests also auch während eines periodischen Tests in der Regel nicht möglich die Laufzeitverzögerungen der Schaltungsausgänge zu messen, die für die Klassifizierung im vorgestellten Verbindungstest notwendig sind. Ein vielversprechender Ansatz ist es die Laufzeitverzögerungen mithilfe eines eingebauten Hochgeschwindigkeitstests zu bestimmen. Durch den Test bei unterschiedlichen Testfrequenzen lässt sich die Pass/Fail Information des Tests als Abtastung der Ausgangssignale der zu testenden Schaltung interpretieren. Mithilfe der so abgeschätzten Signalverläufe lassen sich wieder Verzögerungskarten generieren, die als Eingangsdaten für das mehrlagige Perzeptron verwendet werden können.

Neben der Unterscheidung von Übersprechen und Prozessvariation ist es, wie in [Sadeghi-Kohan2021] dargestellt, notwendig, Schaltungen periodisch auf Alterungseffekte wie Elektromigration zu testen, um so die Alterung der Schaltung zu überwachen. Ähnlich wie bei der Unterscheidung von Übersprechen und Prozessvariation kann das Zeitverhalten der Schaltungen mithilfe von Verfahren des maschinellen Lernens untersucht werden, um Schaltungsinstanzen zu identifizieren, bei denen Elektromigration aufgetreten ist. Durch die Anwendung des eingebauten Hochgeschwindigkeitstests auf den Verbindungstest in Logik-Schaltungen kann das Zeitverhalten der hochintegrierten Schaltung periodisch abgetastet werden, um aktuelle Verzögerungskarten zu erstellen. Hierbei stellt der Einfluss der Abtastung auf das Ergebnis der Klassifizierung eine interessante Forschungsfrage dar.

Eine Herausforderung für diese Erweiterung und den Verbindungstest in Logik-Schaltungen selbst ist die zeitaufwendige Simulation des Zeitverhaltens der Trainingsinstanzen auf Transistorebene. Daher ist die Entwicklung von effizienten Schaltungssimulatoren, die das Zeitverhalten von fehlerhaften Schaltungsinstanzen unter Prozessvariation für moderne Halbleitertechnologien ermöglichen, essenziell. Die Verwendung von GPU-unterstützten Schaltungssimulatoren auf Schalter-Ebene, wie in [Schneider2020; Schneider2019] vorgestellt, ist vielversprechend, um diese Herausforderung zu meistern.

Im vorgestellten Verbindungstest wird immer eine Verzögerungskarte eines Ausgangs für die Klassifizierung der Schaltung verwendet. Da ein Fehler jedoch an mehrere Ausgänge propagiert werden kann, ist es außerdem interessant, zu untersuchen, wie die Verzögerungskarten aller Ausgänge kombiniert werden können, um die Klassifizierungsergebnisse zu verbessern.

Eine weitere vielversprechende Erweiterung der vorgestellten Arbeit ist die Wiederverwendung von eingebauten Testinstrumenten wie die Testantwortkompaktierung von unterschiedlichen Modulen eines System-on-Chips. Im Stand der Technik wird für jedes Modul eines System-on-Chips ein eigener eingebauter Selbsttest entwickelt und in jedes Modul einzeln integriert. Durch die Verwendung von flexiblen Testantwortkompaktierungsverfahren, wie dem modularen Kompaktierer, ist es möglich, die Testantwortkompaktierung auf einem SoC zu zentralisieren. Im Sinne des *Silicon Lifecycle Managements* ist es möglich, die nötigen Informationen zwischen den Modulen auszutauschen. Durch Verwendung eines zentralen Kompaktierungs-Moduls wäre es möglich, redundante Testhardware zu vermeiden, um so die benötigte Chipfläche zu verringern. Alternativ kann die gewonnene Chipfläche genutzt werden, um anspruchsvolle Algorithmen für die Testantwortkompaktierung zu implementieren.

Dies sind nur einige Ansätze zur Bewältigung von Unsicherheiten in modernen Halbleitertechnologien, die eine weitere Erforschung von Testinstrumenten und Methoden zur Testdatenanalyse für das *Silicon Lifecycle Mangagement* notwendig machen.

Symbolverzeichnis

(a_{ij})	Matrix $\boldsymbol{A}$ mit den Elementen a_{ij}
$(x, y]$	Linksoffenes Intervall zwischen x und y
$[x, y]$	Geschlossenes Intervall zwischen x und y
$\mathbf{0}$	Nullvektor
$\bar{\boldsymbol{x}}$	Mittelwert des Trainingdatensatzes
$\boldsymbol{A}$	Matrix $\boldsymbol{A}$
$\boldsymbol{C_{sub}}$	Untermatrix der Kompaktormatrix $\boldsymbol{C}$
$\boldsymbol{C}$	Kompaktormatrix
$\boldsymbol{KV}$	Kovarianzmatrix eines Trainingdatensatzes
$\boldsymbol{l}$	Labelmatrix eines künstlichen neuronalen Netzes
$\boldsymbol{R}$	Zustandsübergangsmatrix des MISRs
$\boldsymbol{S}$	Matrixdarstellung der Testsignaturen
$\boldsymbol{T}$	Matrixdarstellung der Testantworten
$\boldsymbol{T}^{[i]}$	i-te Spalte der Testantwortmatrix
$\boldsymbol{w}$	Vektordarstellung der Gewichte eines künstlichen Neurons
$\boldsymbol{x}$	Eingangsvektor eines künstlichen Neurons
$\boldsymbol{x}^T$	Transponierter Eingangsvektor eines künstlichen Neurons
$\boldsymbol{y}$	Ausgabematrix eines künstlichen neuronalen Netzes
$\boldsymbol{Z}(t)$	Zustandsvektor der den Zustand des MISRs zum Zeitpunkt t beschreibt
δ	Fehlergröße eines kleinen Verzögerungsfehlers
$\Delta(o, op, \varphi_{CC}, p_{add})$	Relative Veränderung der Verzögerung
$\emptyset$	Leere Menge
$\eta(t_i, t_{i'})$	Ähnlichkeits-Index der Prüfpfadbelegungen t_i und $t_{i'}$
γ	X-Kapazität eines Behälters in der Bin-Packing Prüfgruppeneinteilung

$\Gamma(i)$	Menge aller Prüfgruppen, welche den i-ten Prüfpfad enthalten
$\mathbb{R}$	Menge der reellen Zahlen
$\mathbb{R}^m$	m-dimensionaler reeller Raum
$\mathscr{A}$	Menge der Ausgänge der zu testenden Schaltung
$\mathscr{F}$	Fehlermenge eines gegebenen Fehlermodells
$\mathscr{F}(\boldsymbol{T})$	Fehlermenge, welche mit der Testanwortmatrix $\boldsymbol{T}$ erkannt werden kann
$\mathscr{F}(t_{ij})$	Fehlermenge des j-ten Prüfpfadbits des i-ten Prüfpfades
$\mathscr{G}$	Menge der Prüfgruppen
$\mathscr{G}_i$	Die i-te Prüfgruppe
$\mathscr{G}_{Total}$	Prüfgruppe mit allen Prüfpfaden
$\mathscr{M}$	Menge an Zeilen einer Kompaktormatrix
$\mathscr{NC}$	Menge der nicht gruppierten Prüfpfade
$\mathscr{NVC}$	Menge der noch nicht eingruppierten Prüfpfade während der varianzorientierten Prüfgruppeneinteilung
$\mathscr{OP}$	Menge der verwendeten Arbeitspunkte
$\mathscr{S}$	Menge der Prüfpfade in der zu testenden Schaltung
$\mathscr{T}$	Menge der verwendeten Temperaturen
$\mathscr{T}_M$	Testmustermenge
$\mathscr{U}$	Menge der verwendeten Versorgungsspannungen
μ	Mittelwert
$\oplus$	XOR-Operator
$\mathcal{O}$	O-Notation
$\phi(z)$	Aktivierungsfunktion eines künstlichen Neurons
Ψ	Transformationsfunktion eines Kompaktierers
σ	Standardabweichung
σ^2	Varianz
$\sigma^2(X_{Count}(\mathscr{G}_{Total}))$	Varianz des Vektors $X_{Count}(\mathscr{G}_{Total})$
$\varphi_1, \varphi_2, \ldots$	Fehler in einer hochintegrierten Schaltung
φ_{CC}	Größe einer parasitären Koppelkapazität
A	Materialkonstante
b	Bias-Term eines künstlichen Neurons
c	Aktueller Takt
$C_1, C_2, \ldots$	Kapazitäten

c_{ij}	Element der Kompaktormatrix $\boldsymbol{C}$
$d_{pd}(o, op, \varphi_{CC}, p_{add})$	Laufzeitverzögerung
$E(\boldsymbol{w})$	Fehlerfunktion eines künstlichen neuronalen Netzes für die Gewichte $\boldsymbol{w}$
e^x	Exponentialfunktion
f	Anzahl Frequenzen während des Hochgeschwindigkeitstests
$f(X)$	Charakteristische Polynom
f_{CLK}	Taktfrequenz der hochintegrierten Schaltung
f_{FAST}	Taktfrequenz im Hochgeschwindigkeitstest
f_{nom}	Nominelle Taktfrequenz
GND	Massepotential der hochintegrierten Schaltung
$h_1, h_2, \ldots$	Koeffizienten im charakteristischen Polynom eines MISRs, die die lineare Rückkopplung repräsentieren
$h_{\boldsymbol{w}}(\boldsymbol{x})$	Ausgabe eines künstlichen Neurons mit den Gewichten $\boldsymbol{w}$ für den Eingangsvektor $\boldsymbol{x}$
$i, j, k, l, ...$	Laufindex
$i_1, i_2, \ldots$	Eingänge einer hochintegrierten Schaltung
J	Stromdichte
k_1	Anzahl der sichtbaren Fehlverhalten an den Eingängen eines linearen Kompaktierers
k_2	Anzahl der X-Werte an den Eingängen eines linearen Kompaktierers
k_B	Boltzmann-Konstante
k_x	Anzahl X-Werte am Eingang des stochastischen Kompaktierers
KR	Kompaktierungsrate eines Testantwortkompaktierers
L	Anzahl der versteckten Ebenen
$L_1, L_2, \ldots$	Induktivitäten
L_{Gate}	Gatelänge eines Transistors
L_{nom}	Nominelle Gatelänge
L_{var}	Gatelänge mit Prozessvariation
M	Benötigter Speicherplatz für die modulare Kompkatierung
m, n, p, q	Matrix-Dimensionen
$max(x, y)$	Maximumfunktion
n	Modellparameter
n_c	Grad des charakteristischen Polynoms

n_g	Anzahl der Gatter
n_l	Anzahl der Neuronen auf Ebene l
n_s	Anzahl der Signalleitungen
n_z	Anzahl nötiger Zwischensignaturen eines X-Canceling MISRs
$numRuns$	Parameter der varianzorientierten Prüfgruppeneinteilung
$o_1, o_2, \ldots$	Ausgänge einer hochintegrierten Schaltung
op	Arbeitspunkt
p_{add}	Prozentuale Änderung der nominellen Gatelänge
p_{ij}	Wahrscheinlichkeit, dass der Eintrag c_{ij} in der Kompaktormatrix 1 ist
Q	Aktivierungsenergie
$R_1, R_2, \ldots$	Widerstände
$relu(x)$	ReLU-Funktion
s_i	i-te Prüfpfad
s_{ij}	Element der Signaturmatrix $\boldsymbol{S}$
$sig(x)$	Sigmoidfunktion
$softmax(\boldsymbol{x}_j)$	Softmax-Funktion für den Eingangsvektor $\boldsymbol{x}$ am Ausgang j eines künstlichen Neuronalen Netzes mit K Ausgängen
$step(x)$	Stufenfunktion
T	Umgebungstemperatur
t_0	Startzeitpunkt
t_s	Beobachtungszeitpunkt
t_T	Schaltzeitpunkt
T_{CLK}	Taktperiode
T_{FAST}	Taktperiode im Hochgeschwindigkeitstest
t_{ij}	Element der Testantwortmatrix $\boldsymbol{T}$
T_{nom}	Nominelle Taktperiode einer Schaltung
t_{nom}	Nominelle Beobachtungszeit
$tanh(x)$	Tangens hyperbolicus
U_{DD}	Versorgungsspannung der hochintegrierten Schaltung
U_{nom}	Nominelle Versorgungsspannung
U_{TH}	Schwellenspannung
w	Anzahl Wahrscheinlichkeiten des WPRPG
w_i	i-te Gewicht des i-ten Eingangs eines künstlichen Neurons

X	Zufallsvariable zur Beschreibung der Koppelkapazitätsgröße
x_i	i-te Element im Eingangsvektor eines künstlichen Neurons
x_j	Binäre Variable zur Repräsentation der Prüfgruppenauswahl
$X_{Count}(\mathscr{G}_{Total})$	Vektor der X-Verteilung pro Takt in der Prüfgruppe $\mathscr{G}_{Total}$
$X_{Count}(s_i)$	Vektor der X-Verteilung pro Takt des aktuellen Prüfpfades s_i
$X^{new}_{Count}(\mathscr{G}_j)$	Vektor der X-Verteilung pro Takt nach dem Hinzufügen des aktuellen Prüfpfades
$X^{old}_{Count}(\mathscr{G}_j)$	Vektor der X-Verteilung pro Takt vor dem Hinzufügen des aktuellen Prüfpfades
X_{Total}	Anzahl der X-Werte in allen Testantworten
$X_{Total}(\mathscr{G}_j)$	Anzahl der X-Werte in allen Testantworten der Prüfgruppe $\mathscr{G}_j$
Y	Zufallsvariable zur Beschreibung der Prozessvariation
z	Gewichtete Summe der Eingänge eines künstlichen Neurons
$\boldsymbol{x}_i$	i-te Eingangsvektor ein Trainingsdatensatzes
FN	Anzahl der falsch negativen Vorhersagen
FP	Anzahl der falsch positiven Vorhersagen
RN	Anzahl der richtig negativen Vorhersagen
RP	Anzahl der richtig positiven Vorhersagen

Abbildungsverzeichnis

1.1 Aufnahme eines *Apple M1 Ultra* [Apple2022]. 1
1.2 Überblick über das *Silicon Lifecycle Management* [Crosher2022]. 3
1.3 Schema eines eingebauten Selbsttests. 4

2.1 Mögliche Defekte in hochintegrierten Schaltungen [Zivkovic2011]. 8
2.2 Veränderung der Dimensionen von Verbindungsleitungen in hochintegrierten Schaltungen [Geden2011]. 10
2.3 Beispiele von Elektromigration [Geden2011]. 10
2.4 Beispiel eines Haftfehlers-an-1 und eines Haftfehlers-an-0 am Eingang A eines UND-Gatters. 12
2.5 Unterschied zwischen Verzögerungsfehlern und kleinen Verzögerungsfehlern. 15
2.6 RLC Fehlermodell zur Modellierung von Übersprechen [Cuviello1999]. . . 17
2.7 Fehlverhalten aufgrund von Übersprechen [Chen1998]. 17
2.8 Beispiel einer zu testenden Schaltung (CUT) mit Testmustern und Testantworten. 18
2.9 System-on-Chip mit eingebautem Selbsttest. 20
2.10 Beispiel zweier Prüfpfadkonfigurationen. 21
2.11 STUMPS Architektur [Bardell1982]. 21
2.12 Beispiel einer Testantwortmatrix 23
2.13 Beispiel einer linearen Kompaktierung mithilfe zweier XOR-Bäume. . . . 25
2.14 Beispiel einer Fehlermaskierung. 26
2.15 Beispiel einer linearen Kompaktierung mit einem unbekannten Wert. . . 27
2.16 Signaturregister mit mehreren Eingängen (MISR). 29
2.17 Detektion eines versteckten kleinen Verzögerungsfehlers (HDF) mithilfe eines Hochgeschwindigkeitstests [Sprenger2019]. 32
2.18 Testarchitektur für den Hochgeschwindigkeitstest [Sprenger2019]. 33

3.1 Beispiel eines künstlichen Neurons. 35
3.2 Verlauf der Aktivierungsfunktionen $step(x)$, $sig(x)$, $tanh(x)$ und $relu(x)$. 37
3.3 Beispiel eines Perzeptrons. 38
3.4 Linear separierbares Klassifizierungsproblem. 38

3.5 Künstliches neuronales Netz mit zwei versteckten Ebenen. 39
3.6 Schema der Bagging Ensemble-Methode [Géron2020]. 43

4.1 X-Verteilung für b17_1 [Sprenger2019]. 45
4.2 Beispiel einer X-Blockierung. 47
4.3 Beispiel einer X-Maskierung. 47
4.4 Aufbau eines X-Kompaktierers und Matrixrepräsentation. 49
4.5 Implementierung eines stochastischen Kompaktierers [Sprenger2019]. . . 52
4.6 Beispiel stochastischer Kompaktierer. 53
4.7 Testarchitektur für den Hochgeschwindigkeitstest mit modularer Kompaktierung [Sprenger2019]. 54
4.8 Beispielkonfiguration der modularen Kompaktierer [Sprenger2019]. 55
4.9 Beispiel für die testunabhängige Prüfgruppeneinteilung [Sprenger2019]. . 59
4.10 Beispiel für die Bin-Packing Prüfgruppeneinteilung [Sprenger2019]. . . . 61
4.11 Beispiel für die varianzorientierte Prüfgruppeneinteilung [Sprenger2019]. 63
4.12 Prüfgruppeneinteilung mit unterschiedlichen Algorithmen für fünf Testfrequenzen. 65
4.13 Stochastische Kompaktieurng mit X-Pfad-Blockierung und Pfad-Bypass für essenzielle Prüfpfade. 67
4.14 Kompaktierungsrate vs. Prüfgruppengröße [Sprenger2019]. 69
4.15 Einfluss der Kompaktierungsrate auf den X-Reduktionsfaktor (XRF) für die testunabhängige Prüfgruppeneinteilung bei einer Prüfgruppengröße von 32 [Sprenger2019]. 69
4.16 Einfluss der Kompaktierungsrate auf die Fehlerdurchlässigkeit für die testunabhängige Prüfgruppeneinteilung bei einer Prüfgruppengröße von 32 [Sprenger2019]. 70
4.17 Einfluss der Prüfgruppengröße auf den X-Reduktionsfaktor (XRF) für die testunabhängige Prüfgruppeneinteilung bei einer Kompaktierungsrate von 2 [Sprenger2019]. 70
4.18 Einfluss der Prüfgruppengröße auf die Fehlerdurchlässigkeit (FD) für die testunabhängige Prüfgruppeneinteilung bei einer Kompaktierungsrate von 2 [Sprenger2019]. 70
4.19 Vergleich zwischen optimaler Lösung und Heuristiken für die Schaltung b17_1 [Sprenger2018a; Sprenger2018b; Sprenger2019]. 73
4.20 X-Reduktionsfaktor (XRF) und Fehlerdurchlässigkeit (FD) bei zufälliger Prüfpfadkonfiguration [Sprenger2019]. 74
4.21 Vergleich von X-Reduktionsfaktor (XRF) und Fehlerdurchlässigkeit (FD) für zufällige und graphbasierte Prüfpfadkonfigurationen [Sprenger2019]. . 76

4.22 Einfluss von X-Pfad-Blockierung und Pfad-Bypass auf X-Reduktionsfaktor (XRF) und Fehlerdurchlässigkeit (FD) bei einer Kompaktierungsrate von 16 [Sprenger2019]. 76
4.23 Verlorene Fehler für die Benchmark-Schaltung *bench5* (KR=8) [Maaz2019a]. 79
4.24 Hybride Kompaktierung [Maaz2019b; Maaz2019a]. 80
4.25 Beispiel einer Eingangsgruppeneinteilung für die hybride Kompaktierung. 82
4.26 Beispiel für den Ähnlichkeits-Index η [Maaz2019a]. 84
4.27 Stochastische Kompaktierung vs. hybride Kompaktierung - Vergleich der Fehlerdurchlässigkeit (FD) bei einem Kompaktierungsrate (KR) von 16 und 32. 88
4.28 Entwicklung der Fehlerdurchlässigkeit bei steigender Anzahl der wiederverwendeten Testmuster [Maaz2019a]. 89
4.29 Stochastische Kompaktierung vs. hybride Kompaktierung - Vergleich des XRFs bei einer Kompaktierungsrate von 16 und 32. 90
4.30 Speicheraufwand bei einer Kompaktierungsrate (KR) von 16 und 32. . . 92

5.1 Verteilung der Laufzeitverzögerungen für Schaltungen bei denen Prozessvariation und Übersprechen auftritt [Sprenger2020]. 96
5.2 Beispiel eines Fehlerortes für mögliches Übersprechen. 100
5.3 Veränderung der Laufzeitverzögerungen aufgrund von Übersprechen [Sprenger2020]. 101
5.4 Veränderung der Laufzeitverzögerungen aufgrund von Prozessvariationen [Sprenger2020]. 102
5.5 Veränderungen der Laufzeitverzögerungen aufgrund von Prozessvariationen bei einer Koppelkapazität von 100 fF [Sprenger2020]. 103
5.6 Verzögerungskarte für den Ausgang o_0 bei $p_{add} = 6{,}00\,\%$ und $\varphi_{CC} = 20{,}00\,\mathrm{fF}$. 103
5.7 Konfiguration des künstlichen neuronalen Netzes. 105
5.8 Merkmalreduktion mithilfe von *Feature Bagging*. 107
5.9 Klassifizierungsergebnisse. 114

Tabellenverzeichnis

4.1 Eigenschaften der verwendeten Benchmark-Schaltungen [Sprenger2019] . 68
4.2 Einfluss der Kompaktierungsrate (KR) auf X-Reduktionsfaktor (XRF) und Fehlerdurchlässigkeit (FD) bei einer Prüfgruppengröße von 32 71
4.3 Einfluss der Prüfgruppengröße ($|\mathscr{G}_j|$) auf X-Reduktionsfaktor (XRF) und Fehlerdurchlässigkeit (FD) bei einer Kompaktierungsrate von 2 71
4.4 X-Reduktionsfaktor (XRF) [Sprenger2018b] 72
4.5 Fehlerdurchlässigkeit (FD) [Sprenger2018b] 73
4.6 Vergleich zwischen zufälliger und graphbasierter Prüfpfadkonfiguration [Sprenger2019] 75
4.7 Schaltungseigenschaften 87
4.8 Fehlerdurchlässigkeit (FD) für $KR = 16$ und $KR = 32$ bei der Wiederanwendung von 10 % der Testmuster [Maaz2019a] 87
4.9 X-Reduktionsfaktor für $KR = 16$ und $KR = 32$ bei der Wiederverwendung von 10 % der Testmuster [Maaz2019a] 89
4.10 Anzahl der X-Werte an den Ausgängen des Kompaktierers bei $KR = 16$ und $KR = 32$ bei der Wiederverwendung von 10 % der Testmuster [Maaz2019a] 90
4.11 X-Reduktionsfaktor (XRF) während der deterministischen Phase für $KR = 16$ und $KR = 32$ bei einer Wiederanwendung von 10 % der Testmuster [Maaz2019a] 91
4.12 Speichergröße für $KR = 16$ und $KR = 32$ bei der Wiederverwendung von 10 % der Testmuster (kbit) [Maaz2019a] 92

5.1 Datensatzaufteilung [Sprenger2020] 108
5.2 Konfusionsmatrix *GridSearchCV* [Sprenger2020] 109
5.3 Konfusionsmatrix *RandomizedSearchCV* [Sprenger2020] 109
5.4 Evaluierung der Hyperparametersuche [Sprenger2020] 111
5.5 Klassifizierungsergebnisse nach der Merkmalauswahl für die Benchmark-Schaltung s27 [Sprenger2020] 113

5.6 Klassifizierungsergebnisse nach der Merkmalauswahl für die Benchmark-Schaltung s298 113

Algorithmenverzeichnis

4.1 Testunabhängige Prüfgruppeneinteilung [Sprenger2019] 58
4.2 Bin-Packing Prüfgruppeneinteilung [Sprenger2019] 60
4.3 Varianzorientierte Prüfgruppeneinteilung [Sprenger2019] 62
4.4 Eingangsgruppeneinteilung [Maaz2019a] 85
4.5 Algorithmus zur Bildung neuer Eingangsgruppen [Maaz2019a] 85

Abkürzungsverzeichnis

ATE	Automatische Testausrüstung (engl. Automatic Test Equipment)
ATPG	Automatisierte Testmustererzeugung (engl. Automatic Test Pattern Generation)
BTI	Bias Temperaturinstabilität (engl. Bias Temperature Instability)
CUT	Zu testende Schaltung (engl. Circuit Under Test)
DPPM	Fehlerhafte Teile pro Million (engl. Defective Parts Per Million)
ELF	Frühausfall (engl. Early Life Failure)
FAST	Hochgeschwindigkeitstest (engl. Faster-than-At-Speed Test)
FCLGA	Flip Chip Land Grid Array
FD	Fehlerdurchlässigkeit
GPU	Grafikprozessor (engl. Graphics Processing Unit)
HCI	Injektion heißer Ladungsträger (engl. Hot Carrier Injection)
HDF	Versteckter kleiner Verzögerungsfehler (engl. Hidden Delay Fault)
LFSR	Linear rückgekoppeltes Schieberegister (engl. Linear Feedback Shift Register)
MISR	Signaturregister mit mehreren Eingängen (engl. Multiple-Input Signature Register)
MLP	Mehrschichtiges Perzeptron (engl. Multi-Layer Perceptron)
NBTI	Negative Bias-Temperaturinstabilität (engl. Negative Bias Temperature Instability)

PBTI	Positive Bias-Temperaturinstabilität (engl. Positive Bias Temperature Instability)
SDF	Kleiner Verzögerungsfehler (engl. Small Delay Fault)
SoC	System-on-Chip
SRSG	Shift Register Sequence Generator
STUMPS	Self-Test Using MISR and Parallel SRSG
SVM	Support Vektor Maschine (engl. Support Vector Machine)
TAE	Testantwortevaluierung (engl. Testresponse Evaluation)
TMG	Testmustergenerator (engl. Testpattern Generator)
VLSI	Hochintegrierte Schaltung (engl. Very Large Scale Integrated circuit)
WPRPG	Gewichteter Pseudo-Zufallsmustergenerator (engl. Weighted Pseudo-Random Pattern Generator)
XRF	X-Reduktionsfaktor

Literaturverzeichnis

[Abu-Mostafa2012] Y. S. Abu-Mostafa, M. Magdon-Ismail und H.-T. Lin. *Learning Fom Data: A short course*. AMLbook, Jan. 2012. ISBN: 978-1-60049-006-4.

[Aghaee2014] N. Aghaee, Z. Peng und P. Eles. „Process-Variation Aware Multi-Temperature Test Scheduling“. In: *IEEE International Conference on VLSI Design and IEEE International Conference on Embedded Systems (VLSID/ES)*. Jan. 2014, S. 32–37. DOI: 10.1109/vlsid.2014.13.

[Ahmed2006] N. Ahmed, M. Tehranipoor und V. Jayaram. „A Novel Framework for Faster-Than-at-Speed Delay Test Considering IR-drop Effects“. In: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. Nov. 2006, S. 198–203. DOI: 10.1109/iccad.2006.320136.

[Ahmed2010] N. Ahmed und M. Tehranipoor. „A Novel Faster-Than-at-Speed Transition-Delay Test Method Considering IR-Drop Effects“. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 28.10 (Sep. 2010), S. 1573–1582. DOI: 10.1109/tcad.2009.2028679.

[Aitken2008] R. C. Aitken. „Defect or Variation? Characterizing Standard Cell Behavior at 90 nm and Below“. In: *IEEE Transactions on Semiconductor Manufacturing (TSM)* 21.1 (Feb. 2008), S. 46–54. DOI: 10.1109/tsm.2007.913191.

[AMD2023] AMD. *AMD Prozesorren für Desktops*. 2023. URL: https://www.amd.com/de/products/processors-desktop (zuletzt besucht am 24. 03. 2023).

[Amuru2022] D. Amuru, H. V. Vudumula, P. K. Cherupally, S. R. Gurram, A. Ahmad, A. Zahra und Z. Abbas. *AI/ML Algorithms and Applications in VLSI Design and Technology*. Feb. 2022. DOI: 10.

48550/arxiv.2202.10015. URL: https://arxiv.org/abs/2202.10015.

[Apple2022] Apple. *Apple stellt den M1 Ultra vor, den weltweit leistungsstärksten Chip für einen Personal Computer*. 2022. URL: https://www.apple.com/de/newsroom/2022/03/apple-unveils-m1-ultra-the-worlds-most-powerful-chip-for-a-personal-computer/ (zuletzt besucht am 24. 03. 2023).

[Asokan2015] A. Asokan, A. Bosio, A. Virazel, L. Dilillo, P. Girard und S. Pravossoudovitch. „An ATPG Flow to Generate Crosstalk-Aware Path Delay Pattern". In: *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. Juli 2015, S. 515–520. DOI: 10.1109/isvlsi.2015.99.

[Bai2003] X. Bai, S. Dey und A. Krstid. „HyAC: A Hybrid Structural SAT Based ATPG for Crosstalk". In: *IEEE International Test Conference (ITC)*. März 2003, S. 112–121. DOI: 10.1109/test.2003.1270831.

[Balas1976] E. Balas und M. W. Padberg. „Set Partitioning: A Survey". In: *SIAM Review* 18.4 (Okt. 1976), S. 710–760. DOI: 10.1137/1018115.

[Bardell1982] P. H. Bardell und W. H. McAnney. „Self-Testing of Multichip Logic Modules". In: *IEEE International Test Conference (ITC)*. Nov. 1982, S. 200–204.

[Bellman1961] R. E. Bellman. *Adaptive Control Processes*. Princeton University Press, 1961. ISBN: 978-1-40087-466-8.

[Bhatnagar2017] S. Bhatnagar, D. Ghosal und M. H. Kolekar. „Classification of Fashion Article Images using Convolutional Neural Networks". In: *IEEE International Conference on Image Information Processing (ICIIP)*. März 2017, S. 1–6. DOI: 10.1109/iciip.2017.8313740.

[Bishop2016] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2016. ISBN: 978-1-4939-3843-8.

[Black1969] J. R. Black. „Electromigration - A Brief Survey and Some Recent Results". In: *IEEE Transactions on Electron Devices (T-ED)* 16.4 (Apr. 1969), S. 338–347. DOI: 10.1109/t-ed.1969.16754.

[Borkar2011] S. Borkar und A. A. Chien. „The Future of Microprocessors“. In: *Communications of the ACM* 54.5 (Mai 2011), S. 67–77. DOI: 10.1145/1941487.1941507.

[Breiman1996] L. Breiman. „Bagging Predictors“. In: *Machine Learning* 24.2 (1996), S. 123–140. DOI: 10.1007/bf00058655.

[Brglez1989] F. Brglez, D. Bryan und K. Kozminski. „Combinational Profiles of Sequential Benchmark Circuits“. In: *IEEE International Symposium on Circuits and Systems (ISCAS)*. Mai 1989, S. 1929–1934. DOI: 10.1109/iscas.1989.100747.

[Bushnell2000] M. L. Bushnell und V. D. Agrawal. *Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits*. Springer, 2000. ISBN: 0-7923-799-1-8.

[Chakrabarty1998] K. Chakrabarty, B. T. Murray und J. P. Hayes. „Optimal Zero-Aliasing Space Compaction of Test Responses“. In: *IEEE Transactions on Computers* 47.11 (Nov. 1998), S. 1171–1187. DOI: 10.1109/12.736427.

[Chen1998] H. H. Chen und J. S. Neely. „Interconnect and Circuit Modeling Techniques for Full-Chip Power Supply Noise Analysis“. In: *IEEE Transactions on Components, Packaging, and Manufacturing Technology: Part B* 21.3 (Aug. 1998), S. 209–215. DOI: 10.1109/96.704931.

[Chen1999] W.-Y. Chen, S. K. Gupta und M. A. Breuer. „Test Generation for Crosstalk-Induced Delay in Integrated Circuits“. In: *IEEE International Test Conference (ITC)*. Sep. 1999, S. 191–200. DOI: 10.1109/test.1999.805630.

[Chun2009] S. Chun, T. Kim und S. Kang. „ATPG-XP: Test Generation for Maximal Crosstalk-Induced Faults“. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 28.9 (Aug. 2009), S. 1401–1413. DOI: 10.1109/tcad.2009.2028165.

[Cireşan2010] D. C. Cireşan, U. Meier, L. M. Gambardella und J. Schmidhuber. „Deep, Big, Simple Neural Nets for Handwritten Digit Recognition“. In: *Neural Computation* 22.12 (Dez. 2010), S. 3207–3220. DOI: 10.1162/neco_a_00052.

[Corno2000] F. Corno, M. S. Reorda und G. Squillero. „RT-Level ITC'99 Benchmarks and First ATPG Results". In: *IEEE Design and Test of Computers* 17.3 (Juli 2000), S. 44–53. DOI: 10.1109/54.867894.

[Crosher2022] S. Crosher. *Silicon Lifecycle Management: Actionable Silicon Insights Through Intelligent Measurement and Analysis*. White Paper. Synopsys, Nov. 2022. URL: https://www.synopsys.com/content/dam/synopsys/solutions/slm/whitepapers/slm-whitepaper.pdf (zuletzt besucht am 24. 03. 2023).

[Cuviello1999] M. Cuviello, S. Dey, X. Bai und Y. Zhao. „Fault Modeling and Simulation for Crosstalk in System-on-Chip Interconnects". In: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. Nov. 1999, S. 297–303. DOI: 10.1109/iccad.1999.810665.

[Czysz2010] D. Czysz, G. Mrugalski, N. Mukherjee, J. Rajski und J. Tyszer. „On Compaction Utilizing Inter and Intra-Correlation of Unknown States". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 29.1 (Jan. 2010), S. 117–126. DOI: 10.1109/tcad.2009.2035550.

[DasGupta1995] S. DasGupta, R. G. Walther, T. W. Williams und E. B. Eichelberger. „An Enhancement to LSSD and some Applications of LSSD in Reliability, Availability, and Serviceabilit". In: *IEEE International Symposium on Fault-Tolerant Computing (FTCS)*. Juni 1995, S. 289–291. DOI: 10.1109/ftcsh.1995.532648.

[Davidson1999] S. Davidson. *ITC'99 Benchmark Homepage*. 1999. URL: https://www.cerc.utexas.edu/itc99-benchmarks/bench.html (zuletzt besucht am 24. 03. 2023).

[Duda2012] R. O. Duda, P. E. Hart und D. G. Stork. *Pattern Classification*. Wiley, 2012. ISBN: 978-1-11858-600-6.

[Eggersglüß2010] S. Eggersglüß, D. Tille und R. Drechsler. „Efficient Test Generation with Maximal Crosstalk-Induced Noise using Unconstrained Aggressor Excitation". In: *IEEE International Symposium on Circuits and Systems (ISCAS)*. Mai 2010, S. 649–652. DOI: 10.1109/iscas.2010.5537503.

[Eggersglüß2012] S. Eggersglüß und R. Drechsler. *High Quality Test Pattern Generation and Boolean Satisfiability*. Springer, 2012. ISBN: 978-1-4419-9975-7.

[Floudas1995] C. A. Floudas. *Nonlinear and Mixed-Integer Optimization: Fundamentals and Applications*. Oxford University Press, 1995. ISBN: 978-0-19510-056-3.

[Fujiwara1982] Fujiwara und Toida. „The Complexity of Fault Detection Problems for Combinational Logic Circuits“. In: *IEEE Transactions on Computers* C-31.6 (Juni 1982), S. 555–560. DOI: 10.1109/tc.1982.1676041.

[Ganeshpure2010] K. Ganeshpure und S. Kundu. „On ATPG for Multiple Aggressor Crosstalk Faults“. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 29.5 (Mai 2010), S. 774–787. DOI: 10.1109/tcad.2010.2043589.

[Geden2011] B. Geden. *Understand and Avoid Electromigration (EM) & IR-drop in Custom IP Blocks*. White Paper. Synopsys, Nov. 2011.

[Géron2020] A. Géron. *Praxiseinstieg Machine Learning mit Scikit-Learn, Keras und TensorFlow - Konzepte, Tools und Techniken für intelligente Systeme*. O'Reilly, 2020. ISBN: 978-3-96009-124-0.

[Hasib2019] O. A. Hasib, Y. Savaria und C. Thibeault. „Multi-PVT-Point Analysis and Comparison of Recent Small-Delay Defect Quality Metrics“. In: *Journal of Electronic Testing – Theory and Applications (JETTA)* 35.6 (Dez. 2019), S. 823–838. DOI: 10.1007/s10836-019-05832-w.

[Hellebrand2014] S. Hellebrand, T. Indlekofer, M. Kampmann, M. A. Kochte, C. Liu und H.-J. Wunderlich. „FAST-BIST: Faster-than-At-Speed BIST Targeting Hidden Delay Defects“. In: *IEEE International Test Conference (ITC)*. Okt. 2014, S. 1–8. DOI: 10.1109/test.2014.7035360.

[Hellebrand2015] S. Hellebrand, T. Indlekofer, M. Kampmann, M. Kochte, C. Liu und H.-J. Wunderlich. „Effiziente Auswahl von Testfrequenzen für den Test kleiner Verzögerungsfehler“. In: *ITG/GMM/GI Workshop Testmethoden und Zuverlässigkeit von Schaltungen und Systemen (TuZ)*. März 2015.

[Huffman2003] W. C. Huffman und V. Pless. *Fundamentals of Error-Correcting Codes*. Cambridge University Press, 2003. ISBN: 978-0-521-78280-7.

[IEEE1994] IEEE. *IEEE 1149.1b-1994: IEEE Standard Test Access Port and Boundary-Scan Architecture*. 1994. DOI: 10.1109/ieeestd.1995.122623.

[IEEE2014] IEEE. *IEEE 1687-2014: IEEE Standard for Access and Control of Instrumentation Embedded within a Semiconductor Device*. 2014. DOI: 10.1109/ieeestd.2014.6974961.

[Intel2022] Intel. *Spezifikation Intel® Core™ i9-12900T Prozessor*. 2022. URL: https://www.intel.de/content/www/de/de/products/sku/134601/intel-core-i912900t-processor-30m-cache-up-to-4-90-ghz/specifications.html (zuletzt besucht am 24. 03. 2023).

[ISO2018] ISO. *ISO 26262:2018: ISO Standard Road vehicles - Functional safety*. 2018.

[Kampmann2015] M. Kampmann, M. A. Kochte, E. Schneider, T. Indlekofer, S. Hellebrand und H. Wunderlich. „Optimized Selection of Frequencies for Faster-Than-at-Speed Test“. In: *IEEE Asian Test Symposium (ATS)*. Nov. 2015, S. 109–114. DOI: 10.1109/ats.2015.26.

[Kampmann2016] M. Kampmann und S. Hellebrand. „X Marks the Spot: Scan-Flip-Flop Clustering for Faster-than-at-Speed Test“. In: *IEEE Asian Test Symposium (ATS)*. Nov. 2016, S. 1–6. DOI: 10.1109/ats.2016.20.

[Kampmann2017] M. Kampmann und S. Hellebrand. „Design-for-FAST: Supporting X-tolerant Compaction during Faster-than-at-Speed Test“. In: *IEEE Design and Diagnostics of Electronic Circuits and Systems (DDECS)*. Apr. 2017. DOI: 10.1109/ddecs.2017.7934564.

[Kampmann2018] M. Kampmann und S. Hellebrand. „Design For Small Delay Test - A Simulation Study“. In: *Microelectronics Reliability* 80 (Jan. 2018), S. 124–133. DOI: 10.1016/j.microrel.2017.11.019.

[Kampmann2020] M. Kampmann. „Eingebauter Selbsttest für kleine Verzögerungsfehler“. Dissertation. Universität Paderborn, 20. Feb. 2020.

[Kim2010] Y. M. Kim, Y. Kameda, H. Kim, M. Mizuno und S. Mitra. „Low-Cost Gate-Oxide Early-Life Failure Detection in Robust Systems“. In: *IEEE Symposium on VLSI Circuits (VLSIC)*. Juni 2010, S. 125–126. DOI: 10.1109/vlsic.2010.5560326.

[Kingma2015] D. P. Kingma und J. Ba. „Adam: A Method for Stochastic Optimization“. In: *International Conference on Learning Representations (ICLR)*. Mai 2015, S. 1–15. DOI: 10.48550/arxiv.1412.6980.

[Korte2018] B. Korte und J. Vygen. *Kombinatorische Optimierung : Theorie und Algorithmen*. Springer, 2018. ISBN: 978-3-662-57691-5.

[Krstic1998] A. Krstic und K.-T. Cheng. *Delay Fault Testing for VLSI Circuits*. Springer, 1998. ISBN: 978-0-7923-8295-9. DOI: 10.1007/978-1-4615-5597-1.

[LeCun1998] Y. LeCun, C. Cortes und C. J. C. Burges. *The MNIST Database of handwritten digits*. 1998. URL: http://yann.lecun.com/exdb/mnist/ (zuletzt besucht am 24. 03. 2023).

[Lee2008] J. Lee und E. J. McCluskey. „Failing Frequency Signature Analysis“. In: *IEEE International Test Conference (ITC)*. Okt. 2008, S. 1–8. DOI: 10.1109/test.2008.4700561.

[Lienig2018] J. Lienig und M. Thiele. *Fundamentals of Electromigration-Aware Integrated Circuit Design*. Springer, 2018. ISBN: 978-3-319-73557-3.

[Lin1987] C. J. Lin und S. M. Reddy. „On Delay Fault Testing in Logic Circuits“. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 6.5 (Sep. 1987), S. 694–703. DOI: 10.1109/tcad.1987.1270315.

[Liu2018] C. Liu, E. Schneider, M. Kampmann, S. Hellebrand und H.-J. Wunderlich. „Extending Aging Monitors for Early Life and Wear-Out Failure Prevention“. In: *IEEE Asian Test Symposium (ATS)*. Okt. 2018, S. 92–97. DOI: 10.1109/ats.2018.00028.

[Liu2021] Y. Liu, S. Milewski, G. Mrugalski, N. Mukherjee, J. Rajski, J. Tyszer und B. Włodarczak. „X-Tolerant Compactor maXpress for In-System Test Applications With Observation Scan“. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems (TVLSI)* 29.8 (Aug. 2021), S. 1553–1566. DOI: 10.1109/tvlsi.2021.3092421.

[Livshits2012] P. Livshits und S. Sofer. „Aggravated Electromigration of Copper Interconnection Lines in ULSI Devices Due to Crosstalk Noise“. In: *IEEE Transactions on Device and Materials Reliability (T-DMR)* 12.2 (Juni 2012), S. 341–346. DOI: 10.1109/tdmr.2012.2184288.

[Ma1995] S. C. Ma, P. Franco und E. J. McCluskey. „An Experimental Chip to Evaluate Test Techniques - Experiment Results“. In: *IEEE International Test Conference (ITC)*. Okt. 1995, S. 663–672. DOI: 10.1109/test.1995.529895.

[Malandruccolo2011] V. Malandruccolo, M. Ciappa, H. Rothleitner und W. Fichtner. „Design and Experimental Characterization of a New Built-In Defect-Based Testing Technique to Achieve Zero Defects in the Automotive Environment“. In: *IEEE Transactions on Device and Materials Reliability (T-DMR)* 11.2 (Juni 2011), S. 349–357. DOI: 10.1109/tdmr.2011.2135354.

[McCulloch1943] W. S. McCulloch und W. Pitts. „A Logical Calculus of the Ideas Immanent in Nervous Activity“. In: *The Bulletin of Mathematical Biophysics* 5.4 (Dez. 1943), S. 115–133. DOI: 10.1007/bf02478259.

[McPherson2019] J. W. McPherson. *Reliability Physics and Engineering. Time-To-Failure Modeling*. Springer, 2019. ISBN: 978-3-319-93682-6.

[Mitra2004a] S. Mitra und K. S. Kim. „X-Compact: An Efficient Response-Compaction Technique“. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 23.3 (März 2004), S. 421–432. DOI: 10.1109/tcad.2004.823341.

[Mitra2004b] S. Mitra, S. S. Lumetta und M. Mitzenmacher. „X-tolerant Signature Analysis“. In: *IEEE International Test Conference (ITC)*. Okt. 2004, S. 432–441. DOI: 10.1109/test.2004.1386979.

[Moreno2016] J. Moreno, M. Renovell und V. Champac. „Effectiveness of Low-Voltage Testing to Detect Interconnect Open Defects Under Process Variations“. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems (TVLSI)* 24.1 (Jan. 2016), S. 378–382. DOI: 10.1109/tvlsi.2015.2397934.

[Morosov2001] A. Morosov, K. Chakrabarty, M. Gossel und B. Bhattacharya. „Design of Parameterizable Error-Propagating Space Compactors for Response Observation“. In: *IEEE VLSI Test Symposium (VTS)*. Apr. 2001, S. 48–53. DOI: 10.1109/vts.2001.923417.

[Mrugalski2022a] G. Mrugalski, J. Rajski, J. Tyszer und B. Włodarczak. „DIST: Deterministic In-System Test with X-masking“. In: *IEEE International Test Conference (ITC)*. Sep. 2022, S. 20–27. DOI: 10.1109/itc50671.2022.00008.

[Mrugalski2022b] G. Mrugalski, J. Rajski, J. Tyszer und B. Włodarczak. „X-Masking for In-System Deterministic Test“. In: *IEEE European Test Symposium (ETS)*. Mai 2022, S. 1–6. DOI: 10.1109/ets54262.2022.9810407.

[Nair2010] V. Nair und G. E. Hinton. „Rectified Linear Units Improve Restricted Boltzmann Machines“. In: *ACM International Conference on Machine Learning (ICML)*. Haifa, Israel, Juni 2010, S. 807–814.

[Najafi-Haghi2020] Z. P. Najafi-Haghi, M. Hashemipour-Nazari und H.-J. Wunderlich. „Variation-Aware Defect Characterization at Cell Level“. In: *IEEE European Test Symposium (ETS)*. Mai 2020, S. 1–6. DOI: 10.1109/ets48528.2020.9131600.

[Naruse2003] M. Naruse, I. Pomeranz, S. M. Reddy und S. Kundu. „On-chip Compression of Output Responses with Unknown Values Using LFSR Reseeding“. In: *IEEE International Test Conference (ITC)*. Sep. 2003, S. 1060–1068. DOI: 10.1109/test.2003.1271094.

[Nocker2004] R. Nocker. „Modulo-2-Arithmetik“. In: *Digitale Kommunikationssysteme 1: Grundlagen der Basisband-Übertragungstechnik*. Vieweg+Teubner, 2004, S. 223–225. ISBN: 978-3-322-80253-8.

[Patel2003] J. H. Patel, S. S. Lumetta und S. M. Reddy. „Application of Saluja-Karpovsky Compactors to Test Responses with Many Unknowns“. In: *IEEE VLSI Test Symposium (VTS)*. Mai 2003, S. 107–112. DOI: 10.1109/vtest.2003.1197640.

[Pedregosa2011] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot und E. Duchesnay. „Scikit-learn: Machine Learning in Python“. In: *Journal of Machine Learning Research (JMLR)* 12.85 (Okt. 2011), S. 2825–2830.

[Peng2013] K. Peng, M. Yilmaz, K. Chakrabarty und M. Tehranipoor. „Crosstalk- and Process Variations-Aware High-Quality Tests for Small-Delay Defects“. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems (TVLSI)* 21.6 (Juni 2013), S. 1129–1142. DOI: 10.1109/tvlsi.2012.2205026.

[Pomeranz2002] I. Pomeranz, S. Kundu und S. M. Reddy. „On Output Response Compression in the Presence of Unknown Output Values“. In: *IEEE/ACM Design Automation Conference (DAC)*. Juni 2002, S. 255–258. DOI: 10.1109/dac.2002.1012631.

[Qian2010] X. Qian und A. D. Singh. „Distinguishing Resistive Small Delay Defects from Random Parameter Variations“. In: *IEEE Asian Test Symposium (ATS)*. Dez. 2010, S. 325–330. DOI: 10.1109/ats.2010.62.

[Qian2012] X. Qian, C. Han und A. D. Singh. „Detection of Gate-Oxide Defects With Timing Tests at Reduced Power Supply“. In: *IEEE VLSI Test Symposium (VTS)*. Apr. 2012, S. 120–126. DOI: 10.1109/vts.2012.6231090.

[Rabenalt2009] T. Rabenalt, M. Goessel und A. Leininger. „Masking of X-values by Use of a Hierarchically Configurable Register“. In: *IEEE European Test Symposium (ETS)*. Mai 2009, S. 149–154. DOI: 10.1109/ets.2009.11.

[Rajski2005] J. Rajski, J. Tyszer, C. Wang und S. M. Reddy. „Finite Memory Test Response Compactors for Embedded Test Applications“. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 24.4 (Apr. 2005), S. 622–634. DOI: 10.1109/tcad.2005.844111.

[Rajski2008] J. Rajski, J. Tyszer, G. Mrugalski, W.-T. Cheng, N. Mukherjee und M. Kassab. „X-Press: Two-Stage X-Tolerant Compactor With Programmable Selector“. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 27.1 (Jan. 2008), S. 147–159. DOI: 10.1109/tcad.2007.907276.

[Reddy1988] S. M. Reddy, K. K. Saluja und M. G. Karpovsky. „A Data Compression Technique for Built-In Self-Test“. In: *IEEE Transactions on Computers (TC)* 37.9 (Sep. 1988), S. 1151–1156. DOI: 10.1109/12.2271.

[Reimer2019] J. D. Reimer. „SAT-basierte Modellierung und Analyse von Verzögerungsfehlern mit variabler Größe“. Masterarbeit. Universität Paderborn, 2019.

[Roy2011] S. Roy und A. Dounavis. „Efficient Delay and Crosstalk Modeling of RLC Interconnects Using Delay Algebraic Equations“. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems (TVLSI)* 19.2 (Feb. 2011), S. 342–346. DOI: 10.1109/tvlsi.2009.2032288.

[Rumelhart1987] D. E. Rumelhart, G. E. Hinton und R. J. Williams. „Learning Internal Representations by Error Propagation“. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. MIT Press, 1987. ISBN: 978-0-26229-140-8.

[Sadeghi-Kohan2021] S. Sadeghi-Kohan, S. Hellebrand und H.-J. Wunderlich. „Stress-Aware Periodic Test of Interconnects“. In: *Journal of Electronic Testing – Theory and Applications (JETTA)* 37.5 (Dez. 2021), S. 715–728. DOI: 10.1007/s10836-021-05979-5.

[Sammut2017] C. Sammut und G. I. Webb. *Encyclopedia of Machine Learning and Data Mining*. Second Edition. Springer, 2017. ISBN: 978-1-48997-685-7.

[Sauer2012] M. Sauer, A. Czutro, I. Polian und B. Becker. „Small-Delay-Fault ATPG with Waveform Accuracy“. In: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. Nov. 2012, S. 30–36. DOI: 10.1145/2429384.2429391.

[Sauer2013] M. Sauer, Y. M. Kim, J. Seomun, H. Kim, K. Do, J. Y. Choi, K. S. Kim, S. Mitra und B. Becker. „Early-Life-Failure Detection using SAT-based ATPG“. In: *IEEE International Test Conference (ITC)*. Sep. 2013, S. 1–10. DOI: 10.1109/test.2013.6651925.

[Savir1993] J. Savir und S. Patil. „Scan-Based Transition Test“. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 12.8 (Aug. 1993), S. 1232–1241. DOI: 10.1109/43.238615.

[Savir1994] J. Savir und S. Patil. „Broad-Side Delay Test“. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 13.8 (Aug. 1994), S. 1057–1064. DOI: 10.1109/43.298042.

[Schneider2019] E. Schneider und H.-J. Wunderlich. „SWIFT: Switch Level Fault Simulation on GPUs“. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 38.1 (Jan. 2019), S. 122–135. DOI: 10.1109/tcad.2018.2802871.

[Schneider2020] E. Schneider und H.-J. Wunderlich. „Switch Level Time Simulation of CMOS Circuits with Adaptive Voltage and Frequency Scaling“. In: *IEEE VLSI Test Symposium (VTS)*. Apr. 2020, S. 1–6. DOI: 10.1109/vts48691.2020.9107642.

[Sharma2005] M. Sharma und W.-T. Cheng. „X-filter: Filtering Unknowns from Compacted Test Responses“. In: *IEEE International Test Conference (ITC)*. Nov. 2005, S. 1090–1098. DOI: 10.1109/test.2005.1584076.

[SII2023] SII. *15nm Open-Cell Library and 45nm FreePDK*. 2023. URL: https://si2.org/open-cell-library/ (zuletzt besucht am 24. 03. 2023).

[Smith1985] G. L. Smith. „Model for Delay Faults Based upon Paths“. In: *IEEE International Test Conference (ITC)*. Nov. 1985, S. 342–349.

[Strole1991] A. P. Strole und H.-J. Wunderlich. „TESTCHIP: A Chip for Weighted Random Pattern Generation, Evaluation, and Test Control“. In: *IEEE Journal of Solid-State Circuits (JSSC)* 26.7 (Juli 1991), S. 1056–1063. DOI: 10.1109/4.92026.

[Tang2006] Y. Tang, H.-J. Wunderlich, P. Engelke, I. Polian, B. Becker, J. Schloffel, F. Hapke und M. Wittke. „X-Masking During Logic BIST and Its Impact on Defect Coverage“. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems (TVLSI)* 14.2 (Feb. 2006), S. 193–202. DOI: 10.1109/tvlsi.2005.863742.

[Tehranipoor2011] M. Tehranipoor, K. Peng und K. Chakrabarty. *Test and Diagnosis for Small-Delay Defects*. Springer, 2011. ISBN: 978-1-44198-296-4.

[Touba2007] N. A. Touba. „X-Canceling MISR – An X-Tolerant Methodology for Compacting Output Responses with Unknowns Using a MISR“. In: *IEEE International Test Conference (ITC)*. Okt. 2007, S. 1–10. DOI: 10.1109/test.2007.4437576.

[TSMC2023] TSMC. *Logic Technology Overview*. 2023. URL: https://www.tsmc.com/english/dedicatedFoundry/technology/logic (zuletzt besucht am 24. 03. 2023).

[Volkerink2005] E. H. Volkerink und S. Mitra. „Response Compaction with any Number of Unknowns using a new LFSR Architecture“. In: *IEEE/ACM Design Automation Conference (DAC)*. Juni 2005, 117-122. DOI: 10.1145/1065579.1065614.

[Wang2006] L.-T. Wang, C.-W. Wu und X. Wen. *VLSI Test Principles and Architectures: Design for Testability*. Morgan Kaufmann, 2006. ISBN: 978-0-12370-597-6.

[Wang2008a] L.-T. Wang, C. E. Stroud und N. A. Touba. *System-On-Chip Test Architectures: Nanometer Design for Testability*. Morgan Kaufmann, 2008. ISBN: 978-0-12373-973-5.

[Wang2008b] S. Wang, K. J. Balakrishnan und W. Wei. „X-Block: An Efficient LFSR Reseeding-Based Method to Block Unknowns for Temporal Compactors“. In: *IEEE Transactions on Computers (TC)* 57.7 (Juli 2008), S. 978–989. DOI: 10.1109/tc.2007.70833.

[Wohl2004] P. Wohl, J. A. Waicukauski und S. Patel. „Scalable Selector Architecture for X-Tolerant Deterministic BIST“. In: *IEEE/ACM Design Automation Conference (DAC)*. Juni 2004, S. 934–939. DOI: 10.1145/996566.996814.

[Wohl2007] P. Wohl, J. A. Waicukauski und S. Ramnath. „Fully X-tolerant Combinational Scan Compression“. In: *IEEE International Test Conference (ITC)*. Okt. 2007, S. 1–10. DOI: 10.1109/test.2007.4437575.

[Wohl2008] P. Wohl, J. A. Waicukauski und F. Neuveux. „Increasing Scan Compression by Using X-chains“. In: *IEEE International Test Conference (ITC)*. Okt. 2008, S. 1–10. DOI: 10.1109/test.2008.4700646.

[Yan2003] H. Yan und A. D. Singh. „Experiments in Detecting Delay Faults using Multiple Higher Frequency Clocks and Results from Neighboring Die“. In: *IEEE International Test Conference (ITC)*. Sep. 2003, S. 105–111. DOI: 10.1109/test.2003.1270830.

[Zivkovic2011] V. Zivkovic, J. Schipper, R. Kluit, M. Garcia-Sciveres, A. Mekkaoui, M. Barbero, R. Beccherie, D. Gnani, T. Hemperek, M. Karagounis, M. Menouni, D. Fougeron, F. Gensolen, V. Gromov, A. Kruth, G. Darbo, J. Fleury, J. C. Clemens, S. Dube und D. Arutinov. „The Design for Test Architecture in Digital Section of the ATLAS FE-I4 Chip“. In: *Journal of Instrumentation (JINST)* 6.1 (Jan. 2011), S. 1–5. DOI: 10.1088/1748-0221/6/01/c01090.

Eigene Konferenz- und Zeitschriftenbeiträge

[Holst2020] S. Holst, M. Kampmann, A. Sprenger, J. D. Reimer, S. Hellebrand, H.-J. Wunderlich und X. Wen. „Logic Fault Diagnosis of Hidden Delay Defects“. In: *IEEE International Test Conference (ITC)*. 3.–5. Nov. 2020, S. 1–10. DOI: 10.1109/itc44778.2020.9325234.

[Maaz2019a] M. U. Maaz, A. Sprenger und S. Hellebrand. „A Hybrid Space Compactor for Adaptive X-Handling“. In: *IEEE International Test Conference (ITC)*. 11.–17. Nov. 2019, S. 1–8. DOI: 10.1109/itc44170.2019.9000159.

[Sprenger2018b] A. Sprenger und S. Hellebrand. „Tuning Stochastic Space Compaction to Faster-than-at-Speed Test“. In: *IEEE Design and Diagnostics of Electronic Circuits and Systems (DDECS)*. 25.–27. Apr. 2018, S. 73–78. DOI: 10.1109/ddecs.2018.00020.
Best Paper Award.

[Sprenger2019] A. Sprenger und S. Hellebrand. „Divide and Compact - Stochastic Space Compaction for Faster-than-At-Speed Test“. In: *Journal of Circuits, Systems and Computers* 28.supp01 (Apr. 2019), S. 1–23. DOI: 10.1142/s0218126619400012.

[Sprenger2020] A. Sprenger, S. Sadeghi-Kohan, J. D. Reimer und S. Hellebrand. „Variation-Aware Test for Logic Interconnects using Neural Networks - A Case Study“. In: *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*. 19.–21. Okt. 2020, S. 1–6. DOI: 10.1109/dft50435.2020.9250922.
Best Student Paper Award.

Eigene Workshopbeiträge

[Maaz2019b] M. U. Maaz, A. Sprenger und S. Hellebrand. „A Hybrid Space Compactor for Varying X-Rates“. In: *ITG/GMM/GI Workshop Testmethoden und Zuverlässigkeit von Schaltungen und Systemen (TuZ)*. 24.–26. Feb. 2019.

[Sprenger2018a] A. Sprenger und S. Hellebrand. „Stochastische Kompaktierung für den Hochgeschwindigkeitstest“. In: *ITG/GMM/GI Workshop Testmethoden und Zuverlässigkeit von Schaltungen und Systemen (TuZ)*. 4.–6. März 2018.

Betreute wissenschaftliche Arbeiten

[Alsati2021] A. Alsati. „Test Pattern Scheduling for Faster-than-At-Speed Test“. Masterarbeit. Universität Paderborn, 2021.

[Boschmann2018] M. Boschmann. „Implementierung und Analyse eines rekonfigurierbaren X-toleranten Signaturregisters“. Bachelorarbeit. Universität Paderborn, 2018.

[Heuvel2021] J. van den Heuvel. „Anwendung künstlicher neuronaler Netze zur Unterscheidung kleiner Verzögerungsfehler und Prozessvariationen“. Bachelorarbeit. Universität Paderborn, 2021.

[Maaz2017] M. U. Maaz. „Comparison and Development of X-tolerant Compaction Schemes for Faster-than-At-Speed Test“. Masterarbeit. Universität Paderborn, 2017.

[Ngawa2019] H. Ngawa. „Konzeption und Implementierung einer Hardware--Software-Testumgebung für Funkschnittstellen einer modularen Entwicklungsplattform in der Automobilindustrie“. Masterarbeit. Universität Paderborn, 2019.

[Piel2021] A. Piel. „Parametrisierung von Verbindungsdefekten innerhalb integrierter Schaltungen mit Hilfe von Layoutsynthese“. Bachelorarbeit. Universität Paderborn, 2021.

[Reimer2018] J. D. Reimer und V. Tran. „Fehleremulation für den Software-basierten Selbsttest - Eine experimentelle Evaluation für den LEON3-Prozessor“. Projektarbeit. Universität Paderborn, 4. Juli 2018.

[Rheinert2019] E. J. Rheinert. „Fault Tolerance Analysis of Approximate Artificial Neural Networks“. Masterarbeit. Universität Paderborn, 2019.

[Veerappa2017] S. K. Veerappa. „Developing and Analysing a Reconfigurable X-Canceling MISR for Faster-than-At-Speed Test“. Masterarbeit. Universität Paderborn, 2017.

[Zhang2019] Y. Zhang. „Convolutional Compaction for Faster-than-At-Speed-Test (FAST)“. Masterarbeit. Universität Paderborn, 2019.