UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

FAKULTÄT FÜR
ELEKTROTECHNIK,
INFORMATIK UND
MATHEMATIK

Fehlercharakterisierung zuverlässiger Schaltungen im Selbsttest

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

zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften (Dr.-Ing.)

genehmigte Dissertation

von

Dipl.-Wirt.-Ing. Thomas Indlekofer

Erster Gutachter: Prof. Dr. Sybille Hellebrand
Zweiter Gutachter: Prof. Dr. Ilia Polian

Tag der mündlichen Prüfung: 03.03.2016

Paderborn 2016

Diss. EIM-E/325

Danksagung

Es ist mir eine große Ehre, mich bei den Personen zu bedanken, die mich in dieser anstrengenden, herausfordernden, aber auch ungemein lohnenden Phase begleitet und unterstützt haben.

Als erstes möchte ich im Besonderen meiner Doktormutter Prof. Dr. Sybille Hellebrand danken für ihre vielseitige hilfreiche Unterstützung bei der Bearbeitung dieses außerordentlich interessanten Forschungsthemas, den vielen anregenden Diskussionen, aber auch für die unheimlich schöne und schützende Atmosphäre, in der diese Arbeit entstanden ist. Jede Phase dieser Arbeit wurde von ihr intensiv, professionell und warmherzig begleitet.

Ebenfalls bedanken möchte ich mich bei meinem Zweitgutachter, Prof. Dr. Ilia Polian, für seine Bereitschaft meine Arbeit zu begutachten sowie seine hilfreichen Hinweise und Anregungen.

Besonderer Dank gilt Prof. Dr. Hans-Joachim Wunderlich vom Institut für Technische Informatik der Universität Stuttgart und seiner Arbeitsgruppe. Durch viele gemeinsame Publikationen und die damit verbundenen richtungsweisenden Anmerkungen, Denkanstöße und Ideen wurde ein großer Teil dieser Arbeit geprägt. Besonders danken möchte ich auch für die Bereitstellung des Frameworks ADAMA, das zu einem großen Teil für die Simulationen in dieser Arbeit benutzt wurde.

Insgesamt habe ich die Zeit als wissenschaftlicher Mitarbeiter im Fachgebiet Datentechnik am Institut für Elektrotechnik und Informationstechnik der Universität Paderborn sehr genossen und möchte mich deswegen bei meinen Kollegen bedanken. Matthias Kampmann danke ich für die Implementierung der FAST-Gruppeneinteilung, die während seiner Masterarbeit am Lehrstuhl entstanden ist, aber auch für die zielführenden Diskussionen und hilfreichen Ideen bei dem ein- oder anderen Kaffee. Michael Schnittger danke ich für die Hilfe bei der Implementierung der extremen Kompaktierung. Besonders danken möchte ich unserem Laboringenieur Rüdiger Ibers, der neben seinem fachlichen Know-How auch stets ein offenes Ohr für Freude und Leid hatte, aber vor allem mit vielen Vater-Tipps hilfreich sein konnte. Ich bedanke mich auch bei meinem ehemaligen Bürokollegen Viktor Fröse für viele sehr interessante Diskussionen und wertvolle Ideen. Besonders danken möchte ich ihm aber auch für die vielen Lacher, für die er oft genug sorgte. Danken möchte ich auch Marc Hunger, der leider viel zu früh von uns gegangen ist. Vielen Dank auch Ursula Stiebritz und Christine Fricke für ihre Arbeit im Sekretariat, die neben dem Bürokratie- und Organisations-Chaos stets warme und aufbauende Worte gefunden haben.

Ganz besonders bedanken möchte ich meiner liebenden Familie. Meiner geliebten Frau Katharina möchte ich danken für all die Zeit und das Verständnis, die sie während dieser langen anstrengenden Zeit aufgebracht hat. Sie hat mir stets den Rücken freigehalten

und mir mit Lucas das schönste Geschenk gemacht. Meinen Eltern danke ich von ganzem Herzen für all die Unterstützung während meiner Kindheit, Schulzeit, meines Studiums und meiner Zeit als Doktorand. Ohne ihre tatkräftige und liebevolle Unterstützung wäre aus mir nicht der Mensch geworden, der ich heute bin. Danken möchte ich auch meiner Schwester für ihren schwesterlichen Rat und ihre Unterstützung während der letzten Jahre. Allen meinen lieben Freunden danke ich für die Ausdauer, Ruhe und Geduld, womit sie mir stets zur Seite standen und mich immer wieder aufgemuntert haben.

Paderborn, Dezember 2015 *Thomas Indlekofer*

Abstract

As a result of the fact, that today's integrated circuits have smaller features sizes, higher frequencies and are more energy efficient, weak spots can occur in the system. These weak spots can be undetected by the production test, but during system operation they can lead to hard failures, because of increasing susceptibility to intrinsic and external disturbances or aging effects. This early system breakdown is called „early-life failure“ and cannot be detected by a standard test without any adjustments. Indicators of early-life failures could be intermittent faults and also small delay defects (SDDs).

An intermittent fault has a non-permanent behaviour and is at the beginning uncritical, but during the product life cycle and because of aging effects or external disturbances it can lead to a system breakdown. An example of an intermittent fault is a high-frequency power-droop (HFPD), which results from power starvation when multiple cells connected to the same power grid segment suddenly increase their current demand. Dealing with intermittent faults is particularly challenging, because they impact quality but may lead to similar observations during test as uncritical transient faults, which occur for example because of parameter variations or radiation and could be tolerated during system operation. This leads in a tradeoff between product quality and additional yield loss.

Another faulty behavior, which can lead to an early-life failure, is a hidden small delay defect. A hidden SDD is only propagated along short paths where the slack is larger than the defect size. Even if they do not violate the nominal timing at the production test, the delay can increase during the product lifetime, because of external disturbances, aging or accumulation with other effects, and leads to a hard failure.

In this thesis a built-in self-test is presented, which characterizes faulty behavior to detect weak spots and avoid early-life failures, especially caused by SDDs or HFPDs, with low hardware and time overhead by using a standard test set. In a first step, the test procedure can distinguish between permanent and non-permanent faults. After that, a diagnosis process and Bayesian reasoning implement the classification of the non-permanent faults. With this procedure the product quality can be increased without additional yield loss. Furthermore a Faster-than-at-Speed-Test (FAST) will be introduced, which allows detecting SDDs in a built-in self-test environment without any changes in the ATPG flow.

Kurzfassung

Hochintegrierte Schaltungen können immer kleiner, höher getaktet und energieeffizienter hergestellt werden, allerdings können bedingt durch diese technologischen Trends auch vermehrt Schwachstellen im System entstehen. Diese Schwachstellen führen oft während des Produktionstests nicht zu einem Fehlverhalten der Schaltung, während des Betriebs allerdings droht durch die steigende Anfälligkeit gegenüber intrinsischen und äußeren Störeinflüssen sowie Alterungseffekten ein vorzeitiger Ausfall der Schaltung. Solche Frühausfälle werden „Early-Life Fehler“ genannt und können mit einem Standard-Test ohne weitere Anpassungen nicht erkannt werden. Indikatoren für einen Frühausfall können intermittierende Fehler, aber auch kleine Verzögerungsfehler sein. Ein intermittierender Fehler hat ein nicht-permanentes Fehlverhalten und ist anfänglich zwar noch unkritisch, kann aber im Laufe des Produktlebenszyklus' durch Alterung oder weitere Störeinflüsse zu einem Totalausfall der Schaltung führen. Beispielsweise kann ein hochfrequenter Spannungsabfall (engl. *High-Frequency Power-Droop*, *HFPD*), der aufgrund eines nicht ausreichend dimensionierten Versorgungsnetzes auftritt, ein solches intermittierendes Fehlverhalten haben. Problematisch hierbei ist, dass sich ein intermittierender Fehler nur schwer von einem unkritischen transienten Fehler, der nur kurzzeitig aufgrund von äußeren Störeinflüssen oder Parameterschwankungen auftritt und durch die eingebaute Redundanz während des Betriebs toleriert werden kann, unterscheiden lässt. Somit entsteht hierbei ein Konflikt zwischen Produktqualität und zusätzlichen Ausbeuteverlusten. Ein weiteres Fehlverhalten, das zu einem Frühausfall führen kann, ist ein versteckter kleiner Verzögerungsfehler (engl. *Small Delay Defect*, *SDD*). Hierbei handelt es sich um einen Fehler mit einer Verzögerungszeit, die bei Betriebsfrequenz keine Auswirkung auf einen Schaltungsausgang hat. Allerdings kann sich diese Verzögerung im Laufe des Produktlebenszyklus' durch Alterung, äußere Störeinflüsse oder Kumulation mit anderen Effekten noch so erhöhen, dass sich ein dauerhaftes Fehlverhalten an den Schaltungsausgängen einstellt.

In dieser Arbeit wird ein Selbsttest vorgestellt, der eine Fehlercharakterisierung zur Erkennung von Systemschwachstellen und Vermeidung von Frühausfällen, speziell solche, die sich als SDD oder HFPD auswirken, mit geringem Hardware- und Zeitaufwand mittels eines Standard-Tests ermöglicht. Hierzu wird im Selbsttest zunächst zwischen permanenten und nicht-permanenten Fehlern unterschieden und eine Klassifikation der nicht-permanenten Fehler mit Hilfe eines voran geschalteten Diagnoseverfahrens und Bayesschen Berechnungen durchgeführt. Hierdurch lässt sich die Produktqualität ohne zusätzliche Ausbeuteverluste erhöhen. Zusätzlich wird ein Test mit erhöhter Betriebsfrequenz vorgestellt, der im Selbsttest kleine Verzögerungsfehler erkennt.

Inhaltsverzeichnis

1 Einleitung

Gordon E. Moore veröffentlichte 1965 in der Zeitschrift *Electronics Magazine* eine Arbeit, die später als erste Version des „Mooreschen Gesetzes“ bekannt wurde: „Cramming more components onto integrated circuits“ [Moo65]. In diesem Artikel formulierte er die Hypothese, dass sich die Anzahl der Komponenten auf einem Chip jedes Jahr verdoppeln würde. Später korrigierte er diese Annahme auf 18 Monate. Dass er Recht behalten würde, zeigt die Entwicklung der Transistorenanzahl für integrierte Schaltungen der letzten 40 Jahre:

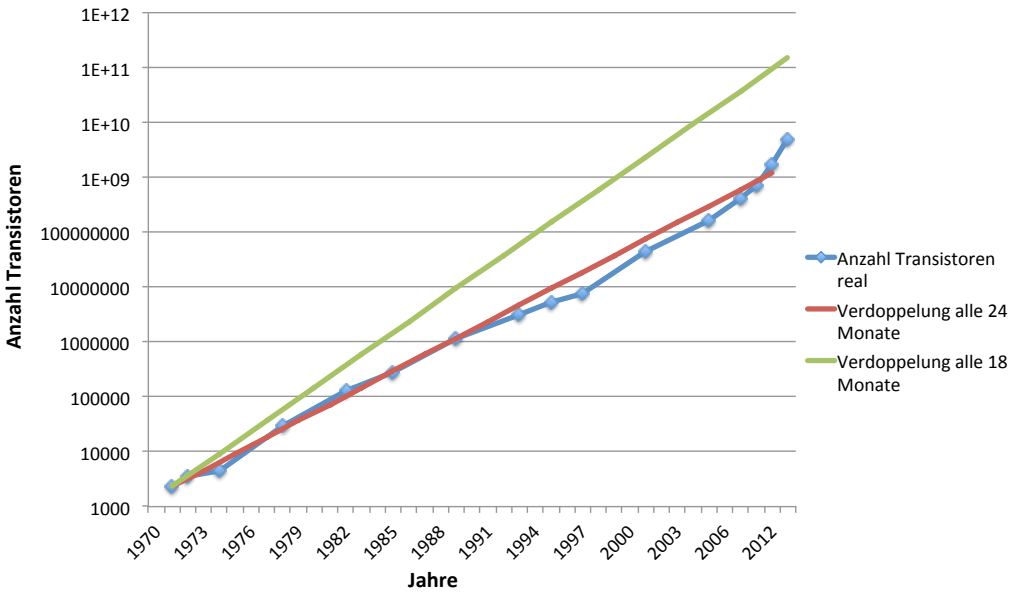


Abbildung 1.1: Entwicklung der Transistorenanzahl im Vergleich zum Mooreschen Gesetz[1]

Wie man an Abbildung 1.1 erkennt, hat sich die Zahl der Transistoren seit 1970 ca. alle 20 Monate verdoppelt. Ein Grund hierfür sind die stetig kleiner werdenden Strukturgrößen, mit denen Transistoren gefertigt werden können: Während in den 1990er-Jahren der erste Pentium Prozessor der Firma Intel noch mit 3 Millionen Transistoren und einer minimalen

[1] Als Basis der Kurve „Anzahl Transistoren real“ dient eine Auswahl von Prozessoren der Firma Intel. Die Tabelle hierzu befindet sich im Anhang A.

Strukturgröße von 0,8 μm hergestellt wurde, hatte der Pentium 4 von 2000 schon über 41 Millionen Transistoren mit einer minimalen Strukturgröße von 180 nm. Die Folgen dieses Trends sind vielfältig: Zum einen können sehr komplexe Funktionen auf immer kleineren Flächen berechnet werden. Zum anderen lässt sich durch die kleineren Strukturgrößen sowohl die Versorgungsspannung reduzieren, was zu einem geringeren Stromverbrauch führt, als auch die Taktfrequenz erhöhen, um die Berechnungen schneller und effizienter durchzuführen. So geht die *ITRS* (*International Technology Roadmap for Secmiconductors*) [ITR] davon aus, dass sich in den nächsten Jahren die Versorgungsspannung auf unter 0,6 V verringert und die Frequenz auf über 6 GHz steigt. Schon 2022 könnte sich außerdem die minimale Strukturgröße auf unter 10 nm reduzieren und die Anzahl an Transistoren auf über 49 Milliarden Transistoren pro Chip erhöhen, wie man Abbildung 1.2 entnehmen kann.

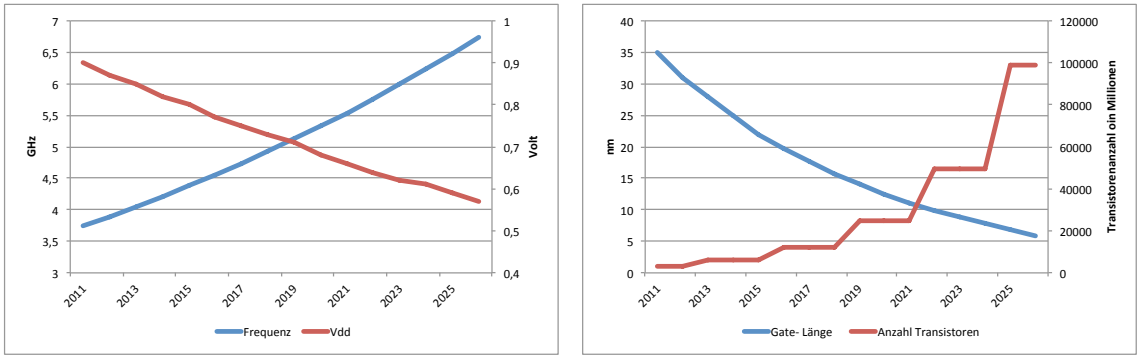


Abbildung 1.2: Entwicklung hochintegrierter Schaltungen [ITR]

Allerdings führen diese Trends auch zu neuen Herausforderungen bei der Fertigung: So benötigen immer kleinere und schnellere Schaltungen eine deutlich höhere Präzision bei der Produktion, und schon kleine Abweichungen während des Produktionsprozesses können zu fehlerhaften Schaltungen oder schlechterer Leistung führen. So haben Variationen der Störstellendichte, Oxiddicke oder Diffusionstiefe einen direkten Einfluss auf die Schaltzeit und das Schaltverhalten eines Transistors, aber auch Ungenauigkeiten beim Auftragen der Verbindungsleitungen können zu Variationen im zeitlichen Verhalten führen. Andererseits können bei Schaltungen mit höherer Integrationsdichte und kleineren Strukturgrößen zusätzliche Effekte wie Leitungsübersprechen, ein durch die Spannungsversorgung bedingtes Rauschen oder resistive Kurzschlüsse und Unterbrechungen auftreten, die in früheren Chip-Generationen nicht oder nur sehr vereinzelt vorkamen [TPC12]. Außerdem spielen äußere Störeinflüsse wie Strahlung oder Temperatur eine sehr große Rolle, die im schlimmsten Fall zu einem kompletten Ausfall des Schaltkreises führen können [KK07].

Damit möglichst frühzeitig ein Fehlverhalten der Schaltung festgestellt werden kann, spielen der Test und die Diagnose eine immer wichtigere Rolle. Hierzu wird in heutigen Schaltungen mehr und mehr Testinfrastruktur direkt auf dem Chip implementiert, um zum einen die Kosten eines aufwendigen externen Testgeräts zu vermeiden und zum anderen die Möglichkeit zu haben, die Schaltung auch während des Betriebs zu testen [AKS93]. Ein solcher Selbsttest kann dabei sowohl die reine Funktionalität der Schaltung

überprüfen als auch auf Strukturebene ansetzen, um die einzelnen Bauteile bezüglich eines Fehlermodells, das mögliche Defekte abstrahiert, zu testen. Ein Defekt, der zu einem permanenten Fehlverhalten führt, kann hierdurch effizienter und frühzeitig identifiziert werden.

Um die Anfälligkeit der Schaltungen während des Betriebs zu minimieren, werden robuste und selbst-adaptive Schaltungen gefertigt. Mit eingebauter Redundanz und verschiedenen fehlerkorrigierenden Techniken wird die Schaltung gegen Fehler geschützt, die während des Betriebs auftreten können [KK07]. Dass diese Techniken nicht mehr ausschließlich in sicherheitskritischen Anwendungen wie Luft- und Raumfahrt oder im militärischen Umfeld nötig sind, zeigt das von ARM und Intel eingesetzte Razor-Register [D$^+$06] [E$^+$04], bei dem die Versorgungsspannung möglichst gering gehalten wird und die hierdurch injizierten Fehler während des Betriebs korrigiert werden.

Besonders problematisch sind allerdings Schwachstellen im System, die unter normalen Umgebungsbedingungen während des Produktionstests nicht zu einem Fehlverhalten der Schaltung geführt haben, während des Betriebs allerdings durch die steigende Anfälligkeit gegenüber intrinsischen und äußeren Störeinflüssen sowie Alterungseffekten zu einem vorzeitigen Ausfall der Schaltung führen. Bereits in [HM93] wurde festgestellt, dass Frühausfälle (oder „Early-Life Fehler“, *engl. early-life failure, ELF*) oft auf produktionsbedingte Schwachstellen im System zurückzuführen sind. Im Folgenden wird auf diese Problematik genauer eingegangen.

1.1 Early-Life Fehler: Neue Herausforderungen für den Test

Frühausfälle von Schaltungen sind für Chiphersteller ein großes Problem, denn sie lassen sich durch den Produktionstest nicht ohne Weiteres vorhersagen und können so zu massiven wirtschaftlichen Schäden führen [HM93]. So führte ein einzelner Transistor im SATA-Taktgenerator der Intel Sandy Bridge im Jahr 2011 zu einem hohen finanziellen Schaden für den Hersteller. Dieser Transistor war bei der Auslieferung funktionstüchtig und hatte somit auch alle Tests bestanden. Da die Versorgungsspannung zu hoch dimensioniert war, veränderten sich im Laufe des Betriebs die Leckströme, so dass es zu unkontrollierten Schaltvorgängen kam und dadurch auch benachbarte Transistoren nicht mehr fehlerfrei funktionierten [intb] [Shi].

Wie bereits in [HM93] beschrieben, können während der Produktion von integrierten Schaltungen Schwachstellen entstehen, die während des Produktionstests nicht zu einem Fehlverhalten führen, aber zusammen mit anderen Effekten wie Alterung o.ä. im weiteren Verlauf des Produktlebenszyklus’ zu einem Ausfall führen und die Zuverlässigkeit wie auch die Produktqualität stark beeinflussen können.

Eine solche Schwachstelle kann sich beispielsweise als kleiner Verzögerungsfehler (*engl. Small Delay Defect, SDD*) auswirken, der die Schaltzeit eines Transistors erhöht, aber zu keinem Fehlverhalten an den Schaltungsausgängen führt. In einem Standard-Test würde eine derartige Schaltung den Test zwar erfolgreich durchlaufen, jedoch könnte sich diese Schwachstelle im Laufe des Produktlebenszyklus’ noch verschlechtern oder durch Kumulierung mit anderen Effekten zu einem Fehlverhalten führen [TPC12]. Andere Schwach-

stellen, wie sie z.B. in [PCKB07] beschrieben sind und als hochfrequenter Spannungsabfall (engl. *high-frequency power-droop, HFPD*) bezeichnet werden, treten durch die hohe Integrationsdichte als auch die sinkende Versorgungsspannung auf und können sich als intermittierende Fehler auswirken: So können gleichzeitige Schaltvorgänge von vielen Bauelementen, die im selben Versorgungsnetz liegen, zu einem fehlerhaften Schaltungsverhalten eines einzelnen Transistors, oder mehreren Transistoren, führen.

Für Anwendungen mit hohen Qualitätsanforderungen werden teilweise sehr aufwendige und teure Testverfahren eingesetzt, um Early-Life Fehler zu erkennen und Frühausfälle zu vermeiden: So wird während des Produktionstests beim Einbrennen (*engl. Burn-in test*) der Chip unter extrem hohen und extrem niedrigen Temperaturen getestet, um eine hohe Beanspruchung zu erzeugen und die Bauteile auf mögliche Schwachstellen zu überprüfen [Wun91]. Eine andere Möglichkeit ist die Analyse analoger Schaltungsparameter, beispielsweise das Messen des Ruhestroms oder das Verhalten der Versorgungsspannung, die Auskünfte darüber geben kann, ob das Schaltverhalten der Transistoren dem erwarteten entspricht [HM93]. Nachteilig bei diesen Testverfahren ist jedoch, dass neben den hohen Kosten und der langen Dauer des Tests auch die Schwächung der Bauteile in Kauf genommen wird. Ein weiterer Nachteil ist auch die Tatsache, dass nicht alle Schwächungen, wie beispielsweise solche, die sich als SDD auswirken, erkannt werden können. Darüber hinaus können Ausbeuteverluste entstehen, wenn die Schaltung nur aufgrund eines kurzzeitigen transienten Fehlers ein Fehlverhalten zeigt, das während des Betriebs durch die eingebaute Redundanz toleriert werden kann.

1.2 Ziel der Arbeit

Ziel dieser Arbeit ist ein integrierter Selbsttest mit Diagnose, der effizient eine Fehlercharakterisierung zur Erkennung von Systemschwachstellen und Vermeidung von Frühausfällen mit einem Standard-Test ermöglicht. Das Verfahren soll hierbei speziell HFPDs und SDDs erkennen und von unkritischen transienten Fehlern unterscheiden, um unnötige Ausbeuteverluste zu vermeiden. Die Anpassungen an die Test- und Diagnoseinfrastruktur sollen bezüglich Hardware- und Zeitaufwand möglichst minimal sein. Der in dieser Arbeit entwickelte Charakterisierungsablauf ist Abbildung 1.3 zu entnehmen. Während Kapitel 2 zunächst einen Überblick über die für diese Arbeit wichtigsten Grundlagen gibt, werden im weiteren Verlauf dieser Arbeit die einzelnen Schritte der Charakterisierung vorgestellt:

- Grundlage ist ein aus der Literatur bekannter **Selbsttest mit Rücksetzpunkten**, der zwischen permanenten und nicht-permanenten Fehlern unterscheidet und dabei unnötige Ausbeuteverluste vermeidet. Darauf aufbauend wird in Kapitel 3 ein neues extremes Kompaktierungsverfahren ausgearbeitet, um den Speicheraufwand der vielen für den weiteren Charakterisierungsablauf wichtigen Informationen, die aus den Testergebnisse gezogen werden können, zu minimieren.

- Im Fehlerfall wird dann zur Analyse ein bereits bekanntes **direktes Diagnoseverfahren** in Kapitel 4 so erweitert, dass nicht nur extrem kompaktierte Signaturen verarbeitet werden können und bei permanenten Fehlern der Fehlerort bestimmt wird, sondern darüber hinaus bei nicht-permanenten Fehlern auch noch Rückschlüsse über eine weitere Klassifikation des Fehlverhaltens in transient und intermittierend zulässt.

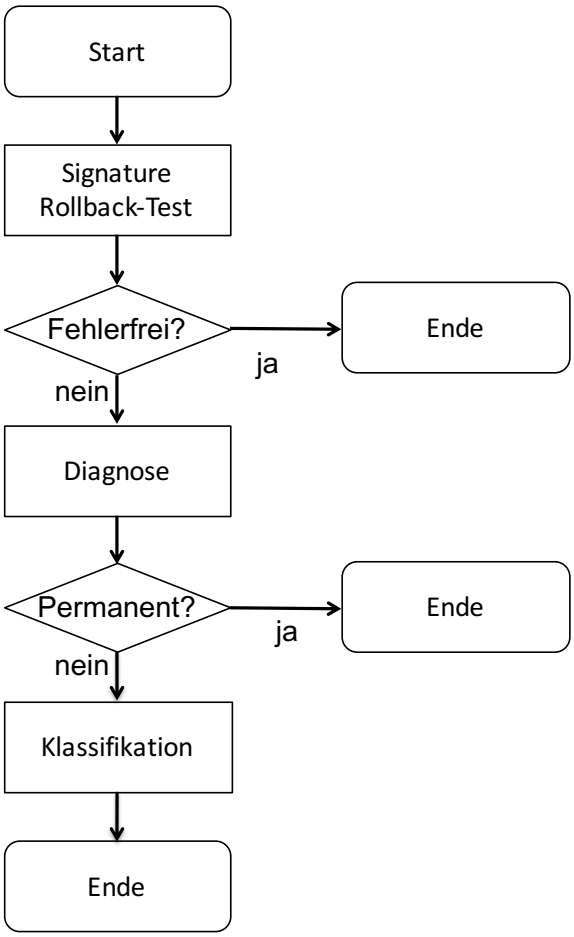


Abbildung 1.3: Charakterisierungsablauf

- Um bei den nicht-permanenten Fehlern zwischen kritischen intermittierenden, wie zum Beispiel HFPD-Fehlern, und nicht-kritischen transienten Fehlern zu unterscheiden, wird in Kapitel 5 eine in dieser Arbeit entwickelte **Klassifikation** beschrieben, die auf Bayesschen Netzen beruht. Durch die Berechnung der bedingten Wahrscheinlichkeiten lässt sich dabei die Fehlerart bestimmen.

- Zusätzlich wird in Kapitel 6 gezeigt, wie sich kleine Verzögerungsfehler im Selbsttest mit Hilfe eines **Tests mit erhöhter Betriebsfrequenz** (engl. *Faster-than-at-Speed-Test, FAST)* erkennen lassen. Hierzu wird eine Testmustermenge einer kleinen Anzahl von FAST-Gruppen zugeteilt und ein Kompaktierer vorgestellt, der trotz der auftretenden unbekannten Werte die Testantworten effizient kompaktieren kann. Durch Abspeichern von Zwischensignaturen können in der Analyse die relevanten Bits extrahiert und mit den fehlerfreien verglichen werden.

In den Experimenten werden die ausgearbeiteten Verfahren validiert: So werden für diverse Benchmarkschaltungen die Test- und Diagnosealgorithmen implementiert und simuliert. Die Ergebnisse zeigen, dass sich für robuste Schaltungen neben der Maximierung der Produktqualität auch eine Minimierung zusätzlicher Ausbeuteverluste erreichen lässt.

2 Grundlagen

In diesem Kapitel werden die für die Charakterisierung notwendigen Grundlagen vorgestellt. Hierzu wird zunächst im Allgemeinen auf Verlässlichkeitskenngrößen wie Zuverlässigkeit und Verfügbarkeit und ihrer Einflussfaktoren eingegangen. Danach werden kurz verschiedene Fehlermodelle und Testkenngrößen definiert. Dann folgen die Grundlagen zum integrierten Selbsttest und zur Diagnose sowie die mit dem Technologiefortschritt einhergehenden neuen Fehlerarten, die zu Frühausfällen führen können.

2.1 Verlässlichkeitskenngrößen

Um die Verlässlichkeit eines Systems, also die Wahrscheinlichkeit, ob das System über eine bestimmte Zeit oder zu einem bestimmten Zeitpunkt fehlerfrei funktioniert, zu bewerten, gibt es verschiedene Kenngrößen. Für bestimmte Anwendungen ist eine besonders hohe Zuverlässigkeit erwünscht. Die Zuverlässigkeit wird definiert als Wahrscheinlichkeit $R(t)$, dass das System im Zeitintervall $[0, t]$ durchgehend fehlerfrei funktioniert. Im Gegensatz dazu gibt die Verfügbarkeit $A(t)$ an, dass zum Zeitpunkt t die Schaltung fehlerfrei arbeitet [KK07]. In neueren Systemen taucht allerdings die Frage auf, welche dieser Maßzahlen sinnvoll ist. Gegeben sei beispielsweise ein System, in dem jede Stunde ein Fehler festgestellt wird. Nach wenigen Sekunden jedoch steht es wieder zur Verfügung. In diesem Fall ist die Zuverlässigkeit schlecht, die Verfügbarkeit allerdings sehr hoch. Dies legt die Frage nahe, was „fehlerhaft“ bedeutet: Ist eine Schaltung fehlerhaft, die innerhalb von 100.000 Stunden einen Fehler verursacht? Oder ist beispielsweise eine Schaltung fehlerhaft, deren ALU ein fehlerhaftes Gatter besitzt und es hierdurch zu einer falschen Rundung bei der Division kommt? Aus diesem Grunde gibt es in der Literatur weiterführende Definitionen der Zuverlässigkeit und Verfügbarkeit nanoelektronischer Systeme, wie z.B. durch Einbringung der durchschnittlichen Rechenkapazität bei Systemen mit mehreren Prozessoren [KK07].

Einer der wichtigsten Parameter zur Analyse der Zuverlässigkeit ist die Fehlerrate, welche die zu erwartende Anzahl von Fehlern pro Zeiteinheit angibt. Sie ist abhängig von vielen internen und externen Faktoren der Schaltung. Abbildung 2.1 zeigt die erwartete Fehlerrate über das Alter der Schaltungskomponenten. Am Anfang, wenn die Schaltung noch jung ist, ist die Fehlerrate sehr hoch, da produktionsbedingte Probleme evtl. nicht durch den Test erkannt werden. Mit zunehmender Zeit sinkt die Fehlerrate erheblich und ist über eine gewisse Zeit konstant. Ab einem gewissen Punkt treten jedoch Alterungseffekte auf und die Fehlerrate steigt wieder an [KK07]. Da die Form der Kurve an eine

Badewanne erinnert, wird diese als *Badewannen-Kurve* bezeichnet.

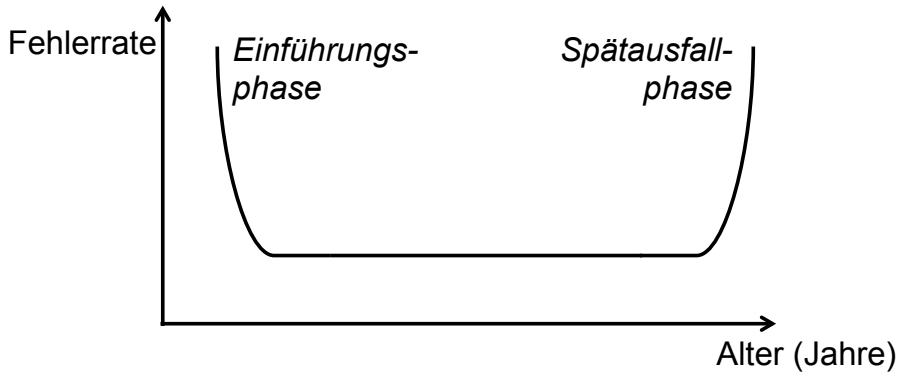


Abbildung 2.1: Badewannen-Kurve

Die Kurve ist allerdings auch von anderen Faktoren abhängig, wie der Lernkurve der Produktionstechnologie (je neuer die Technologie, desto länger die Einführungsphase), der Umgebungstemperatur, dem Schaltverhalten, aber auch der Komplexität der Schaltungsfunktion. Je anfälliger also eine Schaltung gegenüber diesen Effekten ist, desto wichtiger wird ein guter Test, damit die Senke der Kurve möglichst lange anhält und Frühausfälle vermieden werden.

Ist die Fehlerrate λ über die Zeit konstant, entspricht die Lebensdauer der Schaltungskomponente einer exponentiellen Verteilung und es ergibt sich für die Zuverlässigkeit

$$R(t) = e^{-\lambda t} \tag{1}$$

für $t \geq 0$ [KK07].

2.2 Fehlermodellierung und Test

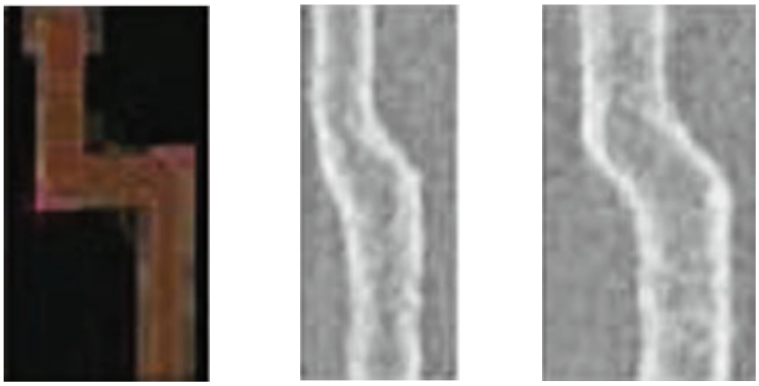
Die Herstellung eines Chips ist ein hochkomplexer Vorgang und lässt sich prinzipiell in drei Teilbereiche einteilen [WMF96] [Hil04]:

1. Herstellung der Siliziumscheibe,
2. Integration der elektrischen Funktionen (Front End),
3. Montage in ein Gehäuse (Verdrahtung und Metallisierung, Back End, Packaging).

In allen Teilbereichen können Defekte und Verunreinigungen der Schaltungsbauteile auftreten, aber besonders die Integration der elektrischen Funktionen, das sogenannte *Front End* ist sehr fehleranfällig. Das Front End wird mit Hilfe der Planartechnik hergestellt, wobei etwa 400 Prozessschritte nötig sind, die in vier Gruppen (*Schichttechnik, Lithographie, Ätztechnik und Dotiertechnik*) eingeteilt werden können [WMF96] [Hil04]. Beispielhaft sollen in den folgenden Ausführungen die Fehler, die während der Lithographie auftreten, detaillierter beschrieben werden:

Während der **Lithographie** wird mit einer geeigneten Maske die Struktur einer Entwurfsebene mit Hilfe von Lichtbestrahlung auf den Fotolack übertragen und anschließend der belichtete Fotolack entfernt. Wenn hierbei beispielsweise ein Partikel auf dem Wafer liegt, sind Abbildungsfehler die Folge, die zu einem Ausfall der Schaltung führen können. Noch kritischer ist ein Defekt auf der Maske, da dieser auf der Scheibe vervielfacht wird. Um Strukturgrößen zu realisieren, die geringer sind als die Wellenlänge von Licht, werden Abbildungssysteme eingesetzt, um trotz der Beugungseffekte des Lichts die Struktur auf das Substrat zu übertragen. Allerdings nimmt hierdurch die Tiefenschärfe ab, so dass Ungenauigkeiten entstehen können [WMF96]. Abbildung 2.2 zeigt ein solches Beispiel für Leiterbahnen: Während 2.2(a) die Layoutspezifikation zeigt, zeigen 2.2(b) und 2.2(c) die produzierten Leitungen auf dem Silizium, die aufgrund der Unschärfe der Lithographie dünner (2.2(b)) oder dicker (2.2(c)) ausgefallen sind. Dies bewirkt einen höheren Widerstand oder eine höhere Koppelkapazität und führt zu Veränderungen im zeitlichen Verhalten der Schaltung.

Aber auch wenn Leiterbahnen sehr dicht angeordnet sind, können elektrische Effekte, z.B. das Leitungsübersprechen, auftreten. Hierdurch werden die Schaltungseigenschaften negativ beeinflusst und es entstehen induktive oder kapazitive Kopplungen, die zu verzögerten oder schnelleren Pegelwechseln führen können. Somit wird auch die Zuverlässigkeit der Schaltung negativ beeinflusst.



(a) Spezifikation (b) mögliche Produktion (c) mögliche Produktion

Abbildung 2.2: Auswirkungen von Prozessvariationen auf Leiterbahnen [TPC12]

Um Defekte nach der Produktion zu entdecken und eine maximale Produktqualität zu erreichen, müssen effiziente Tests mit hoher Fehlerabdeckung eingesetzt werden. Zur Bestimmung der Fehlerabdeckung wird Fehlersimulation eingesetzt, bei der die Defekte als Fehler modelliert werden. Fehlermodelle haben das Ziel, die komplexen und vielfältigen Verhalten möglicher Defekte abstrakt und effizient darzustellen. Die für diese Arbeit wichtigen Fehlermodelle werden im nächsten Kapitel weiter ausgeführt.

2.2.1 Fehlermodellierung

Das Ziel eines guten Fehlermodells ist es, einen Defekt bzw. dessen Auswirkung auf die Schaltungsfunktion möglichst genau abzubilden, so dass dessen Effekte mit wenig

Rechenleistung und -zeit effizient simuliert und analysiert werden können. Es wird hierbei prinzipiell zwischen dem Einfachfehler- und Mehrfachfehlermodell unterschieden.

Das *Einfachfehlermodell* modelliert einen Defekt als Fehler, der zum Zeitpunkt t nur Auswirkungen auf einen Fehlerort hat, beispielsweise auf den Ausgang eines UND-Gatters. Prinzipiell können pro Fehlerort k Typen von Fehlern auftreten, wobei in den meisten Fehlermodellen $k = 2$ gilt. Beispielsweise könnte ein Defekt dazu führen, dass der Ausgang eines UND-Gatters dauerhaft auf 0 oder auf 1 gesetzt wird (sog. Haftfehler). Somit können bei n möglichen Fehlerorten insgesamt

$$2 \cdot n \tag{2}$$

Haftfehler auftreten.

Im *Mehrfachfehlermodell* wird angenommen, dass sich ein Fehler an mehreren Schaltungsknoten auswirkt oder auch mehrere Fehler gleichzeitig auftreten. So können bei n möglichen Schaltungsknoten und k Fehlertypen pro Schaltungsknoten insgesamt

$$(k+1)^n - 1 \tag{3}$$

Fehler auftreten. Wie man sieht, ist das Mehrfachfehlermodell bei größeren Schaltungsstrukturen nicht anwendbar. Allerdings hat sich auch gezeigt, dass eine gute Einfachfehlerabdeckung automatisch eine gute Mehrfachfehlerabdeckung impliziert [BA00], weswegen im weiteren Verlauf dieser Arbeit von dem Einfachfehlermodell ausgegangen wird.

Die für diese Arbeit wichtigsten Fehlermodelle sind das Haftfehler-, das Verzögerungsfehler- und das bedingte Haftfehlermodell, die im weiteren Verlauf genauer definiert werden.

Haftfehlermodell

Das bekannteste, aber auch einfachste Fehlermodell ist das *Haftfehlermodell* (engl. *stuck-at*), welches annimmt, dass ein Ein- oder Ausgang eines Gatters permanent auf 0 oder 1 gesetzt wird. Deswegen spricht man dann auch von einem *stuck-at-0* (SA0) oder *stuck-at-1* (SA1)-Fehler. Abbildung 2.3 zeigt einen möglichen *stuck-at-0*-Fehler am Eingang eines UND-Gatters. Durch dieses Fehlverhalten wird der Ausgang ebenfalls permanent auf 0 gesetzt, da 0 der steuernde Wert[2] des UND-Gatters ist. Tabelle 2.1 zeigt die daraus resultierende Wertetabelle: Der Fehler wirkt sich bei der Eingangskombination $I1 = 1, I2 = 1$ aus und ändert den Ausgang O auf 0 (im fehlerfreien Fall wäre $O = 1$).

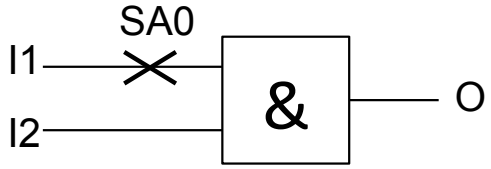


Abbildung 2.3: Haftfehler

I1	I2	O
0	0	0
0	1	0
1	0	0
1	1	1 –> 0

Tabelle 2.1: Wertetabelle

[2] Als steuernder Wert eines Logikgatters wird der Wert bezeichnet, der die Ausgabe des Gatters auf einen bestimmten Wert setzt unabhängig von der Eingangsbelegung der anderen Gattereingänge. Eine 0 ist der steuernde Wert eines UND-Gatters, da der Ausgang immer 0 ist, unabhängig von den anderen Eingangsbelegungen.

Verzögerungsfehlermodell
Das *Verzögerungsfehlermodell* (engl. *delay-fault*) geht davon aus, dass ein Teil der Schaltung langsamer schaltet als vorgesehen. Wirkt sich diese Verzögerung an mindestens einem Ausgang aus, kommt es also zu einer fehlerhaften Ausgabe, spricht man von einem *Transitionsfehler* (*TF*). Es wird zwischen dem *Gatterverzögerungsfehler* und dem *Pfadverzögerungsfehler* unterschieden. Beim Gatterverzögerungsfehler wird eine Verzögerung eines einzelnen Gatters angenommen, während beim Pfadverzögerungsfehler ein gesamter Pfad einer Schaltung betrachtet wird. In den nachfolgenden Ausführungen soll der Gatterverzögerungsfehler genauer erläutert werden:

Ein Gatterverzögerungsfehler kann entweder ein *slow-to-fall-* oder ein *slow-to-rise*-Fehler sein. Während beim slow-to-fall-Fehler, wie der Name schon sagt, eine Transition von 1 nach 0 am Gatterausgang langsamer stattfindet als spezifiziert, findet bei einem slow-to-rise-Fehler ein verzögerter 0-1 Übergang statt. Die Schaltzeit t_g eines Gatters entspricht dann

$$t_g = t_{nominal} + d_g, \tag{4}$$

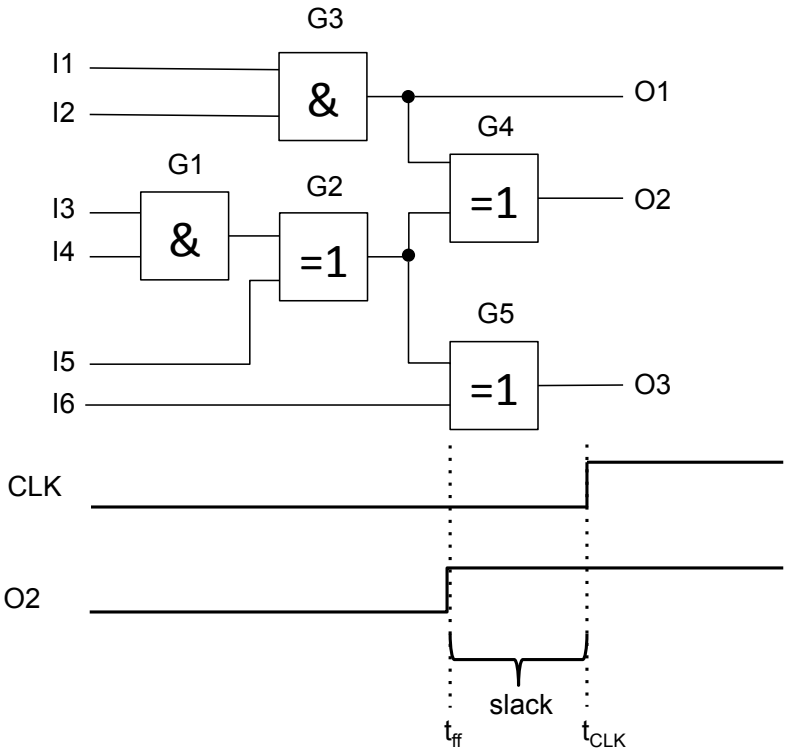
mit der nominalen Gatterschaltzeit $t_{nominal}$ und der Verzögerung d_g. Das Beispiel aus Abbildung 2.4(a) soll diese Fehlermodellierung verdeutlichen: Gegeben sei eine Schaltung mit fünf Gattern, sechs Eingängen und drei Ausgängen sowie einem Takt *CLK* mit der Taktperiode t_{CLK}. Im fehlerfreien Fall schaltet der Ausgang O2 von 0 auf 1 zum Zeitpunkt $t_{ff} < t_{CLK}$. Die Differenz $t_{CLK} - t_{ff}$ wird Pufferzeit (engl. *slack*) genannt. Kommt es nun zu einem Gatterverzögerungsfehler an Gatter G4, so kann es passieren, dass die Transition am Ausgang O2 erst nach t_{CLK} stattfindet (gekennzeichnet mit t_f). Die Schaltung arbeitet fehlerhaft und es liegt ein Verzögerungsfehler vor: Die Pufferzeit ist negativ (Abbildung 2.4(b)).

Bedingtes Haftfehlermodell
In [HW07] wurde das *bedingte Haftfehlermodell* (engl. *conditional stuck-at faultmodel*) vorgestellt, welches beliebig komplexe Fehler modellieren kann. An einem Schaltungsknoten v kann ein bedingter Haftfehler *cond_0_v* oder *cond_1_v* auftreten. Wenn die Bedingung *cond* erfüllt ist, wird die Leitung v auf 0 bzw. 1 gesetzt. Als Bedingungen können Boolesche Ausdrücke angegeben werden. Hiermit können sowohl logische als auch zeitliche Fehler modelliert werden. Der bedingte Haftfehlfer *(v=0)_1_v* beschreibt beispielsweise den Haftfehler SA1 und $(v_{-1} = 0 \wedge v = 1)\_0\_v$ beschreibt einen slow-to-rise-Fehler an Leitung v. Es lassen sich aber auch beliebig komplexere Bedingungen aufstellen.

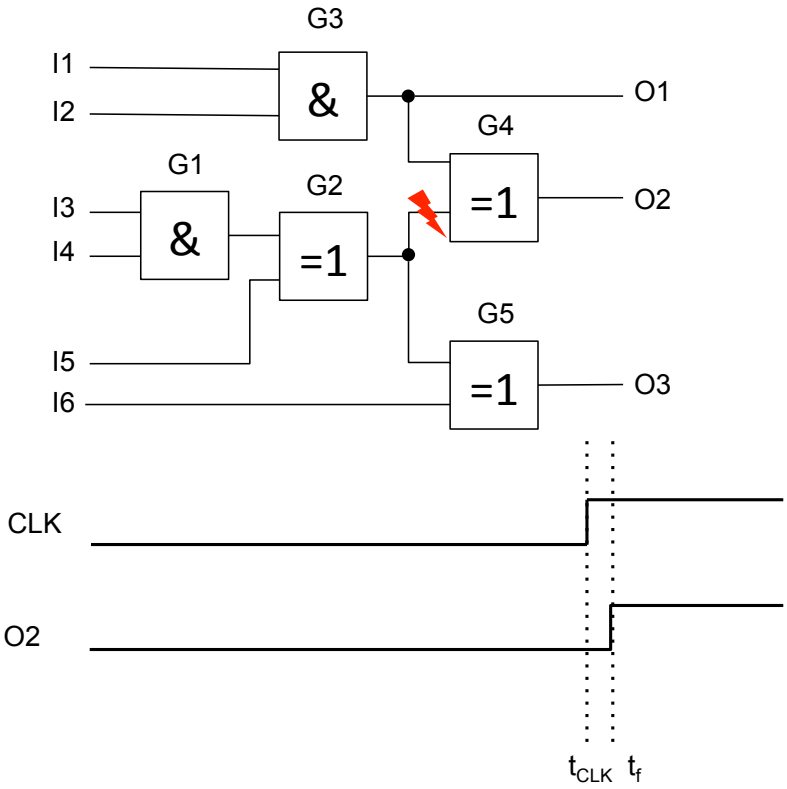
2.2.2 Kennzahlen und Bedeutung des Tests

Aus wirtschaftlicher Sicht ist das Ziel des Herstellers hochintegrierter Schaltungen die Maximierung der *Ausbeute*, das Ziel des Tests ist es, die *Produktqualität* zu maximieren und gleichzeitig unnötige Ausbeuteverluste zu vermeiden. Im Allgemeinen wird die Ausbeute Y durch die Funktion

$$Y = \frac{n_{ff}}{n_{gesamt}} \tag{5}$$



(a) im fehlerfreien Fall



(b) mit Gatterverzögerungsfehler an Gatter G4

Abbildung 2.4: Beispielschaltung

mit der Anzahl der fehlerfreien Chips n_{ff} und der Anzahl aller produzierten Chips n_{gesamt} definiert [WWW06]. Da n_{ff} in der Praxis nur abgeschätzt werden kann, gibt es in der Literatur einige Modelle zur Abschätzung der Ausbeute. Eines der ersten, das *Poissonmodell*, ist zwar ein pessimistischer Ansatz, erklärt jedoch gut die Parameter, die auf die Ausbeute Einfluss haben, und soll deswegen nun im Detail erläutert werden:

Das *Poissonmodell* geht davon aus, dass die Defekte gleichverteilt auf dem Wafer auftreten. Ausgehend von n_{cw} Chips auf dem Wafer ist die Wahrscheinlichkeit, dass ein Defekt auf einem bestimmten Chip auftritt $p = 1/n_{cw}$. Die Wahrscheinlichkeit $p(k)$, dass genau k Defekte auf einem Chip auftreten, kann bei insgesamt n aufgetretenen Defekten mit der Binomialverteilung

$$p(k) = \binom{n}{k} \cdot p^k \cdot (1-p)^{n-k} \tag{6}$$

beschrieben werden. Nach dem Poissonschen Grenzwertsatz konvergiert für $n \to \infty$ und $p \to 0$ der Erwartungswert gegen eine Konstante $\mu = n/n_{cw}$ und die Binomialverteilung nähert sich der Poissonverteilung

$$p(k) = \frac{\mu^k}{k!} \cdot e^{-\mu}. \tag{7}$$

Die Ausbeute Y kann hiermit als Wahrscheinlichkeit, dass $k = 0$ Defekte auf einem Chip vorhanden sind, abgeschätzt werden und entspricht

$$Y = p(0) = e^{-A_c \cdot d}, \tag{8}$$

mit einer Chipfläche A_c und einer Defektdichte d [KK07].

Das Poissonmodell geht von einer gleichmäßigen und gleichverteilten Fehlerdichte aus, die jedoch in der Praxis so nicht, oder zumindest kaum, auftritt. Es ist eine sehr pessimistische Annahme, da sich in der Praxis beobachten lässt, dass Defekte lokal gehäuft auftreten, so dass das Poissonmodell nur eingesetzt wird, um schnell und einfach eine untere Abschätzung zu berechnen. Weitere Ausbeutemodelle werden in [Wun91] und [WWW06] beschrieben. Allerdings sieht man an diesem einfachen Modell schon verschiedene Einflussfaktoren: Je größer die Fläche eines Chips, desto höher die Wahrscheinlichkeit eines Defekts und desto niedriger die Ausbeute. Je größer allerdings der Wafer bei gleichbleibender Defektdichte und Chipfläche, desto geringer die Produktionskosten pro Chip. Dies erklärt, zusammen mit vielen anderen Faktoren, die Vorteile der sinkenden Strukturgrößen und größeren Wafer.

Die entscheidende Größe für den Test ist die *Produktqualität* PQ. Diese gibt das Verhältnis der fehlerfreien Chips zu den als fehlerfrei eingestuften Chips an. Interessant ist hierbei der Zusammenhang zwischen Test und Produktqualität. Hierzu soll zunächst eine weitere wichtige Maßzahl der Testqualität definiert werden:

Die *Fehlerabdeckung fc* (aus dem engl. *fault coverage*) gibt das Verhältnis von den Fehlern, die von einem Test erkannt werden, zu allen möglichen Fehlern an. Die Fehlerabdeckung hängt mit der Produktqualität durch folgende Gleichung zusammen [WB81]:

$$PQ = Y^{1-fc}. \tag{9}$$

Die Abhängigkeit der Produktqualität von der Fehlerabdeckung für gegebene Ausbeute, kann graphisch Abbildung 2.5 entnommen werden.

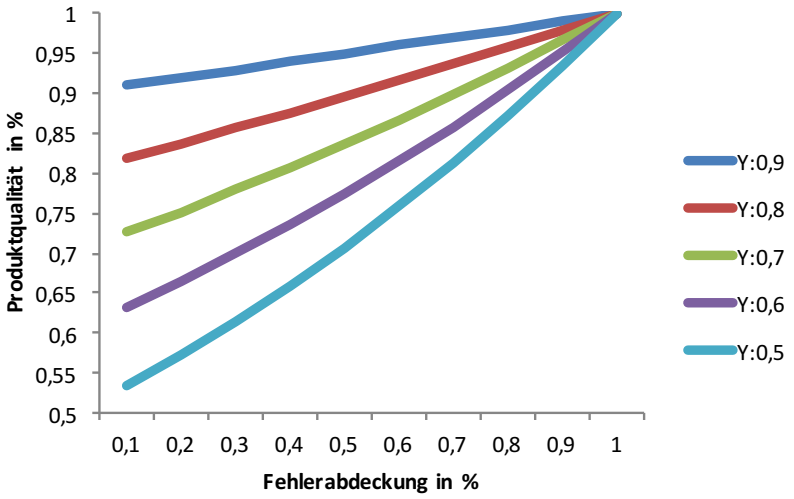


Abbildung 2.5: Produktqualität abhängig von der Fehlerabdeckung

Wie man hieraus erkennen kann, muss das Ziel sein, eine möglichst hohe Fehlerabdeckung zu erreichen, um die Produktqualität zu maximieren. Das bedeutet: Es muss ein Test gefunden werden, der im Optimalfall alle möglichen Fehler in einer Schaltung erkennt. Allerdings kann es vorkommen, dass auf Grund von Fehlern, die nicht erkannt werden können, nie eine 100%-ige Fehlerabdeckung erreicht werden kann. Ein Fehler kann dann nicht erkannt werden, wenn sich an den Schaltungsausgängen kein Unterschied zwischen fehlerfreiem und fehlerhaftem Verhalten feststellen lässt. Aus diesem Grund wird zur Validierung einer Testinfrastruktur oder Testmustermenge oft die Fehlereffizienz (engl. *fault detection efficiency* oder fault efficiency) herangezogen, welche den Anteil der erkannten Fehler von der Anzahl aller Fehler abzüglich der nicht erkennbaren Fehler beschreibt.

Zusätzlich müssen allerdings unnötige, durch den Test bedingte, Ausbeuteverluste vermieden werden. Ausbeuteverluste können beispielsweise dadurch entstehen, dass der Test Schaltungen als fehlerhaft identifiziert, der Fehler aber durch die eingebaute Fehlertoleranz während des Betriebs keine Auswirkungen auf die Funktionalität hätte. Diese Problematik wird in Kapitel 3.1 genauer diskutiert.

2.3 Der Logiktest

Bei der Serienproduktion von Chips durchlaufen diese nach der Herstellung eine Reihe von Schritten zur Überprüfung der Qualität und Funktionsweise. Diese Arbeit konzentriert sich auf Digitalschaltungen, wobei ein wichtiger Teilaspekt des Produktionstests der Logiktest ist. Hierbei wird die Schaltung mit einer Menge von Testmustern stimuliert und die Antworten mit den erwarteten Antworten verglichen. Der Logiktest kann prinzipiell auf Struktur- oder Funktionsebene ansetzen. Während bei einem funktionalen Test Testmuster zur Überprüfung der Ausgabefunktion einer Schaltung erzeugt werden, also das

reine Ein- und Ausgabeverhalten untersucht wird, werden beim Test auf Strukturebene Testmuster auf der Grundlage eines strukturorientierten Fehlermodells erstellt.

Üblicherweise wird ein Produktionstest mit Testautomaten (engl. *Automatic Test Equipment*, *ATE*) durchgeführt, dessen Kosten vor allem durch Testgeschwindigkeit, Anzahl testbarer Anschlüsse, Durchsatz, Genauigkeit und Auflösung bestimmt ist [Wun91]. Durch die steigende Komplexität der hergestellten Schaltungen kann ein ATE-Test sehr teuer und zeitaufwendig werden und es sind unter Umständen bestimmte Module auf dem Chip nicht testbar. Deswegen wird in heutigen Schaltungen die Testinfrastruktur mehr und mehr auf dem Chip selbst implementiert. Ein solcher Selbsttest wird *Built-In Self-Test* (*BIST*) genannt und wird aufgrund der herausragenden Bedeutung für diese Arbeit in Kapitel 2.4 ausführlich beschrieben.

Allgemein lassen sich Digitalschaltungen in zwei Gruppen einteilen: kombinatorische und sequentielle Schaltungen. Während kombinatorische Schaltungen eine Boolesche Funktion

$$I \rightarrow O \tag{10}$$

implementieren, mit $I = \{0,1\}^n$ als Menge der Eingänge und $O = \{0,1\}^m$ als Schaltungsausgangsmenge, ist eine sequentielle Schaltung zusätzlich noch von ihrem internen Zustand abhängig. Dies kann als Funktion

$$I \times Z \rightarrow Z \times O \tag{11}$$

interpretiert werden, mit $Z = \{0,1\}^s$ als Menge der Zustände der Schaltung. Die Zustände sind in internen Registern, die aus Flip-Flops bestehen, gespeichert. Da die Register nicht direkt von außen angesteuert werden können, müssen diese über die primären Eingänge gesetzt werden, was unter Umständen nur schwer oder gar nicht möglich ist und das Testen enorm erschwert. Um die Testbarkeit von sequentiellen Schaltungen zu verbessern, wird der Schaltungsentwurf mit einem Prüfpfad versehen (engl. *Scan-Design*).

2.3.1 Der Prüfpfad

Für die Verwendung eines Prüfpfades wird jedes Flip-Flop durch zusätzliche Hardware in eine Prüfpfad-Zelle umgewandelt, deren einfachster Aufbau für den flankengesteuerten Entwurf schematisch in Abbildung 2.6 gezeigt ist [WWW06].

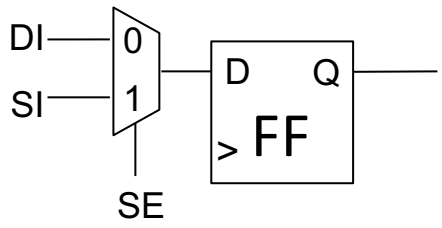


Abbildung 2.6: Schematischer Aufbau einer Prüfpfad-Zelle

Diese Prüfpfad-Zelle erlaubt es, entweder die Daten aus der Logik ($SE = 0$) oder die Daten, die am Eingang SI anliegen ($SE = 1$), in das Flip-Flop aufzunehmen. Die Prüfpfad-Zellen werden zu einem oder mehreren Prüfpfaden verbunden, um über möglichst wenige

zusätzliche Eingänge eine Vielzahl von Zellen zu erreichen. Im Beispiel aus Abbildung 2.7 sind drei Prüfpfad-Zellen zu einem Prüfpfad zusammengefasst, so dass das Testmuster über SI in den Prüfpfad geschoben und über PPI an der Schaltung anliegt. Diese zusätzlichen Schaltungseingänge werden, in Anlehnung an die Bezeichnung „Primäreingang“ (engl. *primary inputs, PI*), Pseudo-Primäreingänge (engl. *pseudo-primary inputs, PPI*) genannt. Die Testantworten werden dann mit dem nächsten Taktsignal über PPO in die Prüfpfad-Zellen aufgenommen und anschließend über den Ausgang SO herausgeschoben. Wegen der Bezeichnung „Primärausgang“ (engl. *primary output, PO*) werden diese zusätzlichen Ausgänge deswegen als Pseudo-Primärausgänge (engl. *pseudo-primary outputs, PPO*) bezeichnet. Ein Test mit einem 3-Bit Testmuster ψ benötigt also insgesamt sieben Takte (drei Takte um das Testmuster in den Prüfpfad zu schieben, einen Takt für die Aufzeichnung der Testantwort und wieder drei Takte für das Herausschieben der Testantwort). Da das Hineinschieben eines neuen Testmusters parallel zum Herausschieben der vorherigen Testantwort geschehen kann, benötigt im Allgemeinen ein Test mit n Testmustern und einer Prüfpfadlänge von l_{pr} insgesamt $n \cdot l_{pr} + n + l_{pr}$ Takte.

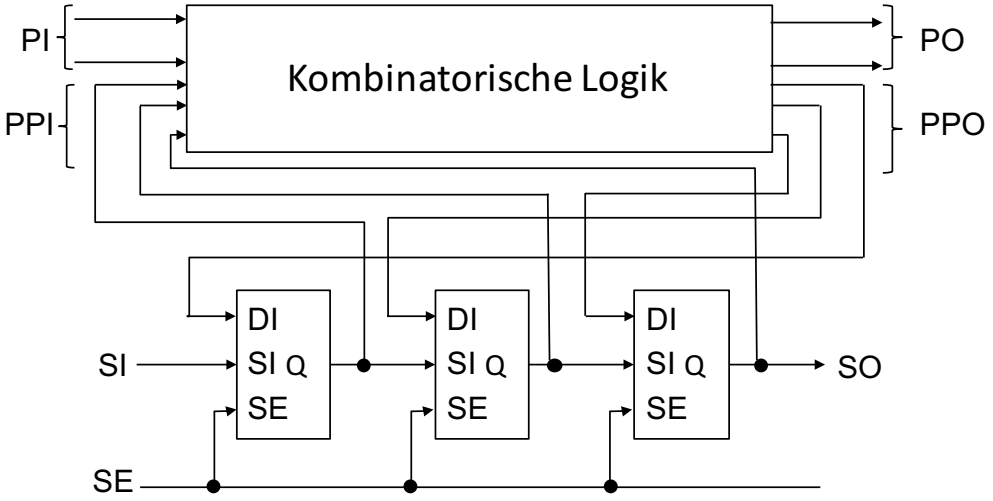


Abbildung 2.7: Full-Scan Design

Mit Hilfe des Entwurfs mit Prüfpfaden ist der Test auf ein rein kombinatorisches Problem reduziert worden. Da durch den Prüfpfad bereits im Entwurfs-Schritt die Testbarkeit deutlich vereinfacht wird, spricht man in diesem Zusammenhang von einem testfreundlichen Entwurf (engl. *Design for Testability*, *DfT*). Um Verzögerungsfehler zu erkennen, muss der Prüfpfad allerdings noch erweitert werden.

2.3.2 Testverfahren für Verzögerungsfehler

Für den Test auf Verzögerungsfehler wird nicht ein einzelnes Testmuster, sondern ein Testmusterpaar $PP = \{\psi_1, \psi_2\}$ benötigt: Das erste Testmuster ψ_1 initialisiert die Logik, während durch das zweite Testmuster ψ_2 der Pegelwechsel erzeugt wird. Erreicht der Pegelwechsel nicht innerhalb der Taktperiode einen der Primär- oder Pseudo-Primärausgänge, wurde ein Fehler erkannt. Problematisch hierbei ist das sequentielle Laden des Prüfpfades und die damit verbundene Verzögerung zwischen den Testmustern, die

einen Test auf Verzögerungsfehler ohne Anpassungen nicht möglich macht. In der Literatur sind deshalb drei verschiedene Arten der Testmustererzeugung für Verzögerungsfehler mit Prüfpfaden bekannt:

- Launch-on-Shift (LOS),
- Launch-on-Capture (LOC) und
- Enhanced Scan (ES).

Bei *Launch-on-Shift (LOS)*, oder auch *Skewed Load-Verfahren* [WWW06] genannt, werden zunächst die ersten $l-1$ Bit des Testmusters ψ_1 mit einem langsamen Takt, in Abbildung 2.8 mit *Schiebetakt* gekennzeichnet, in den Prüfpfad geladen. Durch diese $l-1$ Bits wird die Schaltung initialisiert. Durch das letzte Schieben wird dann der Pegelwechsel erzeugt, so dass dieser Takt als *Launch Takt* bezeichnet wird. Das *SE*-Signal wird auf 0 gesetzt und das Aufzeichnen der Testantwort durch einen schnellen Aufzeichnungstakt sichergestellt (*Capture Takt*). Das Testmuster ψ_2 entspricht somit dem um 1 Bit verschobenen Testmuster ψ_1 und die Testantwort auf das Testmusterpaar PP kann ausgegeben werden. Vorteilhaft an diesem Verfahren ist die Tatsache, dass aus einem einzelnen Testmuster das Testmusterpaar entsteht. Nachteilig ist allerdings die starke Abhängigkeit des Testmusters ψ_2 von ψ_1.

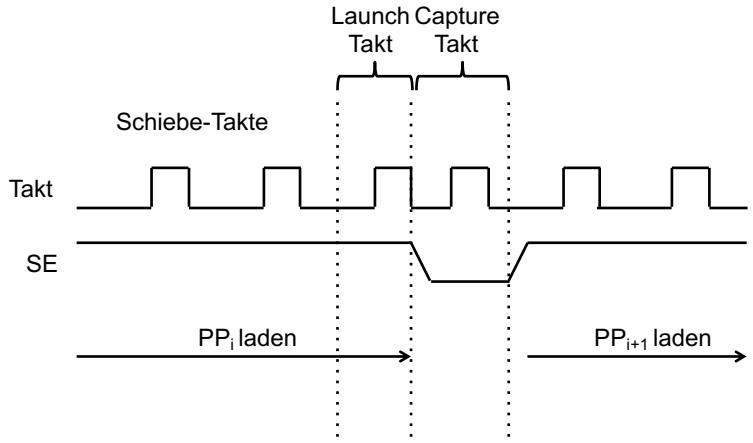


Abbildung 2.8: Launch-on-Shift Verfahren [WWW06]

Im Unterschied zu LOS wird bei *Launch-on-Capture (LOC)*, oder auch *Broadside Test* [WWW06], ψ_2 aus der Testantwort von ψ_1 generiert, siehe Abbildung 2.9. Zunächst wird ψ_1 vollständig in den Prüfpfad geladen. Danach wird mit einem schnellen Takt die Schaltung initialisiert (*Launch Takt*). Durch $SE = 0$ wird die Testantwort von ψ_1 in den Prüfpfad geladen. Diese Testantwort wird dann als zweites Testmuster ψ_2 verwendet und durch einen zusätzlichen schnellen Takt die Antwort auf das Testmusterpaar PP schließlich aufgezeichnet (*Capture Takt*). Im Unterschied zu LOS muss bei LOC das *SE*-Signal nicht mit der schnellen Testfrequenz erzeugt werden, so dass die Anforderungen an das Testequipment geringer sind. Allerdings ist die Fehlerabdeckung eines LOC-Tests im Allgemeinen geringer als bei einem LOS-Test, da nicht immer alle Bits einer Testantwort bekannt sind und sich somit unbekannte Werte in ψ_2 ergeben. Diese unbekannten Werte

an den Schaltungseingängen wirken sich stark auf die entsprechenden aufgezeichneten Antworten von ψ_2 aus.

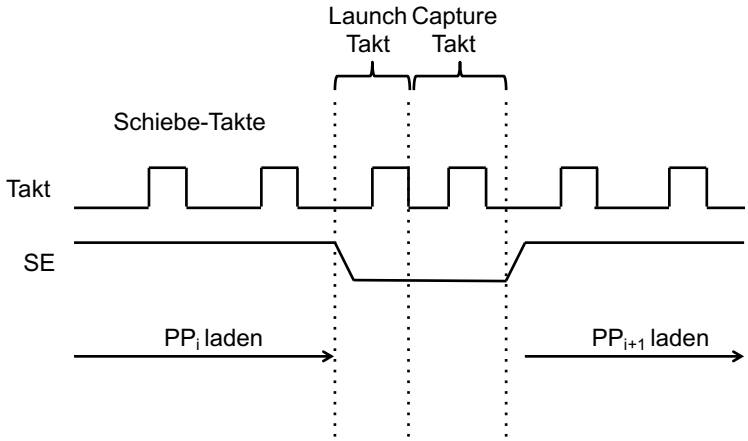


Abbildung 2.9: Launch-on-Capture Verfahren [WWW06]

Für *Enhanced Scan (ES)* wird jede Prüfpfad-Zelle um ein zusätzliches *Halte-Latch* erweitert. Hierdurch lassen sich zwei voneinander unabhängige Testmuster in den Prüfpfad laden. Zunächst wird Testmuster ψ_1 in den Prüfpfad geladen und in die zusätzlichen Latche abgelegt. Dann wird ψ_2 in den Prüfpfad geladen und die Testmuster nacheinander an die Schaltung angelegt. Die Testantwort von ψ_2 wird schließlich mit einem schnellen Takt (*Capture*) aufgezeichnet und ausgegeben. Ein Vorteil von ES ist einerseits, dass durch die Unabhängigkeit von ψ_1 und ψ_2 eine höhere Fehlerabdeckung erreicht wird (es kann Fehler geben, die in LOS und LOC aufgrund der Abhängigkeit nicht erkannt werden können). Andererseits benötigt der Test in den meisten Fällen durch den höheren Freiheitsgrad in ψ_2 sogar weniger Testmusterpaare als bei LOS oder LOC. Nachteilig ist der zusätzliche Hardwareaufwand und die damit einhergehenden Kosten durch die zusätzlichen Latche und deren Verdrahtungen.

2.4 Selbsttest (BIST)

Durch die steigende Komplexität und Leistungsfähigkeit heutiger Schaltungen und den damit verbundenen steigenden Testkosten werden mehr und mehr Testkomponenten auf dem Chip selber implementiert. Hierbei lässt sich zwischen einem autonomen Selbsttest, der als *Built-In Self-Test (BIST)* bezeichnet wird, und einem eingebetteten Test (engl. *embedded test*) unterscheiden. Während bei einem eingebetteten Test die eingebaute Testinfrastruktur noch vom externen Testautomat gesteuert wird, um beispielsweise die Testinformationen komprimiert vom Tester an den Chip zu übertragen und dann auf dem Chip zu dekodieren, werden beim BIST Komponenten zur Testmustergenerierung (*TMG*, engl. *Test Pattern Generator, TPG*), Auswertung der Testantworten (engl. *Test Response Evaluation*, TRE) und die Steuerung auf dem Chip implementiert. Hierdurch lässt sich das zu testende Device bzw. die zu testende Schaltung (engl. *Device-under-Test*, DUT bzw. *Circuit-under-Test*, CUT) ohne externen Testautomaten auf Defekte überprüfen. Von außen werden über den *Test Access Mechanism* (*TAM*) die für den Test notwendigen Daten

bereit gestellt. Ein Selbsttest hat den Vorteil, dass die Kosten für den externen Testautomaten reduziert werden und auch Module getestet werden können, die nur über die Anschlusspins evtl. gar nicht bzw. nur sehr schwer erreicht werden. Zusätzlich gibt es die Möglichkeit, dass sich der Chip während des Produktlebenszyklus' selber testet, beispielsweise nach einem Neustart, und den Benutzer auf mögliche Probleme hinweist. Nachteilig ist der erhöhte Hardwareaufwand, was zum einen höhere Kosten verursacht und zum anderen zusätzliche Fläche für mögliche Defekte bietet. So muss sichergestellt werden, dass die Testinfrastruktur selber fehlerfrei ist, um die Testergebnisse nicht zu verfälschen.

Für Schaltungen mit Prüfpfad ist die in Abbildung 2.10 gezeigte STUMPS Architektur (engl. *self-test using MISR and parallel shift register sequence generator*) eine der populärsten Architekturen [BM82]. Die Schaltung ist mit Prüfpfaden ausgestattet und Testmuster werden vom TMG erzeugt. Die Testantworten werden mit einem MISR (engl. *multiple input signature register*) kompaktiert. Die Funktionsweisen dieser Module sollen in den nächsten Kapiteln genauer erklärt werden.

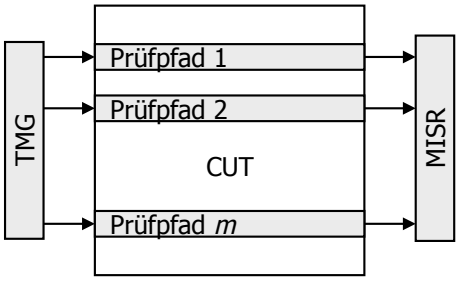


Abbildung 2.10: STUMPS Architektur

2.4.1 Testmustererzeugung im Selbsttest

Als TMG wird im Selbsttest im einfachsten Fall ein linear rückgekoppeltes Schieberegister (engl. *linear feedback shift-register, LFSR*) eingesetzt, welches eine deterministische Zahlenfolge erzeugt. Die Struktur eines LFSR kann komplett durch das charakteristische Polynom $h(Y) \in GF(2)[Y]$ vom Grad k von der Form

$$h(Y) = Y^k + \sum_{j=0}^{k-1} h_j Y^j \tag{12}$$

bestimmt werden, wobei im Falle einer Rückkopplung an Stelle j der Koeffizient $h_j = 1$ gesetzt wird, ansonsten $h_j = 0$. Die Zustandsübergänge können auch als lineare Transformation mit der Matrix $H \in (k \times k, GF(2))$ der Form

$$H = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 & 1 \\ 1 & 0 & 0 & \dots & 0 & h_1 \\ 0 & 1 & 0 & \dots & 0 & h_2 \\ \vdots & 0 & \ddots & \vdots & \vdots & \\ 0 & 0 & 0 & \dots & 1 & h_{k-1} \end{bmatrix} \tag{13}$$

dargestellt werden und entsprechen bei Startzustand z der Folge

$$
\begin{aligned}
&t = 0 : z \\
&t = 1 : Hz \\
&t = 2 : H^2 z \\
&t = 3 : H^3 z \\
&\ldots
\end{aligned}
$$

zum entsprechenden Zeitpunkt t [WWW06]. Wird als charakteristisches Polynom ein primitives Polynom benutzt, kann eine maximale Anzahl von $2^k - 1$ verschiedenen Zuständen erreicht werden (alle außer dem 0-Zustand, engl. *all-zero-state*) [LN96].

Für die Rückkopplung werden bei einem LFSR XOR-Gatter verwendet, da es sich um Berechnungen in $GF(2)$ handelt und die Addition (und auch die Subtraktion) XOR-Operationen entspricht. Ein LFSR lässt sich als modulares LFSR (*MLFSR*, auch Galois-LFSR genannt) oder Standard-LFSR (*SLFSR*, auch Fibonacci-LFSR genannt) implementieren. Abbildung 2.11 zeigt die schematische Darstellung eines k-bit modularen LFSR (die XOR-Gatter sind in diesem Schema als $\oplus$ dargestellt).

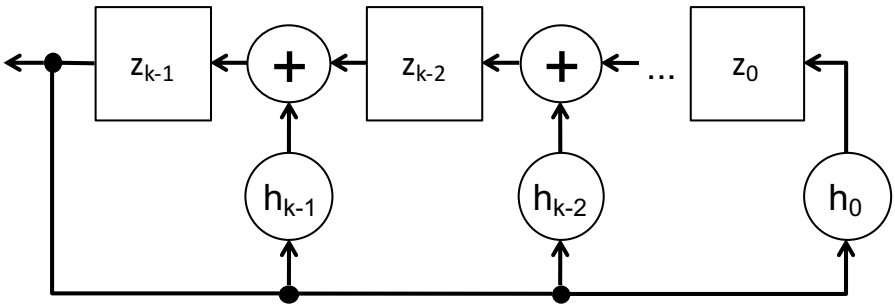
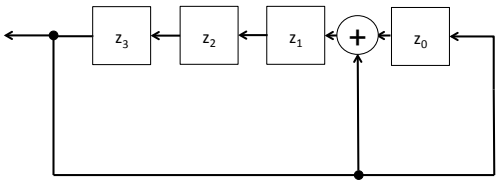


Abbildung 2.11: MLFSR

Im Gegensatz zu einem SLFSR, bei dem sich $z_0 = z_{k-1} + \sum_{j=1}^{k-2} h_j z_j$ und die übrigen $z_i = z_{i-1}$ berechnen lässt, hat das MLFSR die XOR-Gatter zwischen den Flip-Flops. Hierdurch lässt sich $z_0 = h_0 z_{k-1}$ berechnen und die weiteren Bits als $z_i = z_{i-1} + h_i z_{k-1}$ für $i = 1, ..., k-1$. Das MLFSR lässt sich damit schneller takten und bei einem Zustandsübergang entsteht eine höhere Verwürfelung der einzelnen Bits. Mit Hilfe eines LFSRs können pseudo-zufällige Testmuster an die Prüfpfade angelegt werden.

Beispiel: In Abbildung 2.12 wird ein MLFSR vom Grad $k = 4$ gezeigt mit dem primitiven Polynom $Y^4 + Y + 1$. Wird ein Startzustand $z \neq 0$ gewählt, beispielsweise $z_3 = 1, z_2 = 0, z_1 = 0, z_0 = 0$, werden $2^4 - 1 = 15$ verschiedene LFSR-Zustände durchlaufen bis wieder der Startzustand erreicht wird, siehe Tabelle in Abbildung 2.12. Wird der Startzustand $z = 0$ gewählt, sind auch alle folgenden Zustände 0.



Zustand	z_3	z_2	z_1	z_0
0	1	0	0	0
1	0	0	1	1
2	0	1	1	0
3	1	1	0	0
...	...	...	...	...
14	0	1	0	0
0	1	0	0	0

Abbildung 2.12: Implementierung eines MLFSR vom Grad $k = 4$ mit primitivem Polynom und Zustandsübergangstabelle

■

Um lineare Abhängigkeiten zu vermeiden, wird oft noch zusätzlich ein Phasenverschieber eingesetzt. Genauso wird aber auch ein Test mit deterministischen Mustern unterstützt. Die Idee bei einem solchen „*Mixed-Mode*“ BIST ist es, zunächst mit vielen pseudozufälligen Testmustern eine möglichst hohe Fehlerabdeckung zu erreichen und dann gezielt deterministische Muster einzusetzen, um die schwer entdeckbaren Fehler ebenfalls zu erkennen. Die einfachste Idee ist hierbei, die deterministischen Muster in einem zusätzlichen Speicher direkt auf dem Chip abzulegen und bei Bedarf in den Prüfpfad zu laden. Anstatt die kompletten Muster abzuspeichern, können auch nur die Startzustände des LFSRs abgespeichert werden, welche diese deterministischen Muster erzeugen [Koe91]. Hierbei wird ausgenutzt, dass deterministische Muster eine große Anzahl von unspezifizierten Bits, sogenannte „don’t cares“, besitzen. Das heißt, für das deterministische Muster und die Fehlerabdeckung ist es unerheblich, ob es sich bei diesen Bits um eine ’0’ oder eine ’1’ handelt. Das Testmuster wird durch den Startzustand des LFSRs kodiert und hierdurch lässt sich der Speicherbedarf verringern. Diese Flexibilität ist ein großer Vorteil und wird „Reseeding“ genannt.

Viele Arbeiten haben sich damit beschäftigt, die Länge des LFSR und die Speicheranforderungen weiter zu minimieren. In [HRT$^+$95] wird die Idee aufgegriffen, durch ein programmierbares LFSR verschiedene charakteristische Polynome mit dem LFSR darzustellen. Weitere Arbeiten [RTZ98] [KJT01] [AYMM03] [RBO03] [KT04] [KBK$^+$01] [VM03] stellen vor allem geschickte Kodier- und Dekodierungsmaßnahmen für die Startzustände vor. In allen Arbeiten wird ausgenutzt, dass bei bestimmten Fehlern nur wenige Bits eines Testmusters gesetzt werden müssen, um die Fehler zu entdecken.

In [FGGL99] wird ein effizienter Algorithmus zur Berechnung der optimalen LFSR Startzustände für einen Pseudozufallstest eingeführt. Im ersten Schritt des zweistufigen Verfahrens werden die Startzustände so weit wie möglich zusammengefasst und danach werden mittels effizienter Simulationstechniken die optimalen Startzustände berechnet, welche die höchste Fehlerabdeckung garantieren. Eine weitere Möglichkeit ist es, gezielt pseudo-zufällige Testmuster so während des Betriebs zu ändern, dass die gewünschten deterministischen Muster entstehen. Dies kann durch zusätzliche Logik bewerkstelligt werden, um beispielsweise bestimmte Bits zu flippen [WK96] [GWV$^+$04] oder zu fixieren [TM96]. Hierbei wird die Idee ausgenutzt, dass bestimmte Testmuster einer pseudozufälligen LFSR Sequenz keine zusätzlichen Fehler entdecken können und somit nicht

benötigt werden. Durch eine zusätzliche Steuerlogik können somit aus diesen Mustern deterministische Muster erzeugt und diese in die pseudo-zufällige Mustermenge integriert werden. Der Nachteil ist jedoch der zusätzliche Hardwareaufwand, der durch die Steuerung entsteht. Neuere Ansätze, wie beispielsweise [RTKM04], versuchen, beide Ideen miteinander zu verbinden. Hierdurch lassen sich effizient deterministische Testmuster in einem integrierten Selbsttest verwenden.

2.4.2 Testantwortauswertung im Selbsttest

Zur Auswertung des Tests müssen die Testantworten des CUT mit den zu erwartenden Testantworten verglichen werden. Da beim Selbsttest aufgrund des hohen Speicherbedarfs nicht alle Testantworten auf dem Chip vorgehalten werden können, müssen diese mit möglichst wenigen Bits möglichst genau charakterisiert werden. Für diese Art der Kompaktierung wird in der STUMPS Architektur ein MISR eingesetzt. Mit Hilfe dieser zeitlichen Kompaktierungstechnik werden die Testantworten zu einer Signatur zusammengeführt. Grundlage hierfür ist eine Polynomdivision, die mit dem MISR durchgeführt wird. Hierzu wird ein LFSR um zusätzliche Eingänge erweitert, so dass die Testantworten parallel in das MISR geladen werden können. Zur Vereinfachung wird die grundlegende Idee zunächst für eine Schaltung mit nur einem Ausgang und der Ausgangsfolge $A = (a_{l-1}, ...a_0)$ erklärt, die in ein k-Bit breites LFSR mit einem Eingang geschoben wird (SISR genannt, engl. *Single Input Signature Register*), siehe Abbildung 2.13.

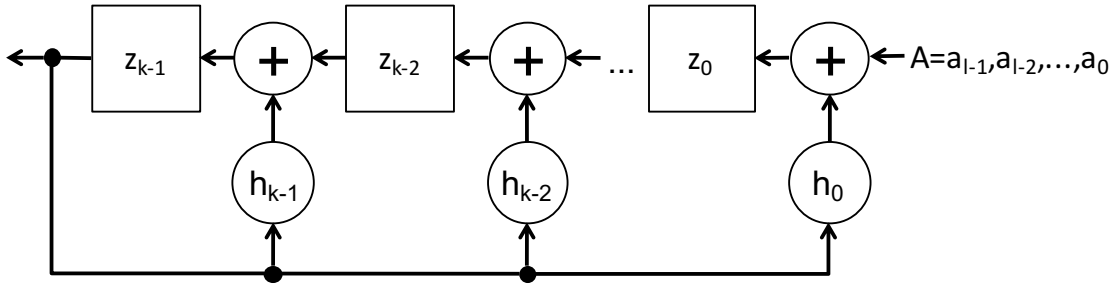


Abbildung 2.13: SISR

Das Rückkopplungspolynom sei wieder $h(Y) \in GF(2)[Y]$. Interpretiert man A als Polynom

$$A(Y) = a_{l-1}Y^{l-1} + ... + a_1Y + a_0 \in GF(2)[Y], \tag{14}$$

wird hier die Polynomdivision $A(Y)/h(Y)$ durchgeführt. Nachdem alle l Testantworten in das MISR geschoben wurden, repräsentiert der Zustand des MISR-Registers die Koeffizienten des Rest-Polynoms $r(Y) = r_{k-1}Y^{k-1} + ... + r_1Y + r_0$ [PJ72]. Der Vektor $(r_0, ..., r_{k-1}) \in GF(2)^k$ wird *Signatur* genannt.

Der Informationsverlust, der durch die Kompaktierung entsteht, kann dazu führen, dass eine falsche Testantwort zu einer richtigen Signatur verarbeitet wird. Dieses Fehlverhalten wird *Aliasing* genannt, allerdings ist die Wahrscheinlichkeit hierfür sehr gering. Für ein k-Bit MISR kann die Wahrscheinlichkeit mit

$$p_{Aliasing} \approx 2^{-k} \tag{15}$$

abgeschätzt werden. Das bedeutet, dass die Qualität dieser Kompaktierung allein von der Größe des MISRs abhängig ist [WWW06].

> **Beispiel**: Für den Test einer Schaltung mit 32 Ausgängen, 10.000 Testmustern und einem 32-bit MISR zur Testantwortkompaktierung wird eine Kompaktierungsrate von 10.000X erreicht, wobei die Aliasing Wahrscheinlichkeit lediglich $p_{Aliasing} \approx 2^{-32} \approx 2,33 \cdot 10^{-10}$ beträgt. ■

Für eine Schaltung mit mehreren parallelen Prüfpfaden wird der Kompaktierer aus Abbildung 2.13 angepasst, um die Testantworten parallel zu laden. Hierzu werden zwischen den Flip-Flops XOR3 oder zwei XOR2-Gatter eingesetzt. Abbildung 2.14 zeigt schematisch ein MISR mit mehreren Eingängen. Die Ausgabefolge der Schaltung $A_i = (a_{i,l-1}, ...a_{i,0})$ für $i = 1, ..., n$ besteht aus n Vektoren der Breite k.

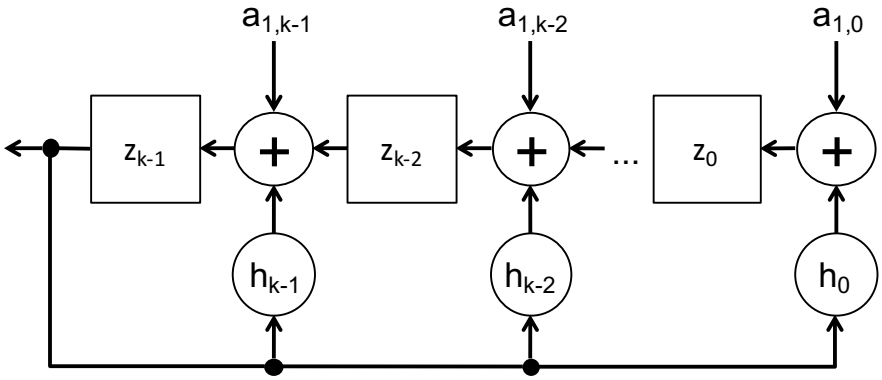


Abbildung 2.14: MISR

Die bereits bekannte Abschätzung der Aliasing-Wahrscheinlichkeit (Gleichung (15)) gilt auch hier [WWW06].

Analog zum LFSR können die Zustandsübergänge des MISRs, auch Signatursequenz genannt, mit primitivem Rückkopplungspolynom $h(Y)$ auch als lineare Transformation mit der Matrix $H \in (k \times k, GF(2))$ dargestellt werden. Die Zustände des MISRs lassen sich somit zum Zeitpunkt t mit Startzustand z als

$$
\begin{aligned}
&t = 0 : z \\
&t = 1 : Hz + A_1 \\
&t = 2 : H^2 z + HA_1 + A_2 \\
&t = 3 : H^3 z + H^2 A_1 + HA_2 + A_3 \\
&... \\
&t = n : H^n z + H^{n-1} A_1 + ... + HA_{n-1} + A_n
\end{aligned}
$$

usw. darstellen. $H^n z + H^{n-1} A_1 + ... + HA_{n-1} + A_n$ ist dann die Signatur.

Testantwortkompaktierung mit unbekannten Werten

Bei XOR-basierten Kompaktierern lässt sich die Sollsignatur nicht berechnen, wenn in der Ausgabefolge A unbekannte Werte (symbolisch mit 'X' gekennzeichnet) auftreten: Da der Ausgang eines XOR-Gatters von den Werten beider Eingänge abhängig ist, also keinen steuernden Wert besitzt, führt ein unbekannter Wert an einem der Eingänge unweigerlich auch zu einem unbekannten Wert am Ausgang. Es gibt vielerlei Gründe, warum unbekannte Werte in der Simulation auftreten können: Zum einen können nicht initialisierte Flip-Flops oder Latches dafür verantwortlich sein, dass mit unbekannten Werten simuliert werden muss. Zum anderen können aber auch Signale, die durch Domänen mit unterschiedlichem Takt laufen oder Signale von analogen Bauteilen, die nicht genau genug aufgelöst werden können als dass eine binäre Repräsentation möglich ist, zu unbekannten Werten führen [TWV+04]. Während des Produktionstests kann also an diesen Stellen eine '0' oder eine '1' auftreten, was sich allerdings nicht korrekt vorhersagen lassen kann. Hierdurch lässt sich nicht feststellen, ob ein Fehler aufgetreten ist oder nicht.

In der Literatur gibt es einige Ansätze, um mit unbekannten Werten umzugehen, die sich prinzipiell in

- X blockierende,
- X maskierende und
- X tolerante

Verfahren einteilen lassen.

Der erste Ansatz zielt darauf ab, zusätzliche Hardware einzubauen, um unbekannte Werte zu eliminieren oder zu **blockieren**. Diese DfT-Maßnahme wird *X-bounding* oder *X-blocking* genannt und verhindert, dass unbekannte Werte an die Prüfpfad-Zellen weitergegeben werden. Während des Entwurfs, werden potentielle Gefahren mittels eines sog. *scan design rule checkers* identifiziert und auf einen fixen Wert gesetzt [NPRK03] [PLR03]. Abbildung 2.15 zeigt eine einfache Darstellung dieser Idee.

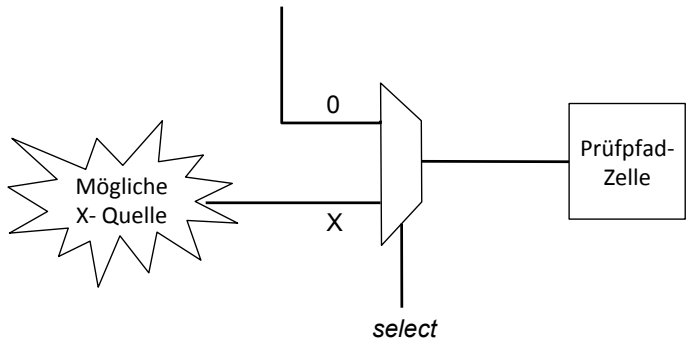


Abbildung 2.15: Vereinfachte Darstellung des X-bounding

Durch einen zusätzlichen Multiplexer, der durch das Steuersignal *select* angesteuert wird, ist es möglich, während des Tests das 'X' durch einen festen logischen Wert, in diesem Fall '0', zu ersetzen und an den Kompaktierer zu übergeben. X-blocking sorgt zwar dafür, dass keine unbekannten Werte übergeben werden, kann aber auch zu ungewolltem

Verlust der Fehlerabdeckung führen: Wenn z.B. ein Fehlverhalten nur durch den Teil der Schaltung, der eine mögliche X-Quelle ist, an die Ausgänge propagiert wird, wird dieser durch den Multiplexer geblockt. Darüber hinaus werden durch die Multiplexer zusätzliche Hardware und eine zusätzliche Verzögerung im Pfad eingebaut. Da in modernen komplexen Schaltungen in mehr als 25% der Test-Zyklen ein oder mehrere unbekannte Werte enthalten sein können, ist es sehr schwierig, diese mit zusätzlicher DfT-Hardware zu eliminieren [WWW06].

Andere Ansätze, die keine Modifikation der Schaltung beinhalten, nutzen die Idee der **Maskierung** und werden deshalb *X-Masking* genannt. Hierbei werden die unbekannten Werte direkt vor dem Kompaktierer maskiert. Abbildung 2.16 zeigt schematisch eine Maskierungslogik. Diese ist aus UND-Gattern aufgebaut und wird von einem Maskierungs-Controller gesteuert[3]. Dieser Controller gibt immer dann eine '1' an das entsprechende Gatter, wenn ein 'X' am Ausgang der Schaltung anliegt. Die Masken müssen in einem vorverarbeitenden Schritt (engl. *pre-processing*) vorberechnet und können dann in einer komprimierten Form auf dem Chip selber gespeichert werden. Die Dekompression kann auf mehrere Arten ablaufen und wird in den verschiedenen Arbeiten auf verschiedene Arten gelöst [WWP04] [HHLL05] [VM05] [RTWR05] [TWV+04].

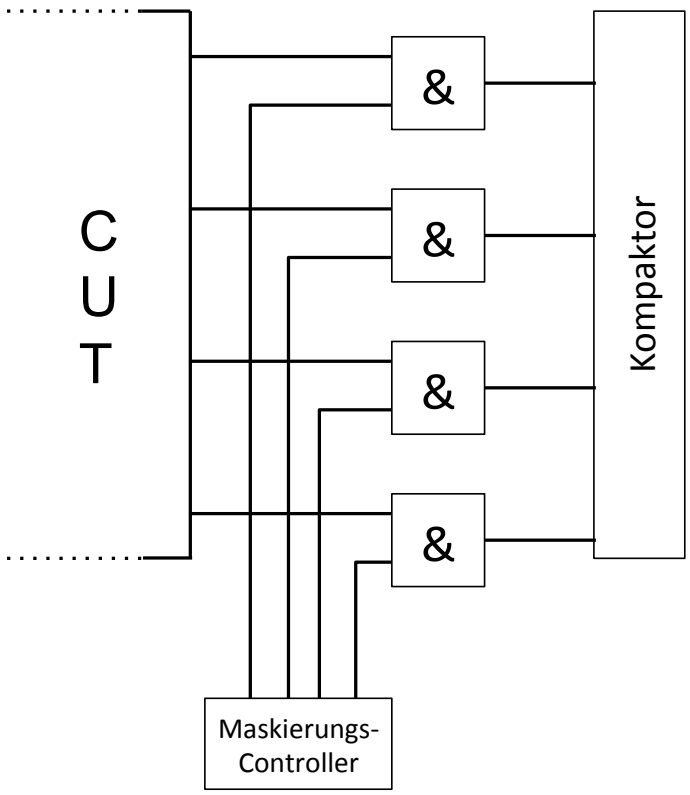


Abbildung 2.16: X-Masking [WWW06]

Hierbei ergibt sich allerdings ein Konflikt zwischen Speicheraufwand und Fehlerabdeckung: Je genauer die Masken um spezielle Bits zu maskieren, umso mehr Speicher wird

[3] Die Maskierungslogik kann genauso gut aus anderen Gattern aufgebaut sein. Wichtig ist, dass das Gatter einen steuernden Wert hat, um das 'X' auszumaskieren. Die Wahl der Gatter kann für die Dekompression von Bedeutung sein, je nachdem wie die Masken am effizientesten kodiert werden können.

für die Masken benötigt. Je ungenauer die Masken, desto höher ist die Gefahr, dass spezifizierte Bits, die evtl. Fehlerinformationen enthalten, ausmaskiert werden. Gerade für sicherheitskritische Anwendungen muss jedoch eine hohe Produktqualität gegeben sein.

Die dritte Gruppe der Kompaktierern, die unbekannte Werte verarbeiten können, heißen **X-tolerante** Kompaktierer, die die Testantworten in Anwesenheit von X-Werten kompaktieren, ohne diese vorher auszumaskieren. Einer der ersten X-toleranten Kompaktierer war das sogenannte *X-compact* [MK04]. Hierbei werden m Schaltungsausgänge mittels einer Logikschaltung bestehend aus XOR-Gattern in jedem Takt zu m_{komp} Bits kompaktiert. Die Logikschaltung wird dabei so entworfen, dass der Fehler garantiert erkannt wird, wenn er sich an einen, zwei oder einer ungeraden Anzahl von Schaltungsausgängen auswirkt (auch wenn gleichzeitig ein unbekannter Wert an einem beliebigen anderen Ausgang anliegt). Abbildung 2.17 zeigt beispielhaft eine solche Schaltung für $m = 8$ und $m_{komp} = 5$.

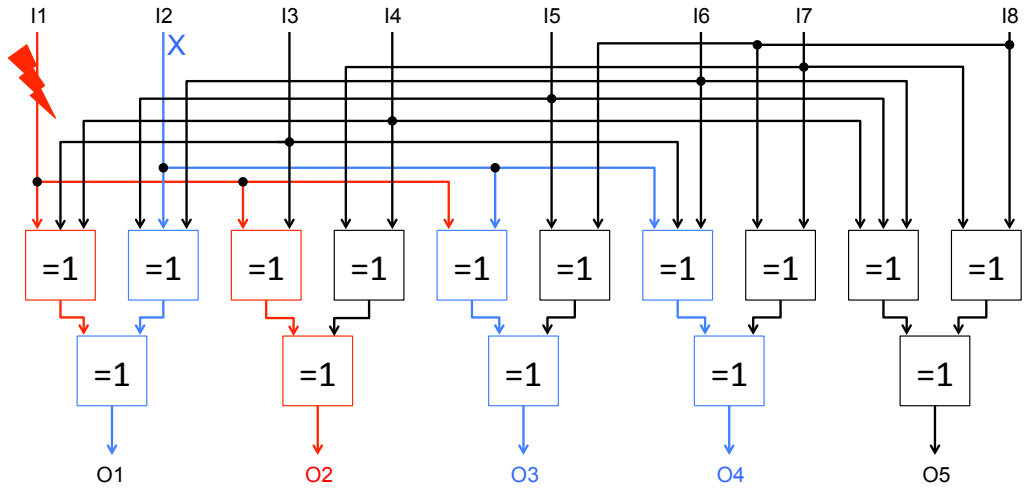


Abbildung 2.17: Beispielhafter X-Kompaktierer

Tritt ein Fehler in der Schaltung auf, der sich auf $I1$ auswirkt, wird dieses Fehlverhalten über die XOR-Gatter an die Kompaktiererausgänge $O1$, $O2$ und $O3$ propagiert. Tritt zur gleichen Zeit ein unbekannter Wert an einem beliebigen anderen Schaltungsausgang auf, beispielsweise an $I2$, kann das Fehlverhalten immer noch an mindestens einem Kompaktiererausgang, nämlich $O2$, erkannt werden.

Der Vorteil dieses Verfahrens ist, dass keine Änderungen in der Testmustererzeugung vorgenommen und nur zusätzliche Logik implementiert werden muss. Allerdings können Fehler, die an einer geraden Anzahl von Prüfpfadausgängen sichtbar sind (und nicht nur an zwei), nicht erkannt werden.

In [Tou07] wurde ein neuer X-toleranter Kompaktierer vorgestellt, der ein MISR benutzt und durch anschließende Berechnungen die unbekannten Werte herausrechnet, das *X-Canceling MISR*. Durch symbolische Simulation wird der Endzustand des MISRs, abhängig von den Bits, die aus der Schaltung in das MISR geschoben werden, bestimmt. Jedes MISR Bit wird somit als Linearkombination der Testantwort-Bits repräsentiert. Dies ist beispielhaft in Abbildung 2.18 gezeigt. Jedes X_i-Symbol steht für einen unbekannten Wert, jedes B_i Symbol ist ein bestimmter Wert. Ohne Beschränkung der Allgemeinheit

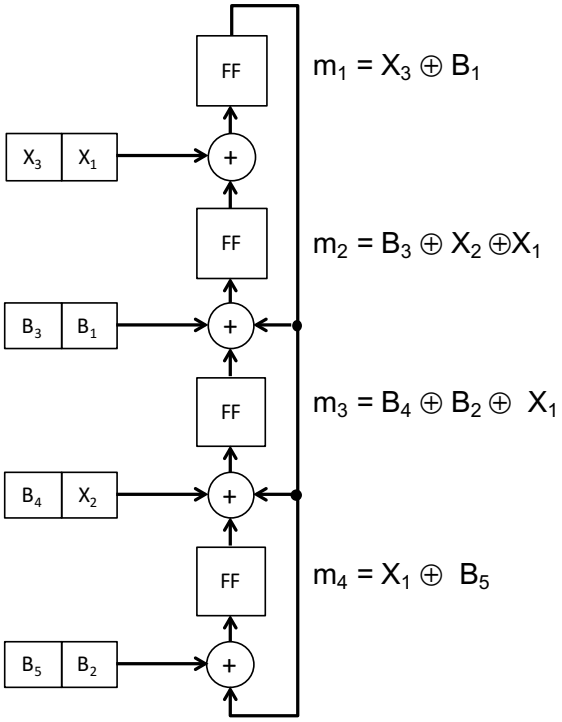


Abbildung 2.18: Beispiel MISR mit symbolischer Darstellung

können alle Bits, die kein 'X' sind, durch '0' ersetzt werden, so dass jede Gleichung nur noch von unbekannten Werten abhängig ist. Die Idee ist nun, durch geschickte Addition der einzelnen MISR-Bits die unbekannten Werte so zu überlagern, dass deren Einfluss maskiert wird.

Beispiel: Ausgehend vom Beispiel aus Abbildung 2.18 ergeben sich die vereinfachten Gleichungen, also die MISR-Bits nur in Abhängigkeit von unbekannten Werten, als

$$
\begin{aligned}
m_1 &= X_3 \\
m_2 &= X_2 \oplus X_1 \\
m_3 &= X_1 \\
m_4 &= X_1.
\end{aligned}
$$

Diese Abhängigkeiten können auch als Matrix dargestellt werden, wobei jede Zeile ein MISR-Bit und jede Spalte einen unbekannten Wert repräsentiert. Eine '1' in Zeile i und Spalte j bedeutet, dass MISR-Bit m_i ein X_j enthält. Die entsprechende Matrix wäre dann

$$
\begin{array}{c} \begin{matrix} X_1 & X_2 & X_3 \end{matrix} \\ \left[\begin{array}{ccc} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{array}\right] \begin{matrix} m_1 \\ m_2 \\ m_3 \\ m_4. \end{matrix} \end{array}
$$

Durch den Gauss-Jordan-Algorithmus [Cul97] können lineare Abhängigkeiten gefunden werden. Hierbei wird die Menge der linear unabhängigen Matrixreihen in die Einheitsmatrix überführt.

Für das Beispiel ergibt sich nach dem Gauss-Jordan-Verfahren folgende Matrix

$$\begin{array}{cccc} X_1 & X_2 & X_3 & \\ \left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{array}\right. & & & \left.\begin{array}{c} m_3 \\ m_2 \oplus m_3 \\ m_1 \\ m_3 \oplus m_4. \end{array}\right] \end{array}$$

Wie man sieht, ergibt sich hieraus in den ersten drei Zeilen die Einheitsmatrix und die unterste Gleichung ist unabhängig von einem X_i. Hierzu müssen nur die entsprechenden MISR Bits, die während des Gauss-Jordan-Verfahrens zur Zeilenaddition verwendet wurden, miteinander addiert werden. Gehen wir von der Gleichung

$$m_3 \oplus m_4$$

aus. Hieraus ergibt sich

$$\begin{aligned} m_3 \oplus m_4 = \quad & (B_4 \oplus B_2 \oplus X_1 \oplus X_1 \oplus B_5) \\ = \quad & (B_4 \oplus B_2 \oplus B_5) \end{aligned}$$

und somit ein X-freies Bit nur noch in Abhängigkeit von B_i. ■

In [Tou07] wurde weiterhin gezeigt, dass über 99% aller Fehler, die sich am MISR auswirken, erkannt werden, wenn mindestens sieben X-freie Bits erzeugt werden können. Hierbei werden die X-freien Kombinationen auf dem Chip selber berechnet, wozu zusätzliche Steuersignale vom Tester notwendig sind. Die Anzahl der unbekannten Werte, die im Allgemeinen von einem m-Bit breiten MISR verarbeitet werden können, ist abhängig von der Anzahl der X-freien Bits n_{Xfrei}, die man aus der Signatur extrahieren möchte und ist laut [Tou07] $m - n_{Xfrei}$.

Nachteilig bei diesem Verfahren ist die Tatsache, dass nur eine geringe Anzahl von unbekannten Werten effizient kompaktiert werden kann und die Steuersignale für die Berechnung der X-freien Kombinationen auf dem Chip vorgehalten oder vom externen Testautomaten übertragen werden muss. Experimente haben gezeigt, dass sich bei einer X-Rate von 1% und einer gewünschten Erkennungswahrscheinlichkeit von über 99% nur 8- bis 14-fache Kompressionsraten erreichen lassen, die mit zunehmender X-Rate stark abnehmen.

Bei allen Verfahren, die unbekannte Werte verarbeiten können, ist gleich, dass die Flexibilität sehr gering ist, da die Steuersignale vom Tester zur Verfügung gestellt werden und geschickt kodiert bzw. dekodiert werden müssen. Weiterhin können nur geringe X-Raten (0,1 % - 3 %) effizient verarbeitet werden.

2.5 Methoden der Logikdiagnose

Logikdiagnose hat, im Gegensatz zum Logiktest, das Ziel, einen Defekt nicht nur zu erkennen, sondern diesen auch auf dem Chip zu lokalisieren. Dabei kann die Diagnose auf unterschiedlichen Entwurfsebenen ansetzen. In dieser Arbeit liegt der Fokus auf der Diagnose auf Gatterebene, bei der sich ein Defekt als Fehler in der Kombinatorik einer Schaltung auswirkt. Hierbei werden zwei unterschiedliche Ansätze verfolgt: Zum einen das „Ursache-Wirkungs-Prinzip“ (engl. *cause-effect*), zum anderen das „Wirkungs-Ursache-Prinzip“ (engl. *effect-cause*) [WWW06]. Bei einer Diagnose nach dem Ursachen-Wirkungs-Prinzip wird zunächst der Defekt auf einen Fehler-Typ beschränkt und mittels Fehlersimulation das Verhalten dieses Fehlers bestimmt. Oft wird ein Fehlerverzeichnis angelegt und während des Tests kann die fehlerhafte Ausgabe einem Fehler zugeordnet werden. Problematisch an diesem Ansatz ist die Komplexität der vielen Fehlersimulationen und des Fehlerverzeichnisses sowie die Tatsache, dass nur Haftfehler eindeutig zugewiesen werden können. Zwar wurde in [WR00] ein Ansatz aufgezeigt, der auch komplexere Fehlermodelle, wie Brückenfehler, behandeln kann, allerdings kann im Allgemeinen eine nicht so hohe Genauigkeit erreicht werden [WWW06].

Im Gegensatz dazu wird bei der Diagnose nach dem Wirkungs-Ursachen-Prinzip von der Wirkung des Fehlers ausgegangen und mit Hilfe von Boolescher Algebra der Fehlerort bestimmt. Die Wirkungs-Ursache-Diagnose wird oft dazu benutzt, die Fehlermenge der Ursache-Wirkungs-Analyse zu minimieren [ANV06] [DPB06].

2.5.1 POINTER-Algorithmus

In der Literatur gibt es eine Reihe von Ansätzen zur Logikdiagnose. Hier soll der POINTER-Algorithmus (engl. *Partially Overlapping Impact couNTER*) [HW09] genauer vorgestellt werden, da er die Grundlage für das in dieser Arbeit verwendete Diagnoseverfahren ist.

Der POINTER-Algorithmus ist ein simulationsbasierter Wirkungs-Ursachen-Diagnosealgorithmus, der adaptiv und unabhängig vom Fehlermodell fehlerhafte Ausgaben ohne Fehlerverzeichnis einem bestimmten Fehlerort zuordnet. Die Grundidee ist hierbei, wie bei der Ursache-Wirkungs-Diagnose, die Testmuster für ein bestimmtes Fehlermodell zu simulieren. Jedoch werden dann die beobachteten Testantworten mit den simulierten Antworten verglichen und mit Hilfe statistischer Abschätzungen der Fehlerort bestimmt, ohne dass ein bestimmtes Fehlermodell für die Simulation genutzt wurde. Die Idee der statistischen Analyse ist eine Erweiterung des SLAT-Ansatzes (engl. *Single Location at a Time*) [BHHS01] und basiert auf dem bereits beschriebenen bedingten Haftfehlermodell.

Sei $FM(\phi)$ die Schaltung mit injiziertem Haftfehler ϕ. Für jedes Testmuster $\psi \in \Psi$ wird eine Evidenz

$$ev(\phi, \psi) = (\triangle\sigma_\psi, \triangle\iota_\psi, \triangle\tau_\psi, \triangle\gamma_\psi) \tag{16}$$

bestimmt, definiert als Tupel natürlicher Zahlen $\triangle\sigma_\psi, \triangle\iota_\psi, \triangle\tau_\psi, \triangle\gamma_\psi \in \mathbb{N}$. Anhand dieser Werte wird bestimmt, wie gut ein Haftfehler ϕ die Ausgabe der zu diagnostizierten Schaltung (engl. *Device under Diagnosis*, DUD) erklärt.

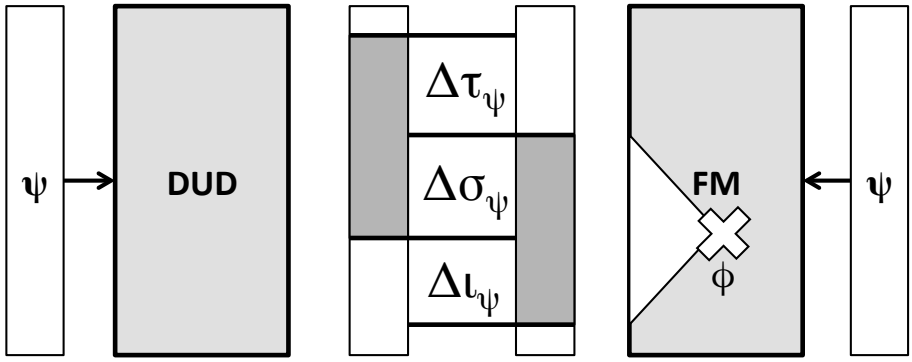


Abbildung 2.19: POINTER Evidenz

Hierbei beschreiben die Elemente des Tupels jeweils die Anzahl der Ausgänge, die für das Testmuster ψ

- $\triangle\sigma_\psi$: sowohl am DUD, als auch in der Fehlersimulation für den Fehler ϕ, fehlerhaft sind,
- $\triangle\iota_\psi$: nur in der Fehlersimulation für den Fehler ϕ fehlerhaft sind, am DUD jedoch fehlerfrei,
- $\triangle\tau_\psi$: am DUD zwar fehlerhaft sind, in der Fehlersimulation für den Fehler ϕ jedoch fehlerfrei sind,

siehe Abbildung 2.19. Der Parameter $\triangle\gamma_\psi$ ist das Minimum von $\triangle\sigma_\psi$ und $\triangle\iota_\psi$. Die Evidenz eines Fehlers ϕ und einer Testmustermenge Ψ ist somit

$$ev(\phi, \Psi) = (\sigma_\Psi, \iota_\Psi, \tau_\Psi, \gamma_\Psi) \tag{17}$$

mit

$$\sigma_\Psi = \sum_{\psi\in\Psi} \triangle\sigma_\psi \text{ , } \iota_\Psi = \sum_{\psi\in\Psi} \triangle\iota_\psi$$
$$\tau_\Psi = \sum_{\psi\in\Psi} \triangle\tau_\psi \text{ und } \gamma_\Psi = \sum_{\psi\in\Psi} \triangle\gamma_\psi \text{ .}$$

Die Fehler werden an Hand ihrer Evidenz so in eine Rangordnung gebracht, dass derjenige Fehler, der am höchsten einsortiert wird, derjenige ist, der am besten das Fehlverhalten des DUD beschreibt. Die Rangordnung $rk(\phi)$ eines jeden Fehlers ϕ mit $ev(\phi, \Psi) = (\sigma_\Psi, \iota_\Psi, \tau_\Psi, \gamma_\Psi)$ kann anhand der folgenden Regeln mit der Rangordnung $rk(\phi')$ eines Fehler ϕ' verglichen werden:

$$rk(\phi) > rk(\phi') \quad \begin{cases} \gamma_\Psi^\phi > \gamma_\Psi^{\phi'} \text{ oder} \\ (\gamma_\Psi^\phi = \gamma_\Psi^{\phi'}) \wedge (\sigma_\Psi^\phi < \sigma_\Psi^{\phi'}) \text{ oder} \\ (\gamma_\Psi^\phi = \gamma_\Psi^{\phi'}) \wedge (\sigma_\Psi^\phi = \sigma_\Psi^{\phi'}) \wedge (\iota_\Psi^\phi > \iota_\Psi^{\phi'}). \end{cases} \tag{18}$$

Zunächst werden die Fehler entsprechend ihrem Parameter γ in eine Rangfolge gebracht, da je höher γ ist, desto größer ist die Wahrscheinlichkeit für einen bedingten Haftfehler an dieser Stelle. Für einen Haftfehler ϕ gilt $\iota^{\phi} = 0, \tau^{\phi} = 0$ und $\sigma^{\phi} > 0$ und ist das Maximum für alle Fehler. Gilt $\gamma_{\Psi}^{\phi} = \gamma_{\Psi}^{\phi'}$, wird der Fehler anhand des Parameters σ sortiert, denn je kleiner σ desto wahrscheinlicher ist der Fehlerort von ϕ für das fehlerhafte Verhalten verantwortlich. Falls $\sigma_{\Psi}^{\phi} = \sigma_{\Psi}^{\phi'}$, wird der Parameter ι in die Analyse mit einbezogen. Durch ι können Informationen auch aus den fehlerfreien Antworten gezogen werden.

Mit Hilfe dieses Algorithmus ist es möglich, verschiedene Defekte durch Annahme eines Haftfehlers zu interpretieren und ihr Verhalten zu erklären. Experimente in [HW07] [HW09] zeigen, dass sowohl für Benchmarkschaltungen als auch für industrielle Schaltungen[4] eine sehr gute diagnostische Auflösung erreicht wird.

2.5.2 Direkte Diagnose im Selbsttest

Im Selbsttest wird die Testinfrastruktur, vor allem die Testmustererzeugung und die Testantwortanalyse, auf dem Chip selber implementiert. In der STUMPS Architektur werden beispielsweise die Testantworten mit Hilfe eines MISRs in eine einzige Signatur kompaktiert (siehe Kapitel 2.4). Ist die Signatur fehlerhaft, hat der Chip den Test nicht bestanden. Schwieriger gestaltet sich dann aber die Diagnose, da aufgrund der hohen Kompaktierungsrate und des Informationsverlustes deutlich weniger Informationen zur Verfügung stehen. Dies erschwert die Identifizierung der Fehlerstelle enorm.

Wird auf dem Chip, analog zum Selbsttest-Ansatz, zusätzliche Hardware implementiert, um neben dem Test die Diagnose zu unterstützen, spricht man von eingebetteter Diagnose (engl. *Built-In Self-Diagnosis, BISD*). Hierbei wird grundsätzlich zwischen *direkter* und *indirekter Diagnose* unterschieden. Während bei der indirekten Diagnose die Prüfpfadelemente, an denen sich das Fehlverhalten ausgewirkt hat, identifiziert und analysiert werden, um dann eine einfache Logikdiagnose durchzuführen [BO02] [RBO03] [LGM04] [LCG02] [PJ72], wird bei der direkten Diagnose direkt aus der Signatur der Fehlerort bestimmt. Der Nachteil bei der indirekten Diagnose ist die Tatsache, dass mehrere Testdurchläufe benötigt werden. Auf der anderen Seite werden bei der direkten Diagnose oft nur einfache Haftfehler betrachtet und die Verfahren sind nicht mit STUMPS kompatibel [GWV$^+$04] [EW10] [WWPM02].

In [CSR$^+$06] wurde ein direkter Diagnosealgorithmus ausgearbeitet, der im Fehlerfall den Test wiederholt und für jedes Testmuster eine eigene unabhängige Signatur bestimmt. Für jedes MISR-Bit wird eine Fehlerfunktion berechnet, die angibt, welches Prüfpfadelement für ein Fehlverhalten an diesem MISR-Bit verantwortlich sein kann. Mit Hilfe dieser Information können dann für jede fehlerhafte Signatur die möglichen Prüfpfadelemente bestimmt werden. Mit Hilfe des SLAT-Ansatzes und einer Fehlersimulation wird dann der mögliche Fehlerort bestimmt. In Kapitel 4 werden im Stand der Technik noch Erweiterungen dieses Ansatzes erläutert, so dass Diagnose effektiver im Selbsttest eingesetzt werden kann.

[4] Die Experimente-Kapitel geben eine Übersicht über die Unterschiede der Schaltungen.

2.6 Early-Life Fehler: Neue Herausforderungen für Test und Diagnose

Frühausfälle sind für Chiphersteller sehr kritisch, da sie sich durch den Produktionstest nur sehr aufwendig vorhersehen lassen. Sie lassen sich oft auf produktionsbedingte Schwachstellen im System zurückführen [HM93] und werden „Early-Life Fehler“ (*ELF*, aus dem engl. *early-life failure*) genannt. Es ist wichtig, diese Schwachstellen schon möglichst während des Produktionstests zu erkennen um spätere Ausfälle, besonders bei sicherheitskritischen Anwendungen, zu vermeiden. In diesem Kapitel werden zwei verschiedene ELF-Arten beschrieben, die in der weiteren Arbeit schwerpunktmäßig behandelt werden.

2.6.1 Intermittierende Fehler

Ein nicht-permanentes Fehlverhalten in einer Schaltung kann auf einen intermittierenden Fehler hindeuten. Charakteristisch für einen intermittierenden Fehler ist zum einen die sehr hohe Fehlerrate, zum anderen, dass er an nur einem bestimmten Ort oder in dessen unmittelbarer Nähe auftritt. Auch wenn es sich nicht um einen permanenten Fehler handelt, kann er zu einem Fehlverhalten der Schaltung führen, ein Indikator für einen ELF sein oder die Robustheit der Schaltung beeinträchtigen. Die Aktivierungsbedingung eines intermittierenden Fehlers ist deterministisch, aber sehr komplex, und lässt sich in der Praxis nur sehr schwer reproduzieren. Bei der Modellierung eines solchen Fehlers wird deswegen ein probabilistischer Aktivierungsfaktor hinzugefügt, um der komplexen Aktivierungsbedingung zu genügen (in Kapitel 3.1.2 wird auf die Modellierung intermittierender Fehler genauer eingegangen).

Die Gründe für intermittierende Fehler können vielfältig sein, Abbildung 2.20 zeigt eine Übersicht.

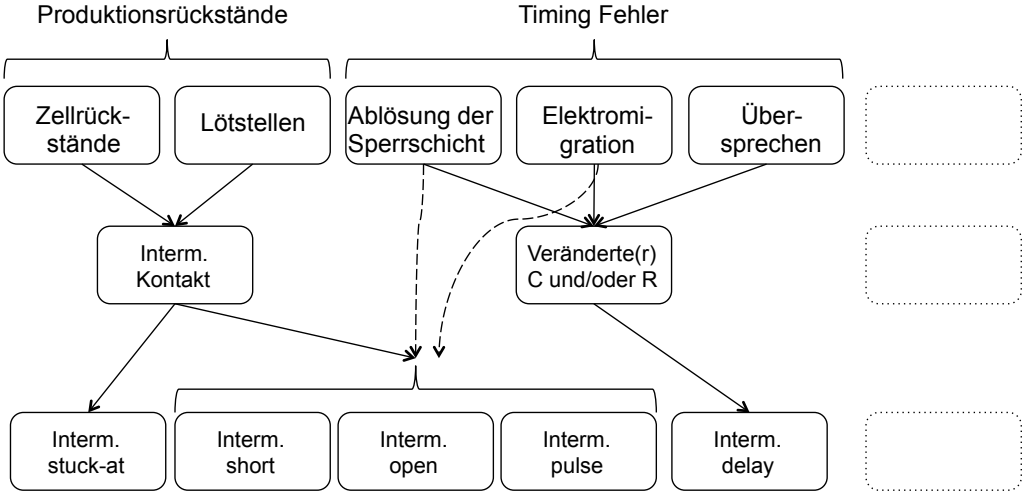


Abbildung 2.20: Ausgewählte Ursachen und Fehlermodelle von intermittierendem Verhalten [GSB+08]

Produktionsrückstände, die beispielsweise während der Beschichtung oder der Einbringung der Dotierung, wie bereits in Kapitel 2.2 beschrieben, auf dem Wafer zurückgeblieben sind, können zu einem intermittierenden Kontakt führen. In Speicher- aber auch in Logikelementen kann dies zu einem intermittierenden Wechsel zwischen zwei logischen Werten führen, so dass diese Art des Fehlers „intermittierender stuck-at“ genannt wird [GSB+08]. Effekte wie Leitungsübersprechen oder Elektromigration, die zunehmend aufgrund der kleiner werdenden Strukturgrößen auftreten, können auch zu einem intermittierenden Verzögerungsfehler führen, da hierdurch der Widerstand bzw. die Kapazität verändert wird [Con07]. Aber auch ein nicht ausreichend dimensioniertes Versorgungsnetz (engl. *Power-Grid*) kann zu einem intermittierenden Verhalten führen, wenn zu viele Transistoren auf einmal von 0 auf 1 schalten und es zu einem Elektronenfluss aufgrund der Verbindung von Source zu Drain kommt. Ein solcher Fehler ist ein hochfrequenter Spannungsabfall (engl. *High-Frequency Power-Droop,HFPD*) [PCKB07] und ist von vielen Faktoren abhängig, siehe Kapitel 3.1.2.

2.6.2 Kleine Verzögerungsfehler

Kleine Verzögerungsfehler (engl. *Small Delay Defects*, *SDDs*) können ebenfalls auf Schwachstellen im System hinweisen, die sich zu einem ELF entwickeln können. Hierbei handelt es sich um Gatterverzögerungsfehler, deren Verzögerungszeit d_g typischerweise deutlich kleiner als die Taktperiode ist. Dies führt zu Problemen, wenn diese Verzögerung für die gegebenen Testmuster nicht zu einer negativen Pufferzeit an einem der Schaltungsausgänge führt (siehe Kapitel 2.2.1). Gibt es allgemein keine Eingangsbelegung, welche die Verzögerung an einem der Ausgänge sichtbar macht, spricht man von einem *versteckten kleinen Verzögerungsfehler*. SDDs deuten auf einen späteren Ausfall der Schaltung hin, denn diese Verzögerung kann sich im Laufe des Produktlebenszyklus' durch Alterung, äußere Störeinflüsse oder Kumulation mit anderen Effekten noch so erhöhen, dass sich ein dauerhaftes Fehlverhalten an den Schaltungsausgängen einstellt. Dies ist besonders für die moderneren Technologien mit Strukturgrößen von 45 nm und geringer relevant: Mit kleineren Strukturgrößen und höheren Betriebsfrequenzen sinkt die Pufferzeit der Pfade und die Schaltung wird noch anfälliger und schon geringe Änderungen im zeitlichen Verhalten können zu einem Ausfall führen [TPC12]. Weiterhin wurde in [TPC12] festgestellt, dass die durch Verzögerungsfehler bedingten Ausfälle meistens auf SDDs zurückzuführen sind. Fortgeschrittenere Werkzeuge zur Testmustergenerierung versuchen deshalb, besonders oft die langen Pfade zu sensibilisieren, so dass die Wahrscheinlichkeit erhöht wird, einen kleinen Verzögerungsfehler an den Ausgängen sichtbar zu machen. SDDs und deren Testverfahren werden in Kapitel 6 detaillierter beschrieben.

2.7 Zusammenfassung und Ausblick

In diesem Kapitel wurden die für diese Arbeit wichtigsten Grundlagen vorgestellt: So wurde zunächst gezeigt, dass die Verlässlichkeit stark von der technologischen Komplexität und von äußeren Einflüssen abhängig ist und es somit wichtig ist, Fehler und auch Schwachstellen im System, die zu einem Frühausfall führen können, durch effiziente Test- und Diagnoseverfahren zu erkennen. Hierzu wurden dann die Grundlagen für Test und Diagnose erläutert. Bedingt durch die steigende Komplexität, wird auch mehr Test- und Diagnoseinfrastruktur auf dem Chip selber implementiert. Allerdings müssen dann Probleme, wie unbekannte Werte oder Diagnose, an Hand von Signaturen gelöst werden. Hiernach wurden dann die Grundlagen der Early-Life Fehler beschrieben und ein besonderes Augenmerk auf kleine Verzögerungsfehler und intermittierende Fehler gelegt, die in dieser Arbeit behandelt werden sollen.

In den nächsten Kapiteln werden die in dieser Arbeit entwickelten Verfahren vorgestellt.

3 Robuster Selbsttest zur Erkennung von nicht-permanenten Fehlern

Heutige Schaltungen sind, wie in Kapitel 2 bereits beschrieben, immer anfälliger gegenüber äußeren Einflüssen und Parameterschwankungen. Deshalb werden sie robust konzipiert, wodurch allerdings, gerade im Hinblick auf sicherheitskritische Anwendungen, stets ein Konflikt zwischen Produktqualität und Ausbeute entsteht: Auf der einen Seite muss der Test auf Strukturebene ansetzen, um Fehler trotz der eingebauten Redundanz zu erkennen und eine hohe Produktqualität zu gewährleisten, auf der anderen Seite können transiente Fehler zu unnötigen Ausbeuteverlusten führen. Zusätzlich möchte man auch Schwachstellen im System erkennen, die sich z.B. als intermittierende Fehler oder kleine Verzögerungsfehler (SDDs) auswirken, um spätere Probleme während des Produktlebenszyklus' zu minimieren. Allerdings müssen diese sehr genau von unkritischen transienten Fehlern, wie in Kapitel 3.1 beschrieben, unterschieden werden, um die Produktqualität nicht zu Lasten großer Ausbeuteverluste zu maximieren. Aus diesem Grund wird in dieser Arbeit ein integrierter Charakterisierungsprozess vorgestellt, der mit minimalem zusätzlichen Test- und Hardwareaufwand sowohl die Produktqualität maximiert als auch die Ausbeuteverluste minimiert.

Hierzu wird in diesem Kapitel zunächst der aus [AHHW08b] [AHHW08a] bekannte *Selbsttest mit Rücksetzpunkten* erweitert. Dieser Test kann dazu eingesetzt werden, zwischen permanenten und nicht-permanenten Fehlern zu unterscheiden, benötigt hierzu allerdings zusätzlichen Hardwareaufwand. Um jedoch effektiv die weiteren Schritte zur Fehlercharakterisierung durchzuführen, wird dieser Hardwareaufwand durch die in dieser Arbeit entwickelte extreme Kompaktierung minimiert. Abbildung 3.1 zeigt nochmal den bereits bekannten Gesamtablauf.

Der Charakterisierungsprozess beginnt mit dem in diesem Kapitel erweiterten Selbsttest mit Rücksetzpunkten. Ist die Schaltung fehlerfrei, sind keine weiteren Schritte notwendig. Im Fehlerfall kann der Selbsttest zwischen permanenten und nicht-permanenten Fehlern unterscheiden. Die Vorteile vom Selbsttest mit Rücksetzpunkten sind zum einen der geringe Hardwareaufwand der Selbsttest-Implementierung, zum anderen die geringe Testzeit: So benötigt der Test im fehlerfreien Fall nicht länger als ein Standard-Test, im Fehlerfall wird durch gezielte Wiederholung einzelner Sitzungen die Testzeit und bei gleichbleibender Produktqualität die Ausbeuteverluste minimiert.

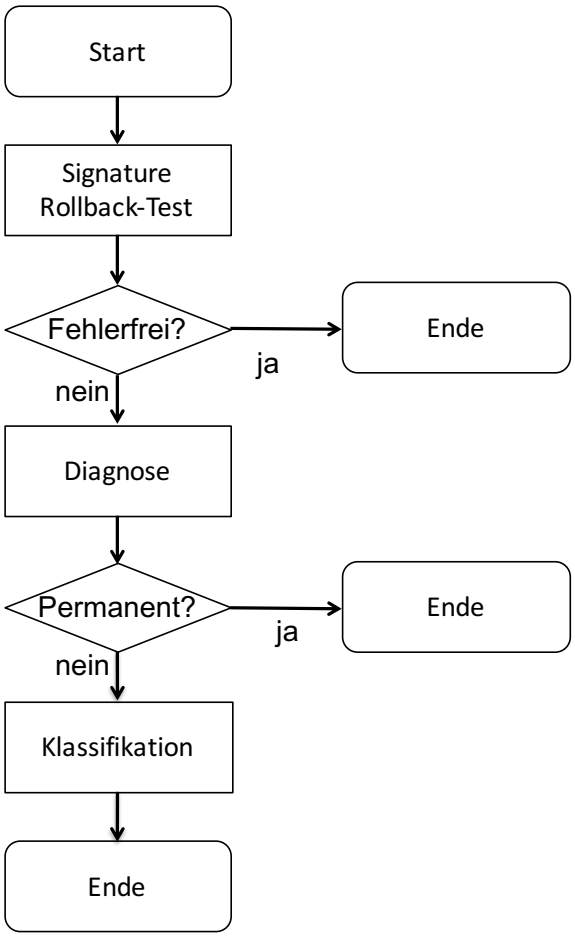


Abbildung 3.1: Charakterisierungsablauf

Im Folgenden wird zunächst der Stand der Technik von robusten Entwürfen und der Selbsttest mit Rücksetzpunkten beschrieben. Um den Hardwareaufwand für den Test zu minimieren, wird in Kapitel 3.3 die in dieser Arbeit entwickelte extreme Kompaktierung vorgestellt, die, was die Experimente zeigen, keine Auswirkungen auf die Fehlerabdeckung hat.

3.1 Stand der Technik: Test robuster Schaltungen

In Kapitel 2 wurden permanente und intermittierende Fehler betrachtet. Allerdings können auch transiente Fehler auftreten, die zu Signaleinbrüchen oder Bitflips führen: Waren durch radioaktive oder kosmische Strahlung injizierte Soft-Errors vor einigen Jahren lediglich für die Zuverlässigkeit von Speicherbereichen oder nur in großen Höhen und für die Luft- und Raumfahrt sowie Medizintechnik relevant, treten diese nun vermehrt auch in Logikelementen und in geringeren Höhen auf. Denn je kleiner die Strukturgrößen, desto größer auch die Anfälligkeit gegen äußere Störeinflüsse, wie zum Beispiel radioaktive oder kosmische Strahlung: Während die Einbringung von Alpha-Teilchen durch Radioaktivität durch eine geeignete Materialauswahl der Schaltung als auch des Gehäuses stark verringert werden kann, werden bei kosmischer Strahlung durch Wechselwirkung der Pro-

tonen und Neutronen mit den Silizium-Atomen Alpha-Teilchen generiert und es entstehen somit kurzzeitig Elektronen-Loch-Paare im Substrat [Juh03]. Je dünner und kleiner die Wannen, desto größer ist der Einfluss der so injizierten Elektronen-Loch-Paare auf den Stromfluss. Dies kann im schlimmsten Fall zu einer kurzfristigen Fehlfunktion des Transistors führen. Abbildung 3.2 zeigt diesen Effekt: Durch Einbringung von Protonen und Alpha-Teilchen entstehen Elektronen-Loch-Paare, die bei einem NMOS Transistor mit Elektronen aus dem Valenzband gefüllt werden. Hierdurch werden die Elektronen im Valenzband beweglich und es kann zu einem ungewollten Stromfluss führen.

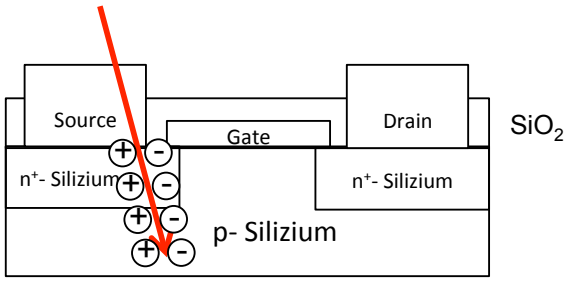


Abbildung 3.2: NMOS Transistor unter Bestrahlung

Ähnliche Effekte treten aber auch bei dynamischen und statischen Parameterschwankungen auf, denn das Verhalten einer integrierten Schaltung ist im Wesentlichen, neben der Schaltungstopologie, von Betriebsparametern, z.B. Versorgungsspannung und Temperatur, abhängig. Sowohl die Versorgungsspannung als auch die Temperatur haben direkten Einfluss auf die Beweglichkeit der Ladungsträger und beeinflussen das Schaltverhalten der Transistoren und somit auch die Laufzeit von Pfaden [KK05] [ABS+05]. Zusätzliche zeitliche Probleme gibt es auch an den Übergängen von Taktdomänen, wenn sich bedingt durch Synchronisationsfehler kurzzeitige, transiente, fehlerhafte Übergänge ergeben.

Durch robuste und selbst-adaptive Schaltungen können solche transienten Fehler zwar zum großen Teil toleriert werden, allerdings muss eine Schaltung weiterhin auf Strukturebene getestet werden, um permanente oder intermittierende Fehler zu erkennen. Hierdurch kann es allerdings passieren, dass Schaltungen aufgrund von nicht-kritischen transienten Fehlern den Test auf Strukturebene nicht bestehen, obwohl dieser während des Betriebs durch die Redundanz hätte toleriert werden können. In den nachfolgenden Kapiteln sollen zunächst verschiedene Fehlertoleranz-Techniken diskutiert werden, bevor dann ein robuster Selbsttest beschrieben wird, der auf Strukturebene ansetzt, aber nicht-permanente Fehler von permanenten Fehlern unterscheiden kann.

3.1.1 Robuster Entwurf

Von einem robusten Entwurf spricht man, wenn das System gewisse Fehler, die während des Betriebs auftreten, erkennt, lokalisiert und behebt. Abbildung 3.3 zeigt eine schematische Darstellung: Das System ist durch eine fehlertolerante Implementierung derart geschützt, dass der Benutzer anhand des Ein- /Ausgabeverhaltens keinen Fehler feststellen kann. An den Ausgaben liegen fehlerfreie Werte.

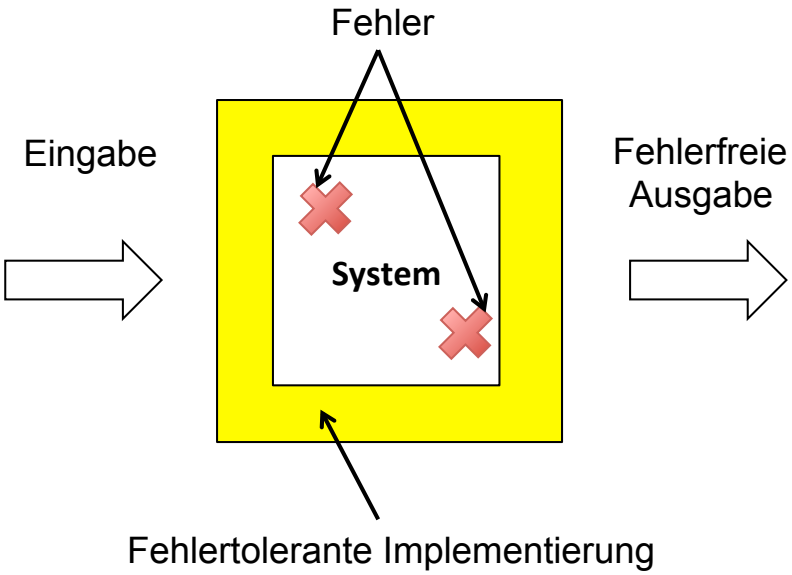


Abbildung 3.3: Schematische Darstellung eines robusten Entwurfs

Um das zu erreichen, muss das System redundant entworfen oder permanent überwacht und bei Auftritt eines Fehlers zurückgesetzt oder umkonfiguriert werden. Dabei beruht die Idee eines robusten Entwurfs auf *Redundanz*.

Redundanz zur Tolerierung von Hardware-Fehlern

Redundanz beschreibt die Eigenschaft, dass mehr Ressourcen für eine bestimmte Funktion benutzt werden als nötig. Es wird zwischen vier Arten der Redundanz unterschieden [KK07]: Hardware-, Informations-, Zeit- und Software-Redundanz, wobei Fehler in der Hardware vor allem von den ersten drei Arten der Redundanz behandelt werden. Allen gleich ist die Überlegung, dass durch Hinzufügen zusätzlicher Ressourcen Fehler erkannt und toleriert werden. Im weiteren Verlauf soll die Zeitredundanz genauer erläutert werden[5].

Zeitredundanz wird vor allem dazu eingesetzt, transiente Fehler während des Betriebs zu erkennen und zu korrigieren. Hierzu muss das System dauerhaft überwacht und auf Fehler überprüft werden. Die grundlegende Idee hierbei ist es, zu bestimmten Zeiten den Zustand des Systems abzuspeichern, die *Rücksetzpunkte* genannt werden, und im Falle eines Fehlers eine Neuberechnung ab dem letzten gültigen Zustand durchzuführen. Um den Fehler zu erkennen, werden z.B. über die Zeit versetzt Mehrfachberechnungen durchgeführt und die Ergebnisse miteinander verglichen. Zeitredundanz kann aber auch eingesetzt werden, wenn die Versorgungsspannung verringert wird um Energie einzusparen. Durch eine geringere Versorgungsspannung verringert sich die dynamische Leistungsaufnahme sehr stark, so dass diese Technik zur Energieeinsparung (engl. *Voltage Overscaling, VOS*) weit verbreitet ist [CB95] [CB95]. Allerdings ändert sich hierdurch das zeitliche Verhalten der Schaltung und es können Verzögerungsfehler entstehen. Adaptive Spannungsan-

[5] [KK07] gibt einen detaillierten Einblick in die weiteren Redundanzverfahren, die für diese Arbeit allerdings nicht weiter relevant sind.

passungen (*Adaptive Voltage Overscaling, AVOS*), wie sie beispielsweise in [KP11] beschrieben werden, implementieren kleine Fehlererkennungsblöcke, um, je nach Verhalten der Schaltung, die Versorgungsspannung wieder anzupassen. Hierbei wird die Versorgungsspannung zunächst heruntergefahren und bei Erkennung eines Fehlers wieder erhöht. Ein von den Firmen Intel und ARM entwickeltes Verfahren, welches in Prozessoren mit Pipeline-Technik eingesetzt wird, ist das sogenannte *Razor Register* [D+06] [E+04]. Hierbei wird Zeitredundanz verwendet, um die Verzögerungsfehler, die aufgrund der verringerten Versorgungsspannung aufgetreten sind, während des Betriebs zu erkennen und zu korrigieren. Hierzu wird jedes Pipeline-Flip-Flop um ein zusätzliches Schatten-Latch erweitert, siehe Abbildung 3.4.

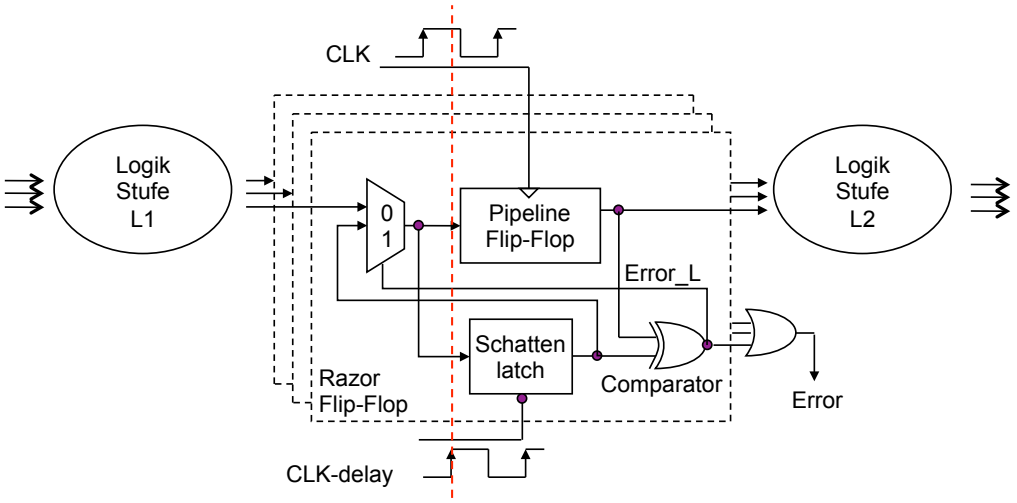


Abbildung 3.4: Architektur Razor-Register [E+04]

Die Schatten-Latches werden von einem zusätzlichen Takt-Signal *CLK-delay* getaktet, welches zu dem Original-Takt *CLK* etwas versetzt ist. Hierdurch können Verzögerungsfehler im Design während des Betriebs entdeckt werden: Während die Werte in den Flip-Flops zum geplanten Zeitpunkt t_0 die Werte der vorangegangenen Logikstufe aufnehmen, wird in den Latches das Ergebnis kurze Zeit später aufgenommen. Während der transparenten Phase wird überprüft, ob sich der Wert noch gegenüber dem im Flip-Flop abgelegten Wert verändert. Ist dies der Fall, hatte die vorangegangene Logikstufe noch nicht den finalen Wert zum Zeitpunkt t_0 berechnet und es ist ein Verzögerungsfehler aufgetreten. In diesem Fall wird der Wert im Flip-Flop durch den Wert im Latch überschrieben und das Error-Signal *Error* auf logisch '1' gesetzt. Die Pipeline wird angehalten und neu gestartet, so dass der richtige Wert an die nächste Logikstufe weitergegeben wird.

Diese dynamische Erkennung und Korrektur wird in modernen Prozessoren vor allem dazu eingesetzt, die Prozessoren möglichst energieeffizient zu betreiben: Ist der Prozessor nicht ausgelastet, wird während des Betriebs schrittweise die Versorgungsspannung verringert. Die hierdurch auftretenden Verzögerungsfehler werden mit dem Razor-Register überprüft und durch ein Rücksetzen korrigiert. Steigt die Fehlerrate zu stark an, kann auch die Versorgungsspannung dynamisch wieder erhöht werden und somit verringert sich auch die Fehlerrate. Durch die dynamische Anpassung der Versorgungsspannung sowie Erkennung und Korrektur der dadurch auftretenden Verzögerungsfehler lassen sich,

vor allem bei mobilen Anwendungen, unnötige Leistungsverluste minimieren. Experimente in [E+04] haben weiterhin gezeigt, dass sich selbst bei sehr hohen Fehlerraten aufwendiges Rücksetzen der Pipeline lohnt und sich noch große Energieeinsparung erreichen lassen. Das Diagramm in Abbildung 3.5 zeigt die dazu gehörenden Daten.

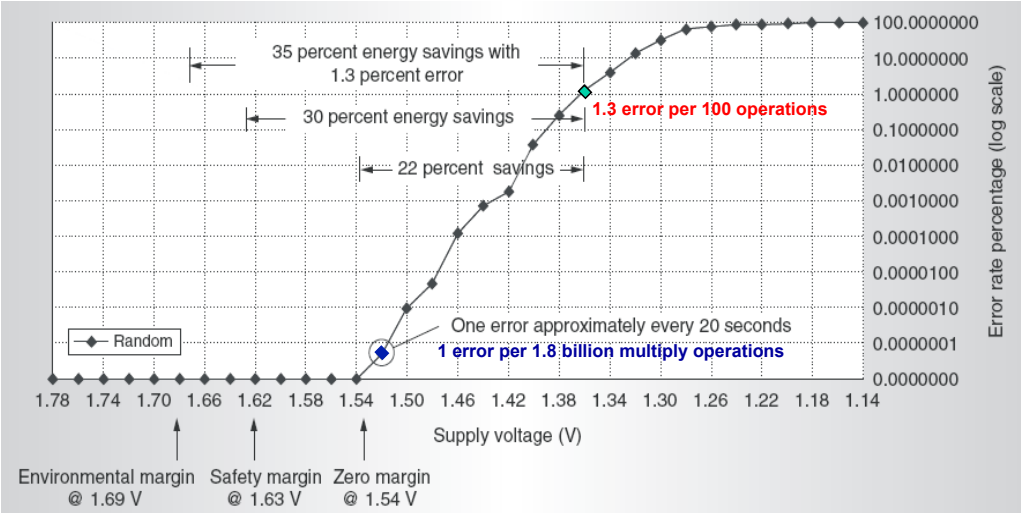


Abbildung 3.5: Zusammenhang Fehlerraten und Versorgungsspannung [E+04]

Es ist eine Kennlinie dargestellt, die für eine abnehmende Versorgungsspannung die gemessene Fehlerrate auf einer logarithmischen Skala zeigt. Zusätzlich sind noch drei Design-Punkte markiert: So zeigt der Punkt „*Zero margin*“ die Spannung an, bei der die Schaltung gerade noch fehlerfrei arbeitet. Der Punkt „*Safety margin*“ gibt an, bei welcher Spannung die Schaltung bei einer Frequenz von 90% von der Nominal-Taktung noch fehlerfrei funktioniert. Der Punkt „*Environmental margin*“ schließlich gibt an, wann die Schaltung bei einer 90%-igen Taktperiode und einer vom Hersteller angegebenen maximalen Temperatur (in diesem Fall 85°C) fehlerfrei funktioniert. Bei Anlegen von Zufallsmustern wurde hierbei herausgefunden, dass bei einer Fehlerrate von 1,3 Fehlern pro 100 Operationen noch 35% der Energie gegenüber der „Environmental margin“, 30% gegenüber der „Safety margin“ und 22% gegenüber der „Zero margin“ eingespart werden kann.

Eine weitere aus der Literatur bekannte Architektur, in die Zeitredundanz eingebracht wird, ist die GRAAL-Architektur [Nic07]. Für ein Latch-Design werden zwei überlappende Taktsignale implementiert, wodurch die Erkennung von transienten Fehlern unterstützt wird. Es ist keine zusätzliche Hardware nötig und durch Zwischenspeicherung der Zustände in Flip-Flops wird das Zurücksetzen bewerkstelligt.

Für Test- und Diagnoseverfahren stellt sich hierdurch allerdings eine Abwägung zwischen Produktqualität und Ausbeute ein: Wird eine Schaltung nur funktional getestet, wird also nur das reine Ein-/ Ausgabeverhalten beobachtet, können Defekte aufgrund der eingebauten Redundanz maskiert werden und das Testergebnis so verfälschen. Wird allerdings eine Schaltung auf Strukturebene getestet, kann das Testergebnis zu pessimistisch sein, wenn die Schaltung durch einen nicht-kritischen transienten Fehler aussortiert wird, der im Systembetrieb aber hätte toleriert werden können. Problematisch ist allerdings, dass kritische

intermittierende Fehler ein ähnliches Fehlverhalten haben wie nicht-kritische transiente Fehler. Zunächst wird deshalb auf die Modellierung von transienten und intermittierenden Fehlern eingegangen, bevor dann der aus [AHHW08b] [AHHW08a] bekannte Selbsttest mit Rücksetzpunkten vorgestellt wird.

3.1.2 Modellierung transienter und intermittierender Fehler

Sowohl transiente als auch intermittierende Fehler sind zeitlich abhängige Fehler, die unter bestimmten Voraussetzungen aktiviert werden. Da Standard-Fehlermodelle diese Besonderheiten nicht in Betracht ziehen können, ist die Grundlage für die Modellierung dieser Fehler das in Kapitel 2.2.1 definierte bedingte Haftfehlermodell [HW07]. Dieses beschreibt für einen Fehlerort v eine Bedingung $cond$, bei der dieser Fehlerort auf 0 oder 1 gesetzt wird, und wird als $cond\_0\_v$ bzw. $cond\_1\_v$ beschrieben.

Ein **transienter Fehler**, der aufgrund von äußeren Störeinflüssen oder dynamischen Parametervariationen auftritt, verändert einen logischen Wert am Fehlerort v für die Dauer eines Taktes bzw. eines Testmusters. Um diesen mit dem bedingten Haftfehlermodell zu modellieren, wird als Bedingung $cond$ ein spezifisches Testmuster ψ_i aus der Testmustersequenz $\Psi = (\psi_1, ..., \psi_n)$ gesetzt, bei dem der Knoten auf einen bestimmten Wert gesetzt wird. So beschreibt der Ausdruck $(\psi_i|\Psi)\_1\_v$ einen transienten Fehler, der den Fehlerort v auf 1 setzt, sobald das Testmuster ψ_i an den Schaltungseingängen anliegt.

Ein **intermittierender Fehler** tritt bei bestimmten sehr seltenen und komplexen Bedingungen auf, die ebenfalls mit dem bedingten Haftfehlermodell beschrieben werden können. In dieser Arbeit wird exemplarisch ein HFPD-Fehler, wie in Kapitel 2.6 beschrieben, angenommen. Hierbei wird ein Fehler, der an einem Gatterausgang v auftritt, dann aktiviert, wenn der Gatterausgang von '0' auf '1' (von '1' auf '0') schaltet **und gleichzeitig** eine bestimmte Anzahl anderer Gatter, die vom selben Teil des Energienetzes (engl. *power grid*) versorgt werden, ebenfalls von '0' auf '1' (von '1' auf '0') schalten [PCKB07]. Da die Aktivierung des Fehlers von einer Vielzahl von komplexen Bedingungen abhängig ist, wird zur Vereinfachung des Modells der Aktivierungsbedingung eine probabilistische Aktivierungswahrscheinlichkeit p_a hinzugefügt. Immer dann, wenn die Zufallsvariable η, die Werte zwischen 0 und 1 annehmen kann, unter dieser vorgegebenen Wahrscheinlichkeit liegt, wird der Fehler aktiviert. Mit Hilfe dieses Faktors kann der Anteil der anderen Aktivierungsbedingungen gewichtet werden.

Als Bedingung für einen bedingten Haftfehler entstehen die Ausdrücke

$$((v_{-1} = 0) \wedge (v = 1) \wedge (|\{w \in N(v) : (w_{-1} = 0) \wedge (w = 1)\}| \geq \tau)) \wedge \eta < p_a \quad (19)$$

und

$$((v_{-1} = 1) \wedge (v = 0) \wedge (|\{w \in N(v) : (w_{-1} = 1) \wedge (w = 0)\}| \geq \tau)) \wedge \eta < p_a \quad (20)$$

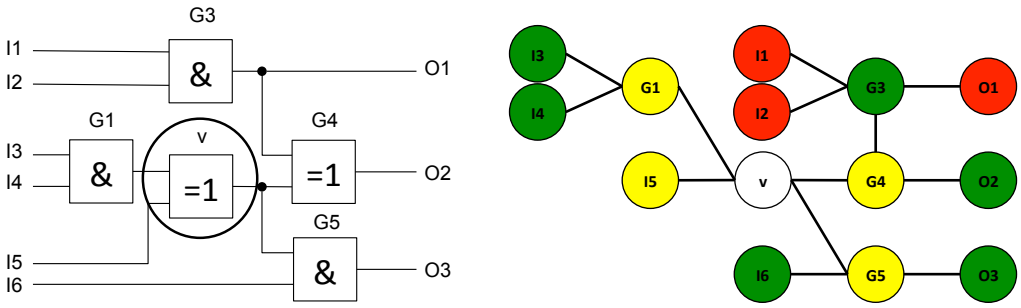
mit der *Nachbarschaft* $N(v)$ eines Fehlerorts v, einer Schranke τ und einem Nachbarknoten $w \in N(v)$.

Die Gatter, die vom selben Versorgungsnetz versorgt werden, werden durch die *Nachbarschaft* eines Gatters ausgedrückt. Für diese Bestimmung sind genauere Layout-Informationen nötig, bei der Fehlermodellierung wird deshalb von einer topologischen Nachbarschaft ausgegangen, wie in Abbildung 3.6 zu sehen ist. Die Netzliste der Schaltung wird hierbei als ungerichteter Graph $G = (V, E)$ dargestellt, der die Gatter als Knoten V und die Leitungen als Verbindungskanten E interpretiert. Die Nachbarschaft wird dann über einen *Radius* r definiert, wobei $r = 1$ *direkte Nachbarschaft* bedeutet und alle Knoten beinhaltet, die direkt mit v verbunden sind. Ein Radius $r = 2$ beinhaltet dann sowohl die direkten Nachbarn, als auch die direkten Nachbarn dieser usw., womit allgemein die Nachbarschaft des Knotens v mit Radius r als

$$N^r(v) := \{w \in V : \text{es existiert ein Pfad von } v \text{ nach } w \text{ mit maximaler Pfadlänge } r\} \tag{21}$$

beschrieben werden kann. Folgendes Beispiel soll dies verdeutlichen:

> **Beispiel:** Gegeben sei die Topologie der bereits bekannten Beispielschaltung aus Abbildung 3.6(a) mit den Gattern $\{G1, G3, G4, G5, v\}$ als ungerichteter Graph, vgl. Abbildung 3.6(b). Ausgehend vom Gatter v sind alle gelb markierten Gatter $(G1, I5, G4, G5)$ in der direkten Nachbarschaft $(r = 1)$. Die direkten Nachbarn dieser Gatter befinden sich mit den direkten Nachbarn zusammen in der Nachbarschaft mit dem Radius $r = 2$ von Gatter v und sind grün markiert $(I3, I4, I6, G3, O2, O3)$, wobei die Ein- und Ausgänge ebenfalls als Knoten dargestellt werden. Die Nachbarschaft mit dem Radius $r = 3$ enthält dann schließlich auch die restlichen Knoten des Graphen $(I1, I2, O1)$, in Abbildung 3.6(b) rot markiert.



(a) Beispielschaltung (b) Topologischer Graph

Abbildung 3.6: Nachbarschaftsdefinition

■

Der Schrankenwert τ im Fehlermodell gibt die Anzahl der Gatter w aus der Nachbarschaft an, die mindestens das gleiche Schaltungsverhalten haben müssen, um den Fehler zu aktivieren. Die Transition an Fehlerort v von '0' auf '1' (von '1' auf '0') wird dabei mit $(v_{-1} = 0) \wedge (v = 1)$ $((v_{-1} = 1) \wedge (v = 0))$ ausgedrückt.

3.2 Selbsttest mit Rücksetzpunkten

Um zunächst zwischen permanenten und nicht-permanenten Fehlern zu unterscheiden, wird in diesem Kapitel der aus [AHHW08b] [AHHW08a] bekannte Selbsttest mit Rücksetzpunkten beschrieben. Die grundlegende Idee ist hierbei die gezielte Wiederholung einzelner Testsitzungen, um zwischen permanenten und nicht-permanenten Fehlern zu unterscheiden. Der beschriebene Ansatz basiert auf der STUMPS Architektur (vgl. Abbildung 2.10), die bereits in Kapitel 2 beschrieben wurde. Der Testmustergenerator stellt pseudo-zufällige oder deterministische Testmuster zur Verfügung. Im MISR wird die Signatur berechnet und nach dem Test mit der berechneten Signatur verglichen.

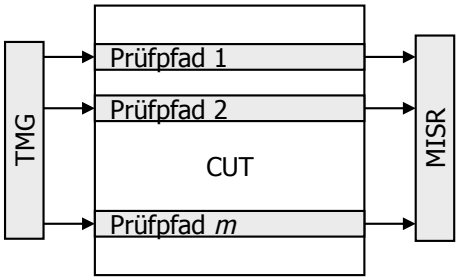


Abbildung 3.7: STUMPS Architektur

Die einfachste Möglichkeit, einen nicht-permanenten Fehler von einem permanenten Fehler zu unterscheiden, ist die Wiederholung des Tests und der Vergleich der beiden Testantworten. Allerdings würde dies eine Verdopplung der Testzeit bedeuten und die Ausbeuteverluste würden sich nicht verringern: Die Wahrscheinlichkeit ist hoch, dass bei Wiederholung des gesamten Tests wieder ein transienter Fehler auftritt, sofern die Fehlerrate hoch genug ist. Aus diesem Grund wird beim Selbsttest mit Rücksetzpunkten ein gegebener Test T mit M Testmustern in N Sitzungen $T_1, T_2, ..., T_N$ aufgeteilt. Die Zahl der Wiederholungen einer Sitzung wird mit einem benutzerspezifizierten Parameter W beschränkt: Wird eine Sitzung W-mal wiederholt, wird der Test abgebrochen und der Chip als defekt klassifiziert. Maßgebend hierfür sind verschiedene Gründe:

- Es liegt ein permanenter Fehler vor, der bei Aktivierung immer zu einer fehlerhaften Signatur führt.
- Transiente Fehler treten mit hoher Fehlerrate auf, so dass auch die Wiederholungen immer wieder fehlerhaft sind.
- Ein intermittierender Fehler mit einer komplexen Aktivierungsbedingung liegt vor.

Hierfür muss für jede Sitzung T_i die Soll-Signatur S_i bekannt sein und auf dem Chip bereitgestellt oder bei einem eingebetteten Test vom externen Testgerät nachgeladen werden. Der Test läuft dann wie in Abbildung 3.8 ab.

Zu Beginn einer jeden Sitzung T_i werden die Testmuster der Sitzungen erzeugt und die Testantworten mit Hilfe des MISRs zu einer Signatur kompaktiert. Nachdem die letzte Testantwort in das MISR geschoben wurde, wird die berechnete Signatur Q_i mit der Soll-Signatur S_i verglichen: Ist die Sitzung fehlerfrei, startet die nächste Sitzung. Kommt es zu

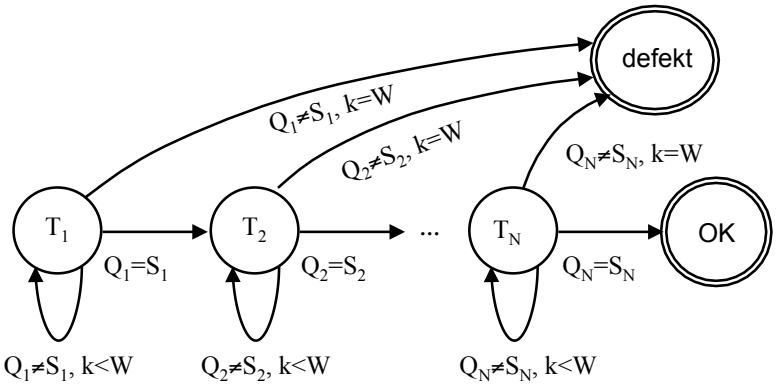


Abbildung 3.8: Testablauf mit Rücksetzpunkten [AHHW08b] [AHHW08a]

einer fehlerhaften Signatur, muss die Sitzung wiederholt werden. Sowohl der Zustand des LFSRs als auch der Zustand des MISRs müssen wiederhergestellt werden. Hierzu werden Backup-Register für LFSR und MISR benötigt, in die zu Beginn einer jeden Sitzung der jeweilige Zustand gesichert wird. Bei einer Wiederholung können so die für diese Sitzung benötigten Testmuster generiert werden und der Zustand der Signatur entspricht dem Zustand nach T_{i-1}. Da mit dem Herausschieben der letzten Testantwort bereits das erste Testmuster der neuen Sitzung in den Prüfpfad geschoben wird, wird ein zweites Backup-Register benötigt, um den Zustand vom LFSR der Sitzung T_{i+1} abzuspeichern. Abbildung 3.9 zeigt die entsprechende Architektur.

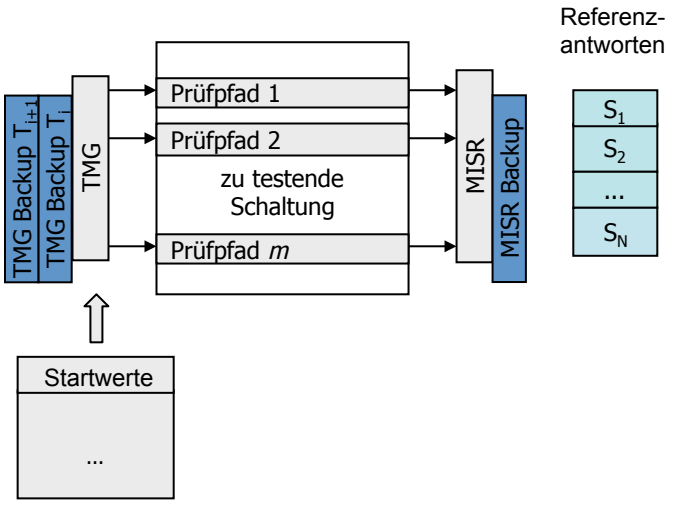


Abbildung 3.9: Architektur des Selbsttests mit Rücksetzpunkten [AHHW08b] [AHHW08a]

Der Testmustergenerator ist üblicherweise ein LFSR, das pseudo-zufällige Testmuster erzeugt, aber auch erlaubt, mit in einem Speicher vorgehaltenen Startwerten deterministische Testmuster zu verwenden (siehe hierzu Kapitel 2).

Die Effizienz des Testablaufs ist vor allem von den Parametern N und W abhängig. Die Anzahl der Wiederholungen W gibt an, bis zu welcher transienten Fehlerrate der Test noch brauchbare Resultate liefert. Spricht nichts gegen ein häufiges Wiederholen einer

Sitzung, kann W entsprechend groß gewählt werden. Eine große Anzahl von Sitzungen N führt zwar dazu, dass die Wahrscheinlichkeit für eine Wiederholung aufgrund eines Fehlers geringer wird und die Ausbeuteverluste vermieden werden, allerdings steigt auch mit wachsender Anzahl von Sitzungen der Hardwareaufwand, da für jede Sitzung die Soll-Signatur auf dem Chip vorgehalten werden muss. Dies lässt sich wie folgt belegen.

Im fehlerfreien Fall ergibt sich die Testzeit wie in Abbildung 3.10 angegeben. Zunächst wird in t_{load} Zeiteinheiten das erste Testmuster in den Prüfpfad geschoben, während dann die Sitzung t_{app} Zeiteinheiten benötigt. Da jede Sitzung genau $\lceil M/N \rceil$ Testmuster besitzt, benötigt jede der N Sitzungen $t_{app}(N)$ Zeiteinheiten. Da mit dem Herausschieben der letzten Testantwort einer jeden Sitzung bereits das erste Testmuster der neuen Sitzung in den Prüfpfad hineingeschoben werden kann, wird keine zusätzliche Zeit hierfür benötigt.

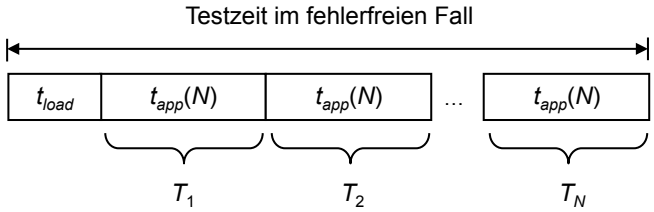


Abbildung 3.10: Testzeit im fehlerfreien Fall

Um die erwartete Testzeit mit Rücksetzen des Tests zu bestimmen, muss zunächst die Wahrscheinlichkeit p, dass ein transienter Fehler auftritt, abgeschätzt werden. Diese lässt sich nach [KK07] für eine konstante Fehlerrate λ mit

$$p = 1 - e^{-\lambda t_{app}(N)} \tag{22}$$

beschreiben. Dementsprechend kann die Wahrscheinlichkeit q, dass die Sitzung fehlerfrei durchläuft, beschrieben werden mit

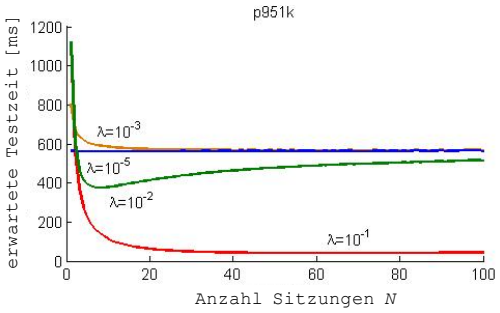
$$q = 1 - p = 1 - (1 - e^{-\lambda t_{app}(N)}) = e^{-\lambda t_{app}(N)}. \tag{23}$$

Die Wahrscheinlichkeit q^N gibt an, wie hoch die Wahrscheinlichkeit ist, dass der Test erfolgreich durchläuft. Dies kann als möglicher Ausbeutegewinn interpretiert werden. Die erwartete Gesamtzeit eines Tests ergibt sich nach [AHHW08b] [AHHW08a] als

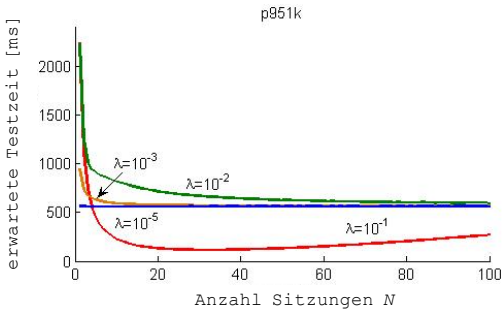
$$E\left(t_{total}(N)\right) = t_{load} + E\left(t_{sess}(N)\right) \cdot \frac{1 - q^N}{1 - q}. \tag{24}$$

$E\left(t_{sess}(N)\right)$ entspricht hierbei der erwarteten Testzeit einer einzelnen Sitzung.

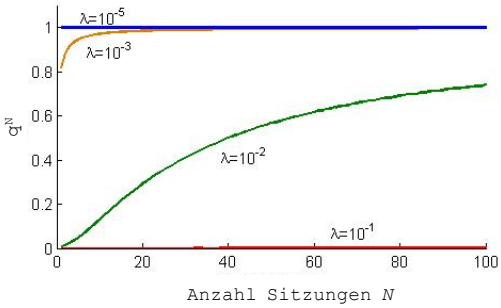
Für die Entwicklung der erwarteten Testzeit ergibt sich für verschiedene Fehlerraten der Verlauf, beispielhaft für die NXP-Schaltung p951k, aus Abbildung 3.11(a) und 3.11(b). Bereits für Fehlerraten von 10^{-3} und niedriger nähert sich die erwartete Testzeit der idealen Testzeit (von knapp 600 ms) bereits bei $N = 20$ an. Für höhere Fehlerraten werden deutlich mehr Sitzungen benötigt, für die Fehlerrate 10^{-1} sogar mehr als 100. Für $\lambda = 10^{-2}$ und $\lambda = 10^{-1}$ liegt die erwartete Testzeit teilweise unter der idealen Testzeit, da die Wahrscheinlichkeit sehr hoch ist, dass bei niedrigem N der Test vorzeitig abbricht und die Testzeit somit verkürzt.



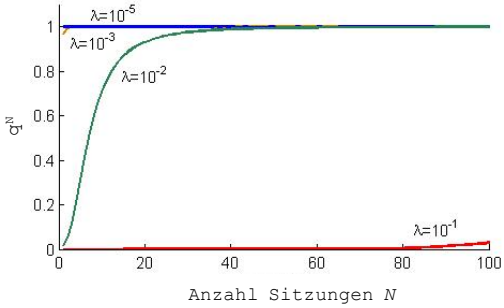
(a) Erwartete Testzeit bei W=2



(b) Erwartete Testzeit bei W=4



(c) Wsk. für fehlerfreien Test bei W=2



(d) Wsk. für fehlerfreien Test bei W=4

Abbildung 3.11: Entwicklung der Testzeit/ Wahrscheinlichkeit für p951k

Man erkennt an der Entwicklung der Wahrscheinlichkeit q^N aus Abbildungen 3.11(c) und 3.11(d), dass der Test alle N Sitzungen durchläuft. Hier sieht man, dass bei $W = 2$ und 100 Sitzungen die Wahrscheinlichkeit für den erfolgreichen Test bei Fehlerraten von 10^{-2} und 10^{-1} unter 95% liegt. Für $W = 4$ erreicht man dieses Ziel für $\lambda = 10^{-2}$ bereits bei ca. $N = 30$, für $\lambda = 10^{-1}$ reichen auch in diesem Fall 100 Sitzungen nicht aus.

Für die Bestimmung der optimalen Sitzungsanzahl lassen sich folgende einfachen Regeln ableiten:

- Allgemein gilt: Je höher die Sitzungsanzahl N, desto kürzer die Testzeit und desto geringer die Ausbeuteverluste.
- Für niedrigere Fehlerraten reicht schon eine geringe Anzahl an Testsitzungen aus, um die Ausbeuteverluste und die Testzeit zu minimieren (bei p951k bereits für $\lambda \leq 10^{-3}$).
- Für höhere Fehlerraten, wenn der Graph der Testzeit bereits bei geringem N einen Wendepunkt hat, hilft bereits eine Anpassung des Parameters W, um die Ausbeute und die Testzeit zu optimieren (z.B. $\lambda = 10^{-2}$ bei p951k).
- Für noch höhere Fehlerraten (bei p951k für $\lambda > 10^{-2}$) muss die Sitzungsanzahl erheblich erhöht werden.

Der entscheidende Parameter für die Vermeidung der Ausbeuteverluste und Verbesserung der Testzeit ist somit die Anzahl der Sitzungen N, wie auch den Experimenten in [AHHW08b] [AHHW08a] zu entnehmen ist. Problematisch an diesem Ansatz ist, dass die

zusätzliche Hardware, die für diesen Test-Ablauf notwendig ist, vor allem von der Sitzungsanzahl N abhängig ist: Je größer N, desto mehr Soll-Signaturen müssen auf dem Chip, oder im Falle eines eingebetteten Tests auf dem externen Testautomaten, vorgehalten werden. Deswegen wird im nächsten Kapitel ein Verfahren vorgestellt, diesen zusätzlichen Hardwareaufwand zu minimieren.

3.3 Extreme Kompaktierung

Das in dieser Arbeit entwickelte Verfahren der extremen Kompaktierung erlaubt es, die Testantworten so zu kompaktieren, dass bei gleichbleibender Produktqualität nicht die komplette Signatur einer Sitzung, sondern nur ein Teil dieser benötigt wird. Im Gegensatz zu anderen Kompaktierungstechniken, wie [EW10] oder [V$^+$06], sind hierbei keine speziellen Anpassungen des Tests erforderlich. Die in dieser Arbeit entworfene extreme Kompaktierung wurde in [ISH10b] [ISH10a] [Ind13] publiziert und für einen integrierten Test- und Diagnose-Ansatz in [CHIW11b] [CHIW11a] angepasst. Die grundlegende Idee ist hierbei, eine zweistufige Kompaktierung zu wählen, wobei in der ersten Stufe mit Hilfe eines MISR eine Signatur berechnet und diese Signatur dann im zweiten Schritt noch weiter räumlich kompaktiert wird. In [ISH10b] [ISH10a] [Ind13] wurde für die zweite räumliche Kompaktierung eine Kompaktierung mit Hilfe einer Paritätslogik entwickelt, während in [CHIW11b] [CHIW11a] das für den Test benötigte Backup-Register in ein zusätzliches MISR umgewandelt wird und nach einigen zusätzlichen Berechnungen nur ein Teil der Signatur benötigt wird. Beide Ansätze werden in den nachfolgenden Ausführungen detailliert beschrieben.

Räumliche Kompaktierung mit Hilfe einer Paritätslogik

Um den Hardwareaufwand zu reduzieren, der durch den Test mit Rücksetzpunkten generiert wird und in erster Linie auf die Speicherung der Zwischensignaturen zurückzuführen ist, kann als zusätzliche räumliche Kompaktierungsstufe eine Paritätslogik verwendet werden, um anstelle der kompletten k-Bit breiten Signaturen

$$S_i = (S_{i,0}, ..., S_{i,k-1}) \in GF(2)^k \tag{25}$$

nur noch wenige Paritätsbits zu speichern. Die Parität der Signatur S_i wird als

$$\pi(S_i) = \sum_{k=0}^{k-1} S_{i,k} \in GF(2)^k \tag{26}$$

berechnet. Die Paritätsberechnung wird für insgesamt L MISR-Zustände $Z_1, ..., Z_L$ durchgeführt, so dass ein *Paritätsfenster*

$$\Pi(S_i) = (\pi(Z_1), ..., \pi(Z_L)) \in GF(2)^L \tag{27}$$

entsteht. Die Paritäten werden am Ende einer Sitzung berechnet. Dieses Soll-Paritätsfenster $\Pi(S_i)$ einer Sitzung T_i wird dann mit dem berechneten Paritätsfenster $\Pi(Q_i)$ der Schaltung verglichen. Abbildung 3.12 veranschaulicht die Idee des Paritätsfensters. Anstatt der kompletten Signatur müssen nur noch L Paritätsbits pro Sitzung auf dem Chip vorgehalten werden.

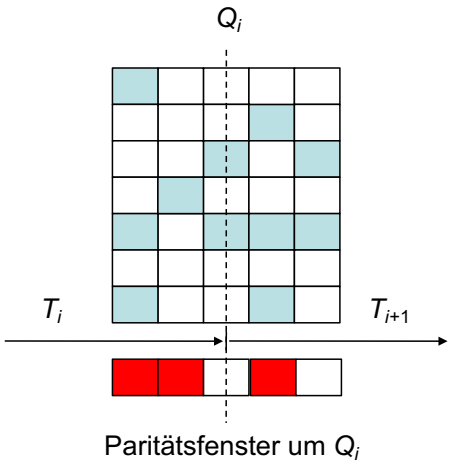


Abbildung 3.12: Paritätsfenster

Die Architektur ist in Abbildung 3.13 schematisch dargestellt. Die Testantworten werden mit einem MISR zunächst zeitlich und dann mit einer Paritätslogik räumlich kompaktiert. Das Paritätsfenster wird in einem L-Bit breiten Register abgelegt und mit den Soll-Paritäten $\Pi(S_i)$ verglichen.

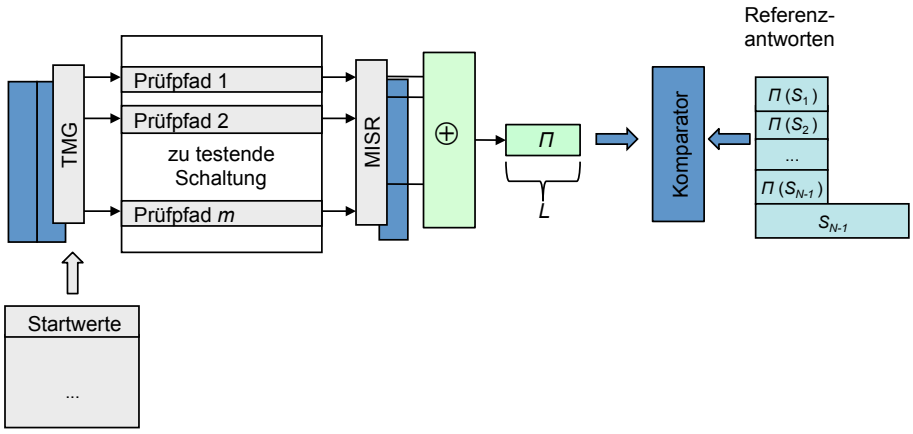


Abbildung 3.13: Architektur Selbsttest mit Rücksetzpunkten und extremer Kompaktierung (Paritätslogik)

Allerdings können durch die zusätzliche Kompaktierung Fehler ausmaskiert werden, obwohl sie in der Signatur erkannt worden wären. Aus diesem Grund wird für die letzte Sitzung die komplette Signatur abgespeichert. Da es sich hierbei um die Signatur des kompletten Tests handelt, wird die Produktqualität durch die zusätzliche Kompaktierung gegenüber einem Standard-Test nicht negativ beeinflusst. Allerdings können durch die extreme Kompaktierung bei transienten Fehlern Ausbeuteverluste entstehen, die mit dem Selbsttest mit Rücksetzpunkten verhindert werden sollten: Tritt ein transienter Fehler beispielsweise in Sitzung T_i auf und wird dieser erst mit einiger Latenz in einer späteren Sitzung erkannt, werden durch die Abspeicherung des falschen MISR-Startzustandes im Schattenregister auch nach der Wiederholung falsche Paritätsbits berechnet. Abbildung

3.14 verdeutlicht dieses Problem anhand eines Beispiels mit einem Paritätsbit: In Sitzung T_i tritt ein transienter Fehler auf, der Auswirkungen auf zwei Bits in der Signatur hat. Da allerdings nur die Parität berechnet und mit der Soll-Parität verglichen wird, bleibt dieser Fehler unerkannt und die nächste Sitzung startet mit Abspeicherung des MISR-Zustands im Schattenregister. Da die Sitzung mit einem falschen Zustand des MISRs gestartet ist, wird auch die Signatur Q_{i+1} der Sitzung T_{i+1} Abweichungen von S_{i+1} haben, in diesem Beispiel allerdings an drei Bits. Hierdurch wird eine Parität $\Pi(Q_{i+1})$ berechnet, die sich von $\Pi(S_{i+1})$ unterscheidet, so dass ein Fehler in Sitzung T_{i+1} erkannt wird. Die Sitzung wird zurückgesetzt und der fehlerhafte MISR-Zustand wird in das MISR geladen, wodurch sowohl Signatur als auch Paritätsbit nach der Wiederholung immer noch fehlerhaft sind. Die Schaltung wird aussortiert, obwohl nur ein transienter Fehler in Sitzung T_i vorlag.

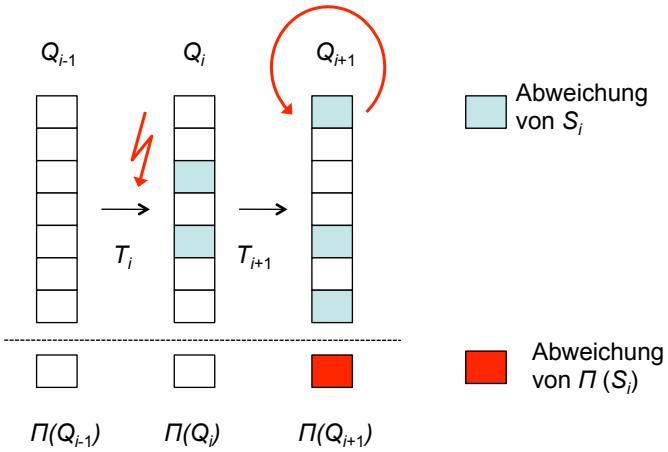


Abbildung 3.14: Beispiel Ausbeuteverlust durch latente Fehlererkennung

Für die Dimensionierung des Registers, also die Abschätzung der Länge des Paritäts-Fensters um Aliasing zu vermeiden, muss das Verhalten eines MISRs im Fehlerfall analysiert werden, welches am Ende dieses Kapitels detaillierter beschrieben wird.

Einsparung durch zusätzliches MISR

Im Gegensatz zur Paritätsberechnung kann auch der Vergleich der Signatur mit der Soll-Signatur auf eine Teilmenge der Bits beschränkt werden. Da allerdings auch hierbei Fehler ausmaskiert werden können, wird anstatt der Berechnung der Parität über mehrere MISR-Zustände ein zusätzliches MISR implementiert, in welches nach dem Ende einer Sitzung die Signatur des MISRs der ersten Kompaktierungsstufe geschoben wird. Der prinzipielle Aufbau ist in Abbildung 3.15 dargestellt. Im Gegensatz zu der Architektur des Selbsttests mit Rücksetzpunkten in Abbildung 3.9 wird anstatt eines Backup-Registers ein zusätzliches MISR eingesetzt und nach einigen Durchläufen werden nur L Bits der Signatur betrachtet. Das MISR der ersten Kompaktierungsstufe wird zu Beginn einer jeden Sitzung zurückgesetzt.

Wie bei der Architektur mit Paritätslogik muss für die Dimensionierung des Parameters L, also die Anzahl der zu vergleichenden Signaturbits, die Fehlerfortpflanzung in einem

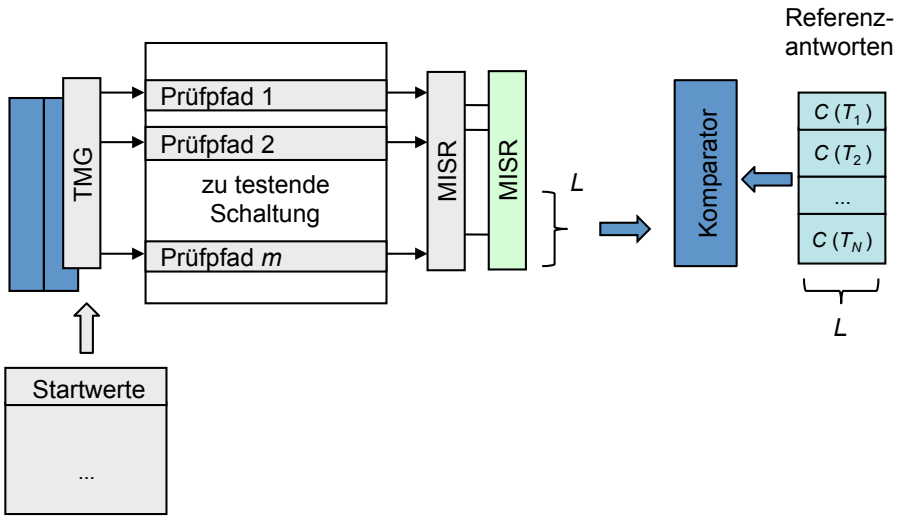


Abbildung 3.15: Architektur Selbsttest mit Rücksetzpunkten und extremer Kompaktierung (zusätzliches MISR)

MISR genauer untersucht werden, damit Aliasing verhindert werden kann. Im Folgenden wird hierauf genauer eingegangen.

Fehlerfortpflanzung

Gegeben sei ein MISR mit dem Startzustand z und der Eingabefolge $(A_i)_{i\geq 1}, A_i = (a_{i,k-1}, ...a_{i,0})$, siehe Abbildung 3.16.

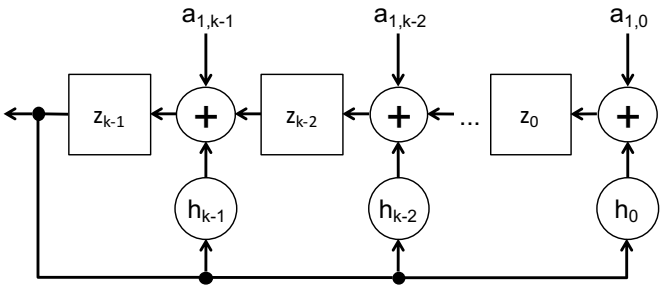


Abbildung 3.16: MISR

Wie in Kapitel 2 beschrieben, lassen sich die Zustandsübergänge eines MISRs mit primitivem Rückkopplungspolynom $h(Y)$ als lineare Transformation mit einer Matrix H beschreiben. Tritt nun zum Zeitpunkt $t = 1$ ein Fehler auf, wird statt A_1 ein $A_1 + e$ mit Fehlervektor e in das MISR geladen. Somit ergeben sich die folgenden Zustände

$$
\begin{aligned}
&t = 0 : z \\
&t = 1 : Hz + A_1 + e \\
&t = 2 : H^2 z + HA_1 + He + A_2 \\
&t = 3 : H^3 z + H^2 A_1 + H^2 e + HA_2 + D_3 \\
&\ldots
\end{aligned}
$$

Vergleicht man nun die fehlerfreie mit der fehlerhaften Zustandsfolge, erkennt man, dass sie sich nur um

$$
\begin{aligned}
&\ldots \\
&t = 1 : e \\
&t = 2 : He \\
&t = 3 : H^2 e \\
&\ldots
\end{aligned}
$$

unterscheiden, welches exakt den Zustandsübergängen eines linear rückgekoppelten Schieberegisters mit Startzustand e und Rückkopplungspolynom $h(Y)$ entspricht. Handelt es sich bei $h(Y)$ um ein primitives Polynom, durchläuft das LFSR alle möglichen Zustände (bis auf den 0-Zustand). Die Wahrscheinlichkeit, dass sich der Fehler e also an einem beliebigen Zustandsbit bemerkbar macht, ist ungefähr $\frac{1}{2}$. Werden somit L zufällige Zustandsbits beobachtet, lässt sich der Fehler mit einer Wahrscheinlichkeit von $1 - 2^{-L}$ erkennen. Dieser Wert liegt schon für kleine L nahe bei 1, allerdings können in ungünstigen Fällen Fehler maskiert werden: Wenn beispielsweise bei einem m-Bit breiten MISR die Zustandsbits z_{m-L},...,z_{m-1} beobachtet werden, sich der Fehler aber an Zustandsbit z_0 auswirkt, werden mindestens $m-L$ Takte benötigt, damit sich der Fehler an den beobachteten Zustandsbits auswirkt. Die gleiche Aussage lässt sich auch für die Paritätsberechnung treffen: Entspricht Bit z_{k-1} einer '1', wird eine ungerade Anzahl an Bits invertiert, so dass sich die Parität ändert. Somit ist die Wahrscheinlichkeit, dass nach L Zuständen mindestens einmal eine fehlerhafte Parität auftritt, ebenfalls $1 - 2^{-L}$.

Hieraus lässt sich leicht berechnen, dass bereits für acht Bits die Wahrscheinlichkeit der Fehlermaskierung bei unter 1% und somit die Erkennungswahrscheinlichkeit bei über 99% liegt. Zur Paritätsberechnung werden $m - 1$ zusätzliche XOR Gatter benötigt, während bei der Einbringung des zusätzlichen MISRs ebenfalls nur XOR Gatter hinzugefügt werden müssen, um aus dem bereits vorhandenen Schatten-Register ein MISR zu implementieren. Neben der Einsparung der Paritätslogik hat die Implementierung mit zusätzlichem MISR den Vorteil, dass der Test im fehlerfreien Fall ohne jegliche Verzögerung abläuft.

3.4 Experimente und Ergebnisse

Um die Ergebnisse der extremen Kompaktierung zu validieren, wurden verschiedene Experimente an den größeren NXP Schaltungen, siehe Tabelle 3.1, durchgeführt. Spalte zwei

zeigt die Schaltungsgröße gemessen an der Anzahl der Logikgatter. Spalten drei und vier sind jeweils die Anzahl der Pseudo-Primäreingänge und -Primärausgänge (PPI und PPO) zu entnehmen. Die Prüfpfad-Elemente wurden gleichmäßig auf die Prüfpfade verteilt, so dass sich die Verteilung gemäß den Spalten fünf (Anzahl der Prüfpfade) und sechs (maximale Länge l_{pr} der Prüfpfade) aus der Tabelle ergibt.

Schaltung	**Anzahl Gatter**	**Anzahl PPI**	**Anzahl PPO**	**Anzahl Prüfpfade**	**max. Länge**
p330k	312.666	18.010	17.468	64	282
p378k	341.315	15.732	17.420	44	396
p388k	489.271	25.005	24.065	50	569
p418k	382.633	30.430	29.809	64	664
p469k	75.572	635	332	1	635
p483k	444.664	33.264	32.610	71	469
p500k	431.439	30.768	30.840	76	406
p533k	586.819	33.373	32.610	71	471
p874k	629.723	61.977	70.863	59	1202
p951k	1.002.883	91.994	104.714	82	1277

Tabelle 3.1: Schaltungscharakteristika und Prüfpfaddaten

In den Experimenten wurde zunächst die Paritätslogik simuliert. Die extreme Kompaktierung mit dem Schatten-MISR wird bei den Experimenten zur Diagnose behandelt. Insgesamt wurden drei verschiedene Experimente durchgeführt, um die optimale Paritätslänge zu bestimmen und das vorgestellte Modell zur Fehlerfortpflanzung im MISR zu validieren. Hierzu wurde ein Test mit 10.000 deterministischen Testmustern in 10 Sitzungen gleicher Länge aufgeteilt. Zunächst wurde die Fehlererkennungslatenz untersucht, das heißt, nach wie vielen Sitzungen ein transienter Fehler erkannt wird, wenn nur eine Parität aus der Signatur am Ende einer Sitzung berechnet wird. In den zweiten Experimenten wurden direkt nach der Fehlerinjektion fortlaufend über die nächsten MISR-Zustände die Paritäten berechnet, um das bereits ausgearbeitete Modell zur Fehlerfortpflanzung in einem MISR zu validieren. Zum Schluss wurden schließlich unterschiedlich lange Paritätsfenster am Ende einer Sitzung miteinander verglichen.

Im ersten Experiment wurde die Fehlererkennungslatenz untersucht: Ein transienter Fehler wurde zufällig während der ersten Sitzung in die Schaltung injiziert. Nach jeder Sitzung T_i wird die Signatur Q_i zu einem einzigen Paritätsbit $\Pi(Q_i)$ kompaktiert und dieses Paritätsbit mit dem fehlerfreien Paritätsbit $\Pi(S_i)$ verglichen. Die Fehlererkennungslatenz l wird dann wie folgt berechnet: Wird der Fehler erstmals in Sitzung T_i erkannt, also $\Pi(Q_i) \neq \Pi(S_i)$, entspricht die Latenz

$$l = i - 1. \tag{28}$$

Für jede Schaltung wurden 100 Experimente durchgeführt, wobei die Simulation mit einem in C++ geschriebenen Logiksimulator durchgeführt wurde. Die Fehlerinjektion wurde mittels Bitflip an einem zufälligen Schaltungsausgang durchgeführt[6]. Tabelle 3.2 zeigt

[6] In diesen Experimenten wurde nicht mit einem klassischen Fehlersimulator gearbeitet sondern dieses Worst Case-Szenario benutzt, um ein direktes Fehlverhalten an den MISR-Eingängen zu simulieren.

die Ergebnisse dieser Experimente. Die Spalte mit $l = 0$ zeigt die Anzahl der Experimente, in denen der Fehler direkt in der ersten Sitzung erkannt wurde. Wurde der Fehler nie erkannt, entspricht die Latenz $l = \infty$. Es ist interessant zu beobachten, dass im günstigsten Fall in 70% der Experimente (p500k), im ungünstigsten Fall nur in 26% der Fälle (p330k) der Fehler direkt in der Sitzung des Auftretens des Fehlers erkannt wird. Der Verlauf dieser beiden Fälle lässt sich in Abbildung 3.17 erkennen. Für den p330k sind sogar solche ungünstigen Fehler aufgetreten, die erst nach sechs weiteren Sitzungen erkannt wurden. Für den p500k war $l = 4$ das Maximum. Man erkennt, dass, wie im Modell bereits beschrieben, ein Paritätsbit nicht ausreicht, um einen transienten Fehler direkt in derselben Sitzung zu erkennen.

Schaltung	**Anzahl der Experimente mit Latenz l**										
	l=0	l=1	l=2	l=3	l=4	l=5	l=6	l=7	l=8	l=9	l=∞
p330k	26	55	3	4	4	0	8	0	0	0	0
p378k	39	49	0	0	2	0	0	0	10	0	0
p388k	33	17	11	25	1	12	0	0	0	1	0
p418k	33	17	11	25	1	12	0	0	0	1	0
p469k	61	22	12	1	3	0	0	0	0	1	0
p483k	61	22	12	1	3	0	0	0	0	1	0
p500k	70	13	8	8	1	0	0	0	0	0	0
p533k	59	26	11	2	0	0	0	0	2	0	0
p874k	50	37	6	0	1	3	3	0	0	0	0
p951k	53	43	3	1	0	0	0	0	0	0	0

Tabelle 3.2: Fehlererkennungslatenz für ein Paritätsbit

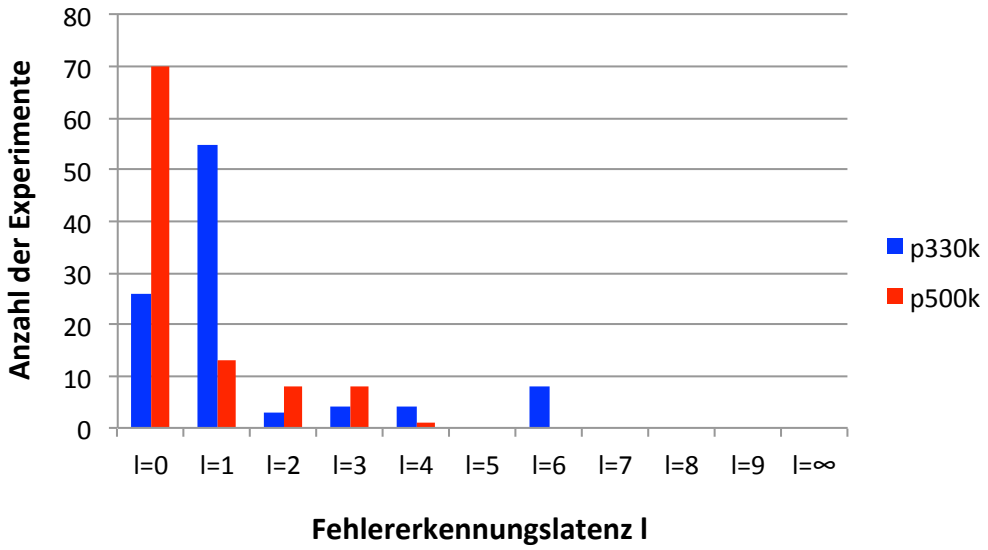


Abbildung 3.17: Verlauf der Fehlererkennungslatenz im günstigsten und ungünstigsten Fall

Um das theoretische Modell der Fehlerauswirkung innerhalb eines MISRs zu validieren, wurden im zweiten Teil der Simulationen folgende Experimente durchgeführt: Es wurde wieder innerhalb der ersten Sitzung ein zufälliger Bitflip an einem der Schaltungsausgänge erzeugt, allerdings wurde dann nach jedem Takt ein Paritätsbit über den MISR-Zustand berechnet. Die Anzahl der Takte n, für die das Paritätsbit dem fehlerfreien Fall entspricht, wird als *Aliasingfolge* bezeichnet. Für jede Schaltung wurden 100 Experimente durchgeführt, die Verteilung der Aliasingfolgen-Länge über maximal 10 Takte ist in Tabelle 3.3 zu erkennen. Wie man leicht sehen kann, entspricht die Verteilung der Aliasingfolge ungefähr $0,5^{-n}$, was auch dem Modell aus Kapitel 3 entspricht. Grafisch lässt sich der Vergleich zwischen Modell und dem Mittelwert aller Experimente darstellen wie in Abbildung 3.18.

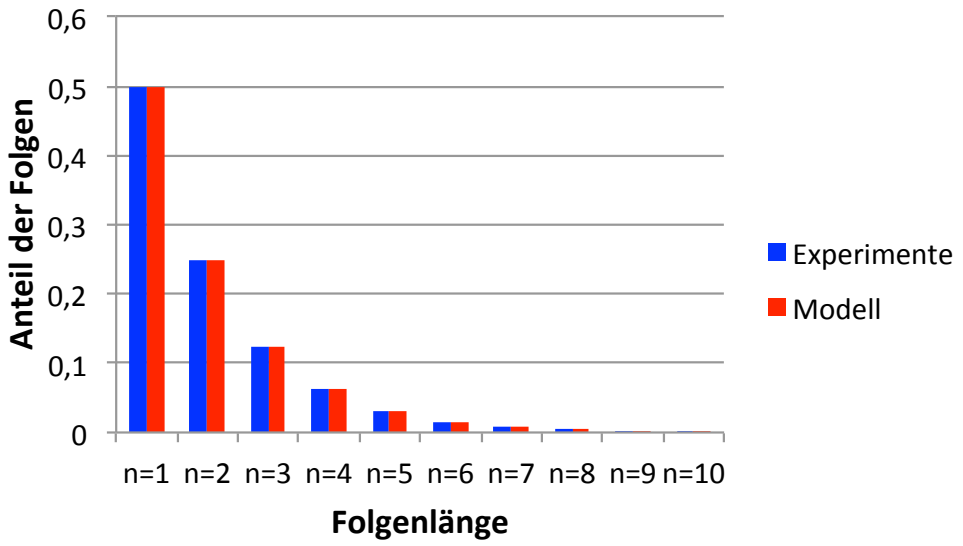


Abbildung 3.18: Vergleich der experimentellen Ergebnisse mit dem theoretischen Modell

In den letzten Experimenten zur Validierung der extremen Kompaktierung wurde nun die Latenz für ein Paritätsfenster mit L Bits untersucht. Dieses Paritätsfenster berechnet die Paritäten zum Zeitpunkt der Signaturberechnung sowie der $L-1$ Zustände um diesen Zeitpunkt herum. Hierzu wurde wieder in der ersten Sitzung ein Bitflip zufällig an einem der Schaltungsausgänge injiziert und jede Schaltung 100 mal simuliert. Die Paritäten wurden dann mit den Soll-Paritäten verglichen und bei Erkennung eines Fehlers die Simulation abgebrochen. Tabelle 3.4 zeigt die Anzahl der Ergebnisse für jede Schaltung in Abhängigkeit von dem Paritätsfenster der Länge L. In jeder Zelle ist die Anzahl der Ergebnisse angegeben, in denen die Fehlererkennungslatenz l zwischen 0 und ∞ liegt. Die obere Schranke ist hier auch sinnvoll, da bei Nicht-Erkennung eines transienten Fehlers dieser auch keine Auswirkungen auf die Testergebnisse hat und somit nicht zu Ausbeuteverlusten führt. Wie man sieht, reicht ein Paritätsfenster von $L = 8$ Bits aus, um alle Fehler direkt in der ersten Sitzung (oder gar nicht) zu erkennen. Somit entsteht durch die extreme Kompaktierung kein zusätzliches Aliasing.

Schaltung	**Anteil der Aliasingfolgen der Länge n**									
	n=1	n=2	n=3	n=4	n=5	n=6	n=7	n=8	n=9	n=10
p330k	0,49914	0,25029	0,12578	0,0622	0,03132	0,01559	0,00797	0,00389	0,00189	0,00099
p378k	0,49934	0,25050	0,12524	0,06249	0,03113	0,01565	0,00771	0,00399	0,00191	0,00095
p388k	0,50051	0,2496	0,12494	0,06234	0,03127	0,01569	0,00786	0,00392	0,00199	0,00096
p418k	0,50114	0,24959	0,12429	0,06217	0,03144	0,01567	0,00779	0,00399	0,00195	0,00098
p469k	0,49944	0,25069	0,12488	0,06244	0,03126	0,01565	0,0078	0,00395	0,00194	0,00096
p483k	0,50038	0,24946	0,12499	0,06248	0,0314	0,01551	0,00789	0,00395	0,00194	0,00097
p500k	0,49923	0,25002	0,12521	0,06247	0,03175	0,01568	0,00776	0,00392	0,00198	0,00097
p533k	0,49939	0,24971	0,12531	0,06258	0,03164	0,01567	0,00782	0,00388	0,00198	0,00097
p874k	0,50083	0,24983	0,1247	0,06206	0,03109	0,01556	0,00803	0,00399	0,00191	0,00096
p951k	0,49983	0,24951	0,12533	0,06305	0,03098	0,01554	0,00786	0,00392	0,00198	0,001

Tabelle 3.3: Anteil der Aliasingfolgen

Schaltung	**Anzahl der Experimente mit Latenz $0 < l < \infty$ bei einem Paritätsfenster mit L Bits**									
	L=1	L=2	L=3	L=4	L=5	L=6	L=7	L=8	L=9	L =10
p330k	74	35	7	8	4	0	0	0	0	0
p378k	61	24	14	0	2	0	0	0	0	0
p388k	67	9	4	0	0	0	0	0	0	0
p418k	63	28	30	1	0	0	0	0	0	0
p469k	39	17	5	5	1	3	1	0	0	0
p483k	66	26	16	15	13	7	0	0	0	0
p500k	30	20	5	2	1	0	0	0	0	0
p533k	41	19	6	0	0	0	0	0	0	0
p874k	50	29	29	0	0	0	2	0	0	0
p951k	47	19	19	3	5	0	0	0	0	0

Tabelle 3.4: Fehlererkennungslatenz in Abhängigkeit von einem L-Bit breiten Paritätsfensters

Ausgehend von einem Paritätsfenster der Länge $L = 8$ zeigt Tabelle 3.5 die Reduktion des Speichers, der benötigt wird, um bei einem Test mit N Sitzungen anstatt der vollen Signaturen nur acht Bits abzuspeichern. Abhängig von der MISR-Größe wird ein Reduktionsfaktor zwischen 4 und 12 erreicht. Die zusätzliche Hardware ist ein XOR-Baum zur Paritätsberechnung, sowie ein Register der Länge L, um die Paritäten des Tests nach jeder Sitzung vorzuhalten und mit den berechneten Paritäten zu vergleichen. Die Größe des XOR-Baums ist abhängig von der Größe des MISRs: Bei einem m-Bit breiten MISR werden $m - 1$ zusätzliche XOR-Gatter benötigt.

Schaltung	**MISR Größe**	**Anzahl Bits für vollständige Signatur**	**Anzahl Bits für Paritätsfenster**	**Reduktions-faktor**
p330k	64	$64N$	$8N$	8
p378k	64	$64N$	$8N$	8
p388k	64	$64N$	$8N$	8
p418k	64	$64N$	$8N$	8
p469k	32	$32N$	$8N$	4
p483k	96	$96N$	$8N$	12
p500k	96	$96N$	$8N$	12
p533k	96	$96N$	$8N$	12
p874k	64	$64N$	$8N$	8
p951k	96	$96N$	$8N$	12

Tabelle 3.5: Fehlererkennungslatenz in Abhängigkeit eines L-Bit breiten Paritätsfensters

3.5 Zusammenfassung

In diesem Kapitel wurde der Selbsttest mit Rücksetzpunkten mit einer extremen Kompaktierung erweitert, welches es erlaubt, anstatt der vollständigen MISR-Signatur nur wenige Bits zu vergleichen und vorzuhalten. Hierzu wurde eine zweistufige Kompaktierung entwickelt, die im ersten Schritt die Testantworten zeitlich mit Hilfe eines MISRs und im nächsten Schritt räumlich kompaktiert. Die zweite Kompaktierungsstufe kann sowohl mit einer Paritätslogik, als auch mit einem zusätzlichen MISR erfolgen, bei dem dann nicht die komplette Signatur, sondern nur einige Bits der Signatur verglichen und auf dem Chip vorgehalten werden. Hiermit lassen sich Kompaktierungsraten von 4X-12X erreichen, ohne Einschränkungen der Produktqualität und Ausbeuteverluste in Kauf zu nehmen. Die theoretische Aliasing-Wahrscheinlichkeit beträgt schon bei Verwendung von $L = 8$ Bits bzw. einem Paritätsfenster der Länge acht weniger als 0,4%, wobei in 1.000 Experimenten kein einziger Fall von Aliasing beobachtet wurde. Je nach MISR Größe m werden $m - 1$ zusätzliche XOR-Gatter für die Paritätsberechnung benötigt oder einige XOR-Gatter, um das Schatten-Register in ein Schatten-MISR umzuwandeln. Auch bei dieser Art der extremen Kompaktierung lässt sich feststellen, dass die Fehlerabdeckung identisch mit der Fehlerabdeckung eines Standard-BIST ist. Zusammen mit dem Selbsttest mit Rücksetzpunkten kann ein sehr effizienter Test implementiert werden, der permanente Fehler von nicht-permanenten Fehlern unterscheidet.

4 Direkte Diagnose permanenter Fehler

Nachdem der Selbsttest, der in Kapitel 3 behandelt wurde, zwischen permanenten und nicht-permanenten Fehlern unterscheiden kann, soll in diesem Kapitel die Integration eines effizienten Diagnoseverfahrens vorgestellt werden, was auch in [CHIW11b] und [CHIW11a] publiziert wurde. Grundlage hierfür ist das in [CEWA11] eingeführte Diagnoseverfahren, welches im weiteren Verlauf zunächst kurz vorgestellt werden soll, sowie das aus Kapitel 2 bekannte bedingte Haftfehlermodell. Das daraus resultierende eingebettete Diagnoseverfahren wird dann in diesem Kapitel auf die Testergebnisse des Selbsttests mit Rücksetzpunkten und extremer Kompaktierung angewandt, um einerseits einen permanenten Fehler zu diagnostizieren und andererseits auch Schlüsse zur weiteren Klassifikation zu ziehen.

4.1 Stand der Technik: Diagnose im Selbsttest

Wie bereits in Kapitel 2.5 beschrieben, wird Diagnose im Selbsttest prinzipiell in *direkte Diagnose* und *indirekte Diagnose* eingeteilt, wobei beide Arten die bereits diskutierten Vor- und Nachteile haben. In [CEWA11] wurde erstmals ein direktes Diagnoseverfahren im Selbsttest eingeführt, welches mit der STUMPS-Architektur kompatibel ist und eine hohe diagnostische Auflösung für beliebige Fehler erreicht. Dieses Verfahren baut auf dem in Kapitel 2.5.1 beschriebenen POINTER-Algorithmus auf und soll in diesem Kapitel detailliert beschrieben werden.

Im Allgemeinen wird beim Verfahren aus [CEWA11] der Test in Sitzungen eingeteilt und nach jeder Sitzung werden die Signaturen auf Korrektheit überprüft. Im Fehlerfall wird die fehlerhafte Signatur in einem Fehlerspeicher abgelegt. Am Ende des Tests werden anhand der fehlerhaften Signaturen mögliche Fehlerorte, die das Fehlverhalten erklären, bestimmt und in eine geordnete Liste gebracht. Um komplexere Fehler zu modellieren, wird das bereits definierte bedingte Haftfehlermodell verwendet. Die Diagnose, basierend auf diesem Fehlermodell, besteht darin, für jedes Testmuster einzeln die möglichen Fehlerkandidaten und deren Aktivierungsbedingung zu identifizieren. Das Problem bei einem Selbsttest ist allerdings, dass die Informationen in einem MISR kompaktiert werden. Deswegen müssen in vorverarbeitenden Schritten (engl. *pre-processing*) die Auswirkungen der Fehler auf die Signatur vorberechnet werden, wobei dies für beliebig komplexe Fehler zu zeitaufwändig wäre. Im weiteren Verlauf dieses Kapitels soll näher auf die vorverarbeitenden Prozesse, sowie die Auswertungen des Fehlerspeichers eingegangen werden.

Ähnlich wie beim Selbsttest mit Rücksetzpunkten wird ein Test T mit M Testmustern in N Sitzungen $T_1, T_2, ..., T_N$ aufgeteilt. Jede Sitzung mit $n = \lceil M/N \rceil$ Testmustern $\psi_1, ..., \psi_n$ generiert eine Signatur Q_i. Das MISR wird durch eine Rückkopplungsmatrix H charakterisiert (siehe Kapitel 2.4). Dann beschreibt die Matrix $M = H^k$ die Funktion des MISRs nach k Takten. Wenn in einer Sitzung T_i ein Fehler ϕ an Fehlerort v nur bei Muster ψ_j aktiv ist, erhält man eine Abweichung d_j^ϕ von der Sollsignatur S_i. Insgesamt erhält man so nach [CEWA11] das Gleichungssystem

$$\begin{bmatrix} d_1^\phi & d_2^\phi & ... & d_{n-1}^\phi & d_n^\phi \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ ... \\ c_{n-1} \\ c_n \end{bmatrix} = S_i \oplus Q_i^\phi \tag{29}$$

mit $d_j^\phi := H^{n-j} e_j^\phi$ und den Konstanten $c_1, ..., c_n \in \{0, 1\}$.

Die Matrix $\begin{bmatrix} d_1^\phi & d_2^\phi & ... & d_{n-1}^\phi & d_n^\phi \end{bmatrix}$ kann in der Vorverarbeitung vorberechnet werden. Gibt es für das Gleichungssystem eine Lösung, beschreiben die Konstanten $c_1, ..., c_n$ die Aktivierungsbedingungen für die Muster $\psi_1, ..., \psi_n$. Gibt es keine Lösung, kann der Fehler am Fehlerort v nicht allein das Fehlverhalten erklären.

Die Evidenz eines Fehlers wird analog zum bereits beschriebenen POINTER-Algorithmus berechnet, allerdings werden jetzt nicht einzelne Ausgänge, sondern die Ergebnisse der Sitzungen zur Berechnung herangezogen: Anhand der Anzahl der Sitzungen, in denen ein Fehler ϕ das Fehlverhalten erklärt, werden die Fehler sortiert. Gibt es zwei Fehler ϕ_1 und ϕ_2, welche die gleiche Anzahl von fehlerhaften Sitzungen erklären, werden die fehlerfreien Sitzungen in die Analyse mit einbezogen: Ist die Anzahl der Sitzungen, in denen eine korrekte Signatur aufgetreten ist, obwohl der Fehler ϕ_1 aktiviert wurde, höher als die entsprechende Anzahl von Sitzungen bei ϕ_2, dann wird ϕ_2 vor ϕ_1 sortiert.

> **Beispiel:** Ein Test sei in vier Sitzungen unterteilt, Sitzung T_1 war fehlerfrei und für die übrigen Sitzungen seien folgende Fehlerkandidaten identifiziert worden
>
> $$T_2 = \{\phi_1\}$$
> $$T_3 = \{\phi_2, \phi_1, \phi_3\}$$
> $$T_4 = \{\phi_2\}.$$
>
> Wie man erkennen kann, können sowohl ϕ_1 als auch ϕ_2 zweimal das fehlerhafte Verhalten erklären. Wenn jetzt der Fehler ϕ_2 die korrekte Signatur für Sitzung T_1 erklärt, ϕ_1 aber nicht, dann wird Fehler ϕ_2 höher einsortiert. ■

Nach der Analyse erhält man eine Liste von bedingten Haftfehlern, die das Fehlverhalten erklären. Experimente in [CEWA11] haben gezeigt, dass mit diesem Ansatz für verschiedene Fehlerarten sehr gute diagnostische Auflösungen erreicht werden, teilweise können sogar bessere Ergebnisse erzielt werden als bei einer Diagnose, die direkt jedes Testmuster

auswertet[7]. Die diagnostische Auflösung gibt an, wie viele Fehler korrekt diagnostiziert wurden, wobei dies in diesem Zusammenhang bedeutet, dass der Fehler an Position 1 der Liste möglicher Fehlerorte steht. Für das gesamte Verfahren muss nur ein zusätzlicher Fehlerspeicher, sowie ein Speicher für die Zwischensignaturen auf dem Chip implementiert werden.

4.2 Einordnung in den Gesamtprozess

Der Selbsttest mit Rücksetzpunkten erlaubt es, zwischen permanenten und nicht-permanenten Fehlern zu unterscheiden. Allerdings lassen sich hieraus keine Rückschlüsse ziehen, ob es sich bei dem nicht-permanenten Fehler um einen nicht-kritischen transienten oder einen kritischen intermittierenden Fehler handelt. Zusätzlich müssen, um das in Kapitel 4.1 beschriebene Diagnoseverfahren für stark kompaktierte Signaturen einzusetzen, Anpassungen an der Testinfrastruktur aus Kapitel 3 vorgenommen werden. Diese Erweiterungen werden nun präzisiert.

4.2.1 Erweiterung der Testarchitektur

Das in Kapitel 4.1 beschriebene Diagnoseverfahren setzt voraus, dass die Signaturen der Teilsitzungen $Q_1, ..., Q_N$ unabhängig voneinander sind. Um dies zu gewährleisten, muss das MISR während des Tests nach jeder Sitzung zurückgesetzt werden. Somit wird im Folgenden davon ausgegangen, dass die extreme Kompaktierung als zusätzliche Kompaktierungsstufe ein Schatten-MISR verwendet. Zusätzlich muss die Architektur um einen Fehlerspeicher erweitert werden, wie in Abbildung 4.1 gezeigt. Am Ende einer Sitzung T_i werden L Bits der Signatur, mit $C(Q_i)$ symbolisiert, mit der extrem kompaktierten Referenzsignatur $C(S_i)$ verglichen. Im Fehlerfall wird die komplette Signatur des Schatten-MISRs, $Q_{Schatten,i}$ in den Fehlerspeicher abgelegt und die Sitzung wiederholt.

Für die weitere Klassifikation der nicht-permanenten Fehler werden im Fehlerspeicher neben den fehlerhaften Signaturen $Q_{Schatten,i}$ bzw. den fehlerhaften Signaturen der wiederholten Sitzung $R_{Schatten,i}$ auch die Sitzungsnummern und die gewonnenen Informationen über die Art des jeweiligen Fehlers (permanent, intermittierend oder transient) abgespeichert, siehe Abbildung 4.2. Hierzu lassen sich folgende Beobachtungen feststellen (wobei aufgrund einer übersichtlicheren Darstellung in den nachfolgenden Ausführungen nur der Fall für $W = 2$, also maximal zwei Durchläufe einer Sitzung, betrachtet wird):

Da ein permanenter Fehler, im Gegensatz zu einem transienten oder intermittierenden, auf jeden Fall immer zur selben fehlerhaften Signatur führt, kann dieser leicht durch einen Vergleich der fehlerhaften Signaturen identifiziert werden. Unterscheiden sich die beiden fehlerhaften Signaturen, handelt es sich mit hoher Wahrscheinlichkeit um einen nicht-permanenten Fehler (transient oder intermittierend) und ein genauerer Klassifikationsprozess ist nötig (siehe Kapitel 5). Sind die beiden fehlerhaften Signaturen gleich, wird von einem permanenten Fehler ausgegangen und der genaue Fehlerort muss mittels Diagnose bestimmt werden. In beiden Fällen wird ein Fehlerspeicher benötigt. Der angepasste Testablauf ist wie folgt:

[7] Hierbei ist zu beachten, dass der Fehlerspeicher begrenzt ist und so unter Umständen mehr Fehlerinformationen aus den abgelegten Signaturen extrahiert werden können, als wenn einzelne fehlerhafte Testmusterantworten abgelegt werden.

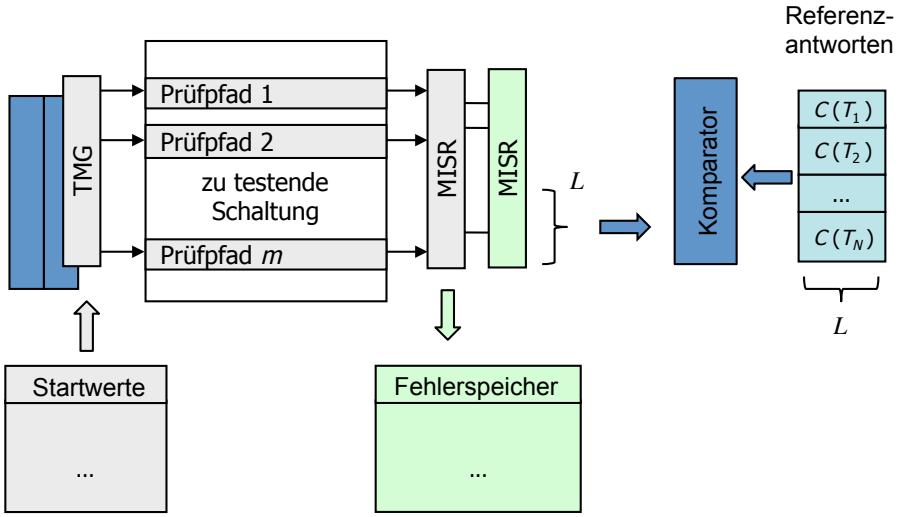


Abbildung 4.1: Architektur für den Selbsttest mit Diagnose

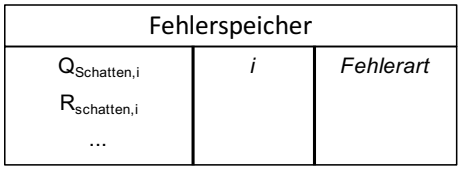


Abbildung 4.2: Fehlerspeicher

Ist eine Sitzung fehlerhaft, entspricht also die extrem kompaktierte Signatur $C(Q_i)$ nicht der Soll-Signatur $C(S_i)$, wird die Sitzung wiederholt und die fehlerhafte Signatur des Schatten-MISRs $Q_{Schatten,i}$ sowie die Nummer der Sitzung i im Fehlerspeicher abgelegt. Nach der Wiederholung kommt es, in Abhängigkeit der Signatur $R_{Schatten,i}$ der wiederholten Sitzung, zur Fallunterscheidung:

- Die Signatur nach der Wiederholung entspricht der Soll-Signatur ($R_{Schatten,i} = S_{Schatten,i}$): Die nächste Sitzung wird gestartet und die Information, dass es sich wahrscheinlich um einen transienten oder intermittierenden Fehler handelt, wird im Fehlerspeicher abgelegt.

- Die Signatur nach der Wiederholung entspricht nicht der Soll-Signatur ($R_{Schatten,i} \neq S_{Schatten,i}$), aber der fehlerhaften Signatur des ersten Durchlaufs ($R_{Schatten,i} = Q_{Schatten,i}$): Da $W = 2$ gewählt wurde, wird die nächste Sitzung gestartet. Die Wahrscheinlichkeit für einen permanenten Fehler ist sehr hoch, die Information, dass es sich wahrscheinlich um einen permanenten Fehler handelt, wird im Fehlerspeicher zur Signatur $R_{Schatten,i}$ abgespeichert.

- Die Signatur nach der Wiederholung entspricht nicht der Soll-Signatur ($R_{Schatten,i} \neq S_{Schatten,i}$) und auch nicht der fehlerhaften Signatur des ersten Durchlaufs ($R_{Schatten,i} \neq Q_{Schatten,i}$): Die Wahrscheinlichkeit für einen nicht-permanenten Fehler ist sehr hoch und die nächste Sitzung wird gestartet. Die neue

fehlerhafte Signatur und die Information, dass es sich wahrscheinlich um einen transienten oder intermittierenden Fehler handelt, werden im Fehlerspeicher hinterlegt.

Durch diese zusätzlichen Informationen kann eine weitere Klassifikation effizienter durchgeführt werden. Für die Diagnose eines permanenten Fehlers werden lediglich die fehlerhaften Signaturen $Q_{Schatten,i}$ analysiert.

4.2.2 Analyse der fehlerhaften Signaturen

Aus den fehlerhaften Signaturen $Q_{Schatten,i}$, die im Fehlerspeicher abgelegt wurden, wird nun im ersten Schritt versucht, eine Kombination von fehlerhaften und fehlerfreien Antworten zu finden, die für jede Sitzung separat die fehlerhafte Signatur erklärt. Wie bereits in Kapitel 4.1 beschrieben, wird für jeden Fehler ϕ die Auswirkung d_j^ϕ auf die Signatur $Q_{Schatten,i}$ des Musters ψ_j bestimmt. Für jeden Fehlerort v wird dann versucht, das lineare Gleichungssystem

$$[d_1^\phi ... d_n^\phi] \begin{bmatrix} c_1 \\ ... \\ c_n \end{bmatrix} = S_{Schatten,i} \oplus Q_{Schatten,i} \tag{30}$$

zu lösen, wobei die Abweichungen $[d_1^\phi, ..., d_n^\phi]$, wie bereits beschrieben, vorberechnet werden können. Ist das lineare Gleichungssystem lösbar, beschreibt $[c_1, ..., c_n]$ die Bedingung für die Testmuster $[\psi_1, ..., \psi_n]$. Gibt es keine Lösung, kann v kein Fehlerort sein, der das Fehlverhalten (im Fall eines Einfachfehlers) erklärt. Nachdem alle Sitzungen ausgewertet wurden, werden die Fehler entsprechend ihrer *Evidenz* sortiert, wie in Kapitel 2.5.2 bereits ausgeführt. Experimente, die in Kapitel 4.3 vorgestellt werden, haben gezeigt, dass auch mit der extremen Kompaktierung sehr gute diagnostische Auflösungen erreicht werden können.

Folgendes Beispiel soll das integrierte Test- und Diagnoseverfahren verdeutlichen:

Beispiel: Gegeben sei die bereits bekannte Schaltung aus Abbildung 4.3. Die Testmustermenge Ψ besteht aus drei Testmustern $\psi_0 = [110110]$, $\psi_1 = [010011]$ und $\psi_2 = [101100]$. Zur extremen Kompaktierung wird ein 3-Bit MISR mit der in Abbildung 4.3 gezeigten Verdrahtung eingesetzt, wobei nur MISR-Bit m_3 verglichen wird (aus Gründen der Übersichtlichkeit wird das Schatten MISR sowie die Hardware zum Vergleich nicht gezeigt). Zu Beginn sei das MISR im 0-Zustand $[m_1, m_2, m_3] = [0, 0, 0]$.

Im fehlerfreien Fall ergibt sich die Wahrheitstabelle aus Tabelle 4.1. Die ersten beiden Spalten zeigen jeweils die Ein- und Ausgabebelegung und die dritte Spalte den MISR-Zustand, nachdem die jeweilige Ausgabe in das MISR geschoben wurde.

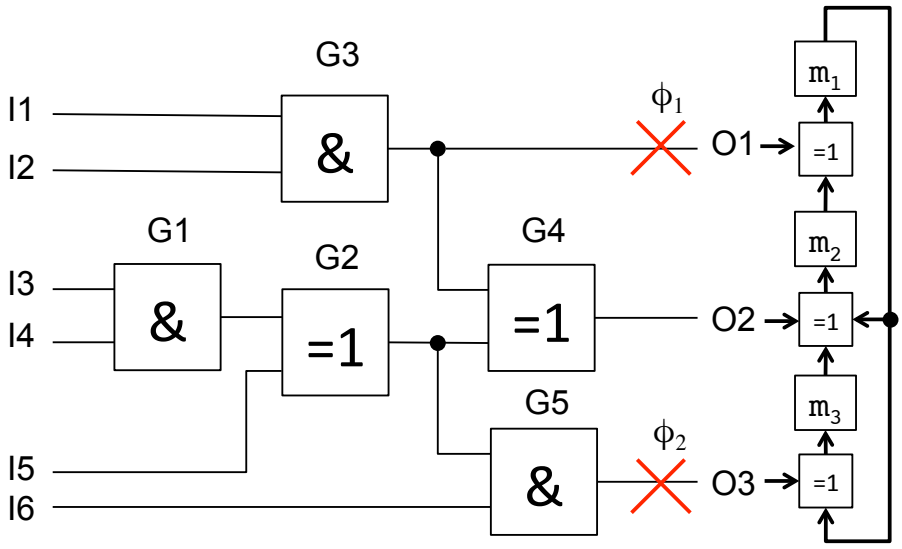


Abbildung 4.3: Beispielschaltung

Eingabe [I1,I2,I3,I4,I5,I6]	**Ausgabe [O1,O2,O3]**	**MISR Zustand** $[\mathbf{m_1}, \mathbf{m_2}, \mathbf{m_3}]$
$\psi_0 = [110110]$	$[1, 0, 0]$	$[1, 0, 0]$
$\psi_1 = [010011]$	$[0, 1, 1]$	$[0, 0, 0]$
$\psi_2 = [101100]$	$[0, 1, 0]$	$[0, 1, 0]$

Tabelle 4.1: Wahrheitstabelle

Um Aliasing zu vermeiden, berechnet das MISR noch die nächsten drei Zustände ohne zusätzliche Eingaben (siehe Kapitel 3.3). Es ergibt sich dann für

$$t = 4 : [m_1, m_2, m_3] = [1, 0, 0]$$
$$t = 5 : [m_1, m_2, m_3] = [0, 1, 1]$$
$$t = 6 : [m_1, m_2, m_3] = [1, 1, 0].$$

Somit ergibt sich im fehlerfreien Fall die Signatur als

$$S_{Schatten,i} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

und die extrem kompaktierte Signatur entsprechend als $m_3 = 0$.

Ist allerdings während des Tests ein Fehler aufgetreten, so dass zum Zeitpunkt $t = 6$ das MISR-Bit $m_3 = 1$ entspricht, muss die abgespeicherte Signatur $Q_{Schatten,i}$ aus dem Fehlerspeicher geladen und mit dem aus Kapitel 4.1 bekannten Verfahren analysiert werden. Nehmen wir an, ein bedingter

Haftfehler $\phi = (\psi_2|\Psi)\_1\_O_3$ sei während des Tests aufgetreten und die fehlerhafte Signatur sei

$$Q_{Schatten,i} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}.$$

Insgesamt können sowohl an den Schaltungsein- und ausgängen als auch an den Gatterein- und ausgängen Fehler auftreten, womit es insgesamt 15 mögliche Fehlerorte gibt. Zur Veranschaulichung betrachten wir nur zwei mögliche Fehlerorte O_1 und O_3 und die zwei Fehlerkandidaten $\phi_1 = s@1\_O_1$ und $\phi_2 = s@1\_O_3$. Für d_j^{ϕ} ergibt sich

$$d_0^{\phi_1} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, d_1^{\phi_1} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, d_2^{\phi_1} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix},$$

$$d_0^{\phi_2} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, d_1^{\phi_2} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, d_2^{\phi_2} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}.$$

Hierdurch lässt sich das lineare Gleichungssystem gemäß Gleichung (30) für den Fehler ϕ_1 aufstellen als

$$\begin{aligned} 0 \cdot c_1 + 0 \cdot c_2 + 1 \cdot c_3 &= 1 + 1 \\ 0 \cdot c_1 + 0 \cdot c_2 + 0 \cdot c_3 &= 1 + 0 \\ 0 \cdot c_1 + 1 \cdot c_2 + 1 \cdot c_3 &= 0 + 1 \end{aligned}$$

und es lässt sich leicht erkennen, dass das Gleichungssystem nicht lösbar ist. ϕ_1 kann kein möglicher (alleiniger) Fehler sein.

Für Fehler ϕ_2 ergibt sich das Gleichungssystem

$$\begin{aligned} 1 \cdot c_1 + 0 \cdot c_2 + 1 \cdot c_3 &= 1 + 1 \\ 0 \cdot c_1 + 0 \cdot c_2 + 1 \cdot c_3 &= 1 + 0 \\ 1 \cdot c_1 + 0 \cdot c_2 + 0 \cdot c_3 &= 0 + 1 \end{aligned}$$

und hier kann mit $c_1 = 1, c_3 = 1$ eine mögliche Lösung gefunden werden. Somit wurde ϕ_2 korrekterweise als Fehlerkandidat diagnostiziert. ■

Die Wahrscheinlichkeit für *Aliasing*, welches im Zusammenhang mit der Diagnose als gleiche fehlerhafte Auswirkung von zwei unterschiedlichen Fehlern interpretiert wird, kann wie folgt abgeschätzt werden:

In [CEWA11] wurde gezeigt, dass die Anzahl der Testmuster einer Sitzung $|\Psi|$ nicht größer sein sollte als die Anzahl der MISR-Bits m. Im ungünstigsten Fall wird ein Fehler von allen n Mustern einer Sitzung erkannt und die MISR-Signaturen sind linear unabhängig. Dann wäre die Anzahl der fehlerhaften Signaturen 2^n. Ausgehend von einer statistischen Unabhängigkeit und einer normal verteilten Auftrittswahrscheinlichkeit der Fehler kann die Aliasing-Wahrscheinlichkeit mit

$$\frac{2^n - 1}{2^m - 1} \approx 2^{n-m} \text{ für } n < m \tag{31}$$

abgeschätzt werden.

4.3 Experimente und Ergebnisse

In diesem Kapitel wird, im Gegensatz zur Untersuchung der Fehlerfortpflanzung in einem MISR aus Kapitel 3.4, zunächst die Fehlerabdeckung eines Schatten-MISRs für $L = 8$ Bits untersucht. Die nächsten Experimente sollen dann das diagnostische Verfahren validieren sowie den Hardwareaufwand bestimmen. Hierzu wurden einige NXP Schaltungen mit vollständigem Prüfpfad verwendet, wobei die Prüfpfad-Informationen ebenfalls von NXP zur Verfügung gestellt wurden. Tabelle 4.2 zeigt die Schaltungscharakteristika, die Prüfpfadinformationen mit der Anzahl der Prüfpfade und deren maximale Länge sowie die Anzahl der injizierten Haftfehler.

Schaltung	**Anzahl Gatter**	**Anzahl PPI**	**Anzahl PPO**	**Anzahl Prüfpfade**	**max. Länge**	**Anzahl Haftfehler**
p100k	84.356	5.902	5.829	270	53	162.129
p141k	152.808	11.290	10.502	264	45	283.548
p239k	224.597	18.692	18.495	260	61	455.992
p259k	298.796	18.713	18.495	360	61	607.536
p267k	239.687	17.332	16.621	260	62	366.871
p269k	239.771	17.333	16.621	360	62	371.209
p279k	257.736	18.074	17.835	385	59	493.844
p286k	332.726	18.351	17.835	385	60	648.044
p295k	249.747	18.508	18.521	330	62	472.124
p330k	312.666	18.010	17.468	320	64	540.758
p378k	341.315	15.732	17.420	325	64	816.534

Tabelle 4.2: Schaltungscharakteristika und injizierte Fehler

Die Simulationen wurden im Java-Framework *ADAMA* (*Adaptive Diagnosis for Arbitrariy Manifold Artifacts*) der Universität Stuttgart[8] durchgeführt, welches unter anderem Klassen und Funktionen für

[8] ADAMA wurde vom Institut für Technische Informatik (ITI) der Universität Stuttgart entwickelt und wird stetig aktualisiert. Im Rahmen von diversen Forschungs- und Projektarbeiten und im Rahmen dieser Dissertation konnte auf dieses Framework zugegriffen werden.

- Logik- und Fehlersimulation,
- einfache Fehlermodellierung und
- das vorgestellte Diagnoseverfahren

zur Verfügung stellt. In den ersten Experimenten wurde zunächst die extreme Kompaktierung validiert: Hierzu wurden mehrere Haftfehler injiziert (Anzahl siehe Spalte sieben der Tabelle 4.2), mit einer von einem kommerziellen Werkzeug erzeugten Mustermenge simuliert und die Fehlerabdeckung der extrem kompaktierten Signaturen mit der Fehlerabdeckung eines Tests ohne Kompaktierung verglichen. Tabelle 4.3 zeigt, dass schon bei $L = 8$ beobachteten MISR-Bits, im Vergleich zu der Simulation ohne Kompaktierung, eine nahezu identische Fehlerabdeckung erreicht wird. Die Fehlerabdeckung weicht maximal, in nur drei Fällen (p267k, p279k und p330k), um 0,02% von der Fehlerabdeckung ohne Kompaktierung ab, aber bei drei Schaltungen (p141k, p259k und p278k) ist die Fehlerabdeckung sogar auf zwei Nachkommastellen identisch.

Schaltung	**Anzahl Muster**	**Fehlerabdeckung ohne Kompaktierung**	**Fehlerabdeckung mit Kompaktierung**
p100k	5.397	99,56%	99,55%
p141k	5.642	98,86%	98,86%
p239k	4.778	98,84%	98,83%
p259k	4.919	99,10%	99,10%
p267k	5.191	99,60%	99,58%
p269k	5.164	99,60%	99,59%
p279k	5.360	97,89%	97,87%
p286k	6.224	98,34%	98,33%
p295k	7.916	99,15%	99,14%
p330k	9.165	98,95%	98,93%
p378k	664	100,00%	100,00%

Tabelle 4.3: Vergleich Fehlerabdeckung ohne/ mit Kompaktierung

Man kann leicht erkennen, dass auch für die Fehlerabdeckung permanenter Fehler die extreme Kompaktierung nahezu identische Ergebnisse erzielt wie ein Test ganz ohne Kompaktierung. Für die Diagnose wird dann allerdings die komplette Signatur in den Fehlerspeicher abgelegt. Im nächsten Schritt soll die diagnostische Auflösung des Verfahrens bestimmt werden.

Für die Bestimmung der diagnostischen Auflösung wurden verschiedene Fehlerarten, nämlich

- Haftfehler,
- Übersprechfehler,
- Verzögerungsfehler und
- Brückenfehler („Wired And“)

in die Schaltung injiziert und die fehlerhafte Signatur samt der Sitzungsnummer in einem Fehlerspeicher abgelegt. Der Fehlerspeicher wurde auf maximal 50 Einträge begrenzt. Es wurden pro Fehlermodell jeweils 100 zufällige Fehler injiziert und mit einer von einem kommerziellen Werkzeug erzeugten Mustermenge simuliert. Ein Fehler wird nur dann als korrekt diagnostiziert bezeichnet, wenn er nach dem Diagnoseverfahren als einziger Kandidat an die erste Stelle der Kandidatenliste gesetzt wurde. Tabelle 4.4 zeigt die diagnostische Auflösung des Verfahrens, wenn die Testantworten nicht kompaktiert, sondern jede Testantwort vollständig ausgewertet wurde (auch als „Bypass“ bezeichnet). Über alle Fehler und Schaltungen werden durchschnittlich 72% aller Fehler korrekt diagnostiziert.

Schaltung	**Haftfehler**	**Crosstalk**	**Verzögerungsfehler**	**Wired-And**
p100k	70%	68%	76%	75%
p141k	83%	61%	79%	67%
p239k	80%	76%	85%	82%
p259k	78%	70%	82%	77%
p267k	79%	63%	70%	69%
p269k	72%	66%	74%	75%
p279k	67%	60%	73%	67%
p286k	76%	56%	67%	68%
p295k	66%	45%	47%	54%
p330k	71%	65%	72%	71%
p378k	87%	91%	95%	93%

Tabelle 4.4: Ergebnisse der diagnostischen Auflösung ohne Kompaktierung

Schaltung	**Haftfehler**	**Crosstalk**	**Verzögerungsfehler**	**Wired-And**
p100k	83%	67%	78%	76%
p141k	85%	62%	86%	78%
p239k	86%	77%	85%	83%
p259k	79%	69%	83%	77%
p267k	87%	64%	75%	78%
p269k	83%	64%	78%	85%
p279k	79%	56%	68%	73%
p286k	79%	56%	69%	70%
p295k	73%	51%	53%	58%
p330k	71%	66%	73%	73%
p378k	87%	91%	95%	93%

Tabelle 4.5: Ergebnisse der diagnostischen Auflösung mit Kompaktierung (32 Testmuster pro Sitzung, $L = 8$ Bits)

Tabelle 4.5 zeigt die diagnostische Auflösung, wenn 32 Testmuster zu einer Sitzung zusammengefasst und $L = 8$ Bits pro Sitzung ausgewertet werden. Wie man sieht, wird in nahezu allen Fällen mit Kompaktierung eine höhere diagnostische Auflösung erreicht als ohne, im Durchschnitt 75%. Der Grund hierfür ist die vorgegebene Größe des Fehlerspeichers: Ohne Kompaktierung wird jede Testantwort einzeln ausgewertet und jede fehlerhafte Antwort in den Fehlerspeicher abgelegt. Mit Kompaktierung wird maximal ein Eintrag pro Sitzung in den Fehlerspeicher geschrieben, was zur Folge hat, dass Fehlerinformationen über mehrere Sitzungen in den Fehlerspeicher abgelegt werden können und somit, trotz der Kompaktierung, auch mehr Fehlerinformationen. In Tabelle 4.6 werden die Zahlen noch detaillierter miteinander verglichen. Für jede Schaltung wird eine Verbesserung der diagnostischen Auflösung mit Kompaktierung, im Vergleich zur diagnostischen Auflösung ohne Kompaktierung, erreicht. In einigen Fällen sogar 5,8%, im Durchschnitt 3,1%.

Schaltung	**„Bypass“ Durchschnitt**	**kompaktiert (32 Muster pro Teilsitzung)**	
		Durchschnitt	**Verbesserung**
p100k	72,2 %	76,0%	+3,8
p141k	72,5 %	75,2%	+2,8
p239k	80,8 %	82,8%	+2,0
p259k	76,8 %	77,0%	+0,2
p267k	70,2 %	76,0%	+5,8
p269k	71,8 %	77,5%	+5,8
p279k	66,8 %	69,0%	+2,2
p286k	66,8 %	68,5%	+1,8
p295k	53,0 %	58,8%	+5,8
p330k	69,8 %	70,8%	+1,0
p378k	91,5 %	91,5%	0,0
Gesamt	72,0 %	75,1 %	+3,1

Tabelle 4.6: Vergleich der diagnostischen Auflösungen (ohne/mit Kompaktierung)

Zur Bewertung der Hardwarekosten wurden die zusätzlichen Kosten, die für die Implementierung des Fehlerspeichers nötig sind, ins Verhältnis zu den Kosten eines Speichers gesetzt, der für deterministische Testmuster benötigt wird. Hierzu wird angenommen, dass die Startzustände des TMG kodiert auf dem Chip abgelegt werden. Als Vergleichswerte dienen die Werte aus [HHW+09], da dieses Verfahren eine sehr effiziente Kodierung vorsieht.

Tabelle 4.7 zeigt, dass die Hardwarekosten für Antwort- und Fehlerspeicher vergleichbar mit den Kosten für den Speicher der deterministischen Testmustermenge sind und in fast allen Fällen sogar unter 2% Mehraufwand liegen. Lediglich für die Schaltung p100k liegt der Mehraufwand über 5%. Dies hat allerdings mit der sehr guten Kodierungsmöglichkeit der Testmustermenge zu tun, wie man an dem geringen Speicheraufwand der Testmustermenge erkennen kann.

Schaltung	Startwerte für TMG ([HHW+09]) [KB]	Mehraufwand (%)
p100k	7,25	5,18%
p141k	36,18	1,06%
p239k	17,97	1,98%
p259k	23,54	1,53%
p267k	47,95	0,77%
p269k	47,44	0,78%
p279k	48,37	0,77%
p286k	63,69	0,63%
p295k	nicht bekannt	—
p330k	76,56	0,64%
p378k	nicht bekannt	—

Tabelle 4.7: Hardwarekosten bei 32 Mustern pro Teilsitzung ($L = 8$)

Insgesamt lässt sich erkennen, dass das Diagnoseverfahren auch mit extrem kompaktierten Daten eine hohe diagnostische Auflösung erreicht. Der Speicheraufwand ist vergleichsweise niedrig, vor allem im Vergleich zum Speicher für die deterministische Testmustermenge.

4.4 Zusammenfassung

In diesem Kapitel wurde ein bereits bekanntes diagnostisches Verfahren in den Selbsttest mit Rücksetzpunkten und extremer Kompaktierung integriert. Nach jeder Sitzung wird überprüft, ob sie fehlerhaft oder fehlerfrei war. Im fehlerfreien Fall wird die nächste Sitzung gestartet, im fehlerhaften Fall wird die Sitzung wiederholt. Je nach Ergebnis der wiederholten Sitzung, werden eine oder beide fehlerhafte Signaturen im Fehlerspeicher abgelegt, ein Code für das Verhalten nach der Wiederholung der fehlerhaften Sitzung sowie die entsprechende Sitzungsnummer. Wurde während des Tests ein Fehlverhalten festgestellt, wird dann für jede Sitzung die Auswirkung aller möglichen Fehlerorte auf die Signatur berechnet und die Abweichung von der Referenzsignatur bestimmt. Der Fehler, der die meisten fehlerhaften Signaturen erklärt, wird als Fehlerort angenommen. Somit können permanente Fehler genau diagnostiziert werden, doch auch für nicht-permanente Fehler lassen sich wichtige Informationen für den weiteren Klassifikationsschritt aus dem Diagnoseverfahren ziehen, denn intermittierende Fehler treten an einem Fehlerort auf, während transiente Fehler zufällig an unterschiedlichen Fehlerorten auftreten können (siehe Kapitel 5). In den Experimenten wurde gezeigt, dass gegenüber des Verfahrens ohne Kompaktierung sogar eine höhere diagnostische Auflösung erzielt werden konnte, welches allerdings an der Beschränkung des Fehlerspeichers liegt. Der Mehraufwand für den Fehlerspeicher ist vergleichbar mit dem Mehraufwand, den man für die effiziente Speicherung deterministischer Muster in Kauf nimmt.

5 Klassifikation nicht-permanenter Fehler

Nachdem im vorangegangenen Kapitel ein Diagnoseverfahren vorgestellt wurde, welches anhand der Signatur effizient den Fehlerort bestimmen kann, soll in diesem Kapitel die Klassifikation der nicht-permanenten Fehler in unkritische transiente und kritische intermittierende behandelt werden. Die Unterscheidung ist wichtig, da es sich bei einem transienten Fehlverhalten um einen Fehler handelt, der zufällig aufgrund von externen Störeinflüssen oder Parametervariationen auftritt und durch ein robustes Design während des Betriebs toleriert werden kann. Demgegenüber entsteht ein intermittierender Fehler bei einer Schwachstelle im System, welche sich im Laufe der Zeit zu einem kritischen permanenten Defekt entwickeln kann. Problematisch hierbei ist die Tatsache, dass das Verhalten eines intermittierenden Fehlers dem eines transienten stark ähnelt, wie beispielsweise schon an Hand eines hochfrequenten Spannungsabfalls (engl. *High-Frequency Power-Droop, HFPD*) erläutert. Der Unterschied dabei ist allerdings, dass der Fehlerort hierbei konstant bleibt, während ein transienter Fehler zufällig an unterschiedlichen Orten innerhalb einer Schaltung auftritt. Abbildung 5.1 zeigt diesen Unterschied: Während sich der Fehlerort zum Zeitpunkt t_2 gegenüber dem Zeitpunkt t_1 bei einem transienten Fehler ändert, bleibt dieser bei einem intermittierenden Fehler nahezu konstant.

Bei der Klassifikation der Fehler gibt es prinzipiell einen Konflikt zwischen Produktqualität und Ausbeuteverlusten: Ein Klassifikationsverfahren muss auf der einen Seite „sensibel“ genug sein, einen kritischen intermittierenden Fehler zu erkennen, um so Probleme mit der Produktqualität zu vermeiden, darf aber auf der anderen Seite nicht jedes nicht-permanente Fehlverhalten als kritischen intermittierenden Fehler klassifizieren, was zu Ausbeuteverlusten führen würde.

Um transiente von intermittierenden Fehlern zu unterscheiden, wird in diesem Kapitel ein Klassifikationsverfahren vorgestellt, welches auch in [CGH+13] [GCI+14] publiziert wurde. Da sich ein intermittierender Fehler von einem transienten Fehler vor allem durch das unterschiedliche Verhalten bezüglich seines Fehlerorts unterscheidet, werden bei diesem Klassifikationsverfahren die Informationen aus dem vorangegangenen Test- und Diagnoseprozess (Kapitel 3 und 4) verwertet und mit Hilfe eines Bayesschen Modells weiter analysiert. Bayessche Netze, die für die Auswertung statistischer Daten ein wichtiges Werkzeug darstellen [AG04] [Bar12] [BG07] [Pea88], werden typischerweise bei Klassifikationsproblemen eingesetzt. Problematisch bei diesen Ansätzen ist allerdings die Komplexität: So können die Standard-Verfahren aufgrund der großen Anzahl von möglichen Fehlerorten nur für kleine Schaltungen angewendet werden. Deshalb ist die Idee des in diesem Kapitel vorgestellten Verfahrens, die Informationen, die im Test und der Diagnose

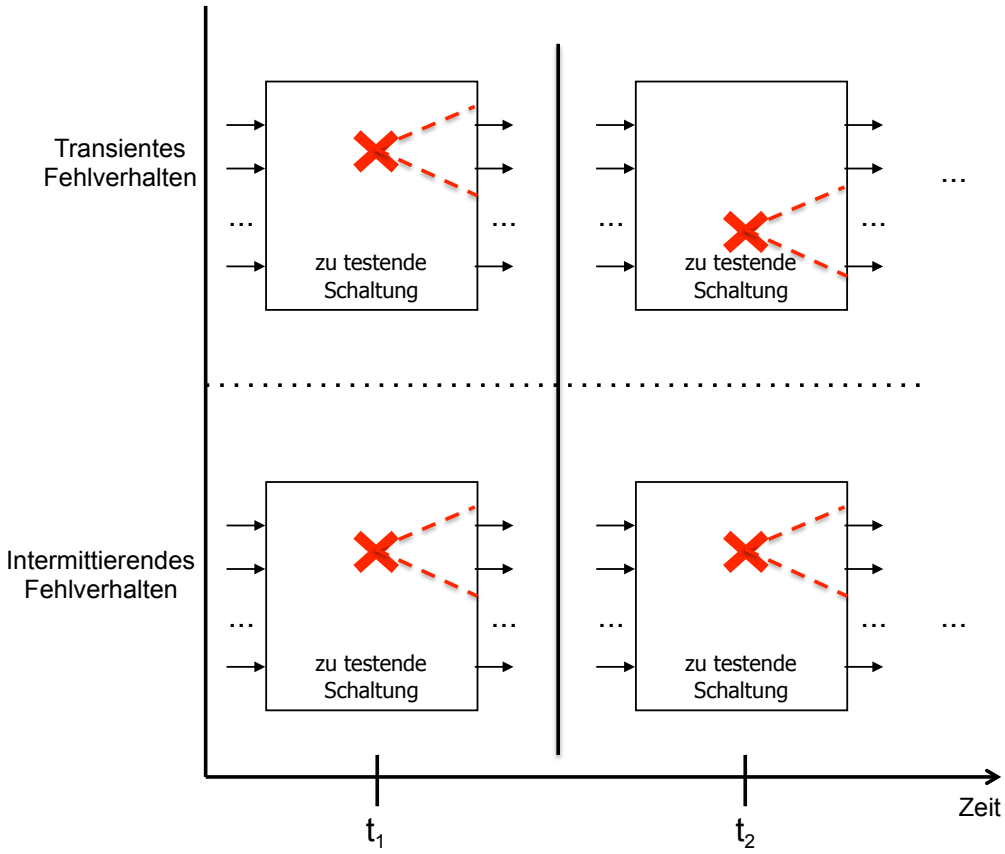


Abbildung 5.1: Nicht-permanentes Fehlverhalten

bereits gewonnen wurden, für die genauere Klassifikation mit Hilfe von Bayesschen Netzen zu nutzen und geschickt auszuwerten. Hierbei wird der Fokus auf die Auswertung des im Tests beschriebenen Fehlerspeichers und auf die aus der Diagnose gezogenen Schlüsse zum Verhalten der unterschiedlichen Fehlerorte gelegt. Mit diesen Informationen wird das Bayessche Netz modelliert. Die Berechnung mit Hilfe des in [Gom11] entwickelten Verfahrens und die Kalibrierung des Netzes sollen anschließend ebenfalls kurz vorgestellt und diskutiert werden.

5.1 Stand der Technik: Fehlerklassifikation

Wie in Kapitel 2.6 beschrieben, kann ein Fehlverhalten prinzipiell permanent, transient oder intermittierend sein. Während permanente Fehler durch Produktionsfehler auftreten, sind transiente und intermittierende Fehler zeitlich und von anderen Faktoren wie Strukturgrößen, Schaltungsaktivität und Versorgungsspannungen abhängig. Da transiente und intermittierende Fehler ein sehr ähnliches Verhalten zeigen, aber deutlich unterschiedliche Auswirkungen auf die Produktqualität einer Schaltung haben, ist das Ziel in der Fehlercharakterisierung, diese Fehler zu erkennen und voneinander zu unterscheiden. So gibt es in der Literatur einige Arbeiten, wie man prinzipiell zeitliche (transiente und intermittierende) Fehler erkennt, während andere Ansätze die Fehler genauer charakterisieren. In [Fec08] wird beispielsweise ein Ansatz beschrieben, der mit verschiedenen

Spannungsgrenzen die Fehlerrate bestimmt, anhand derer dann zwischen den Fehlerarten unterschieden wird. Allerdings wird für diesen Ansatz eine kontinuierliche Messung der Spannung vorausgesetzt, die nicht auf dem Chip selber implementiert werden kann. In [SAK08] wird dagegen eine Selbsttest-Architektur vorgeschlagen, welche die *Soft-Error-Rate* (*SER*) bestimmt. Die Grundidee hierbei ist es, jedes Testmuster zweimal hintereinander anzulegen und die Testantworten miteinander zu vergleichen. Hierzu werden zwei identische MISR-Bausteine zur Testantwortanalyse und ein modifiziertes LFSR zur Testmustergenerierung verwendet. Bei unterschiedlichen Testantworten wird ein zusätzlich implementierter Zähler inkrementiert, da ein transienter Fehler vorliegen muss. Gleichzeitig werden die MISR-Bausteine zurückgesetzt. Die Architektur der Antwortkompaktierung ist in Abbildung 5.2 dargestellt.

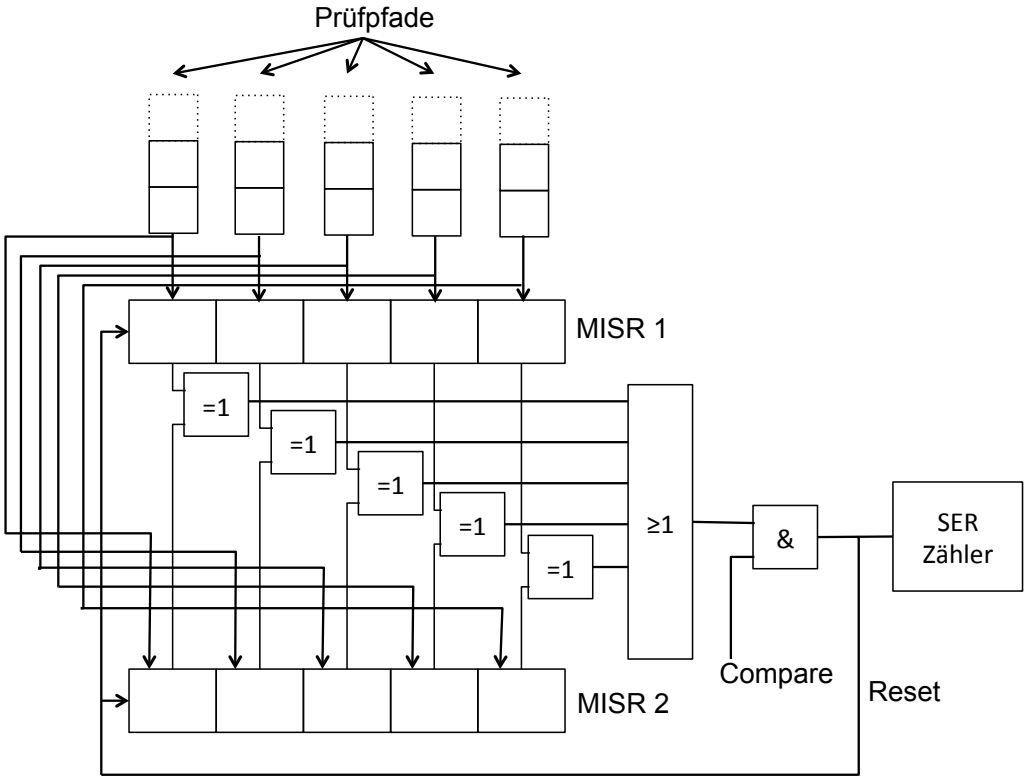


Abbildung 5.2: BIST zur Bestimmung der SER [SAK08]

Für die Testmustererzeugung muss ein LFSR dahingehend angepasst werden, dass ein Testmuster zweimal hintereinander an die Schaltung gegeben werden kann. Hierzu wird jedem LFSR Flip-Flop ein zusätzliches Latch hinzugefügt, in Abbildung 5.3 mit „H“ gekennzeichnet. Der Anfangszustand, der dieses Testmuster generiert hat, wird in diesem zusätzlichen Latch gespeichert (durch das *sample*-Signal). Nachdem das Testmuster in den Prüfpfad geschoben wurde, wird es vom H-Latch in das S-Latch übertragen (durch das *transfer*-Signal), um es nochmals zu erzeugen. Dieses modifizierte Flip-Flop mit drei Latches wird zu einem LFSR zusammengeführt, wie es beispielsweise in Abbildung 5.4 für ein 5-Bit LFSR mit angeschlossenem Phasenverschieber gezeigt ist. Die zusätzlichen Multiplexer unterstützen das Schieben des Anfangszustandes in die Flip-Flops in Abhängigkeit vom Signal *LFSR scan_enable*. Zwar ist bei diesem Ansatz nur ein geringer

zusätzlicher Hardwareaufwand nötig und es ist für nahezu jeden Selbsttest geeignet, allerdings ist die Testzeit doppelt so hoch wie bei einem Standard-Test und es lässt sich nicht zwischen intermittierenden und transienten Fehlern unterscheiden. In [SAK10] wurde die Architektur dahingehend erweitert, dass das Verhalten der Schaltung bei unterschiedlichen Frequenzen analysiert werden kann. Es bleiben dabei aber die Nachteile der langen Testzeit und der geringen Klassifikations-Möglichkeiten bestehen.

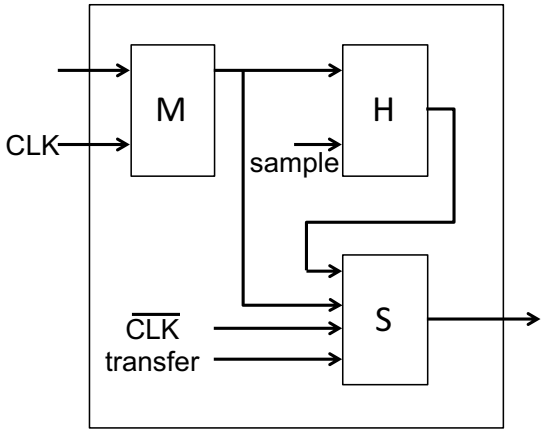


Abbildung 5.3: Modifiziertes LFSR Flip-Flop [SAK08]

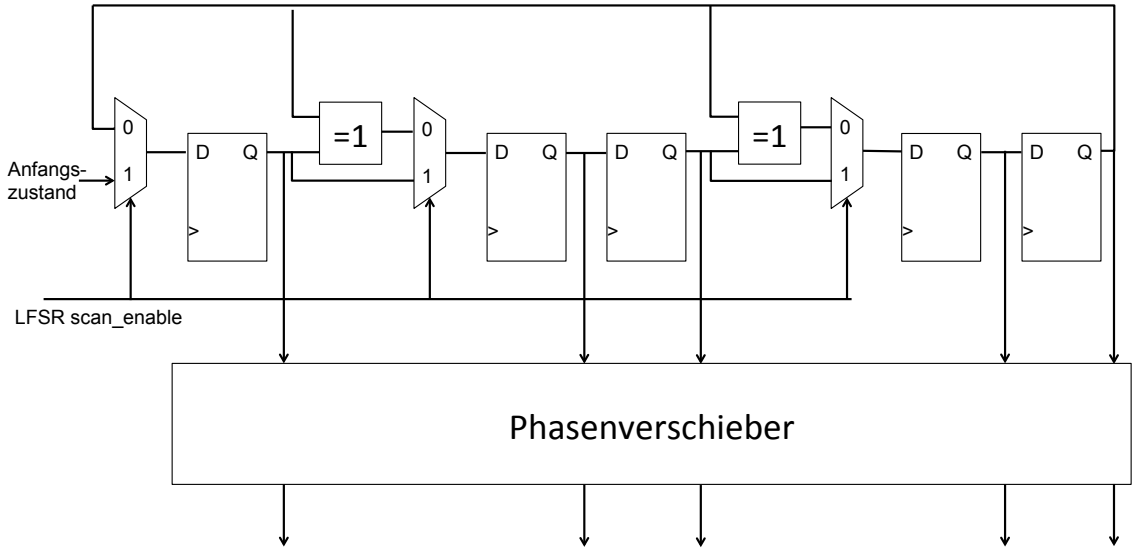


Abbildung 5.4: Modifiziertes LFSR [SAK08]

In vielen ingenieurwissenschaftlichen Bereichen der statistischen Datenanalyse haben sich Bayessche Netze als nützliches Werkzeug durchgesetzt, vor allem bei Klassifikationsproblemen [AG04] [Bar12] [BG07] [Pea88]. Im Kontext des Tests elektronischer Schaltungen gibt es einige Ansätze für Diagnose von analogen Schaltungen [AA01] [LNO06] [KDBK10], aber auch auf System- und Board-Ebene wurden Bayessche Netze eingesetzt [BKK04] [OMCE05] [ZWGC10]. Im nächsten Kapitel werden Bayessche Netze detaillierter erläutert.

5.1.1 Bayessche Netze

In diesem Kapitel werden zunächst die Grundlagen von Bayesschen Netzen vorgestellt und danach auf die Bedeutung für die Fehlerklassifikation und Diagnose eingegangen.

Grundlagen Bayesscher Netze

Ein Bayessches Netz ist ein gerichteter azyklischer Graph, in dem jeder Knoten eine Zufallsvariable Z_i darstellt, die zwei oder mehr Werte annehmen kann. Die Kanten geben den kausalen Zusammenhang zwischen den verbundenen Variablen an und die Stärke dieses Zusammenhangs wird durch bedingte Wahrscheinlichkeiten ausgedrückt. Enthält der Graph die Variablen $Z_1, ..., Z_n$ und ist $\mathbf{Y_i}$ die Menge der Eltern für die Variable Z_i, so lässt sich die Stärke des Zusammenhangs durch die bedingte Wahrscheinlichkeit $p(Z_i|\mathbf{Y_i})$ ausdrücken. Bei Vorgabe dieser Größen erhält man ein globales und konsistentes probabilistisches Modell. Das Produkt

$$p(Z_1, ...Z_n) = \prod_{i=1}^{n} p(Z_i|\mathbf{Y_i}) \tag{32}$$

definiert die gemeinsame Verteilung des Modells. Ist umgekehrt die Verteilung

$$p(Z_1, ..., Z_n)$$

in Gleichung (32) gegeben und berechnet man hieraus die bedingten Wahrscheinlichkeiten $p(Z_i|\mathbf{Y_i})$, so führt das wieder zurück zu den vorgegeben Werten.

Je nach Anwendung lassen sich unterschiedliche weiterführende Berechnungen durchführen. So erlaubt die Bayessche Inversionsformel (oder auch Bayessches Gesetz)

$$p(B|A) = \frac{p(A|B) \cdot p(B)}{p(A)}, \tag{33}$$

welche die Grundlage der Bayesschen Netze darstellt, die Umrechnungen der Wahrscheinlichkeit des Ereignisses A unter der Bedingung des Ereignisses B in die Wahrscheinlichkeit von B unter der Bedingung, dass A eingetreten ist. Mit Hilfe des Gesetzes der totalen Wahrscheinlichkeit lässt sich die Wahrscheinlichkeit des Ereignisses A und der paarweise disjunkten Ereignisse $\{B_i\}(i = 1, ..., n)$, unter der Voraussetzung $A \subset \cup_{i=1}^{n} B_i$ als

$$p(A) = \sum_{i=1}^{n} p(A|B_i)p(B_i) \tag{34}$$

berechnen. Folgendes Beispiel soll dies verdeutlichen:

> **Beispiel:** Abbildung 5.5 zeigt ein Bayessches Netz „Arbeitsweg“. Es wird die Situation einer Person dargestellt, die zur Arbeit fährt und die Pünktlichkeit (Variable Z, Zustände {ja, nein}) davon abhängig ist, ob ein Stau auf dem Arbeitsweg war (Variable Y_1, Zustände {ja,nein}) und ob der Bus erreicht wurde (Variable Y_2, Zustände {ja,nein}).
>
> Die Wahrscheinlichkeit $p(Z = ja) := p(Z)$, ob die Person pünktlich zur Arbeit kommt, lässt sich gemäß Formel (34) als

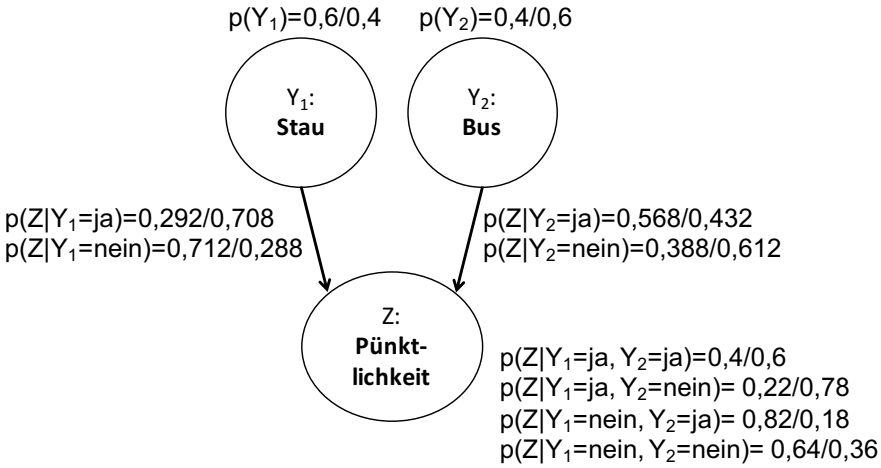


Abbildung 5.5: Beispiel eines Bayesschen Netzes

$$\begin{aligned} p(Z) &= p(Z|Y_1 = ja, Y_2 = ja) \cdot p(Y_1 = ja) \cdot p(Y_2 = ja) \\ &+ p(Z|Y_1 = ja, Y_2 = nein) \cdot p(Y_1 = ja) \cdot p(Y_2 = nein) \\ &+ p(Z|Y_1 = nein, Y_2 = ja) \cdot p(Y_1 = nein) \cdot p(Y_2 = ja) \\ &+ p(Z|Y_1 = nein, Y_2 = nein) \cdot p(Y_1 = nein) \cdot p(Y_2 = nein) \\ &= 0,46 \end{aligned}$$

berechnen.

Ist man beispielsweise interessiert daran, ob ein Stau auf dem Arbeitsweg der Person war, die pünktlich zur Arbeit gekommen ist, so lässt sich mit Gleichung (33) die Wahrscheinlichkeit berechnen als

$$\begin{aligned} p(Y_1 = ja|Z = ja) &= \frac{p(Z = ja|Y_1 = ja) \cdot p(Y_1 = ja)}{p(Z = ja)} \\ &= \frac{0,292 \cdot 0,6}{0,46} \\ &\approx 0,38. \end{aligned}$$

■

Das Ziel bei Bayesschen Netzen ist also, die „*a posteriori*“ Wahrscheinlichkeiten herzuleiten. Da die exakte Berechnung ein NP-vollständiges Problem ist, werden Algorithmen wie das *message passing* eingesetzt, welches auch als Propagierungsprozess bezeichnet wird. Das Modell dieses Prozesses aus [Pea88] wird im Folgenden kurz zusammengefasst.

Im Modell werden jedem Wert z jeder einzelnen Variablen Z drei Wahrscheinlichkeiten zugeordnet:

- $\pi(z)$: Maß für die Stärke der Unterstützung des Zustandes z durch die Vorgängerknoten,
- $\lambda(z)$: Maß für die Stärke der Unterstützung des Zustandes z durch die Nachfolgerknoten,
- BEL(z): entspricht der absoluten Wahrscheinlichkeit des Zustandes z.

Die BEL-Werte lassen sich aus den π- und λ-Werten mit der Formel

$$BEL(z) = \alpha\pi(z)\lambda(z) \tag{35}$$

berechnen, wobei α so gewählt wird, dass

$$\sum_{z} BEL(z) = 1 \tag{36}$$

ist.

Prinzipiell beruht der Propagierungs-Algorithmus darauf, dass jeder Knoten λ-Nachrichten an die Elternknoten und π-Nachrichten an die Kinderknoten sendet und den BEL-Wert aufgrund der eingegangenen Nachrichten aktualisiert. Für einen Knoten Z mit m Kindern $C_1, ..., C_m$ und n Eltern $Y_1, ..., Y_n$ bezeichnet $\pi_Z(y_i)$ die von Y_i und $\lambda_{C_j}(z)$ die von C_j eingehenden Nachrichten (siehe Abbildung 5.6).

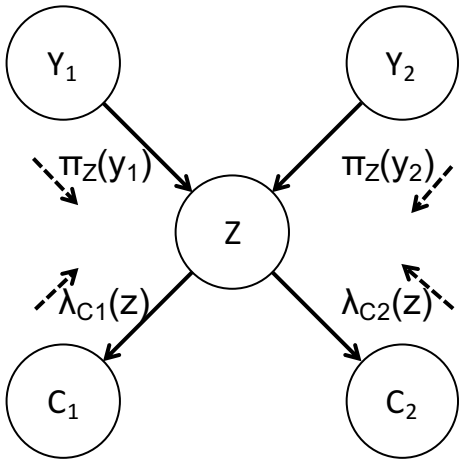


Abbildung 5.6: Propagierungs-Algorithmus im Bayesschen Netz

Dann lässt sich mit der gegebenen bedingten Wahrscheinlichkeitsverteilung $p(z|y_1, ..., y_n)$ der aktuelle BEL Wert nach (35) berechnen mit

$$\lambda(z) = \prod_{j} \lambda_{C_j}(z) \tag{37}$$

und

$$\pi(z) = \sum_{y_1,...,y_n} p(z|y_1, ..., y_n) \prod_{i} \pi_Z(y_i). \tag{38}$$

Die an den Kinderknoten C_j gerichtete π-Nachricht lässt sich dann berechnen als

$$\pi_{C_j}(z) = \alpha \left[\prod_{k \neq j} \lambda_{C_k}(z) \right] \sum_{y_1,...,y_n} p(z|y_1, ..., y_n) \prod_i \pi_Z(y_i) \tag{39}$$

mit Normierungsfaktor α. Die an den Elternknoten Y_i zu sendende λ-Nachricht ergibt sich zu

$$\lambda_Z(y_i) = \beta \sum_z \left[\lambda(z) \sum_{y_k : k \neq i} p(z|y_1, ..., y_n) \prod_{k \neq i} \pi_Z(y_k) \right] \tag{40}$$

mit Normierungsfaktor β.

Vor der Propagierung einer neuen Information werden die Knoten initialisiert, in dem man alle λ-Werte auf 1 setzt, d.h. die π-Werte sind gerade die vorgegebenen bedingten Wahrscheinlichkeiten bzw. bei Knoten ohne Vorgänger die bekannten Verteilungen der entsprechenden Variable. Man kann davon ausgehen, dass neue Informationen stets über einen Knoten ohne Kinder in das Netz eingeführt werden (vgl. [Pea88]). Von diesen Knoten startet der Algorithmus. Falls das Bayessche Netz ein Baum ist, d.h. für jeden Knoten die Menge der Eltern höchstens aus einem Element besteht, so ist die Komplexität des Algorithmus $O(\text{Anzahl der Knoten})$. Sind lokal Eltern $Y_1, .., Y_n$ vorhanden, so wird in Gleichungen (38), (39) und (40) über alle möglichen Kombinationen der Elternvariablen summiert und diese Summation ist exponentiell in n. Eine große Anzahl von Eltern beeinflusst die Komplexität des Algorithmus erheblich.

Bayessche Netze zur Fehlerklassifikation

Bayessche Netze können zur Fehlerklassifikation, beispielsweise zur Fehler-Diagnose, eingesetzt werden. Abbildung 5.7 zeigt hierzu ein Beispiel. Die Knoten ϕ_1 bis ϕ_3 repräsentieren drei mögliche Fehler und werden durch ihre „*a priori*“ Wahrscheinlichkeiten $p(\phi_1)$, $p(\phi_2)$ und $p(\phi_3)$ charakterisiert. Die Punkte Q_1 und Q_2 repräsentieren die Testantworten, beispielsweise die Signaturen, die während des Tests gespeichert wurden. Kann ein Fehler für eine der fehlerhaften Signaturen verantwortlich sein, gibt die Verbindung zwischen den Knoten die bedingte Wahrscheinlichkeit $p(Q_i|\phi_j)$ an, dass die Signatur Q_i

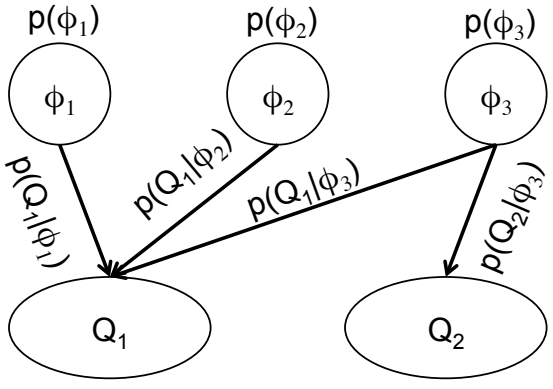


Abbildung 5.7: Bayessches Netz zur Fehler-Diagnose

beobachtet wurde unter der Bedingung, dass ϕ_j aufgetreten ist. Andersherum zeigt die bedingte Wahrscheinlichkeit $p(\phi_j|Q_i)$ an, dass Fehler ϕ_j für die Signatur Q_i verantwortlich ist.

Dieses Netz kann durch die Einfachfehlerannahme vereinfacht werden: Hierzu kann eine mehrwertige Zufallsvariable Φ eingesetzt werden, welche alle möglichen Einfachfehler $\phi_1, ..., \phi_n$ als Werte annehmen kann [PT00]. Abbildung 5.8 zeigt diese Idee für drei Einfachfehler ϕ_1, ϕ_2 und ϕ_3 und zwei fehlerhafte Signaturen Q_1 und Q_2. Hierdurch wird der vorgestellte Propagierungsalgorithmus deutlich vereinfacht, da bei Auftritt einer fehlerhaften Signatur nur noch λ-Nachrichten an den einen Elternknoten Φ geschickt werden müssen.

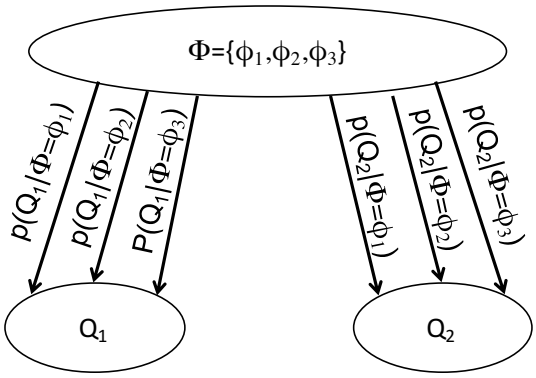


Abbildung 5.8: Bayessches Netz zur Fehler-Diagnose von Einfachfehlern

In der Literatur gibt es Arbeiten wie [Kle09], die mit Hilfe von Bayesschen Netzen zwischen permanenten und intermittierenden Fehlern unterscheiden. Das Verfahren überprüft zu bestimmten Zeitpunkten das System und aktualisiert das Bayessche Netz je nach festgestelltem Verhalten. Die Diagnose wählt interne Beobachtungspunkte aus, so dass die *a posteriori* Wahrscheinlichkeiten für falsche Fehlerorte schnell abnehmen. Für kleine Schaltungen führt dieser Ansatz zwar zum Erfolg, für größere Schaltungen wird das Bayessche Netz allerdings zu groß und ist somit nicht anwendbar, da zu viele Fehler und Fehlerorte in Betracht kommen können.

5.2 Einordnung der Klassifikation in den Gesamtprozess

Die Klassifikation von Fehlern mit Hilfe von Bayesschen Netzen ist problematisch, da bei großen Schaltungen die Anzahl der möglichen Fehlerorte ansteigt und somit die Komplexität des Verfahrens zu hoch wird. Aus diesem Grund werden für die vorgestellte Klassifikation die Erkenntnisse aus dem in Kapitel 3 und 4 beschriebenen Test- und Diagnoseverfahren verwendet, um die Anzahl der möglichen Fehlerorte zu begrenzen und eine erste Abschätzung der Fehlerart zu liefern: Tritt ein Fehler auf, sei er permanent oder nicht-permanent, startet das Diagnoseverfahren (Kapitel 4). Handelt es sich bei dem Fehler um einen nicht-permanenten Fehler, wird mit den gewonnenen Informationen die Klassifikation mit Bayesschen Berechnungen durchgeführt. Im Falle eines nicht-permanenten Fehlers übernimmt das Klassifikationsverfahren folgende Informationen des Test- und Diagnoseverfahrens:

- die Liste der möglichen Fehler $\{\phi_1, ...\phi_n\}$,
- die Anzahl der Muster, die zu einer fehlerhaften Ausgabe geführt haben sowie
- die Anzahl der Muster, die trotz eines Fehlers zu einer fehlerfreien Ausgabe geführt haben.

Aus diesen Informationen kann dann das Bayessche Netz aufgebaut werden.

5.3 Adaptive Klassifikation

Um intermittierende Fehler von transienten zu unterscheiden, werden die Test- und Diagnoseergebnisse aus den vorherigen Kapiteln mit Bayesschen Netzen kombiniert. Die Idee hierbei ist, das Netzwerk adaptiv mit den Test- und Diagnoseergebnissen zu aktualisieren.

Wie bereits beschrieben, ist der Test in N Sitzungen mit jeweils n Testmustern unterteilt. Im Falle eines Fehlers, wenn also die Signatur Q_i der Sitzung T_i nicht der erwarteten Signatur S_i entspricht, wird die Sitzung wiederholt. Die Signatur für die wiederholte Sitzung wird mit R_i gekennzeichnet. Die Anzahl der möglichen Sitzungen ist durch den benutzerspezifischen Parameter W beschränkt. Wie bereits in Kapitel 3 gezeigt, steigt zwar die Qualität des Tests mit steigenden Werten für W, allerdings wird im weiteren Verlauf, wie auch vorher, zur Vereinfachung der Darstellung ohne Beschränkung der Allgemeinheit von $W = 2$ ausgegangen. Zunächst wird im folgenden Kapitel beschrieben, wie der Fehlerspeicher ausgewertet wird. Aus diesen Erkenntnissen wird im darauffolgenden Kapitel das Bayessche Netz aufgestellt, wobei auch kurz auf die Berechnungen eingegangen wird[9]. Da es bei der Klassifikation konträre Ziele gibt, zum einen die Minimierung der Ausbeuteverluste und zum anderen die maximale Produktqualität, wird die Kalibrierung und deren Auswirkung zum Schluss dieses Kapitels diskutiert.

5.3.1 Auswertung des Fehlerspeichers

Zunächst wird nach dem Test der Fehlerspeicher ausgelesen und analysiert. Im Speicher wurden während des Tests neben den fehlerhaften Signaturen auch Informationen über das adaptive Verhalten abgelegt (siehe Abbildung 4.2). Tabelle 5.1 fasst die Kodierungsregeln nochmal zusammen.

Im ersten Schritt wird überprüft, ob eine weitere Klassifikation überhaupt nötig ist. Hierbei wird neben dem adaptiven Verhalten auch die Anzahl der fehlerhaften Sitzungen analysiert:

- Wird für eine Sitzung zweimal dieselbe fehlerhafte Signatur abgespeichert ($R_i = Q_i$), dann wird ein permanenter oder intermittierender Fehler angenommen (Kodierung 00). Es ist keine weitere Klassifikation nötig, das Fehlverhalten ist während des Tests bereits als kritisch eingestuft worden.

[9] Die Berechnungen innerhalb des Bayesschen Netzes wurden mit einem von der Universität Stuttgart entwickelten Verfahren durchgeführt und implementiert. In [Gom11] sind die detaillierten Informationen zu finden, die nicht Gegenstand dieser Arbeit waren.

Verhalten	**Kodierung**	**Interpretation**
$Q_i \neq S_i, R_i = S_i$	10	Transienter oder intermittierender Fehler
$Q_i \neq S_i, R_i \neq S_i, Q_i = R_i$	00	Permanenter oder intermittierender Fehler
$Q_i \neq S_i, R_i \neq S_i, Q_i \neq R_i$	01	1. Intermittierender Fehler, der durch unterschiedliche Muster aktiviert wurde oder 2. ein transienter Fehler

Tabelle 5.1: Adaptives Verhalten für W=2 Wiederholungen

- Ist auch die wiederholte Sitzung fehlerhaft, wobei jedoch eine andere fehlerhafte Signatur abgespeichert wurde ($R_i \neq Q_i$ und $R_i \neq S_i$), ist ebenfalls keine weitere Klassifikation nötig (Kodierung 01). Grund für dieses Fehlverhalten kann zwar auch ein transienter Fehler sein, allerdings ist die Fehlerrate so hoch, dass auch nach der Wiederholung eine fehlerhafte Signatur aufgetreten ist. Auch eine zu hohe transiente Fehlerrate kann kritisch sein, gerade für sicherheitskritische Anwendungen. Sowohl mit den Parametern N (Anzahl Sitzungen), als auch W (Anzahl Wiederholungen), kann der Test auf die gewünschte Toleranz justiert werden (siehe Kapitel 3).

- Zu viele fehlerhafte Sitzungen sprechen ebenfalls dafür, dass es sich um ein kritisches Fehlverhalten handelt. Der entsprechende Schwellenwert T_{max} kann wie folgt abgeschätzt werden: Sei λ die erwartete Fehlerrate für einen transienten Fehler, dann entspricht die Wahrscheinlichkeit, dass eine Sitzung mit n Testmustern aufgrund eines transienten Fehlers fehlerhaft ist, $1-(1-\lambda)^n$. Abhängig von der Anzahl der Wiederholungen W ist die Gesamtanzahl der Sitzungen N_{Gesamt} zwischen N und $W \cdot N$. Die Wahrscheinlichkeit, dass T aus N_{Gesamt} Sitzungen aufgrund eines transienten Fehlers eine fehlerhafte Signatur erzeugen, ist somit

$$\binom{N_{Gesamt}}{T} \left(1-(1-\lambda)^n\right)^T \left((1-\lambda)^n\right)^{N_{Gesamt}-T}. \tag{41}$$

  Hieraus kann dann T_{max} bestimmt werden, so dass die Wahrscheinlichkeit von T_{max} fehlerhaften Sitzungen aufgrund eines transienten Fehlers unterhalb dieses benutzerdefinierten Schwellenwertes liegt. Ist die Anzahl der fehlerhaften Signaturen größer als T_{max}, wird ein intermittierender Fehler angenommen.

In allen drei oben aufgeführten Fällen (Kodierung 00, Kodierung 01 und Anzahl fehlerhafter Sitzungen größer als T_{max}) wird der Chip als kritisch klassifiziert. Für die anderen Fälle wird ein Bayessches Modell aufgestellt, um zwischen transienten und intermittierenden Fehlern zu unterscheiden. Das Bayessche Netz wird hierbei während der Analyse der fehlerhaften Signaturen automatisch, wie im Folgenden beschrieben, entwickelt.

5.3.2 Aufstellen des Bayesschen Netzes

Gegeben sei die vom Diagnoseverfahren identifizierte Liste mit k möglichen Fehlerorten $v_1, ...v_k$ und für jede Sitzung T_i die beobachtete Signatur Q_i sowie im Falle einer Wiederholung auch R_i. Für jeden Fehlerort v wird ein intermittierender Fehler ϕ_v generiert und, ausgehend von den Überlegungen aus Kapitel 5.1.1, dem Wertebereich einer mehrwertigen Zufallsvariablen F hinzugefügt. Zusätzlich wird ein transienter Fehler ϕ_{trans} als „Hintergrundrauschen“ modelliert, das heißt, ein transienter Fehler kann zufällig zu einer bestimmten Zeit auftreten und den Effekt eines intermittierenden Fehlers maskieren oder modifizieren. Dieser wird ebenfalls F hinzugefügt. Jede beobachtete Signatur $Q_1, ...Q_N$ und für jede fehlerhafte Sitzung T_i mit Signatur R_i werden als einzelne *Kinderknoten* eingeführt und mit F verbunden, siehe Abbildung 5.9. Die Beschriftung der Kanten, also die Wahrscheinlichkeiten, dass die Signatur Q_i beobachtet wurde unter der Bedingung, dass Fehler ϕ_j aufgetreten ist, lässt sich formal aufstellen als $p(Q_i|\phi_j)$ bzw. $p(R_i|\phi_j)$. Die Initialisierung, also die *a priori* Wahrscheinlichkeiten, können aus Kenntnissen über den Produktionsprozess gewonnen werden, es kann aber ebenso zunächst von einer gleichverteilten Wahrscheinlichkeit ausgegangen werden[10]. Die bedingten Wahrscheinlichkeiten $p(Q_i|\phi_j)$ bzw. $p(R_i|\phi_j)$ basieren sowohl auf einer probabilistischen als auch auf einer deterministischen Charakterisierung der Fehler. Die Wahrscheinlichkeit eines transienten Fehlers ist dabei $\lambda = \lambda_a \cdot \lambda_p$ mit der Wahrscheinlichkeit λ_a, dass der Fehler aktiviert wurde, und der Wahrscheinlichkeit λ_p, dass das Fehlverhalten zu einem der Ausgänge propagiert wurde. Ein intermittierender Fehler wird mit seiner Aktivierungswahrscheinlichkeit μ, den Mustern, die das Fehlverhalten an den Ausgang propagieren und dem Fehler-Aktivierungs-Profil charakterisiert.

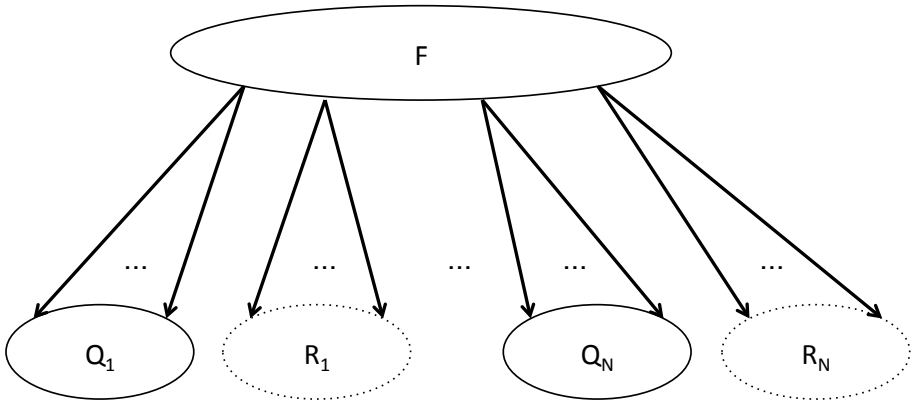


Abbildung 5.9: Bayessches Netz für die adaptive Klassifizierung

Relevante Testmuster: Sei Ψ die Menge aller Testmuster in einer Sitzung und sei v ein möglicher Fehlerort. Weiterhin sei $b \in \{0, 1\}$ die Polarität eines bedingten Haftfehlers, dann beschreibt die Menge $\Psi_d(v) := \{\psi \in \Psi | d\left((\psi|\Psi)\_b\_v\right) \neq 0\}$ die relevanten Testmuster für v und $n_d(v) := |\Psi_d(v)|$ ist die Anzahl der relevanten Testmuster für v.

Fehler-Aktivierungs-Profil: Sei Ψ die Menge aller Testmuster in einer Sitzung, sei Q die beobachtete Signatur und sei v der mögliche Fehlerort. Weiterhin sei $b \in \{0, 1\}$ die

[10] In Kapitel 5.3.3 wird auf die Kalibrierung des Netzes genauer eingegangen.

Polarität eines bedingten Haftfehlers, dann beschreiben die Mengen

$$\Psi_a(v, Q) := \{\psi \in \Psi_d(v) | (\psi|\Psi)\_b\_v \text{ muss aktiviert sein, um } Q \text{ zu erklären}\} \quad (42)$$

und

$$\Psi_i(v, Q) := \{\psi \in \Psi_d(v) | (\psi|\Psi)\_b\_v \text{ darf nicht aktiviert sein, um } Q \text{ zu erklären}\} \quad (43)$$

das Fehler-Aktivierungs-Profil von v, um die beobachtete Signatur Q zu erklären. Die entsprechenden Kardinalitäten werden dann mit $n_a(v, Q) := |\Psi_a(v, Q)|$ und $n_i(v, Q) := |\Psi_i(v, Q)|$ bezeichnet.

Für einen Test mit Testmustermenge Ψ, einer fehlerhaften Signatur $Q \neq S$ und einem intermittierenden Fehler ϕ an Fehlerort v, werden in der Vorverarbeitung (*pre-processing*) die Informationen über die relevanten Testmuster und das Fehler-Aktivierungs-Profil berechnet.

Das folgende Beispiel soll diese Ideen verdeutlichen:

> **Beispiel:** Gegeben sei die bereits bekannte Beispielschaltung in Abbildung 5.10 mit den möglichen Fehlerorten $v = \{v_1, v_2, v_3\}$ und der Testmustermenge $\Psi = \{\psi_1, \psi_2, \psi_3, \psi_4, \psi_5\}$ mit $\psi_1 = (111101)$, $\psi_2 = (111111)$, $\psi_3 = (011010)$, $\psi_4 = (001100)$ und $\psi_5 = (110111)$. Um Anschaulichkeit zu gewährleisten, sei in diesem Beispiel keine Kompaktierung an den Schaltungsausgängen implementiert. Die Wahrheitstabelle ergibt sich im fehlerfreien Fall gemäß der zweiten Spalte (mit *ff* gekennzeichnet) von Tabelle 5.2. Die Polarität eines jeden Fehlers sei ausschließlich $b = 0$. Die fehlerhaften Ausgaben sind den Spalten drei bis fünf von Tabelle 5.2 zu entnehmen. Hieraus lassen sich leicht die relevanten Testmuster $\Psi_d(v)$, für die $d((\psi|\Psi)\_b\_v) \neq 0$ gilt, ablesen als
>
> $$\Psi_d(v_1) = \{\psi_1, \psi_2, \psi_5\}$$
> $$\Psi_d(v_2) = \{\psi_1, \psi_3, \psi_4, \psi_5\}$$
> $$\Psi_d(v_3) = \{\psi_1, \psi_5\}$$
>
> und es ergibt sich die Anzahl $n_d(v)$ der relevanten Testmuster der jeweiligen Fehlerorte $n_d(v_1) = 3$, $n_d(v_2) = 4$ und $n_d(v_3) = 2$.

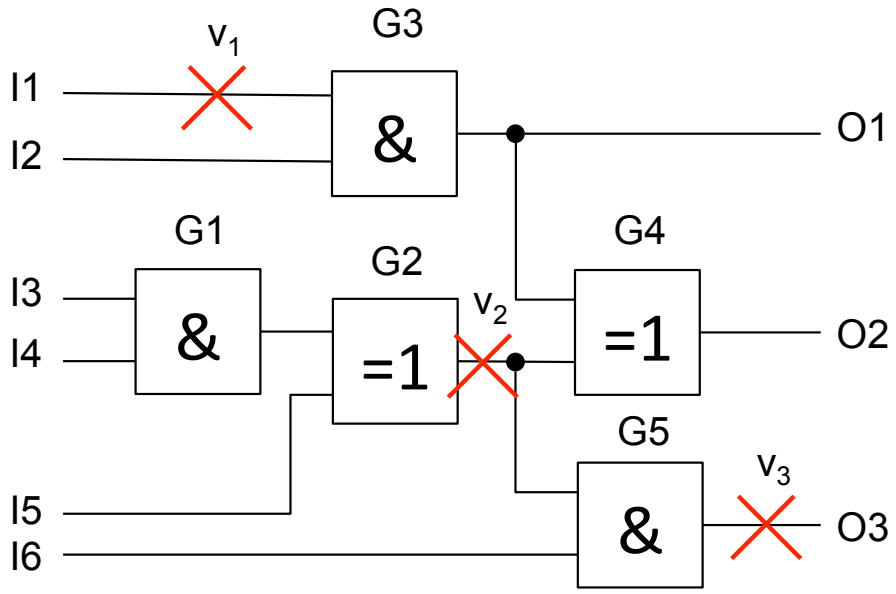


Abbildung 5.10: Beispielschaltung

Eingabe (I1,I2,I3,I4,I5,I6)	**Ausgabe (O1,O2,O3)**			
	ff	0_v_1	0_v_2	0_v_3
$\psi_1 = (111101)$	$(1,0,1)$	$(0,1,1)$	$(1,1,0)$	$(1,0,0)$
$\psi_2 = (111111)$	$(1,1,0)$	$(0,0,0)$	$(1,1,0)$	$(1,1,0)$
$\psi_3 = (011010)$	$(0,1,0)$	$(0,1,0)$	$(0,0,0)$	$(0,1,0)$
$\psi_4 = (001100)$	$(0,1,0)$	$(0,1,0)$	$(0,0,0)$	$(0,1,0)$
$\psi_5 = (110111)$	$(1,0,1)$	$(0,1,1)$	$(1,1,0)$	$(1,0,0)$

Tabelle 5.2: Wahrheitstabelle

Eingabe (I1,I2,I3,I4,I5,I6)	**Testausgabe (O1,O2,O3)**
$\psi_1 = (111101)$	$(1,0,0)$
$\psi_2 = (111111)$	$(0,0,0)$
$\psi_3 = (011010)$	$(0,0,0)$
$\psi_4 = (001100)$	$(0,1,0)$
$\psi_5 = (110111)$	$(1,0,1)$

Tabelle 5.3: Testantworten

Nehmen wir an, die Testantworten seien wie in Tabelle 5.3 gegeben, so dass $Q = (100\ 000\ 000\ 010\ 101)$ ist. Hieraus ergibt sich, dass für Testmuster ψ_1, ψ_2 als auch ψ_3 fehlerhafte Ausgaben zu beobachten waren und bei Testmuster ψ_4 und ψ_5 die Ausgabe fehlerfrei war. Folgende Beobachtungen lassen sich für die jeweiligen Fehlerkandidaten festhalten:

- 0_v_1: Der Fehler 0_v_1 wird nur bei den Testmustern ψ_1, ψ_2 und ψ_5 sichtbar. Da bei ψ_5 eine fehlerfreie Ausgabe beobachtet wird, kann er

durch ψ_5 nicht erkannt werden. Allerdings entspricht Q bei ψ_1 nicht der für diesen Fehler erwarteten Antwort, so dass sich zwei Möglichkeiten ergeben: Entweder wird der Effekt des aktivierten intermittierenden Fehlers von einem zusätzlichen transienten Hintergrundrauschen überlagert, oder das transiente Fehlverhalten ist alleine für die fehlerhafte Signatur verantwortlich und der intermittierende Fehler wurde nicht aktiviert. Die fehlerhafte Ausgabe bei Muster ψ_3 kann ebenfalls nicht durch 0_v_1 allein erklärt werden und kann durch ein (zusätzliches) transientes Hintergrundrauschen bedingt aufgetreten sein. Die Antwort auf Muster ψ_2 entspricht der für diesen Fehler erwarteten, so dass sich $\Psi_a(v_1, Q) = \{\psi_2\}$ und $\Psi_i(v_1, Q) = \{\psi_5\}$ mit Kardinalitäten $n_a(v_1, Q) = n_i(v_1, Q) = 1$ ergeben.

- 0_v_2: Der Fehler 0_v_2 wird durch die Testmuster ψ_1, ψ_3, ψ_4 und ψ_5 erkannt, so dass er für ψ_4 und ψ_5 inaktiv sein muss, da hier eine fehlerfreie Ausgabe beobachtet wurde. Allerdings entspricht auch hier Q bei ψ_1 nicht der für diesen Fehler erwarteten Antwort, so dass ein zusätzliches transientes Hintergrundrauschen oder der transiente Fehler ausschließlich hierfür verantwortlich sein kann, welches auch die fehlerhafte Antwort auf Muster ψ_2 erklärt. Die Antwort auf Muster ψ_3 entspricht der für diesen Fehler erwarteten, so dass sich $\Psi_a(v_2, Q) = \{\psi_3\}$ und $\Psi_i(v_2, Q) = \{\psi_4, \psi_5\}$ mit Kardinalitäten $n_a(v_2, Q) = 1$ und $n_i(v_2, Q) = 2$ ergeben.
- 0_v_3: Der Fehler 0_v_3 lässt sich durch die Testmuster ψ_1 und ψ_5 beobachten, so dass er für ψ_5 inaktiv sein muss, da hier eine fehlerfreie Ausgabe beobachtet wurde. Die fehlerhaften Ausgaben für Muster ψ_2 und ψ_3 kann ebenfalls durch ein zusätzliches oder alleiniges transientes Hintergrundrauschen erklärt werden. Die Antwort auf Muster ψ_1 entspricht der für diesen Fehler erwarteten, so dass sich $\Psi_a(v_1, Q) = \{\psi_1\}$ und $\Psi_i(v_1, Q) = \{\psi_5\}$ mit Kardinalitäten $n_a(v_3, Q) = n_i(v_3, Q) = 1$ ergeben.

■

Wie man erkennt, bleiben nach der Auswertung des Fehlerspeichers (siehe Ausführungen in Kapitel 5.3.1) folgende Fälle übrig:

- Ein intermittierender Fehler kann die aufgetretene fehlerhafte Signatur erklären.
- Ein intermittierender Fehler kann die aufgetretene fehlerhafte Signatur nicht erklären.
- Eine fehlerfreie Signatur ist aufgetreten trotz des intermittierenden Fehlers.

Die bedingten Wahrscheinlichkeiten dieser Fälle, ausgehend von den Informationen aus dem Diagnoseverfahren, können wie folgt berechnet werden:

- Wenn ϕ die fehlerhafte Signatur Q beschreiben kann, wird der Fehlereffekt nicht von einem transienten Rauschen überdeckt, so dass die bedingte Wahrscheinlichkeit

$$p(Q \neq S|\phi) = \mu^{n_a(v,Q)}(1-\mu)^{n_i(v,Q)}(1-\lambda)^n \tag{44}$$

  entspricht. Hierbei ist $p(Q \neq S|\phi)$ das Produkt der Wahrscheinlichkeiten, dass ϕ von allen Testmustern aus $\Psi_a(v,Q)$ aktiviert wurde, ϕ von den verbleibenden $n_i(v,Q)$ relevanten Testmustern nicht aktiviert wurde und kein transienter Fehler während der Sitzung mit n Testmustern aufgetreten ist.

- Erklärt der intermittierende Fehler ϕ am Fehlerort v nicht die fehlerhafte Signatur Q, dann muss entweder der Fehler von einem transienten Rauschen überdeckt worden sein oder der intermittierende Fehler wurde nicht aktiviert und die fehlerhafte Signatur ist nur aufgrund eines transienten Fehlers aufgetreten. Die bedingte Wahrscheinlichkeit $p(Q \neq S|\phi)$ wird dann berechnet mit

$$\begin{aligned} p(Q \neq S|\phi) &= \left(1-(1-\mu)^{n_d(v)}\right)\left(1-(1-\lambda)^n\right) + (1-\mu)^{n_d(v)}\left(1-(1-\lambda)^n\right) \\ &= 1-(1-\lambda)^n, \end{aligned} \tag{45}$$

  wobei $(1-\mu)^{n_d(v)}$ die Wahrscheinlichkeit dafür ist, dass der Fehler während der kompletten Sitzung nicht aktiviert wurde, und $1-(1-\mu)^{n_d(v)}$ ist die Wahrscheinlichkeit, dass der Fehler von einem oder mehreren Mustern aktiviert wurde. Der Ausdruck $1-(1-\lambda)^n$ entspricht der Wahrscheinlichkeit, dass mindestens ein transienter Fehler während der Sitzung aufgetreten ist. Für einen transienten Fehler ϕ_{trans} beschreibt die bedingte Wahrscheinlichkeit $p(Q \neq S|\phi_{trans})$, dass ein transienter Fehler aktiviert wurde und das Fehlverhalten durch mindestens eines der n Testmuster auch an einen der Ausgänge propagiert wurde. Dies kann mit $1-(1-\lambda_\psi)^n$ berechnet werden.

- Ist die Signatur fehlerfrei ($Q = S$), obwohl ein intermittierender Fehler in der Schaltung aufgetreten ist, kann der Fehlereffekt durch ein transientes Hintergrundrauschen maskiert worden sein oder der Fehler wurde nicht aktiviert und es ist auch kein transienter Fehler aufgetreten. Die bedingte Wahrscheinlichkeit $p(Q = S|\phi)$ kann dann berechnet werden als

$$\begin{aligned} p(Q = S|\phi) &= \left(1-(1-\mu)^{n_d(v)}\right)\left(1-(1-\lambda)^n\right) + (1-\mu)^{n_d(v)}(1-\lambda)^n \\ &= 1-(1-\lambda)^n. \end{aligned} \tag{46}$$

Tritt wiederum eine fehlerfreie Signatur $Q = S$ auf, obwohl ein transienter Fehler aufgetreten ist, kann dies nur dadurch erklärt werden, dass das Fehlerverhalten nicht an einen der Ausgänge propagiert wurde, hierfür ist die bedingte Wahrscheinlichkeit $p(Q = S|\phi_{trans})$ und kann als $(1-\lambda_p)^n$ berechnet werden.

Die Aktualisierungen der bedingten Wahrscheinlichkeiten werden, wie im Kapitel 5.1.1 beschrieben, durch einen aus [Pea88] bekannten Lernprozess innerhalb des Bayesschen

Netzes aktualisiert. Hierzu wurde das von der Universität Stuttgart entwickelte Verfahren und Implementierung aus [Gom11] eingesetzt.

Nach der Analyse der kompletten Testdaten und der Berechnung der bedingten Wahrscheinlichkeiten werden mit Hilfe der in Kapitel 5.1.1 beschriebenen Regeln die *a posteriori* Wahrscheinlichkeiten berechnet und die Fehler gemäß ihrer Wahrscheinlichkeiten sortiert. Ist die Wahrscheinlichkeit für einen intermittierenden Fehler am größten, wird das Fehlverhalten entsprechend als intermittierend klassifiziert, sonst als transient.

Bei der Bayesschen Analyse sind in der Praxis die Modell-Parameter nicht bekannt und das Netzwerk muss während des Betriebes kalibriert werden. Das nächste Kapitel diskutiert die verschiedenen Möglichkeiten und Auswirkungen auf die Lösung.

5.3.3 Kalibrierung des Bayesschen Netzes

Die *a priori* Wahrscheinlichkeiten der Fehler im Bayesschen Netz können im Idealfall aus den statistischen Kenntnissen über den Produktionsprozess gewonnen werden. In der Praxis kann dies allerdings nur selten gewährleistet werden, so dass dann die Fehlerorte für intermittierende Fehler als gleichverteilt angenommen werden. Die Bayessche Analyse aus [Gom11] ist gegen diese Ungenauigkeit robust: Die Klassifikation wird dann für verschiedene intermittierende Fehlerraten μ durchgeführt und für jeden Fehler ϕ wird die höchste Wahrscheinlichkeit $p(\phi|Q)$ ausgewählt. Dies erlaubt der Methode einen flexiblen Umgang mit verschiedenen Fehlermechanismen, auch wenn detaillierte Statistiken für die möglichen Defekte nicht verfügbar sind. Die Abschätzung der transienten Fehlerrate $\lambda = \lambda_a \cdot \lambda_p$ ist insofern einfacher, als dass diese aus vorherigen Beobachtungen abgeleitet und die Propagierung eines transienten Fehlers an einem zufälligen Ort mittels Fehlersimulation abgeschätzt werden kann. Diese flexible Anpassung der Modellparameter erlaubt auch eine Flexibilität in der Analyse des Fehlverhaltens je nach gewünschtem Ziel: Sollen die Ausbeuteverluste minimiert werden, fließen die minimalen *a priori* Wahrscheinlichkeiten der intermittierenden Fehler in die Analyse ein. Soll die Produktqualität maximiert werden, werden die maximalen Wahrscheinlichkeiten berücksichtigt. Die Auswirkungen auf das Modell lassen sich wie folgt zusammenfassen: Werden im Modell hohe *a priori* Wahrscheinlichkeiten für intermittierende Fehler angenommen, wird auch ein nur kurzzeitiges transientes Fehlverhalten als intermittierendes Fehlverhalten interpretiert. Die Produktqualität ist maximal, da ein intermittierendes Fehlverhalten auch mit einer geringen Fehlerrate als kritisch eingestuft wird. Auf der anderen Seite kann allerdings auch ein transienter Fehler mit leicht erhöhter Fehlerrate fälschlicherweise als intermittierend interpretiert werden, so dass unnötige Ausbeuteverluste entstehen. Werden dem gegenüber geringe *a priori* Wahrscheinlichkeiten angenommen, können die Ausbeuteverluste minimiert werden. Transiente Fehler mit hohen Fehlerraten werden als unkritisch eingestuft, allerdings können hieraus auch Probleme mit der Produktqualität entstehen, wenn intermittierende Fehler mit geringen Fehlerraten ebenfalls als transient eingestuft werden. Die Definition von *kritischem* und *unkritischem* Fehlverhalten wird somit auch eine Frage der Interpretation der Ergebnisse mit besonderem Fokus auf die spätere Anwendung.

5.4 Experimente und Ergebnisse

In diesem Kapitel werden die Experimente zur Fehlerklassifikation mit Hilfe der vorgestellten Bayesschen Netze beschrieben. Zur Validierung des Verfahrens wurden drei verschiedene Experimente gemacht: Zunächst wurden intermittierende Fehler injiziert, dann wurden Simulationen mit injizierten transienten Fehlern durchgeführt und abschließend wurden Experimente durchgeführt, bei denen überprüft werden sollte, ob bei einem transienten Hintergrundrauschen mit zusätzlichen intermittierenden Fehlern diese Fehler trotzdem richtig klassifiziert und entdeckt werden. Als Simulationsumgebung diente wiederum das Java-Framework ADAMA, in dem die intermittierenden und transienten Fehler gemäß der Modellierung aus Kapitel 3 injiziert werden konnten. Für das Aufstellen und die Berechnungen des Bayesschen Netzes ist die Implementierung aus [Gom11] eingebunden worden. Für alle drei Experimente wurden Simulationen mit 10.000 Testmustern durchgeführt, wobei es sich bei den Testmustern um Zufallsmuster und deterministische Muster, die aus einem kommerziellen Werkzeug zur Testmustererzeugung erzeugt wurden, handelte. Die Testantwortkompaktierung wurde mit einem 48-Bit breiten MISR durchgeführt und der Test wurde in Sitzungen zu jeweils 32 Mustern eingeteilt. Während des Tests wurden, wie in den vorherigen Experimenten auch, nur 8 Bits der jeweiligen Signaturen beobachtet und mit den Sollwerten verglichen. Die Klassifikation mit Hilfe der Bayesschen Netze benutzte für die Modellierung im Netz verschiedene Aktivierungswahrscheinlichkeiten für intermittierende Werte und zwar $0,4; 0,1; 0,001$. Die Charakteristika der untersuchten Schaltungen sind Tabelle 5.4 zu entnehmen.

Schaltung	Anzahl Gatter	Anzahl PPI	Anzahl PPO
p45k	22.414	3.739	2.550
p100k	84.356	5.902	5.829
p141k	152.808	11.290	10.502
p239k	224.597	18.692	18.495
p259k	298.796	18.713	18.495
p267k	239.687	17.332	16.621
p269k	239.771	17.333	16.621
p279k	257.736	18.074	17.835
p286k	332.726	18.351	17.835
p295k	249.747	18.508	18.521

Tabelle 5.4: Schaltungscharakteristika der untersuchten Schaltungen

In den ersten Experimenten wurden intermittierende Fehler an unterschiedlichen, zufälligen Fehlerorten v injiziert. Wie bereits in Kapitel 3 beschrieben, ist die Aktivierung eines intermittierenden Fehlers von den Parametern

- Radius r der Nachbarschaft $N(v)$,
- Schrankenwert τ
- sowie einer Aktivierungswahrscheinlichkeit p_a

abhängig. Hierbei gibt der Radius an, welche Gatter in die Nachbarschaft $N(v)$ des Fehlerortes v aufgenommen werden, und τ bestimmt den Anteil der Gatter mit gleicher Transition, um den Fehler zu aktivieren. Ist diese Bedingung erfüllt, wird am Knoten v ein Bitflip erzeugt, so dass eine Verzögerung der Transition stattfindet. Für die ersten Experimente wurden für den Radius Werte zwischen 1 und 3 angenommen und der Schrankenwert τ konnte 15%, 30% oder 50% annehmen. Um die komplexen Aktivierungsbedingungen zusammenzufassen, wurde zusätzlich eine Aktivierungswahrscheinlichkeit von $0,5$ vorgesehen. Insgesamt 120 zufällige Fehlerorte wurden mit allen möglichen Kombinationen der Parameter r und τ kombiniert. Da die Schaltungen als Netzlisten vorliegen, konnten besonders unrealistische Fehlerorte, also Gatter mit extrem großen Fanouts oder unrealistisch großen Nachbarschaften, die bei der Synthese optimiert werden würden, ausgeschlossen werden[11]. Die beobachteten Fehleraktivierungsraten μ_{exp} reichten von sehr geringen Raten wie 10^{-5} bis hin zu hohen Raten wie $5 \cdot 10^{-1}$. Die maximale Anzahl von fehlerhaften Sitzungen T_{max} wurde für die Experimente auf 10 gesetzt. Nach dem Test und der ersten Einordnung des Fehlverhaltens ergeben sich die Ergebnisse aus Tabelle 5.5.

Hier werden für jede Schaltung zunächst die Anzahl der durchgeführten Experimente in Spalte zwei angegeben. In der dritten Spalte ist die Anzahl der Experimente angegeben, in denen kein fehlerhaftes Verhalten beobachtet wurde und der Fehlerspeicher leer war, obwohl ein intermittierender Fehler injiziert wurde. Im Gegensatz dazu gab es auch einige Experimente, in denen das Fehlerverhalten als permanent eingestuft wurde, da in mindestens einer Sitzung zweimal hintereinander die gleiche fehlerhafte Signatur beobachtet wurde (Kodierung 00). Im Fall, dass in mindestens einer Sitzung zwei unterschiedliche fehlerhafte Signaturen abgespeichert wurden, wurde nach dem Test bereits dieses Fehlverhalten als intermittierend eingestuft (Kodierung 01). Spalte sechs zeigt schließlich die Experimente, in denen in mehr als 10 Sitzungen Fehler aufgetreten sind. In den restlichen Fällen war eine detailliertere Bayessche Klassifikation nötig, siehe Spalte sieben. Die Ergebnisse der vollständigen Klassifikation sind Tabelle 5.6 zu entnehmen.

Hier wird in Spalte zwei die Anzahl der Experimente (Anzahl Exp.) angegeben, in denen ein Fehler erkannt wurde. In Spalten drei und vier kann abgelesen werden, in wie vielen Experimenten das kritische Verhalten korrekt als kritisch klassifiziert wurde: In Spalte drei stehen die Anzahl der Experimente, in denen das Fehlverhalten bereits nach dem Test als kritisch, in Spalte vier steht die Anzahl der Experimente, in denen das Fehlverhalten erst nach der Bayesschen Klassifikation als kritisch eingestuft wurde. Hier werden auch diejenigen Experimente dazu gezählt, in denen das Fehlverhalten nach dem Test als permanent eingeordnet wurde, denn das Fehlverhalten wurde korrekt als kritisch angenommen. Insgesamt ergibt sich dann die Anzahl der Experimente, in denen das Fehlverhalten korrekt als kritisch klassifiziert wurde, in Spalte fünf. Prozentual entspricht das den Werten aus Spalte sechs, die zwischen 95,4% und 99,4% liegen, im Durchschnitt über alle Experimente aber 97,8% beträgt. Allerdings erkennt man, dass die Bayessche Klassifikation nicht alle Fehler korrekt klassifiziert, Tabelle 5.7 analysiert deshalb die Ergebnisse der Bayesschen Klassifikation etwas detaillierter.

[11] Ein Fanout bezeichnet die Anzahl der an den Ausgang eines Gatters angeschlossene Gatter. Eine zu hohe Anzahl an angeschlossenen Gattern würde während der Synthese zu einer Optimierung führen, so dass die Fehlerorte mit solch großen Nachbarschaften aussortiert wurden.

Schaltung	**Anzahl Exp.**	Fehlverhalten klassifiziert als **fehlerfrei**	**perm.**	**interm.**	$> T_{max}$	**Anz. Exp. für Bayes. Klassifikation**
p45k	1044	171	692	140	1	40
p100k	1059	112	807	110	1	29
p141k	1014	116	761	107	2	28
p239k	1050	84	827	116	2	21
p259k	966	54	835	55	0	22
p267k	1053	56	733	234	3	27
p269k	1041	151	741	119	0	30
p279k	1044	227	671	108	0	38
p286k	891	101	693	69	0	28
p295k	1047	271	663	60	0	53

Tabelle 5.5: Klassifikation nach Test für intermittierende Fehler

Schaltung	**Experimente mit Fehler**	**Kritisch eingestuft nach Test (IT)**	**Bayes (IB)**	**insgesamt (IT+IB)**	**Anteil korrekt klassifiziert**
p45k	873	833	11	844	96,7%
p100k	947	918	13	931	98,3%
p141k	898	870	7	877	97,7%
p239k	966	945	15	960	99,4%
p259k	912	890	13	903	99,0%
p267k	997	970	10	980	98,3%
p269k	890	860	12	872	98,0%
p279k	817	779	17	796	97,4%
p286k	790	762	10	772	97,7%
p295k	776	723	17	740	95,4%

Tabelle 5.6: Ergebnisse des gesamten Klassifikationsprozesses bei intermittierenden Fehlern

In Spalte drei ist die Anzahl der Experimente angegeben, in denen nur eine Sitzung fehlerhaft war und Spalten vier und fünf zeigen die Ergebnisse der Bayesschen Klassifikation ((T) für transientes, (I) für intermittierendes Fehlverhalten). Wie man sieht, werden in keinem der Fälle die Fehlverhalten korrekt als intermittierend, also kritisch, eingestuft. Die Spalten sechs bis neun zeigen die Ergebnisse für diejenigen Experimente, in denen zwei und mehr Sitzungen fehlerhaft waren (maximal 9, da $T_{max} = 10$). Hier lässt sich erkennen, dass die Anzahl der korrekten Klassifikationen fast der Anzahl der gesamten Experimente mit diesem Fehlverhalten entspricht. Der Anteil der korrekt klassifizierten Fehler ist in Spalte neun angegeben. Wie man erkennen kann, wurde durch die Bayessche Klassifikation nahezu jeder intermittierende Fehler korrekterweise als solcher erkannt, wenn mehr als eine Sitzung fehlerhaft war. Allerdings wurde jedoch fast jedes Fehlverhalten als unkritisch eingestuft, wenn nur eine Sitzung von diesem Fehlverhalten betroffen war.

Schaltung	**Anzahl**	**1 fehlerhafte Sitzung**			**2-9 fehlerhafte Sitzungen**			
	Exp.	**Exp.**	**T**	**I**	**Exp.**	**T**	**I**	**Anteil I**
p45k	40	29	29	0	11	0	11	100%
p100k	29	15	15	0	14	1	13	92,9%
p141k	28	21	21	0	7	0	7	100%
p239k	21	6	6	0	15	0	15	100%
p259k	22	9	9	0	13	0	13	100%
p267k	27	17	17	0	10	0	10	100%
p269k	30	18	18	0	12	0	12	100%
p279k	38	21	21	0	17	0	17	100%
p286k	28	18	18	0	10	0	10	100%
p295h	53	38	38	0	19	2	17	89,5%

Tabelle 5.7: Detaillierte Ergebnisse der Bayesschen Klassifikation bei intermittierenden Fehlern

In den nächsten Experimenten wurden transiente Fehler in die Schaltungen injiziert, um zu überprüfen, ob dieses Fehlverhalten auch als kritisch eingestuft wird und somit zu Ausbeuteverlusten führt. Transiente Fehler wurden mit Fehlerraten $\lambda \in \{2 \cdot 10^{-3}; 2 \cdot 10^{-4}; 2 \cdot 10^{-5}\}$ an zufälligen Fehlerorten v injiziert und führen bei Aktivierung zu einem Bitflip für einen Taktzyklus. Die Fehlerraten sind deutlich höher als die Fehlerraten, die üblicherweise für durch Strahlung injizierte transiente Fehler angenommen werden. Allerdings werden hiermit auch Parametervariationen simuliert, welche solch hohe oder noch höhere Fehlerraten nach sich ziehen. Für jede Schaltung wurden 140 Experimente mit unterschiedlichen Fehlerorten und Fehlerraten durchgeführt, die Ergebnisse für die Auswertung des Fehlerspeichers, wieder mit $T_{max} = 10$, sind Tabelle 5.8 zu entnehmen.

Schaltung	**Anzahl**	Fehlverhalten klassifiziert als			$> T_{max}$
	Experimente	**fehlerfrei**	**permanent**	**intermittierend**	
p45k	140	11	0	43	14
p100k	140	15	0	15	15
p141k	140	16	0	22	29
p239k	140	13	0	36	32
p259k	140	14	0	15	18
p267k	140	14	0	22	50
p269k	140	13	0	32	45
p279k	140	16	0	31	23
p286k	140	16	0	14	14
p295k	140	13	0	14	17

Tabelle 5.8: Klassifikation nach dem Test für transiente Fehler ($T_{max} = 10$)

Wie man sieht, sind zwar viele Fehler fälschlicherweise als fehlerfrei klassifiziert worden, da es sich allerdings um unkritische transiente Fehler handelt und das Ziel ist, dass dieses Fehlverhalten nicht zu Ausbeuteverlusten führt, ist diese Klassifikation in diesem Fall richtig. Problematisch sind die Experimente, in denen das Fehlverhalten als intermittierend angenommen wurde (Spalte fünf). Allerdings fällt weiterhin auf, dass in sehr vielen Experimenten mehr als zehn fehlerhafte Sitzungen aufgetreten sind. Dies hat den Grund, dass die Fehlerraten sehr hoch waren, so dass es häufig zu fehlerhaften Sitzungen kam. So sind alle Experimente, in denen mehr als zehn fehlerhafte Sitzungen aufgetreten sind, auf die Injektion von transienten Fehlern mit einer Fehlerrate $\lambda = 2 \cdot 10^{-3}$ zurückzuführen. Ist dieses Wissen der Fehlerraten bekannt, kann T_{max} für das Modell angepasst werden.

Tabelle 5.9 zeigt die Ergebnisse der Klassifikation nach dem Test mit dem angepassten Wert $T_{max} = 21$. Die Anzahl der Experimente, in denen eine weitere Bayessche Klassifikation notwendig war, stehen in Spalte sieben. Die Ergebnisse des gesamten Klassifikationsprozesses sind Tabelle 5.10 zu entnehmen. Hier zeigen wieder, analog zu Tabelle 5.6, Spalten drei bis fünf die Anzahl der Experimente, in denen das Fehlverhalten nach dem Test (Spalte drei), nach der Bayesschen Klassifikation (Spalte vier) und insgesamt (Spalte fünf) als kritisch eingestuft wurde. Da allerdings transiente Fehler injiziert wurden, ist diese Klassifikation falsch, so dass sich der Anteil der korrekt klassifizierten Fehler aus Spalte sechs ergibt. Über alle Experimente mit transienten Fehlern wurden 79,7% aller transienten Fehler korrekt als unkritisch eingestuft.

Eine detaillierte Analyse der Bayesschen Klassifikation ist in Tabelle 5.11 zu sehen. Hier lässt sich erkennen, dass sowohl bei nur einer fehlerhaften Sitzung, aber bemerkenswerterweise auch bei mehreren fehlerhaften Sitzungen, die Klassifikation die Mehrzahl der transienten Fehler als unkritisch einstufte. So lässt sich aus Spalte neun ablesen, dass bei mehr als zwei fehlerhaften Sitzungen immer noch zwischen 96,2% und 100 %, im Durchschnitt aber in 99,1% aller Experimente der transiente Fehler korrekt als unkritisch klassifiziert wurde.

Schaltung	**Anzahl Exp.**	Fehlverhalten klassifiziert als **fehlerfrei**	**perm.**	**interm.**	$> T_{max}$	**Anz. Exp. für Bayes. Klassifikation**
p45k	140	11	0	43	0	86
p100k	140	15	0	15	0	110
p141k	140	16	0	22	0	102
p239k	140	13	0	36	0	91
p259k	140	14	0	15	0	111
p267k	140	14	0	22	0	104
p269k	140	13	0	32	0	95
p279k	140	16	0	31	0	93
p286k	140	16	0	14	0	110
p295k	140	13	0	14	0	113

Tabelle 5.9: Klassifikation nach dem Test für transiente Fehler ($T_{max} = 21$)

Schaltung	**Experimente mit Fehler**	**Kritisch eingestuft nach**		**insgesamt (IT+IB)**	**Anteil korrekt klassifiziert**
		Test (IT)	**Bayes (IB)**		
p45k	129	43	1	44	65,9%
p100k	125	15	4	19	84,8%
p141k	124	22	1	23	81,5%
p239k	127	36	1	37	70,9%
p259k	126	15	2	17	86,5%
p267k	126	22	1	23	81,7%
p269k	127	32	2	34	73,2%
p279k	124	31	0	31	75,0%
p286k	124	14	0	14	88,7%
p295h	127	14	0	14	89,0%

Tabelle 5.10: Ergebnisse des gesamten Klassifikationsprozesses bei transienten Fehlern ($T_{max} = 21$)

Schaltung	**Anzahl Exp.**	**1 fehlerhafte Sitzung**			**2-20 fehlerhafte Sitzungen**			
		Exp.	**T**	**I**	**Exp.**	**T**	**I**	**Anteil I**
p45k	86	6	6	0	80	79	1	98,8%
p100k	110	6	6	0	104	100	4	96,2%
p141k	102	6	5	1	96	96	0	100,0%
p239k	91	3	3	0	88	87	1	98,9%
p259k	111	4	4	0	107	105	2	98,1%
p267k	104	2	1	1	102	102	0	100,0%
p269k	95	5	4	1	90	89	1	98,9%
p279k	93	1	1	0	92	92	0	100,0%
p286k	110	6	6	0	104	104	0	100,0%
p295h	113	6	6	0	107	107	0	100,0%

Tabelle 5.11: Detaillierte Ergebnisse der Bayesschen Klassifikation bei transienten Fehlern ($T_{max} = 21$)

Im dritten und letzten Experiment zur Validierung des Klassifikationsprozesses wurden, wie oben beschrieben, intermittierende Fehler injiziert, allerdings bei gleichzeitigem Auftreten von transienten Fehlern mit Fehlerraten von $2 \cdot 10^{-3}, 2 \cdot 10^{-4}$ und $2 \cdot 10^{-5}$. Hiermit sollte herausgefunden werden, ob intermittierende Fehler bei gleichzeitigem transienten Hintergrundrauschen trotzdem als kritisch identifiziert werden würden. Um die Ergebnisse mit denen des ersten Experiments, bei dem nur intermittierende Fehler injiziert wurden, vergleichen zu können, wird T_{max} wieder auf 10 gesetzt. Tabelle 5.12 zeigt die Klassifikation, die aufgrund des Tests entstanden ist. Interessant zu beobachten ist die Tatsache, dass jetzt wesentlich weniger Schaltungen als fehlerfrei klassifiziert wurden. Dies liegt an der Tatsache, dass zusammen genommen deutlich mehr Fehler injiziert wurden. Die Anzahl der Fälle, in denen eine detaillierte Bayessche Klassifikation nötig war, kann Spalte sieben entnommen werden.

Schaltung	**Anzahl Exp.**	Fehlverhalten klassifiziert als			$> T_{max}$	**Anz. Exp. für Bayes. Klassifikation**
		fehlerfrei	**perm.**	**interm.**		
p45k	1044	51	696	157	25	115
p100k	1788	44	1.369	255	19	101
p141k	1014	36	761	133	30	54
p239k	1050	31	817	143	32	27
p259k	966	32	832	62	7	33
p267k	1053	65	734	159	52	43
p269k	1041	54	737	178	36	36
p279k	1044	73	670	159	36	106
p286k	891	57	693	76	10	55
p295h	1047	80	665	85	41	176

Tabelle 5.12: Klassifikation nach dem Test für intermittierende Fehler mit gleichzeitigem transienten Hintergrundrauschen

Es ist nicht überraschend, dass bei allen Schaltungen, im Vergleich zu den ersten Experimenten mit der Injektion von intermittierenden Fehlern ohne transientem Hintergrundrauschen (siehe Tabelle 5.5), wesentlich mehr Experimente eine detailliertere Bayessche Klassifikation benötigten. Die Ergebnisse der gesamten Klassifikation sind Tabelle 5.13 zu entnehmen.

Schaltung	**Experimente mit Fehler**	**Kritisch eingestuft nach**		**insgesamt (IT+IB)**	**Anteil korrekt klassifiziert**
		Test (IT)	**Bayes (IB)**		
p45k	993	878	20	898	90,4%
p100k	1.744	1.643	47	1.690	96,9%
p141k	978	924	30	954	97,5%
p239k	1.019	992	12	1.004	98,5%
p259k	934	901	13	914	97,9%
p267k	988	945	27	972	98,4%
p269k	987	951	19	970	98,3%
p279k	971	865	55	920	94,7%
p286k	834	779	19	798	95,7%
p295h	967	791	28	819	84,7%

Tabelle 5.13: Ergebnisse des gesamten Klassifikationsprozesses bei intermittierenden Fehlern mit gleichzeitigem transienten Hintergrundrauschen

Man kann zwar erkennen, dass insgesamt immer noch sehr viele intermittierende Fehler korrekterweise als kritisch eingestuft wurden, zwischen 84,7% und 98,5%, im Durchschnitt über alle Experimente 95,4%, diese allerdings leicht geringer sind als bei den ersten Experimenten (Verschlechterung um 2,4%). Dies hat den Grund, dass die transienten Fehler mit einer vergleichsweise hohen Fehlerrate und somit, in einigen Fällen, mit einer ähnlichen Auftrittswahrscheinlichkeit wie die intermittierenden Fehler injiziert wurden, was die Klassifikation stark erschwert. Tabelle 5.14 zeigt, im bereits bekannten Format,

die detaillierte Auswertung der Bayesschen Klassifikation. Auch hier sieht man, dass bei nur einer fehlerhaften Sitzung das Fehlverhalten immer als transient eingestuft wurde. Bei mehr fehlerhaften Sitzungen führt das transiente Hintergrundrauschen zwar zu schlechteren Ergebnissen, in zwei Fällen (p45k und p295k) sogar unter 20% korrekt klassifizierte Fehler, bei den übrigen Schaltungen führt die Bayessche Klassifikation jedoch im Durchschnitt über alle Experimente in 48,8% aller Fälle zu einer korrekten Klassifikation des Fehlverhaltens.

Schaltung	**Anzahl**	**1 fehlerhafte Sitzung**			**2-9 fehlerhafte Sitzungen**			
	Exp.	**Exp.**	**T**	**I**	**Exp.**	**T**	**I**	**Anteil I**
p45k	115	8	8	0	107	87	20	18,7%
p100k	101	7	7	0	94	47	47	50,0%
p141k	54	2	2	0	52	22	30	57,7%
p239k	27	4	4	0	23	11	12	52,2%
p259k	33	4	4	0	29	16	13	44,8%
p267k	43	2	2	0	41	14	27	65,9%
p269k	36	4	4	0	32	13	19	59,4%
p279k	106	3	3	0	103	48	55	53,4%
p286k	55	8	8	0	47	28	19	40,4%
p295h	176	10	10	0	166	138	28	16,9%

Tabelle 5.14: Detaillierte Ergebnisse der Bayesschen Klassifikation bei intermittierenden Fehlern mit gleichzeitigem transienten Hintergrundrauschen

Insgesamt kann festgehalten werden, dass mit einer kombinierten Klassifikation von Test, Diagnose und Bayesscher Klassifikation in mehr als 98% aller Fälle ein intermittierendes Verhalten korrekt als kritisch eingestuft wurde.

5.5 Zusammenfassung

In diesem Kapitel wurde ein Klassifikationsprozess basierend auf Bayesscher Analyse vorgestellt, der nach dem Test den Fehlerspeicher ausliest und mittels der Testergebnisse das Fehlverhalten in transient und intermittierend einteilen kann. Hierzu werden Informationen aus dem Diagnoseprozess herangezogen. Es werden nicht nur die möglichen Fehlerorte, also die Größe des Bayesschen Netzes minimiert, sondern auch hilfreiche Informationen über die relevanten Testmuster und das Fehler-Aktivierungs-Profil bereitgestellt. Während der Bayesschen Analyse werden dann schrittweise die fehlerhaften Signaturen ausgewertet und die Wahrscheinlichkeit bestimmt, dass ein intermittierender Fehler an Fehlerort v für dieses Fehlverhalten verantwortlich ist. Nach der Auswertung des Fehlerspeichers kann dann mit einer gewissen Wahrscheinlichkeit angegeben werden, ob ein intermittierender Fehler vorliegt oder es sich doch um einen transienten Fehler handelt, der für eine robuste Schaltung die Zuverlässigkeit nicht beeinflussen würde.

Allerdings stellt sich hierbei auch die Frage, ab wann ein transienter Fehler kritisch oder auch ein intermittierender Fehler unkritisch ist. Durch geschickte Auswahl der *a priori* Wahrscheinlichkeiten kann die Analyse, je nach Wunsch, versuchen, möglichst auch

solche Fehler als kritisch einzustufen, die eine hohe transiente Fehlerrate haben. Entsprechend kann es auch für gewisse Anwendungen interessant sein, selten auftretende intermittierende Fehler noch als vertretbar anzunehmen. Auch dieser Fall kann mit Hilfe der passenden Parameter abgedeckt werden. Zusätzlich führen transiente Fehler nicht zu unnötigen Ausbeuteverlusten. Durch Anpassung der *a priori* Wahrscheinlichkeiten lässt sich die Abwägung zwischen Ausbeute und Produktqualität je nach Anwendung anpassen: So führen geringere *a priori* Wahrscheinlichkeiten der intermittierenden Fehler zwar zu einer höheren Produktqualität, da auch Fehler mit geringeren Fehlerraten korrekt als kritisch eingestuft werden, auf der anderen Seite führt dies auch zu höheren Ausbeuteverlusten, da transiente Fehler mit höheren Fehlerraten ebenfalls als kritisch eingestuft werden könnten. Durch höhere *a priori* Wahrscheinlichkeiten verhält es sich genau entgegengesetzt: Transiente Fehler, auch solche mit höheren Fehlerraten, werden zwar besser erkannt, was zu einer höheren Ausbeute führt, allerdings steigt die Gefahr, dass intermittierende Fehler nicht mehr als kritisch eingestuft werden und somit die Produktqualität verschlechtern. Je nach Anwendungsfall kann so die Klassifikation angepasst werden, siehe die Diskussion in Kapitel 5.3.3.

Die Experimente haben gezeigt, dass durch die kombinierte Klassifikation, bestehend aus Testauswertung und Bayesscher Analyse, intermittierende Fehler mit einer durchschnittlichen Genauigkeit von über 97% korrekt als kritisch eingestuft werden. Auf der anderen Seite werden aber auch transiente Fehler als solche erkannt, so dass unnötige Ausbeuteverluste minimiert werden. Auch bei kombinierten Fehlern, also bei intermittierenden Fehlern mit gleichzeitigem transienten Hintergrundrauschen, können noch durchschnittlich 95,4% der intermittierenden Fehler korrekt klassifiziert werden.

Im nächsten Kapitel werden die Schwachstellen im System, die sich als kleine Verzögerungsfehler auswirken und die von Standard-Testverfahren ohne weitere Anpassungen nicht erkannt werden können, behandelt.

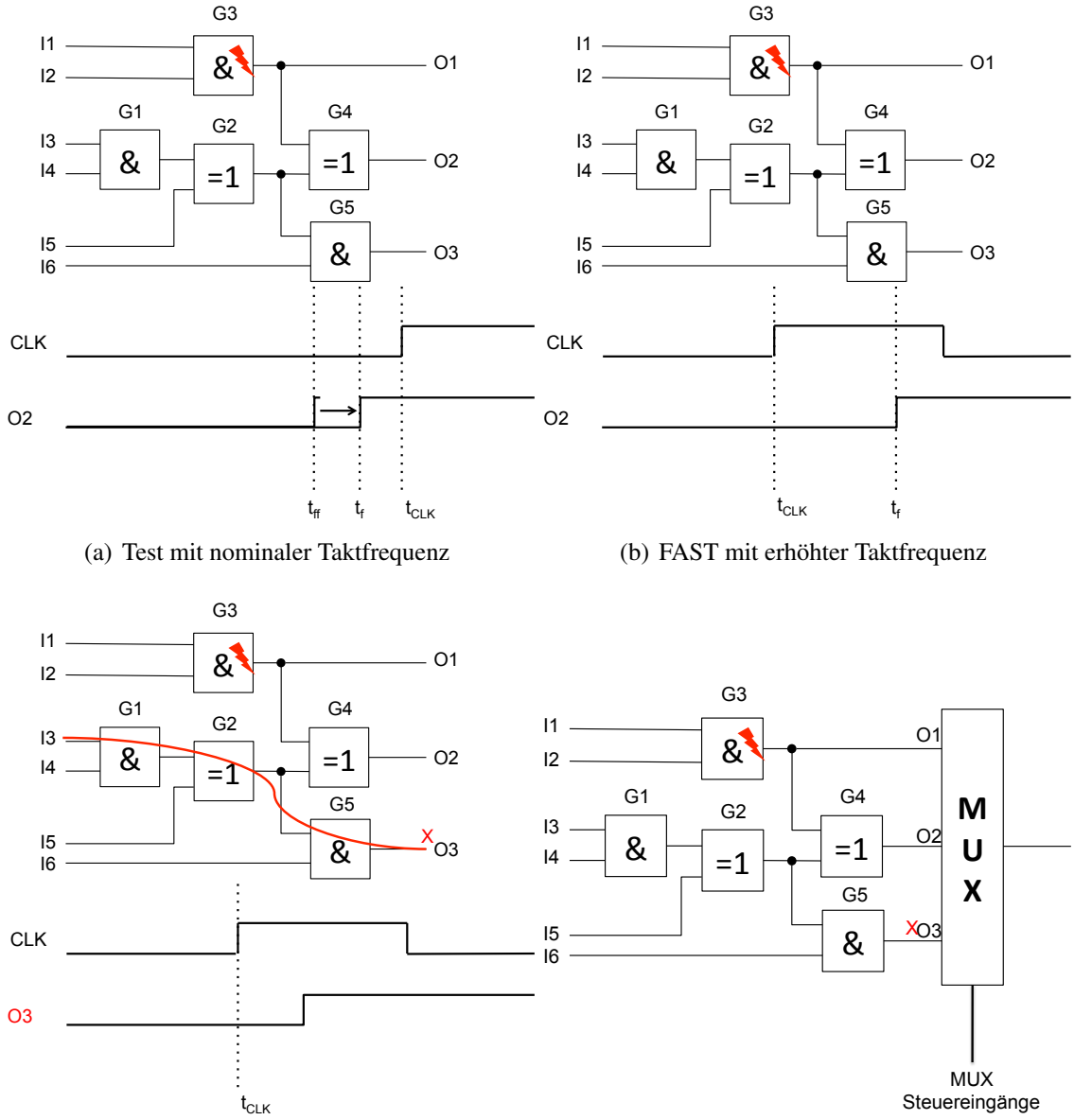
6 FAST-BIST: Faster-than-at-Speed-Test für kleine Verzögerungsfehler im Selbsttest

Kleine Verzögerungsfehler (engl. *Small Delay Defects, SDDs*) sind schwierig zu testen und es müssen umfangreiche und aufwendige Verfahren zur Testmustererzeugung eingesetzt werden, um geeignete Testmusterpaare zu finden. Allerdings treten in Schaltungen auch versteckte kleine Verzögerungsfehler auf, die sich nur über Pfade, deren Pufferzeit größer als die Fehlergröße ist, propagieren lassen: Das Fehlverhalten kann mit einem Standard-Test nicht erkannt werden. Es ist wichtig, auch einen solchen Fehler bereits im Produktionstest zu erkennen, da er ein Indiz für einen vorzeitigen Ausfall des Systems sein kann: So kann er, wie in Experimenten in [KKK+10] gezeigt, im Laufe des Produktlebenszyklus zu schwerwiegenden Fehlverhalten führen. Hier wurden gezielt Transistoren mit dünner Gate-Oxidschicht einem Stresstest mit hohen Versorgungsspannungen unterzogen und das Verhalten der Schaltung bis zur Auswirkung des Fehlverhaltens an den Schaltungsausgängen ausgewertet. Es ließ sich beobachten, dass sich vor dem Systemausfall ein Anstieg der Verzögerungszeit des Transistors und somit auch eine höhere Verzögerung der Transition am Schaltungsausgang einstellte. Somit dürfen auch kleine Verzögerungsfehler nicht ignoriert werden.

Eine Möglichkeit, das Fehlverhalten eines versteckten kleinen Verzögerungsfehlers an den Ausgang einer Schaltung zu treiben, ist ein Test mit erhöhter Betriebsfrequenz, welcher *Faster-than-at-Speed-Test* (*FAST*) genannt wird [YHIF11] [FLL12] [ATJ06] [TA08]. Im Gegensatz zum Speed-Binning, bei dem der kritische Pfad überprüft wird, um die Funktionalität der Schaltung sicherzustellen, wird im FAST das Ziel verfolgt, die kurzen Pfade durch Anlegen von hohen Frequenzen an den Ausgängen sichtbar zu machen. Dies macht zum einen die Testerzeugung mittels externem Testautomaten sehr teuer. Zum anderen wird die Messung durch störende Kapazitäten und andere elektrische Effekte beeinflusst [TA08].
Um diese Probleme zu vermeiden wird in diesem Kapitel erstmals ein FAST vorgestellt, der sich

- im Selbsttest
- kompatibel mit der STUMPS-Architektur
- mit minimaler Hardware und minimaler zusätzlicher Testzeit

implementieren lässt.



(a) Test mit nominaler Taktfrequenz

(b) FAST mit erhöhter Taktfrequenz

(c) Unbekannte Werte bei einem langen Pfad

(d) Filterung mit Multiplexer

Abbildung 6.1: Beispielschaltung mit SDD an Gatter G3

Um FAST im Selbsttest durchzuführen, müssen folgende Problemstellungen gelöst werden: Zum einen müssen die Taktperioden, mit denen der Test durchgeführt wird, auf dem Chip selber erzeugt werden, so dass neben der geschickten Auswahl der Taktperioden auch die technische Umsetzung gewährleistet sein muss. Zum anderen können, bedingt durch die verkürzte Taktperiode, unbekannte Werte an den langen Pfaden auftreten, die bei typischerweise im Selbsttest eingesetzten XOR-basierten Kompaktierern zu fehlerhaften Signaturen führen.

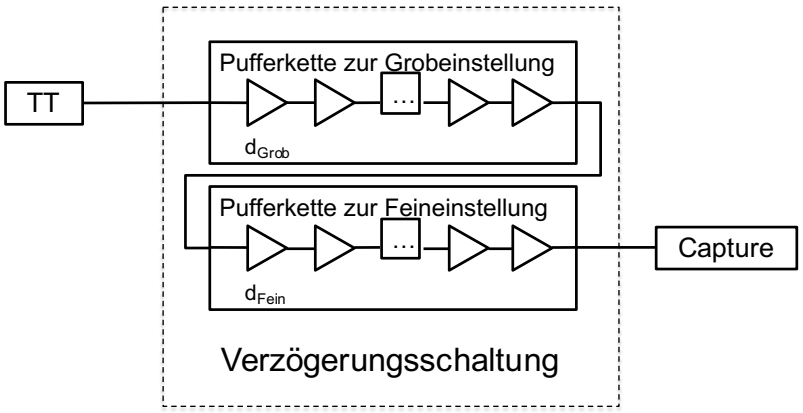
Die bereits aus Kapitel 2 bekannte Beispielschaltung soll diese Idee verdeutlichen: Tritt am Ausgang des Gatters G3 ein Verzögerungsfehler auf, dessen Verzögerung d_g allerdings nicht zu einem Fehlverhalten am Ausgang O2 führt (und somit auch nicht an O1,

welcher aber zur Veranschaulichung außen vorgelassen wird), spricht man von einem SDD, siehe Abbildung 6.1(a). Gibt es allgemein keine Eingangsbelegung, welche dieses Fehlverhalten entdeckt, spricht man von einem *versteckten kleinen Verzögerungsfehler*. Beim FAST wird nun mit einer höheren Frequenz getestet, so dass eine negative Pufferzeit entstehen kann. Abbildung 6.1(b) zeigt das Verhalten bei der doppelten Frequenz: Die Pufferzeit ist nun negativ, das Fehlverhalten am Ausgang O2 wird erkannt. Allerdings gibt es Probleme mit den Pfaden, die länger sind als die FAST-Taktperiode, wie z.B. der rot eingezeichnete Pfad in Abbildung 6.1(c). Dieser Pfad benötigt mehr Zeit als die Dauer der kurzen FAST-Taktperiode. Der Ausgang O3 hat noch keinen stabilen Wert und kann nicht ohne Weiteres vorhergesagt werden: Er entspricht einem unbekannten Wert ('X'). Ein unbekannter Wert ist allerdings problematisch für XOR-basierte Kompaktierer, wie sie in heutigen Selbsttest-Schaltungen eingesetzt werden (vgl. 2.4.2).

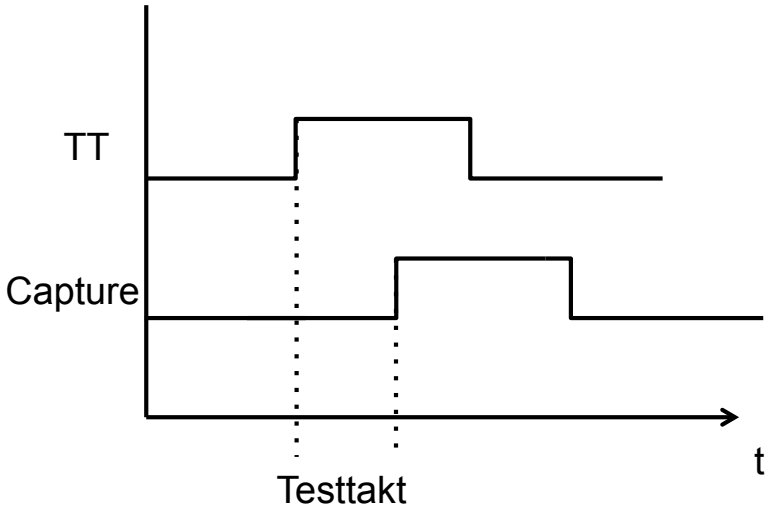
Diese Probleme aufgreifend wird in diesem Kapitel zunächst eine geeignete Heuristik zur Bestimmung geeigneter Testfrequenzen für Standard-Transitionsfehler-Testmusterpaare beschrieben. Hierzu werden zunächst die relevanten Fehler und deren Erkennungsbereiche mittels Simulation bestimmt, bevor dann die Heuristik ausgeführt wird, welche die Fehler in eine minimale Anzahl von FAST-Gruppen einteilt. Weiterhin wird eine programmierbare Lösung der Testantwortkompaktierung vorgestellt, welche ein X-Canceling MISR [Tou07] mit einem kleinen zusätzlichen Speicher für Zwischensignaturen und einer anschließenden Analyse kombiniert. Auch hierfür sind einige vorverarbeitende Prozesse und Simulationen nötig, die ebenfalls im Folgenden beschrieben werden. Die in diesem Kapitel erarbeiteten Lösungsstrategien wurden im Rahmen einer Abschlussarbeit implementiert [Kam14] und in [HIK+14] publiziert. Zunächst wird im nächsten Kapitel jedoch ein Überblick über den Stand der Technik beim Test kleiner Verzögerungsfehler gegeben.

6.1 Stand der Technik: Testverfahren für kleine Verzögerungsfehler im Selbsttest

Versteckte kleine Verzögerungsfehler können nicht mit der nominalen Frequenz erkannt werden und stellen somit eine Besonderheit dar. Eine Möglichkeit, das Fehlverhalten an den Ausgang einer Schaltung zu treiben, ist die Erhöhung der Taktfrequenz [YHIF11] [FLL12] [ATJ06] [TA08]. Wird der FAST auf dem Chip selber implementiert, kann die FAST-Testfrequenz, wie in [PB06] und [TA08], durch Manipulation der Phasenregelschleife oder durch Einbringung eines programmierbaren Taktgenerators (engl. *Programmable On-Chip Delay-Generator*, *PCG*) erzeugt werden. Während bei der Manipulation der Phasenregelschleife, wie sie z.B. bei den *ColdFire*®-Prozessoren von Motorola angewandt wird [MF00], nur eine geringfügige Änderung in der Implementierung nötig ist, kann sie jedoch nur bedingt sehr hohe und feingranulare Frequenzen generieren. Es werden spezielle *Chop-Register* verwendet, um schnellere Taktpulse aus dem Ausgang der Phasenregelschleife zu erzeugen, allerdings sind nur ganze Vielfache der ursprünglichen Taktfrequenz einstellbar. Dies wird zwar bei der Lösung mit dem PCG gelöst, allerdings ist hier mehr zusätzliche Hardware auf dem Chip nötig. So wird, wie in Abbildung 6.2(a) gezeigt, ein vorgegebenes Referenzsignal (*Test Trigger (TT)- Signal*) durch eine Schaltung verzögert. Der eigentliche Testtakt wird aus der Differenz zwischen der ansteigenden Flanke des Referenzsignals und der ansteigenden Flanke des verzögerten



(a) Schematischer Aufbau



(b) Definition Testtakt

Abbildung 6.2: Takterzeugung mit einem Programmable Capture Generator (PCG) [TA08]

Ausgangssignals der Schaltungsanordnung, in Abbildung 6.2(b) mit *Capture* benannt, generiert. Die Verzögerungsschaltung setzt sich aus zwei Pufferketten zusammen, wobei eine Pufferkette zur Grobeinstellung und eine Pufferkette zur Feineinstellung des Testtaktes benutzt werden kann. Die Verzögerungszeiten der Pufferketten lassen sich durch die Anzahl der Pufferelemente, die in Reihe geschaltet werden, einstellen und während des Betriebs verändern. Hierdurch lassen sich feingranularere und höhere Testfrequenzen einstellen, allerdings muss mehr Hardware eingesetzt werden.

Um die Anforderungen an die Hardware möglichst gering zu halten, wird beim FAST versucht, bereits bei der Testmustererzeugung SDDs möglichst über lange Pfade zu propagieren, um die FAST-Frequenz so niedrig wie möglich zu halten. Weiterhin werden die Fehler in Frequenzgruppen eingeteilt, um ein häufiges Wechseln der Testfrequenz zu vermeiden. Gleichwohl werden Pfade aktiviert, die bei der höheren Frequenz die Berechnung noch nicht beendet haben. Aufgrund von möglichen kurzzeitigen Pegelwechseln (*Glitch*) können die alten Werte überschrieben werden, so dass hierbei unbekannte Werte auftreten

(siehe Beispiel in Abbildung 6.1).

Wie bereits in Kapitel 2.4 beschrieben, gibt es in der Literatur bereits Ansätze für X-tolerante Kompaktierer wie X-Filter [SC05] oder X-Compact [MK04], aber auch Kompaktierer, die das Auftreten der unbekannten Werte analysieren und diesen entgegenwirken können, wie X-Masking [TWV+04] und X-Canceling MISR [Tou07]. Allerdings können zum einen nur eine sehr geringe Anzahl unbekannter Werte verarbeitet werden, es lassen sich aber hohe Kompaktierungsraten mit hoher Fehlerabdeckung erreichen. Zum anderen ändern sich die X-Raten während eines FAST sehr stark, abhängig von der Frequenz und der Verteilung der aktivierten langen Pfade. Zusätzlich kann durch Parametervariation und Änderung der Testmustermenge die Menge der SDDs variieren. Somit muss ein X-toleranter Kompaktierer sehr flexibel sein und eine hohe Anzahl von unbekannten Werten verarbeiten können, womit keine Standard-Technik aus der Literatur ohne weitere Anpassung angewandt werden kann.

In [SHQ10] wird eine Kompaktierung eingeführt, die speziell für den FAST-Ansatz entwickelt wurde. Dort werden mehrere Prüfpfade an einen Multiplexer angelegt und mittels Steuersignalen wird jeweils ein Prüfpad aktiviert sowie mit dem Ausgang verbunden, siehe Abbildung 6.3.

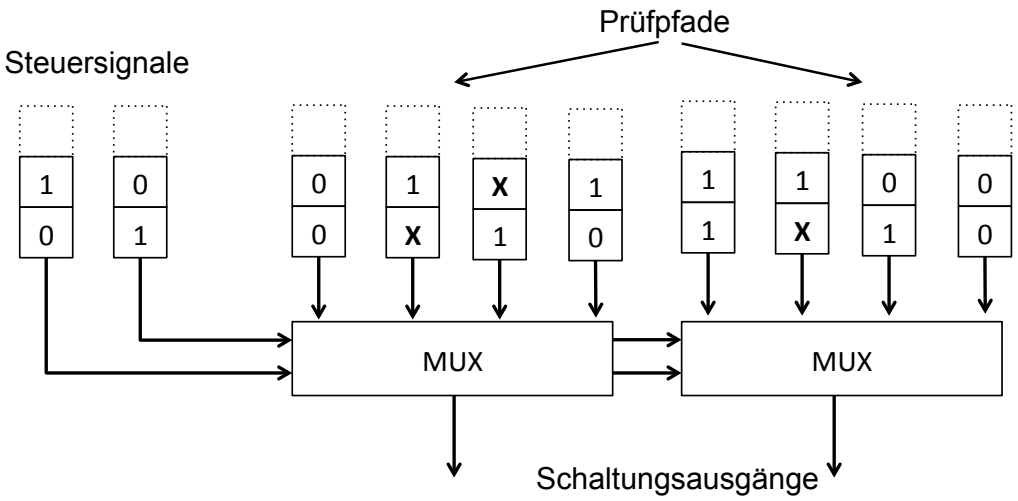


Abbildung 6.3: Ausgangskompaktierung im FAST [SHQ10]

Die Steuersignale werden in separaten Registern abgelegt und steuern für jeden Ausgangsvektor die Multiplexer. Hierdurch werden diejenigen Pfade, die für die FAST-Frequenz nicht ausgelegt sind, maskiert. Um die Steuersignale zu berechnen, wird eine modifizierte Schaltung mit einem Standard-Programm zur Testmustererzeugung durchlaufen. Die Schaltung wird dabei so modifiziert, dass nur die Ausgänge sichtbar sind, die gleichzeitig aktiv sein können. Es gehen bei diesem Ansatz jedoch viele Informationen an den Ausgängen verloren, so dass die Anzahl der Testmuster, um eine ähnliche Fehlerabdeckung wie ein Ansatz ohne Kompaktierung zu erreichen, sehr stark ansteigt. Abbildung 6.1(d) zeigt diesen Ansatz für unsere Beispielschaltung. Ein Multiplexer am Ausgang schaltet nur einen Ausgang weiter und somit kann für den Test auf einen SDD an Gatter G3 lediglich Ausgang O2 durch den Multiplexer ausgewählt werden.

6.2 Überblick FAST-BIST

In diesem Kapitel wird ein Überblick über die entwickelte FAST-Architektur und die Lösungsschritte gegeben, während in den darauffolgenden Abschnitten die Details beschrieben werden.

Wenn der FAST im Selbsttest durchgeführt wird, muss neben der Taktgenerierung auch die Testmustererzeugung sowie die Auswertung der Testantworten auf dem Chip selber implementiert werden. Zur Generierung des FAST-Takts kann auf die bereits angesprochenen Literaturstellen verwiesen werden [PB06] [TA08], hieraus ergeben sich allerdings Rahmenbedingungen für den FAST, die zwar nicht nur im Selbsttest, aber bedingt durch die Implementierungsgrenzen besonders dort zu beachten sind: So sollte sowohl die Anzahl der Frequenzen als auch die Anzahl der Frequenzwechsel möglichst gering sein und die Frequenzen nicht beliebig hoch dimensioniert werden. Für den FAST bedeutet das, dass mit möglichst wenigen FAST-Frequenzen eine maximale Fehlerabdeckung erreicht werden soll. Spezielle FAST-Testmuster werden hierbei nicht verwendet, so dass die Ausgangsbasis eine gegebene deterministische Testmustermenge Ψ_{Gesamt} für Transitionsfehler ist. Eine deterministische Testmustermenge kann, wie bereits in Kapitel 2.4.1 diskutiert, auch im Selbsttest effizient vorgehalten werden.

Ein zusätzliches Problem beim FAST im Selbsttest ist allerdings die Auswertung der Testantworten, denn unbekannte Werte können von Standard-, XOR-basierten, Kompaktierern nicht verarbeitet werden. Auch hierfür gibt es in der Literatur bereits Lösungen (siehe Kapitel 2.4.2), allerdings lassen sie sich, wie bereits diskutiert, aufgrund der hohen X-Raten nicht direkt für den FAST verwenden. Zusätzlich sollte die Kompaktierung möglichst flexibel sein, um die Signatur trotz schwankender X-Raten zu analysieren. Aus diesem Grund werden für die Antwortkompaktierung zunächst die Bits bestimmt, welche die gewünschten Fehlerinformationen enthalten (sogenannte deterministische Bits, symbolisch mit 'D' gekennzeichnet), um diese dann durch Ausnutzung von linearen Abhängigkeiten aus der Signatur X-frei zu extrahieren (ähnlich zu dem in Kapitel 2.4.2 vorgestellten X-Canceling MISR). Um hohe X-Raten zu verarbeiten und eine hohe Fehlerabdeckung zu erreichen, wird ein zusätzlicher Speicher implementiert. In diesem werden Zwischensignaturen des MISRs ablegt und nach dem Test ausgewertet, um die deterministischen Bits aus den Signaturen ohne Einfluss eines unbekannten Wertes zu berechnen. Nach der Speicherung der Zwischensignaturen wird das MISR zurückgesetzt.

Der vorgestellte FAST-BIST kann mit der STUMPS-Architektur sowie Launch-on-Shift, Launch-on-Capture und Enhanced-Scan Techniken für die Generierung der Transitionsfehler-Testmusterpaare arbeiten. Der Test wird in n FAST-Gruppen eingeteilt, wobei jede FAST- Gruppe i durch ihre FAST-Testfrequenz $f_i > f_{nom}$ und ihre FAST-Testmusterpaare Ψ_i charakterisiert wird. Abbildung 6.4 zeigt schematisch die Architektur.

Die bereits bekannte STUMPS-Architektur wird mit einem Zähler erweitert, der die Abspeicherung der Zwischensignaturen und das Zurücksetzen des MISRs über das *reset*-Signal steuert. Weiterhin kann hierüber auch die geeignete FAST-Frequenz ausgewählt werden. Für die Testantwortkompaktierung ist ein zusätzlicher Speicher vorgesehen, der nach dem Test für die Analyse ausgelesen werden kann. Hierdurch ist der Kompaktierer sehr flexibel und kann auch hohe X-Raten verarbeiten, wie in den Experimenten in Kapitel 6.5 gezeigt wird.

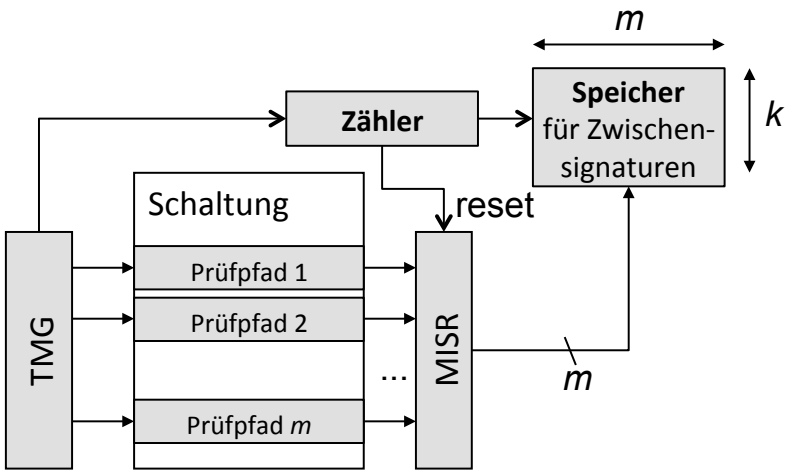


Abbildung 6.4: Architektur für einen FAST-BIST

Für die Implementierung des FAST im Selbsttest müssen zum einen die Auswahl der FAST-Testfrequenzen und zum anderen die Kompaktierung der Testantworten optimiert werden. Hierfür sind einige vorverarbeitende Simulationen notwendig, Abbildung 6.5 zeigt den prinzipiellen Ablauf.

Ausgangspunkt für das Verfahren ist eine von einem kommerziellen Programm generierte deterministische Testmustermenge für Transitionsfehler Ψ_{Gesamt} sowie die nominale Taktfrequenz der Schaltung f_{nom}. Hieraus wird zunächst mittels Simulation die Menge der versteckten Verzögerungsfehler Φ_{rel} generiert. Als „versteckt“ wird ein Verzögerungsfehler ϕ mit Fehlergröße $\Delta(\phi)$ dann genannt, wenn er mit der Frequenz f_{nom} und der Testmustermenge Ψ_{Gesamt} nicht erkannt werden kann. Im nächsten Schritt werden für jeden versteckten Fehler die Erkennungsbereiche bestimmt, in denen der Fehler an mindestens einem Ausgang erkannt werden kann. Die Menge der Zeitpunkte t, bei denen der Fehler ϕ mit einem Musterpaar aus Ψ_{Gesamt} erkannt werden kann, wird dabei als Erkennungsbereich $E(\phi)$ definiert (vgl. hierzu auch [CHE+08]). Anhand der Erkennungsbereiche werden dann die Fehler in FAST-Gruppen eingeteilt. Hierbei ist das Ziel, möglichst wenige Gruppen mit möglichst niedrigen Test-Frequenzen zu finden. Sind solche Gruppen gefunden, ist die Gruppierung abgeschlossen. Das sich hieraus ergebene Optimierungsproblem lässt sich wie folgt formal definieren:

> **FAST-Optimierungsproblem I:** Gegeben sei eine Menge von versteckten kleinen Verzögerungsfehlern Φ_{rel} sowie die Erkennungsbereiche $E(\phi)$ für alle $\phi \in \Phi_{rel}$. Ziel ist es, eine minimale Anzahl von Beobachtungszeitpunkten $\mathscr{T} = \{t_1, ..., t_n\}$ zu finden, so dass für jeden Fehler $\phi \in \Phi_{rel}$ der Durchschnitt $E(\phi) \bigcap \mathscr{T}$ nicht leer ist. Bei zwei oder mehr Lösungen wird die Lösung gewählt, die größere Beobachtungszeitpunkte beinhaltet.

Aus den Beobachtungszeitpunkten kann leicht die jeweilige FAST-Frequenz f_i als $f_i = 1/t_i$ bestimmt werden. Folgendes Beispiel soll dies verdeutlichen:

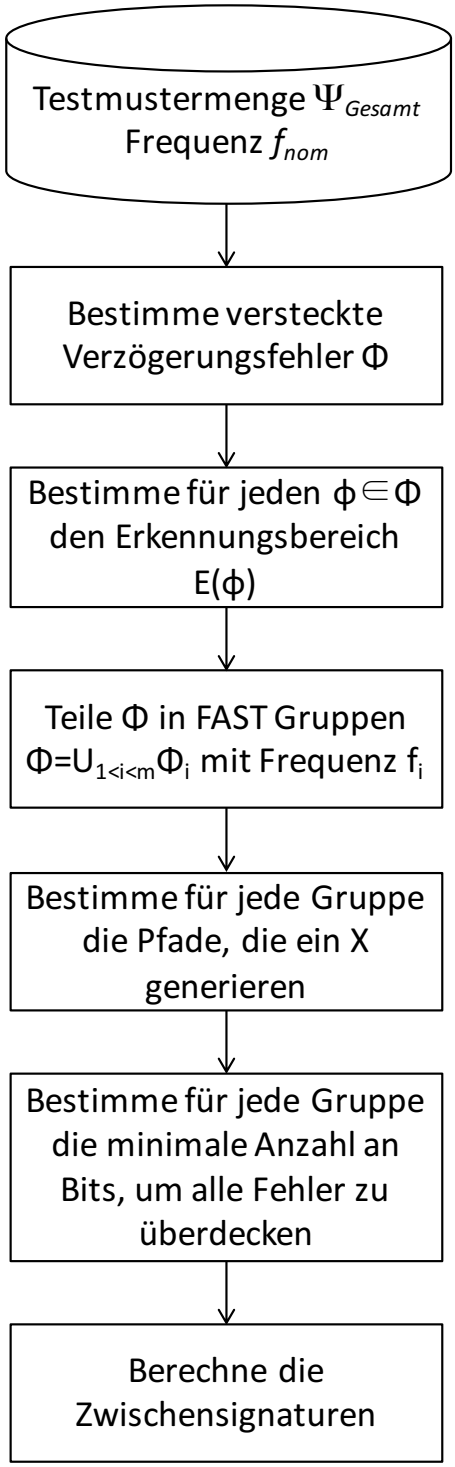


Abbildung 6.5: FAST-BIST Ablauf

Beispiel: Gegeben sei $\Phi_{rel} = \{\phi_1, \phi_2, ..., \phi_7\}$ mit den in Abbildung 6.6 eingezeichneten Erkennungsbereichen. Für kleine Beispiele lässt sich die optimale Lösung schnell bestimmen als $\mathscr{T} = \{t_1, t_2\}$, wie sie in Abbildung 6.7 eingezeichnet ist. Zum Zeitpunkt t_1 lassen sich ϕ_1, ϕ_3 und ϕ_4 erkennen. Für diese Überdeckung wurde t_1 so gewählt, dass er zusätzlich noch einen möglichst großen Wert erreicht, was wiederum der niedrigsten FAST-Taktfrequenz für die Überdeckung dieser vier Fehler entspricht. Die Erkennungsbereiche der restlichen Fehler ϕ_2, ϕ_5, ϕ_6 und ϕ_7 besitzen eine gemeinsame Überdeckung und auch hier wurde t_2 so gewählt, dass eine möglichst niedrige FAST-Taktfrequenz ausreicht.

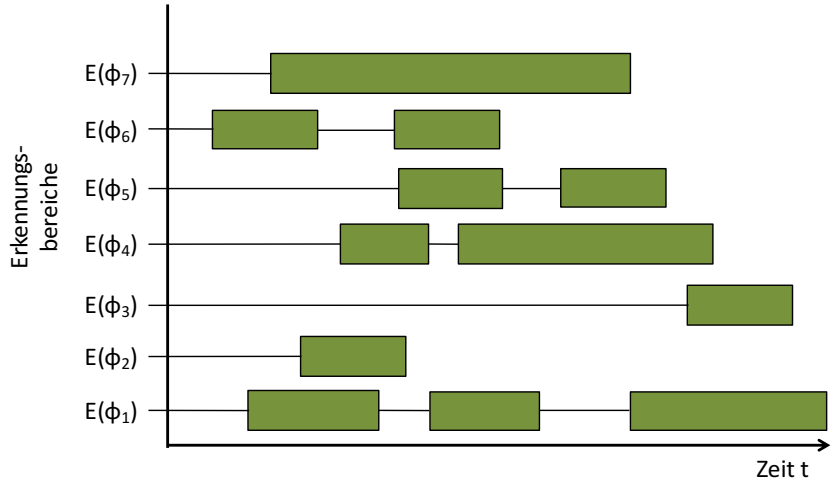


Abbildung 6.6: Erkennungsbereiche

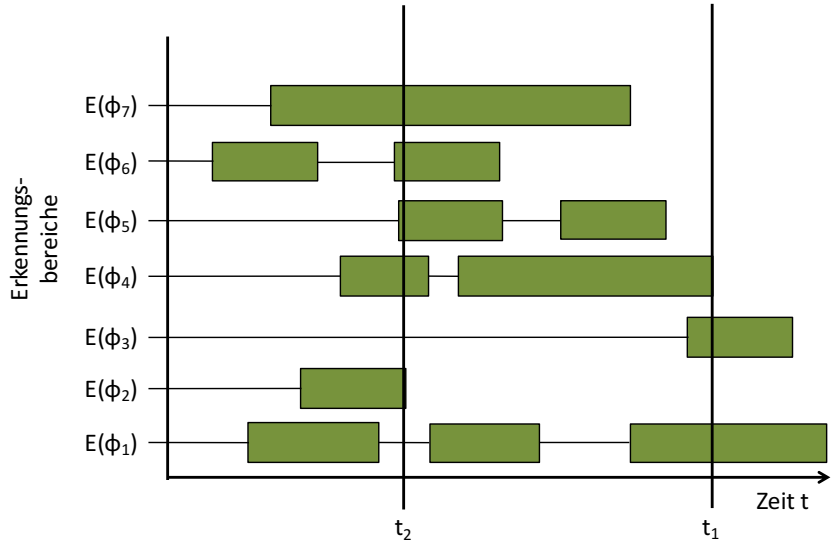


Abbildung 6.7: Optimale Lösung

■

Für große Schaltungen mit sehr vielen Fehlern erweist sich das Finden einer optimalen Lösung allerdings als Problem, welches NP-vollständig ist, so dass effektive Heuristiken herangezogen werden müssen. Kapitel 6.3.3 stellt eine solche Heuristik detailliert vor.

Nachdem eine minimale Anzahl von FAST-Gruppen mit den jeweiligen Frequenzen gefunden wurde, wird für jede Gruppe und jedes Testmuster die Position der unbekannten Werte bestimmt sowie die Bits, die Fehlerinformationen enthalten, sogenannte deterministische Antwortbits oder *D-Bits*. Abbildung 6.8 zeigt ein einfaches Beispiel:

> **Beispiel:** Gegeben sei eine Schaltung mit 3 Eingängen und 2 Ausgängen sowie 2 Gattern. Weiterhin sei das Testmusterpaar (ψ_1, ψ_2) mit $\psi_1 = [111]$ und $\psi_2 = [101]$ gegeben, welches einen kleinen Verzögerungsfehler, der am Ausgang von Gatter $G1$ auftritt, erkennt. Der Fehler wirkt sich auf den Ausgang $O1$ aus. In diesem Fall enthält die Ausgabe an $O1$ Fehlerinformationen, so dass dieses Bit symbolisch mit D gekennzeichnet und D-Bit genannt wird. Da sich für dieses Testmuster kein Fehlereffekt am Ausgang $O2$ auswirkt und für die weitere Verarbeitung keine Rolle spielt, wird diese Ausgabe als „don't care" bezeichnet und symbolisch mit einer 0 dargestellt.

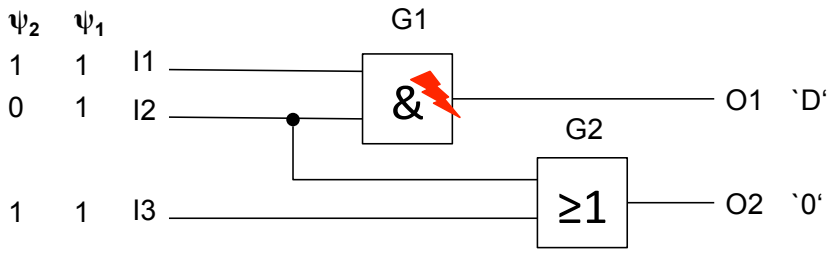


Abbildung 6.8: Beispielschaltung mit D-Bits ■

Um die Anzahl der zu speichernden Zwischensignaturen und somit den Speicheraufwand zu minimieren, sollte die Anzahl der unbekannten Werte möglichst minimal sein. Für eine gegebene Frequenz und gegebene Testmustermenge ist die Anzahl der unbekannten Werte vorgegeben. Allerdings kann durch eine geschickte Auswahl der D-Bits die Anzahl der Zwischensignatur weiter verringert werden. Jedoch wäre ein enormer Rechenaufwand nötig, um die minimale Anzahl von Zwischensignaturen zu finden. Da aber die Anzahl der D-Bits schon einen großen Einfluss auf die Anzahl der Zwischensignaturen hat, wird in dieser Arbeit die Minimierung der Anzahl der D-Bits bei gleichzeitiger maximaler Fehlerabdeckung als Heuristik herangezogen, um die Anzahl der Zwischensignaturen möglichst klein zu halten (siehe Kapitel 6.3.5). Hieraus folgt das zweite Optimierungsproblem, welches sich wie folgt formal definieren lässt:

> **FAST-Optimierungsproblem II:** Gegeben sei eine Menge von versteckten kleinen Verzögerungsfehlern Φ_{rel}. Für jedes D-Bit D_i bezeichne $\Phi_{rel}(D_i)$ die Menge aller Fehler, die an D_i erkannt werden. Ziel ist es, eine Untermenge $\mathscr{D}$ der deterministischen Antwortbits zu finden, so dass $\Phi_{rel} = \bigcup_{D_i \in \mathscr{D}} \Phi_{rel}(D_i)$ minimal ist.

Durch Simulation wird dann bestimmt, zu welchen Zeitpunkten die Zwischensignaturen im Speicher abgelegt werden müssen.

6.3 FAST-Vorverarbeitung

In diesem Kapitel wird detailliert die Vorverarbeitung (engl. *pre-processing*) beschrieben, um zum einen eine minimale Anzahl von FAST-Gruppen (FAST-Optimierungsproblem I) und zum anderen die minimale Anzahl von D-Bits (FAST-Optimierungsproblem II) zu bestimmen. Zunächst werden die versteckten Verzögerungsfehler bestimmt, also die Fehler, die für die gegebene Testmustermenge und gegebene Frequenz nicht erkannt werden können. Hierzu muss zunächst die Fehlergröße eines Fehlers bestimmt werden.

6.3.1 Bestimmung der relevanten Fehler

Zur Bestimmung der relevanten Fehlermenge, also der Menge der versteckten Verzögerungsfehler Φ_{rel}, muss zunächst die relevante Fehlergröße festgelegt werden. Da, bedingt durch Prozessvariationen, vor allem die Transistoren auf einem Chip unterschiedliche Schaltzeiten haben, kann die Schaltzeit d_g eines Gatters nicht mit einem konstanten Wert angegeben werden, sondern folgt einer Verteilungsfunktion $F(d_g)$ [TPC12]. Diese Funktion ist von vielen Faktoren abhängig und soll im Folgenden als Normalverteilung angenommen werden. Da eine gewisse Variation der Schaltzeit gegeben und toleriert werden kann, kann die minimale Fehlergröße $\Delta(\phi)_{min}$ als

$$\Delta(\phi)_{min} = c \cdot \sigma \tag{47}$$

mit der Standardabweichung σ von $F(d_g)$ und einer Konstanten c definiert werden. Je nach Anwendung und Sicherheitsrelevanz kann c dynamisch angepasst werden. Sollte die Schaltzeit eine gegebene tolerierte Abweichung übertreffen, wird diese Schaltzeit als fehlerhaft klassifiziert. Die Grundmenge Φ_G aller Fehler bestehen aus einem Tupel (Fehlerort, Verzögerung), wobei zunächst alle möglichen Fehlerorte ausgewählt werden. Aus dieser Fehlermenge wird nun die relevante Fehlermenge Φ_{rel} bestimmt, für die ein FAST nötig ist. Hierzu werden zunächst äquivalente Fehler identifiziert und aus der Fehlermenge entfernt. Für Haftfehler gibt es eine Vielzahl von Reduktionsregeln, für Verzögerungsfehler sind die Regeln allerdings restriktiver. Deswegen werden in diesem Schritt nur die einfachsten Äquivalenzregeln angewandt, um robustes oder noch strengeres Testen zu ermöglichen [FLL12]: für ein Gatter mit nur einem Eingang sind Verzögerungsfehler an den Ein- und Ausgängen äquivalent. Wenn ein Gatter nur ein einziges anderes Gatter treibt, so ist ein Verzögerungsfehler am Gatterausgang äquivalent zu dem Fehler am Eingang des getriebenen Gatters. Für die restlichen Fehler wird zunächst eine topologische Analyse und dann für jedes Testmusterpaar aus Ψ_{Gesamt} eine Logiksimulation durchgeführt. Jeder Fehler ϕ, der nur auf einem sensibilisierten Pfad η mit der Pfadverzögerungszeit $length(\eta)$ liegt und für den, bei nominaler Taktperiode t_{nom} , Gleichung

$$length(\eta) + \Delta(\phi) < t_{nom} \tag{48}$$

gilt, wird in Φ_{rel} aufgenommen. Wenn eine maximale Frequenz f_{max} vorgegeben ist, werden nur diejenigen Fehler, für die folgende Gleichung gilt, in Φ_{rel} aufgenommen:

$$\frac{1}{f_{max}} < length(\eta) + \Delta(\phi) < t_{nom}. \tag{49}$$

Im nächsten Schritt werden dann die Erkennungsbereiche $E(\phi)$ eines jeden Fehlers $\phi \in \Phi_{rel}$ bestimmt.

6.3.2 Bestimmung der Erkennungsbereiche

Zur Bestimmung der Erkennungsbereiche werden die sensibilisierten Pfade, die aus der Logiksimulation mit der Testmustermenge Ψ_{Gesamt} gewonnen wurden, analysiert: Liegt ein relevanter Fehler $\phi \in \Phi_{rel}$ auf einem Pfad, wird der Erkennungsbereich $E(\phi)$ aktualisiert. Dieser Bereich gibt an, mit welchen FAST-Taktperioden der jeweilige Fehler erkannt werden kann und wird für die Bestimmung der FAST-Gruppen benötigt. Formal lässt sich das Intervall für einen Fehler ϕ, der über den Pfad η erkannt werden kann, als

$$E(\phi, \eta) = [length(\eta), length(\eta) + \Delta(\phi)] \tag{50}$$

beschreiben. Die Menge der Erkennungsintervalle eines Fehlers über alle Pfade $H(\phi)$, auf denen der Fehler ϕ an die Schaltungsausgänge propagiert, beschreibt den Erkennungsbereich $E(\phi) = \bigcup_{\eta \in H(\phi)} E(\phi, \eta)$ und dient als Grundlage für die Bestimmung der FAST Gruppen. Das nachfolgende Beispiel soll dies verdeutlichen.

Beispiel: Abbildung 6.9 zeigt für eine Schaltung mit minimaler FAST-Taktperiode t_{FAST}, einem Ausgang und einem Testmuster den Pegelverlauf im fehlerfreien Fall und für den Fall eines Fehlers ϕ. Wie man erkennt, unterscheiden sich die Pegelverläufe, allerdings ist der Fehlereffekt nicht zur nominalen Taktperiode t_{nom} erkennbar. Der Erkennungsbereich $E(\phi)$ des Fehlers ϕ ist die Vereinigung der Erkennungsintervalle E_1 und E_2, in denen sich die fehlerfreie und die fehlerhafte Ausgabe voneinander unterscheidet. Tritt ein kurzzeitiger Pegelwechsel auf, wird dieser nicht als Erkennungsintervall interpretiert und dem Erkennungsbereich hinzugefügt (engl. *glitch filtering*). Dies liegt daran, dass ein Signal eine gewisse Zeit an einem Flip-Flop anliegen muss, bevor der Wert aufgenommen und gespeichert werden kann.

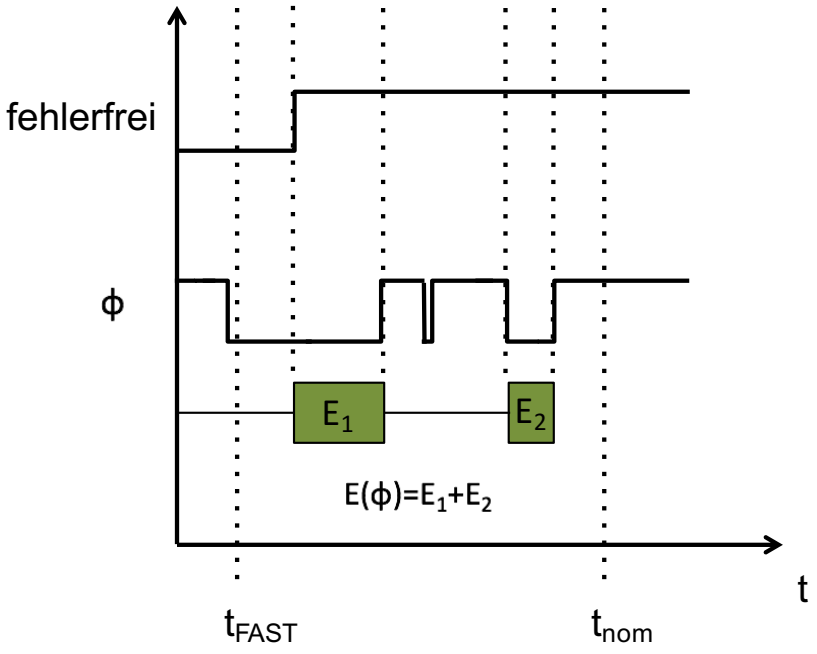


Abbildung 6.9: Bestimmung der Erkennungsbereiche

■

6.3.3 Bestimmung der FAST-Gruppen

Die Bestimmung der Testfrequenzen zur Erkennung der versteckten Verzögerungsfehler hat zwei konträre Ziele: Zum einen sollte die Testfrequenz möglichst hoch sein, um die maximale Anzahl an Fehlern zu testen. Zum anderen bedeutet eine hohe Taktfrequenz auch eine hohe Anforderung an die Taktgenerierung und die Hardware. Zusätzlich erschweren Nebenbedingungen, wie eine eventuell nicht feingranulare Einstellungsmöglichkeit der Betriebsfrequenz oder eine hohe Verzögerungszeit bei einem Frequenzwechsel, die Bestimmung der optimalen Anzahl von FAST-Gruppen. Vereinfacht besteht das Ziel darin, möglichst wenige FAST-Gruppen mit vorzugsweise geringen Frequenzen zu erzeugen, um die Anforderungen an die Hardware gering zu halten. Hierbei können auch mehrere Lösungen optimal bezüglich dieser Anforderungen sein. Um die optimale Lösung zu finden, also die minimale Anzahl an FAST-Gruppen, kann das Problem als Hitting-Set-Problem angesehen werden, wobei es bei diskreten Erkennungsbereichen $E(\phi)$ NP-vollständig ist [Kar72]. Interpretiert man dieses Problem geometrisch, so lassen sich effiziente Heuristiken angeben [MR10]. Um eine zeit- und rechenintensive Analyse der Lösungen zu vermeiden, wird ein einfacher aber effektiver Algorithmus zur Bestimmung der FAST-Gruppen verwendet. Hierzu wird zunächst die Menge aller versteckten Verzögerungsfehler Φ_{rel} betrachtet und folgender Algorithmus durchgeführt:

Algorithm 1 FAST Gruppeneinteilung

1: Φ_{rel} = {alle versteckten Verzögerungsfehler}
2: $\mathscr{T} = \emptyset, i = 0$
3: **while** Φ_{rel} $\emptyset$ **do**
4: $\quad i = i + 1$
5: $\quad t_i = max_{\phi \in \Phi - rel} E(\phi)_{LB}$
6: $\quad \mathscr{T} = \mathscr{T} \bigcup \{t_i\}$
7: $\quad \Phi_{rel} = \Phi_{rel} \backslash \{\phi | t_i \in E(\phi)\}$
8: **end while**

Zunächst wird aus der Fehlermenge Φ_{rel} der Fehler ϕ ausgewählt, dessen Erkennungsbereich $E(\phi)$ die größte untere Grenze $E(\phi)_{LB}$ besitzt. Dieser Wert wird als FAST-Taktperiode t_i der Menge $\mathscr{T}$ hinzugefügt und aus Φ_{rel} alle Fehler entfernt, bei denen t_i innerhalb des Erkennungsbereiches liegt. So lange Φ_{rel} nicht leer ist, werden diese Schritte wiederholt.

Für das Beispiel von möglichen Erkennungsbereichen aus dem bereits bekannten Beispiel in Abbildung 6.6 ergibt sich folgende Gruppeneinteilung (vgl. Abbildung 6.10):

> **Beispiel:** Die Menge der versteckten Verzögerungsfehler Φ_{rel} sei $\{\phi_1, ..., \phi_7\}$ mit den Erkennungsbereichen $E(\phi_1), ..., E(\phi_7)$. Die größte untere Grenze $E(\phi)_{LB}$ besitzt in diesem Fall Fehler ϕ_3 und $E(\phi_3)_{LB}$ wird als t_1 zu $\mathscr{T}$ hinzugefügt. Jetzt werden ϕ_3 und alle Fehler aus Φ_{rel}, bei denen $E(\phi_3)_{LB}$ innerhalb des jeweiligen Erkennungsbereiches liegt (Fehler ϕ_1 und ϕ_4), gestrichen. Hiernach ergibt sich $\Phi_{rel} = \{\phi_2, \phi_5, \phi_6, \phi_7\}$. Im nächsten Schritt wird die nächste FAST-Taktperiode als größte unterste Grenze t_2 der verbleibenden Fehler festgelegt, was in diesem Fall $E(\phi_5)_{LB}$ entspricht. Da t_2 innerhalb der Erkennungsbereiche von ϕ_7 und ϕ_4 liegt, werden auch diese Fehler

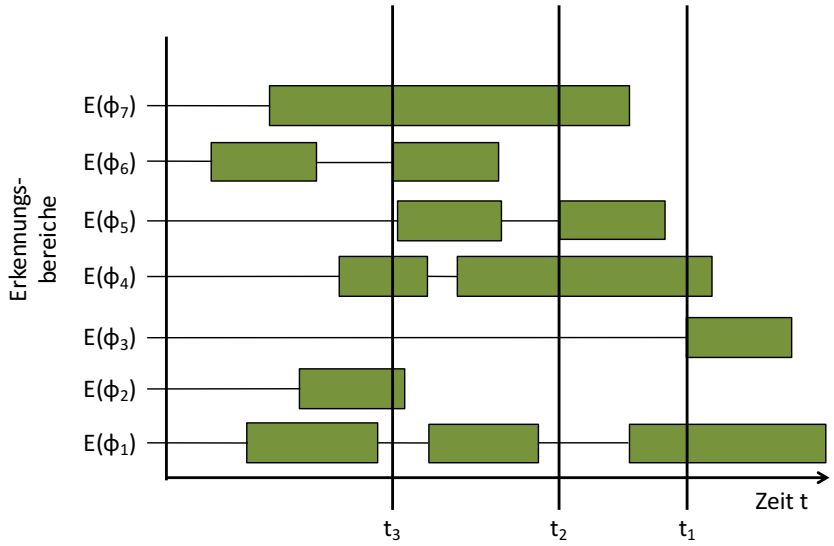


Abbildung 6.10: Bestimmung der Beobachtungszeitpunkte

aus Φ_{rel} gestrichen und es verbleiben nur noch $\Phi_{rel} = \{\phi_2, \phi_6\}$. Die größte untere Grenze entspricht dann der unteren Grenze von $E(\phi_6)$ und es lässt sich leicht erkennen, dass hierdurch auch ϕ_2 erkannt wird. Somit ergeben sich nach Abschluss des Algorithmus folgende Gruppen:

Gruppe 1 : $\{\phi_1, \phi_3, \phi_4\}$ mit FAST-Taktperiode t_1,
Gruppe 2 : $\{\phi_5, \phi_7\}$ mit FAST-Taktperiode t_2,
Gruppe 3 : $\{\phi_2, \phi_6\}$ mit FAST-Taktperiode t_3.

■

Einige umfangreichere Algorithmen zur Gruppenbildung sind beispielsweise das *Clustering* [KR05] oder *Hypergraphenalgorithmen* [LSC10]. Clustering wird vor allem in der Bildverarbeitung eingesetzt. Die prinzipielle Idee beruht darauf, zunächst jedes Element, in diesem Fall jeden Fehler, in eine eigene Gruppe einzusortieren, um sie dann schrittweise zusammenzuführen. Ziel ist es, möglichst wenige und große Gruppen zu erhalten. Beim Hypergraphalgorithmus nach [LSC10] wird das Problem als Hypergraph modelliert, wobei ein Hypergraph aus Kanten und Knoten besteht und eine Kante beliebig viele Knoten miteinander verbinden kann. Wenn die FAST-Taktperioden $\mathscr{T}$ als Menge der Knoten angesehen werden und die Erkennungsbereiche als Kanten, muss eine minimale Menge von Knoten gefunden werden, so dass alle Kanten abgedeckt werden. Nachteilig bei beiden Verfahren ist jedoch die Komplexität und der Rechenaufwand, der nötig ist, eine annäherungsweise optimale Lösung zu finden. Experimente, die in Kapitel 6.5 beschrieben werden, zeigen jedoch, dass mit der in diesem Kapitel vorgestellten einfachen Heuristik gute Ergebnisse mit wenigen FAST-Gruppen erzielt werden können.

6.3.4 Bestimmung der unbekannten Werte

Durch die im vorherigen Kapitel beschriebene Heuristik ist es möglich, effizient und ohne großen Rechenaufwand, Fehler gemäß ihrer Erkennungsbereiche in Gruppen einzuteilen. Hierbei war das Optimierungsziel, eine möglichst geringe Anzahl von möglichst kleinen Frequenzen zu finden. Jetzt müssen die Testmuster den jeweiligen FAST-Gruppen zugeordnet werden: Die Information, welcher Fehler durch welche Testmuster aktiviert und zum Ausgang propagiert wird, wurde schon für die Einteilung der FAST-Gruppen herangezogen und so können diese Informationen nun leicht zur Anordnung der Testmuster benutzt werden. Durch eine einfache Pfad-basierte Analyse können dann die Ausgänge, die erst nach der jeweiligen FAST-Taktperiode einen stabilen Wert erreichen, als unbekannter Wert ('X') angesehen werden. Abbildung 6.11 veranschaulicht diese Überlegung.

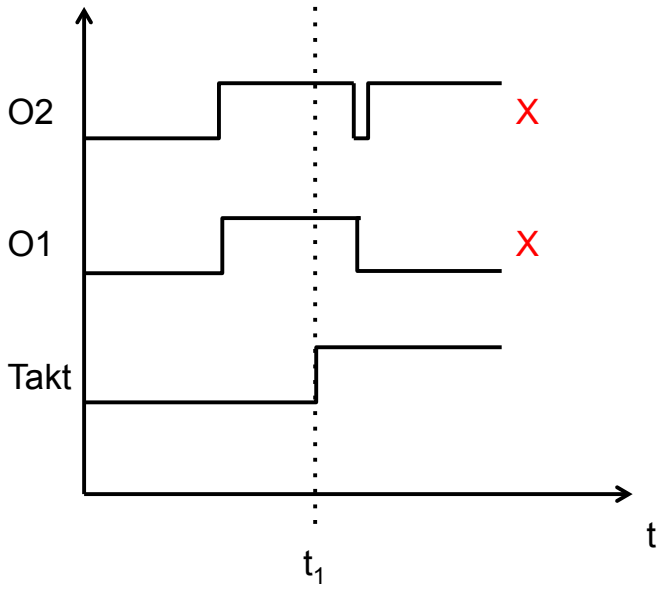


Abbildung 6.11: X-Charakterisierung

Gegeben sei ein Taktsignal mit der Taktperiode t_1 und am Ausgang $O1$ findet nach t_1 noch ein Pegelwechsel von 1 auf 0 statt. Gehen wir davon aus, dass keine Fehlerinformationen an $O1$ anliegt, lässt sich erkennen, dass der Ausgang $O1$ zum Zeitpunkt t_1 noch nicht stabil ist. Da eine feingranulare Timing-Simulation unter Berücksichtigung von Prozess-Variationen notwendig wäre, um den genauen Wert zum Zeitpunkt t_1 zu bestimmen, wird dieser Wert vereinfacht als unbekannt ('X') angenommen. Allerdings sieht man am Ausgang $O2$ auch, warum diese Annahme eine Worst Case-Annahme darstellt: Denn nach dem Zeitpunkt t_1 findet nur ein kurzer Signaleinbruch statt, der keinen dauerhaften Pegelwechsel des Ausgangswertes forciert. Also könnte hier der Wert als stabil '1' angesehen werden, allerdings werden in dieser Arbeit auch diese Effekte als unbekannte Werte interpretiert. Der Grund dafür liegt ebenfalls an der Tatsache, dass eine sehr feingranulare Timing-Simulation unter Berücksichtigung von Prozess-Variationen notwendig wäre. Vereinfacht gesagt wird die Annahme getroffen: Sobald noch ein Pegelwechsel nach dem Beobachtungszeitpunkt t_i auftritt, wird ein 'X' an diesem Ausgang angenommen.

Nach Abschluss der Analyse können dann für jede Testantwort die genauen Positionen

der X-Werte bestimmt werden. Im nächsten Schritt werden diejenigen Bits der Testantworten bestimmt, die verlustfrei aus der Kompaktierung extrahiert werden müssen, um eine maximale Fehlerabdeckung zu erreichen.

6.3.5 Bestimmung der deterministischen Bits

Während die Anzahl der unbekannten Werte für die gegebene Testmustermenge und FAST-Frequenzen vorgegeben ist und sich ohne Weiteres nicht weiter minimieren lässt, können die übrigen Bits innerhalb einer Testantwort prinzipiell in zwei Klassen eingeteilt werden: Zum einen gibt es Bits, die sich beim Auftreten bestimmter Fehler ändern und somit Fehlerinformationen enthalten (D-Bits) [GPT08]. Zum anderen treten Bits auf, die für die relevanten SDDs immer konstant sind. Diese Bits enthalten keine Fehlerinformationen und werden, wie bereits beschrieben, 'don't care'-Bits genannt und symbolisch mit '0' gekennzeichnet. Wie bereits im FAST-Optimierungsziel II definiert, ist das Ziel, eine minimale Anzahl von D-Bits mit maximaler Fehlerabdeckung zu finden, um die Anzahl der abzuspeichernden Zwischensignaturen möglichst gering zu halten. Da ein D-Bit unterschiedlich viele Fehler erkennen kann, entstehen sogenannte Erkennungsprofile.

Beispiel: Sei r_i der Ausgangsvektor des Testmusters ψ_i und durch Simulation und Analyse, wie in den vorherigen Kapiteln beschrieben, sind die Werte des Vektors $\{D, D, 0, D, X\}$. Die ersten beiden und das vierte Bit der Testantwort enthalten Fehlerinformationen und werden mit D_1, D_2 und D_3 entsprechend ihrer Reihenfolge bezeichnet. Ferner sei das Erkennungsprofil für die Fehlermenge $\Phi = \{\phi_1, \phi_2, \phi_3\}$ gegeben durch:

$$\Phi(D_1) = \{\phi_1, \phi_2\}, \Phi(D_2) = \{\phi_3\}, \Phi(D_3) = \{\phi_2\}. \tag{51}$$

Hieraus lässt sich leicht erkennen, dass es für die maximale Fehlerabdeckung ausreicht, nur D_1 und D_2 für den Test heranzuziehen, da mit diesen beiden Bits alle Fehler erkannt werden können. ■

Das FAST-Optimierungsproblem II kann somit als Überdeckungsproblem angesehen werden und es kann eine einfache „greedy"-Heuristik verwendet werden: Das erste Bit, welches einen neuen Fehler erkennt, wird als D-Bit gekennzeichnet und alle Fehler, die hiermit erkannt werden, brauchen nicht weiter betrachtet zu werden. Abbildung 6.12 verdeutlicht die Idee anhand eines Beispiels:

Beispiel: Gegeben seien die Ausgangsbits O_1, O_2, O_3 und die Fehler ϕ_1, ϕ_2, ϕ_3. Bit O_1 erkennt die Fehler ϕ_1 und ϕ_2, Bit O_2 die Fehler ϕ_1 und ϕ_3 und Bit O_3 den Fehler ϕ_2. Da das erste Bit direkt Fehler ϕ_1 und ϕ_2 entdeckt, wird es als D_1 symbolisiert und beide Fehler müssen nicht weiter betrachtet werden. Da das Bit O_2 nur Fehler ϕ_2 erkennt, dieser aber bereits von O_1 erkannt wird, kann dieses Ausgangsbit als '0' symbolisiert werden. Bit O_3 entdeckt den Fehler ϕ_3 und wird mit D_2 symbolisiert.

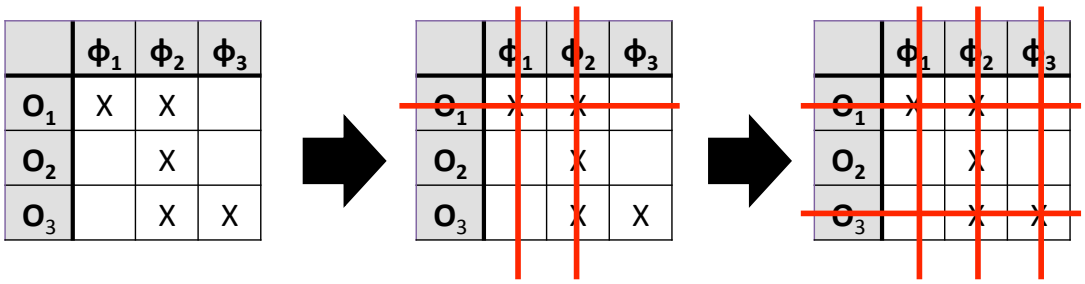


Abbildung 6.12: Beispiel für einfache Heuristik zur D-Bit Auswahl

Der Hardwareaufwand für eine gezielte Auswahl der Bits, beispielsweise mit Multiplexern, wäre zu feingranular und der Hardwareaufwand würde hierfür stark ansteigen, weswegen in dieser Arbeit eine auf einem MISR basierte Kompaktierung entwickelt wird.

6.4 X-tolerante Kompaktierung

Die Verteilung der unbekannten Werte ist je nach FAST-Gruppe sehr unterschiedlich, da mit wechselnder Frequenz unterschiedliche Pfade eine negative Pufferzeit haben und somit an unterschiedlichen Ausgängen einen unbekannten Wert erzeugen. Aus diesem Grund muss der Kompaktierer, der eingesetzt wird, sehr flexibel sein und eine hohe Anzahl von unbekannten Werten tolerieren können. Die aus der Literatur bekannten Lösungen, die auch in Kapitel 2.4 bereits vorgestellt wurden, haben den Nachteil, dass sowohl die X-Rate als auch die Flexibilität der Implementierungen sehr beschränkt sind. Zusätzlich sollen möglichst alle Bits, die Fehlerinformationen enthalten (D-Bits), aus den X-freien Bits extrahiert werden können. Aus diesem Grund wird in diesem Kapitel eine flexible X-tolerante Kompaktierung vorgestellt. Zunächst soll die Grundidee beschrieben werden.

6.4.1 Grundidee

Der X-tolerante Kompaktierer basiert auf dem in Kapitel 2.4 beschriebenen X-Canceling MISR [Tou07], welches mittels symbolischer Simulation die MISR Bits als Linearkombinationen in Abhängigkeit der unbekannten Werte darstellt und durch geschickte Addition den Effekt der unbekannten Werte eliminiert. Zusätzlich sollen in diesem Zusammenhang auch noch D-Bits extrahiert werden, die Fehlerinformationen enthalten und gezielt für eine höhere Fehlerabdeckung sorgen. Um die hohe Anzahl an X- und D-Bits effizient und flexibel verarbeiten zu können, werden zusätzlich Zwischensignaturen gespeichert. Hierfür wird ein Zähler verwendet, der die Abspeicherung der Zwischensignaturen und das Zurücksetzen des MISRs steuert. Für die Abspeicherung ist, wie bereits diskutiert und in Abbildung 6.4 dargestellt, ein zusätzlicher Speicher vorgesehen, der nach dem Test für die Analyse ausgelesen werden kann. Um den Speicherbedarf möglichst klein zu halten, soll die Anzahl der Zwischensignaturen minimal sein. Die Auswahl und die Berechnungen werden im nächsten Kapitel genauer beschrieben.

6.4.2 Berechnungen der Zwischensignaturen

Um eine höhere Anzahl von unbekannten Werten zu verarbeiten, als die bereits aus der Literatur bekannten Verfahren (siehe Kapitel 2.4.2), und im Gegensatz zu [SHQ10] eine flexible Lösung mit maximaler Fehlerabdeckung und minimalem Hardwareaufwand in der STUMPS-Architektur zu gewährleisten, werden Zwischensignaturen in einem zusätzlichen Speicher abgelegt. Hierbei ist jedoch entscheidend, zu welchem Zeitpunkt der Zustand des MISRs abgespeichert und das MISR zurückgesetzt werden soll. Die obere Schranke für die Anzahl der unbekannten Werte, die ein m-Bit breites MISR verarbeiten kann, ist mit $m - n_{Xfrei}$ angegeben, wobei n_{Xfrei} die Anzahl der X-freien Bits ist, die man aus der Signatur extrahieren möchte [Tou07]. Allerdings sind die unbekannten Werte in einem FAST sehr unterschiedlich verteilt und es reicht aus, die deterministischen Bits aus der Signatur X-frei zu extrahieren. Aus diesem Grund muss mittels Simulation herausgefunden werden, wann das MISR keine weiteren X- oder D-Bits mehr aufnehmen kann, um dies zu gewährleisten. Hieraus folgen dann auch die Zeitpunkte, wann der Zustand des MISRs abgespeichert und das MISR zurückgesetzt werden muss.

Ein kurzes Beispiel soll die Idee verdeutlichen:

Beispiel: Gegeben sei eine Schaltung mit drei Prüfpfaden, wobei die Testantworten zur Kompaktierung parallel in ein 3-Bit MISR geschoben werden. Die Bits, die während eines Schiebetaktes in das MISR geschoben werden, werden Testvektor genannt. Die Testvektoren können aus folgenden Symbolen bestehen:

- D-Bits: Deterministische Bits, die Fehlerinformationen über einen oder mehrere Fehler enthalten.
- X-Bits: Unbekannte Werte, da zur FAST-Taktperiode noch kein stabiler Wert anliegt.
- 0: „Don't care"-Bits, die für die symbolische Simulation als 0 dargestellt werden, also keine Auswirkungen auf die Berechnungen haben.

Abbildung 6.13 zeigt schematisch den Aufbau. Nachdem der erste Testvektor in das MISR geschoben wurde, können die MISR Bits m_0, m_1 und m_2 als

$$\begin{aligned} m_0 &= X_0 \\ m_1 &= 0 \\ m_2 &= 0 \end{aligned}$$

dargestellt werden. Die MISR-Matrix der MISR Bits in Abhängigkeit von X_0 ergibt sich trivialerweise als

$$\begin{array}{c} \\ m_0 \\ m_1 \\ m_2 \end{array} \begin{array}{c} X_0 \\ \left[\begin{array}{c} 1 \\ 0 \\ 0 \end{array}\right] \end{array}.$$

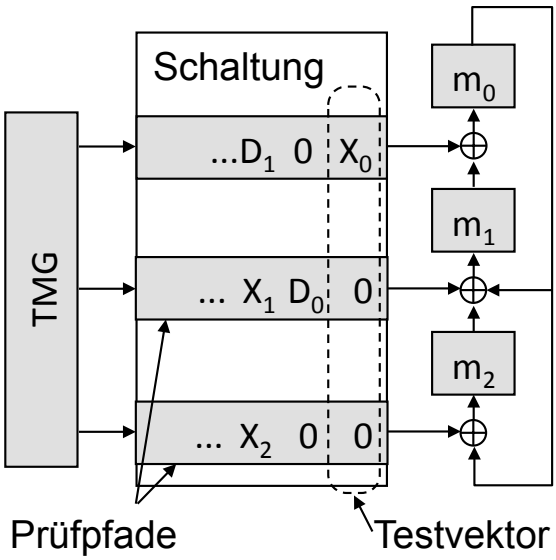


Abbildung 6.13: Beispiel eines X-Canceling MISRs

Sowohl m_1 als auch m_2 sind unabhängig von einem unbekannten Wert und es ist keine weitere Analyse nötig.

Mit dem nächsten Testvektor ergeben sich die MISR-Bits als

$$\begin{aligned} m_0 &= 0 \\ m_1 &= X_0 \oplus D_0 \\ m_2 &= X_0. \end{aligned}$$

m_1 enthält ein deterministisches Bit, ist aber auch abhängig von einem unbekannten Wert (X_0). Die Matrix-Repräsentation der MISR Bits in Abhängigkeit von X_0 und D_0 ist

$$\begin{array}{c} \\ m_0 \\ m_1 \\ m_2 \end{array} \begin{array}{c} \begin{array}{cc} X_0 & D_0 \end{array} \\ \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 0 \end{bmatrix} \end{array}.$$

Es lässt sich leicht erkennen, dass sich durch Addition von m_1 und m_2 folgende Matrix ergibt

$$\begin{array}{c} \\ m_0 \\ m_1 \oplus m_2 \\ m_2 \end{array} \begin{array}{c} \begin{array}{cc} X_0 & D_0 \end{array} \\ \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \end{array}.$$

Durch diese Berechnung ergibt sich

$$\begin{aligned} m_0 &= 0 \\ m_1 \oplus m_2 &= X_0 \oplus D_0 \oplus X_0 = D_0 \\ m_2 &= X_0. \end{aligned}$$

Es lassen sich also zwei Bits, die unabhängig von einem unbekannten Wert sind, und das deterministische Bit aus dieser Signatur extrahieren.

Nachdem der nächste Testvektor in das MISR geladen wird, ergeben sich für die MISR Bits die Gleichungen

$$\begin{aligned} m_0 &= X_0 \oplus D_0 \oplus D_1 \\ m_1 &= X_0 \oplus X_1 \\ m_2 &= X_2. \end{aligned}$$

Die entsprechende Matrix ergibt sich als

$$\begin{array}{c} \\ m_0 \\ m_1 \\ m_2 \end{array} \begin{array}{c} \begin{array}{ccccc} X_0 & X_1 & X_2 & D_0 & D_1 \end{array} \\ \left[\begin{array}{ccccc} 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{array}\right] \end{array}.$$

Es gibt keine Möglichkeit, die D-Bits unabhängig von X zu extrahieren oder X-freie Kombinationen zu erhalten. Aus diesem Grund wird der vorherige Zustand des MISRs für die genauere Analyse abgespeichert und das MISR anschließend zurückgesetzt. ■

An diesem Beispiel erkennt man, dass es von Zeit zu Zeit nötig ist, Zwischensignaturen für die spätere Analyse auf dem Chip abzulegen. Dies ist immer dann der Fall, wenn sich durch weitere Aufnahme von X- oder D-Bits die D-Bits nicht mehr X-frei extrahieren lassen. Durch Simulation werden die Zeitpunkte bestimmt und durch die Matrix-Repräsentation der Gleichungen ist eine effiziente Analyse mit Hilfe des Gauss-Jordan-Verfahren möglich, um die linearen Abhängigkeiten zu bestimmen [Tou07] [GPT08]. Durch eine minimale Anzahl von D-Bits ist zwar keine minimale Anzahl von Zwischensignaturen garantiert, da die minimale Anzahl von Zwischensignaturen auch von der Verteilung der D-Bits abhängig ist und diese in der vorgestellten Heuristik nicht beachtet wird. Allerdings zeigen die experimentellen Ergebnisse, die im nächsten Kapitel analysiert werden, dass sich mit dieser einfachen Heuristik schon deutliche Einsparungen erreichen lassen ohne Verluste bei der Fehlerabdeckung.

6.5 Experimente und Ergebnisse

In diesem Kapitel werden die Experimente und Ergebnisse des in den vorangegangenen Kapiteln vorgestellten FAST-BIST beschrieben. Hierbei wird in den ersten Experimenten vor allem auf die FAST-Generierung, die Erzeugung der FAST-Gruppen und -Testfrequenzen eingegangen. Im zweiten Teil werden dann die Simulationen und Ergebnisse des X-toleranten Kompaktierers ausgewertet.

Zunächst seien in Tabelle 6.1 die Schaltungen angegeben, für welche die Simulationen durchgeführt wurden und deren für die Experimente wichtigen Eigenschaften.

In Spalten eins bis vier sind die Namen der ITC- und NXP-Benchmarkschaltungen, die Schaltungsgröße sowie die Anzahl der Pseudo-Primäreingänge und -Primärausgänge (PPI und PPO) angegeben. Alle Schaltungen wurden mit 64 Prüfpfaden implementiert, so dass sich die in Spalte fünf aufgezeigten maximalen Längen der Prüfpfade ergeben. Spalte sechs zeigt die Zeit des mit der initialen Testmustermenge längsten sensibilisierten Pfades an, welche in den Simulationen als nominale Taktperiode t_{nom} festgelegt wurde. Die minimale FAST-Taktperiode t_{FAST}, die für die Experimente angenommen wurde, ist in Spalte sieben angegeben und entspricht 30% der nominalen Taktperiode. Diese Annahme geht auch einher mit den aus der Literatur bekannten Werten [AC05] [ATJ06] [LM08]. Spalte acht zeigt die Anzahl der betrachteten kleinen Verzögerungsfehler, wobei als Fehlergröße $6 \cdot \sigma$ angenommen wurde. Mittels topologischer Analyse wurden aus dieser Fehlermenge all diejenigen Fehler gestrichen, die sich selbst mit der maximalen FAST-Frequenz nicht erkennen lassen, sowie die Fehler, für die es mindestens einen Pfad gibt, der den Fehlereffekt während der nominalen Taktperiode an einen Ausgang propagiert. Spalte neun zeigt somit die Anzahl der Fehler, die in den weiteren Simulationen analysiert werden müssen.

Für jede Schaltung aus Tabelle 6.1 wurden mit einem kommerziellen Werkzeug zur Testmustererzeugung 1.000 deterministische Muster für Transitionsfehler (*TF*) generiert. Diese Muster wurden dann mittels dem Java-Framework ADAMA analysiert und die Fehlerabdeckung für die kleinen Verzögerungsfehler mit Fehlergröße $6 \cdot \sigma$ bestimmt. Die Ergebnisse sind in Tabelle 6.2 abzulesen. Spalte zwei zeigt die Fehlerabdeckung der TF-Muster für die kleinen Verzögerungsfehler, also die Anzahl der Fehler, die mit der Verzögerung von $6 \cdot \sigma$ bei nominaler Taktfrequenz von den TF-Mustern erkannt werden. Hieraus folgt in Spalte drei die Anzahl der Fehler, die nicht mit den TF-Mustern bei nominaler Taktfrequenz entdeckt und *versteckte Fehler* genannt werden. Über alle Schaltungen werden im Durchschnitt nur 3.08% aller kleinen Verzögerungsfehler bei nominaler Taktfrequenz erkannt. Die nicht erkannten Fehler, also die versteckten kleinen Verzögerungsfehler, sind die Ausgangsbasis für die FAST-Gruppengenerierung.

Hierzu wurde in ADAMA der Algorithmus zur Gruppengenerierung aus Kapitel 6.3.3 implementiert: Zunächst wurden für jeden Fehler die Erkennungsbereiche bestimmt, wobei Intervalle, die kleiner als 10 ps waren, nicht dem Erkennungsbereich hinzugefügt wurden (siehe Kapitel 6.3.2). Dazu wurden die TF-Muster eingelesen, für jeden Fehlerort die sensibilisierten Pfade bestimmt und deren Verzögerungszeit mit der Fehlerzeit addiert. Für die Menge der Fehlerintervalle aller Fehler wurden dann mit der in Kapitel 6.3.3 eingeführten einfachen Heuristik die Taktperioden der verschiedenen Gruppen bestimmt. Tabelle 6.3 zeigt die Ergebnisse der FAST-Gruppeneinteilung.

In Spalte zwei ist die Anzahl der generierten FAST-Gruppen angegeben. Wie man sieht, werden selbst bei den NXP Schaltungen nicht mehr als 14 FAST-Gruppen pro Schaltung

Schaltung	**Anzahl Gatter**	**Anzahl PPI**	**Anzahl PPO**	**max. Länge**	t_{nom}**[ps]**	t_{FAST} **[ps]**	**Anzahl alle Fehler**	**Anzahl zu betrachtende Fehler**
b14_1	12.438	260	214	4	4.124	1.237	25.126	20.848
b17_1	21.858	1.827	1.348	22	3.576	1.073	97.878	62.890
b18_1	75.618	4.116	3.085	49	4.484	1.345	255.960	148.404
b20_1	25.547	533	450	8	4.269	1.281	59.155	49.570
b21_1	25.564	534	450	8	4.276	1.283	59.761	49.644
b22_1	38.568	786	664	11	4.445	1.334	93.262	80.803
p35k	22.803	2.861	2.229	35	3.159	948	125.898	68.084
p45k	22.414	3.739	2.550	40	3.499	1.050	127.295	62.126
p78k	46.504	3.148	3.484	58	1.475	443	268.989	228.644
p81k	78.665	4.029	3.952	62	1.586	476	453.115	384.706

Tabelle 6.1: Schaltungscharakteristika

Schaltung	**Fehlerabdeckung rel. Fehler**	**Anzahl versteckte Fehler**
b14_1	0,03%	20.841
b17_1	0,76%	62.412
b18_1	0,04%	148.341
b20_1	0,09%	49.524
b21_1	0,10%	49.594
b22_1	0,04%	80.774
p35k	0,05%	68.050
p45k	28,20%	44.604
p78k	1,18%	225.937
p81k	0,31%	383.502

Tabelle 6.2: Fehlerabdeckung kleiner Verzögerungsfehler mit TF-Mustern

Schaltung	**Anzahl FAST Gruppen**	**Fehlerabdeckung versteckte Fehler**
b14_1	7	88,91 %
b17_1	8	86,77 %
b18_1	10	87,93 %
b20_1	8	90,64 %
b21_1	9	90,77 %
b22_1	11	91,35 %
p35k	14	61,29 %
p45k	4	71,80 %
p78k	10	89,04 %
p81k	7	97,74 %

Tabelle 6.3: Ergebnisse FAST-Gruppeneinteilung

generiert, über alle FAST-Gruppen im Durchschnitt sogar nur 8. Dies hat den Vorteil, dass die Anforderungen an die Hardware, im Speziellen an den Taktgenerator und die Steuerung, minimal sind. Die maximale FAST-Taktperiode wurde, wie bereits erklärt, auf $0,3 \cdot t_{nom}$ festgelegt. In Spalte drei ist die zusätzliche Fehlerabdeckung angegeben, die durch den FAST erreicht wird. Bis auf zwei Ausreißer (p35k und p45k) liegt die zusätzliche Fehlerabdeckung bei >85%, im Durchschnitt bei 85,62%. Hierbei ist zu beachten, dass es sich bei diesen Ergebnissen um eine zusätzliche Fehlerabdeckung handelt, die speziell nur die versteckten kleinen Verzögerungsfehler betrachtet. Weiterhin ist die Fehlerabdeckung natürlich abhängig von der Testmustermenge der aus dem kommerziellen Werkzeug zur Testmustererzeugung generierten TF-Testmuster. Durch Anpassungen im Testmustererzeugungsprozess würde sich die Fehlerabdeckung noch weiter erhöhen.

Im zweiten Teil der Experimente wird nun die X-tolerante Kompaktierung validiert. Hierzu werden die Ergebnisse aus der Simulation der FAST-Gruppen dazu verwendet, unbekannte ('X'), deterministische ('D') und „don't care" ('0') Bits zu generieren. Hierzu werden die Testmuster gemäß ihrer Fehleraktivierung den jeweiligen FAST-Gruppen zu-

geteilt und für jedes Testmuster folgende Analyse durchgeführt:

- Endet an einem Ausgang ein Pfad, dessen Länge größer ist als die aktuelle FAST-Taktperiode, wird dieser Ausgang mit 'X' gekennzeichnet.
- Endet an einem Ausgang ein Pfad, der einen Fehler $\phi \in \Phi_{rel}$ propagiert und der mit der aktuellen FAST-Taktfrequenz erkannt werden kann, wird dieser Ausgang mit D gekennzeichnet. Der Fehler ϕ_i wird aus Φ_{rel} gestrichen gemäß der Heuristik zur D-Bit Auswahl aus Kapitel 6.3.3. Dies gilt für alle Fehler, deren Fehlerinformationen in diesem Bit enthalten sind.
- In allen anderen Fällen wird dieser Ausgang mit '0' gekennzeichnet.

Schließlich werden die D- und X-Bits durchnummeriert und das MISR symbolisch in ADAMA simuliert. Ziel ist es, die Anzahl der zu speichernden Zwischensignaturen und den damit einhergehenden Speicheraufwand zu berechnen. Hierzu wurde nach jeder Testantwort die MISR-Matrix aufgestellt und überprüft, ob eine Lösung des linearen Gleichungssystems möglich ist. Hierbei wurden zwei verschiedene Ansätze für „korrekte" Lösungen miteinander verglichen: Einerseits wurde versucht, alle im MISR befindlichen D-Bits unabhängig von einem unbekannten Wert zu separieren. Andererseits wurden Experimente gemäß [Tou07] durchgeführt, die jedes D-Bit ebenfalls als „don't care" Bit behandelt haben und die „korrekte" Lösung dann gegeben war, wenn sich sieben X-freie Bits aus der Signatur berechnen ließen. Sobald keine Lösung gefunden wurde, wurde der letzte Zustand des MISRs in einen Speicher abgelegt und das MISR zurückgesetzt. Um die Gruppen unabhängig voneinander zu analysieren, wurde das MISR auch nach jeder FAST-Gruppe zurückgesetzt sowie die Zwischensignatur abgespeichert. Tabelle 6.4 zeigt zunächst die Ergebnisse für sieben X-freie Kombinationen in Abhängigkeit von verschiedenen MISR Größen. Da 64 Prüfpfade in jeder Schaltung vorhanden sind, kann es auch vorkommen, dass bei einem 64-Bit MISR keine Lösung möglich ist, wenn zu viele X- und/oder D-Bits in einer Testantwort in das MISR geschoben werden. Dies ist mit „n.m." (nicht möglich) in der Tabelle markiert.

Man erkennt, dass die Anzahl der unbekannten Werte sehr hoch ist und, wie man in Spalte zwei erkennen kann, je nach Schaltungsgröße variiert. Spalte drei gibt die X-Rate an, also das Verhältnis von unbekannten Werten zu bekannten Werten. Hierbei bestätigt sich auch die Annahme, dass in einem FAST sehr hohe X-Raten auftreten. Spalten vier bis sieben geben dann die Anzahl der zu speichernden Zwischensignaturen in Abhängigkeit von der MISR Größe m an, wobei bei einer Verdopplung der MISR Größe eine Halbierung der Anzahl der Zwischensignaturen zu beobachten ist. Ein Ausreißer ist, aufgrund seiner Schaltungsstruktur mit vielen kurzen Pfaden, die Schaltung p35k. Hier werden viele FAST-Gruppen benötigt, allerdings wird nur eine geringe Fehlerabdeckung erreicht und die X-Rate ist sehr gering, was dazu führt, dass die Anzahl der Signaturen deutlich geringer ist als bei der ähnlich großen ITC-Schaltung b_17.

Schaltung	**Anzahl X-Bits**	**X-Rate**	**Anzahl Zwischensignaturen für m=**			
			64	**128**	**256**	**512**
b14_1	17.861	4,51%	283	141	72	37
b17_1	108.175	2,22%	1798	868	430	217
b18_1	321.633	1,25%	n.m.	2.562	1.270	635
b20_1	56.273	2,74%	848	434	221	113
b21_1	68.174	3,24%	1.043	531	270	137
b22_1	120.233	2,48%	1.842	934	470	240
p35k	31.228	0,27%	588	272	134	68
p45k	246.238	2,88%	4.426	2.059	994	490
p78k	571.104	17,24%	n.m.	5.185	2.416	1.162
p81k	1.153.733	7,29%	n.m.	9.522	4.623	2.285

Tabelle 6.4: Anzahl Zwischensignaturen in Abhängigkeit der MISR Größe für 7 X-freie Kombinationen

Tabelle 6.5 zeigt schließlich die Ergebnisse, wenn neben den X- auch D-Bits in die Analyse mit einbezogen werden. Spalte zwei zeigt die Anzahl der D-Bits pro Schaltung und Spalte drei die D-Rate, welche bei allen Schaltungen teilweise deutlich unter der X-Rate ist. Spalte vier zeigt genauere Informationen der D-Bit Heuristik: So werden pro D-Bit im Durchschnitt mehr als drei Fehler erkannt. Die Anzahl der Zwischensignaturen, und somit der Speicheraufwand, sind, wie Spalten fünf bis acht für die verschiedenen MISR-Größen m zu entnehmen ist, ebenfalls geringer als bei der Lösung mit sieben X-freien Kombinationen, bei kleineren MISR Größen teilweise 30%.

Schaltung	**Anzahl D-Bits**	**D-Rate**	**durchschnittl. Anzahl Fehler pro D-Bit**	**Anzahl Zwischensignaturen für m=**			
				64	**128**	**256**	**512**
b14_1	5.745	1,45 %	3,2	177	105	64	35
b17_1	23.916	0,49 %	2,3	1.125	671	380	204
b18_1	56.665	0,22%	2,3	n.m.	2.562	1.270	635
b20_1	15.216	0,74%	3,0	560	337	196	107
b21_1	15.192	0,72%	3,0	661	409	239	131
b22_1	25.229	0,52%	2,9	1.188	723	412	226
p35k	5.939	0,05%	7,0	462	243	130	68
p45k	15.359	2,88%	2,9	2.260	1.409	813	446
p78k	71.520	2,16%	2,8	6.650	3.798	2.055	1.072
p81k	103.566	0,65%	3,6	n.m.	6.693	3.845	2.077

Tabelle 6.5: Anzahl Zwischensignaturen mit D-Bits in Abhängigkeit der MISR-Größe

Schaltung	Fehlerabdeckung		Fehlereffizienz		Speicheraufwand [KB]	
	7 X-freien Komb.	D-Bit Heuristik	7 X-freien Komb.	D-Bit Heuristik	7 X-freien Komb.	D-Bit Heuristik
b14_1	89,87%	89,87%	100%	100%	18,9	17,9
b17_1	95,95%	95,95%	100%	100%	111,1	104,4
b18_1	95,78%	95,78%	100%	100%	325,1	307,2
b20_1	91,99%	91,99%	100%	100%	57,9	54,8
b21_1	90,19%	90,19%	100%	100%	70,1	67,1
b22_1	91,66%	91,66%	100%	100%	122,9	115,7
p35k	89,75%	89,75%	100%	100%	34,8	34,8
p45k	91,80%	91,80%	100%	100%	250,9	228,4
p78k	87,50%	87,50%	99,99%	99,99%	594,9	548,9
p81k	97,00%	97,00%	99,99%	99,99%	1.169,9	1.063,0

Tabelle 6.6: Vergleich Fehlerabdeckung und Speicheraufwand für 512-Bit MISR

Tabelle 6.6 zeigt den Vergleich für den realistischeren Fall eines 512 Bit breiten MISRs. Spalten zwei und drei zeigen jeweils pro Schaltung die Fehlerabdeckungen, die mit der Auswertung der Zwischensignaturen für die Lösung mit sieben X-freien Kombinationen und die Lösung mit der D-Bit Heuristik erreicht werden. Die Fehlerabdeckung erreicht nie 100%, allerdings liegt das daran, dass bei einigen Fehlern der Fehlereffekt durch einen unbekannten Wert am Schaltungsausgang überdeckt wird. Spalten vier und fünf geben jeweils für die D-Bit Heuristik als auch die Lösung mit 7 X-freien Kombinationen die Fehlereffizienz an, um die entwickelte Kompaktierung zu validieren. Auffällig ist, dass die D-Bit Heuristik bei allen Schaltungen die gleiche Fehlerabdeckung sowie Fehlereffizienz erreicht wie die Lösung mit 7 X-freien Kombinationen und die Fehlereffizienz, bis auf zwei Ausnahmen, stets 100% beträgt. Somit entsteht kein (bis auf zwei Schaltungen) Aliasing durch die D-Bit Heuristik und das MISR. Spalten sechs und sieben geben den Speicheraufwand in Kilobyte an, die für die zusätzlichen Zwischensignaturen benötigt werden. Wie man sieht, lässt sich mit der D-Bit Heuristik Speicher einsparen, im Durchschnitt ca. 8%.

Insgesamt zeigen die Experimente, dass FAST effektiv als Selbsttest implementiert werden kann. Die Anzahl der Gruppen ist gering, genauso wie die Anpassungen der Hardware und der zusätzliche Speicheraufwand: So müssen neben der Taktgenerierung lediglich einige hundert Signaturen auf dem Chip abgelegt werden, was bei einer Schaltung mit fast 80.000 Gattern einem zusätzlichen 1 MByte-Speicher entspricht.

6.6 Zusammenfassung

In diesem Kapitel wurde ein Faster-than-at-Speed-Test vorgestellt, der erstmals im Selbsttest (FAST-BIST) kleine Verzögerungsfehler (SDDs) erkennen kann, die mit nominaler Frequenz nicht hätten erkannt werden können. SDDs weisen auf ein Verhalten hin, welches außerhalb der Spezifikation liegt, allerdings noch nicht direkt zu einem fehlerhaften Verhalten führt. Allerdings ist dieses Verhalten deswegen so kritisch, da es während des Betriebs durch externe Störeinflüsse oder Alterungseffekte zu einem Early-Life Fehler führen kann und somit einen negativen Einfluss auf die Zuverlässigkeit der Schaltung hat.

Die Grundidee im FAST ist es deshalb, die Ausgaben der Schaltung im Test mit einer höheren als der nominalen Frequenz zu beobachten, um so das Fehlverhalten festzustellen. Es werden mehrere vorverarbeitende Schritte durchgeführt und ein neuer Kompaktierer auf dem Chip implementiert, damit der FAST im Selbst- oder eingebetteten Test durchgeführt werden kann. Von einer gegebenen Testmustermenge werden zunächst diejenigen SDDs bestimmt, die nicht mit der nominalen oder maximalen FAST-Frequenz erkannt werden können. Für jeden relevanten Fehler werden dann, mit Hilfe einer Fehlersimulation, die Erkennungsbereiche bestimmt und in FAST-Gruppen eingeteilt. Die jeweiligen Testmuster werden mit den entsprechenden Frequenzen simuliert, um die Ausgänge zu identifizieren, an denen unbekannte Werte ('X') anliegen, da der Pfad nicht für die höhere FAST-Frequenz ausgelegt ist. Zusätzlich werden auch die Ausgänge bestimmt, an denen sich Fehlerinformationen auswirken. Mit Hilfe dieser Informationen kann das MISR, welches als Kompaktierer eingesetzt wird, eine hohe Fehlerabdeckung und hohe Ausbeute garantieren: Durch Abspeicherung von Zwischensignaturen und Zurücksetzen des MISRs ist der Kompaktierer so flexibel, dass er Testantworten mit stark ändernden und

hohen X-Raten, die bei FAST-Gruppen mit hohen Frequenzen zu erwarten sind, verarbeiten kann.

Als Optimierungsprobleme ergeben sich hieraus zum einen die Minimierung der FAST-Gruppen als auch die Minimierung der Zwischensignaturen. Für beide Optimierungsprobleme werden Annäherungen mit Heuristiken gefunden: Die FAST-Gruppen werden mittels eines Algorithmus' bestimmt, der die größten unteren Grenzen der Erkennungsbereiche als FAST-Taktperiode übernimmt. Für die Minimierung der Zwischensignaturen wird das Problem auf die Minimierung der Anzahl der D-Bits reduziert, welches als Überdeckungsproblem mit einem greedy-Algorithmus gelöst wird.

Experimente haben gezeigt, dass selbst für große Schaltungen mit vielen möglichen kleinen Verzögerungsfehlern die Anzahl der FAST-Gruppen vierzehn nicht übersteigt und durchschnittlich über 85% zusätzliche Fehlerabdeckung durch den Test mit höheren Taktfrequenzen erreicht wird. Die hohe Anzahl der unbekannten Werte, die während des Tests auftreten, können durch den vorgestellten Kompaktierer effizient durch Abspeicherung von wenigen zusätzlichen Zwischensignaturen und einer Auswertung toleriert werden. Der Speicheraufwand betrug hierbei selbst im ungünstigsten Fall weniger als 1 MByte.

Im nächsten Kapitel wird die gesamte Arbeit zusammengefasst, zukünftige Denkanstöße für weitere Arbeiten gegeben, sowie die mit dieser Arbeit einhergehenden Publikationen zusammengefasst.

7 Zusammenfassung

In dieser Arbeit wurden neben der Problematik des Selbsttests von hochzuverlässigen Schaltungen für sicherheitskritische Anwendungen vor allem die Probleme von frühzeitigen Systemausfällen (Early-Life Fehler) auf die Produktqualität und Ausbeute behandelt: Auf der einen Seite werden durch eingebaute Redundanzen Fehler zwar toleriert, trotzdem sollen im Produktionstest Fehler erkannt werden, so dass der Test auf Strukturebene ansetzen muss. Auf der anderen Seite sollen transiente Fehler nicht zu unnötigen Ausbeuteverlusten führen. Zusätzlich soll der Test auch Schwachstellen im System erkennen, die im weiteren Verlauf des Produktlebenszyklus' zu einem Totalausfall führen könnten. Aus diesem Grund wurde in dieser Arbeit ein Selbsttest mit minimalem Hardwareaufwand vorgestellt, der es erlaubt, zwischen nicht-kritischen transienten Fehlern und kritischen permanenten bzw. intermittierenden Fehlern zu unterscheiden. Hierzu wurde der aus [AHHW08a] [AHHW08b] bekannte Selbsttest mit Rücksetzpunkten, der den Test in Sitzungen einteilt und gezielt fehlerhafte Sitzungen wiederholt, durch eine extreme Kompaktierung optimiert. Dabei sieht die extreme Kompaktierung entweder eine Paritätslogik oder ein zusätzliches MISR als zusätzliche räumliche Kompaktierung vor. Es wird entweder ein Paritätsfenster über die Zustände des MISRs berechnet oder es werden, nach einigen zusätzlichen Zustandsübergängen im Schatten-MISR, nur wenige Bits der Signatur ausgewertet. In den Experimenten wurde gezeigt, dass die Signatur 4X- 12X kompaktiert werden kann, was für den Selbsttest mit Rücksetzpunkten mit seinen zusätzlichen Zwischensignaturen eine große Hardware-Einsparung bedeutet, bei gleichzeitiger Minimierung des Aliasings und Maximierung der Fehlerabdeckung.

Für die weitere Klassifikation und Diagnose wurde ein zusätzlicher Fehlerspeicher implementiert, durch dessen Auswertung Rückschlüsse auf das zeitliche Verhalten des Fehlers gezogen werden konnten: Permanente, aber auch intermittierende Fehler sind kritisch, da ein intermittierender Fehler eine Schwachstelle im System darstellt und zu einem frühzeitigen Systemausfall führen kann. Diese benötigen keine weitere Klassifikation. Wenn jedoch nicht zwischen transient und intermittierend unterschieden werden kann, wird eine Analyse mit Hilfe von Bayesschen Netzen durchgeführt, um das Fehlverhalten zu klassifizieren. In den Experimenten wurde gezeigt, dass durch dieses integrative Verfahren von Test und Diagnose sowie Bayesschen Modellen zur gezielten Klassifikation eine hohe Rate von intermittierenden Fehlern korrekt als kritisch eingestuft wurden, bei gleichzeitiger Minimierung der Ausbeuteverluste durch transiente Fehler.

Für kleine Verzögerungsfehler wiederum, die ebenfalls Indikatoren für frühzeitige Systemausfälle sind und nicht mit Standard-Testverfahren erkannt werden können, wurde

ein integrierter Faster-than-at-Speed-Selbsttest vorgestellt. Dieser FAST-BIST erlaubt es, durch Erhöhung der Testfrequenz auch kleine versteckte Verzögerungsfehler zu erkennen, die auf kurzen Pfaden liegen. Hierzu wird die Testmustermenge in FAST-Gruppen unterschiedlicher Test-Frequenzen eingeteilt. Für die Kompaktierung auf dem Chip wurde ein X-toleranter Kompaktierer eingeführt, der es erlaubt, sehr flexibel unterschiedliche und hohe X-Raten zu tolerieren. Hierzu werden Zwischensignaturen in einem zusätzlichen Speicher abgelegt und die Fehlereffekte anschließend durch Linearkombinationen der MISR-Bits unabhängig von unbekannten Werten separiert. Gerade für größere Schaltungen ist dieses Verfahren sehr effektiv, da nur wenige zusätzliche Zwischensignaturen abgelegt werden müssen und dadurch der Hardwareaufwand nur gering ansteigt.

7.1 Zusammenfassung der eigenen Veröffentlichungen

In der vorliegenden Arbeit wurden Ideen und Experimente dargestellt, welche die Fehlercharakterisierung in hochzuverlässigen Schaltungen behandeln. Im Vordergrund stand hierbei die Identifikation von Schwachstellen im System, die sich als intermittierende Fehler und kleine Verzögerungsfehler auswirken, bei gleichzeitiger Minimierung der Ausbeuteverluste durch transiente Fehler. Im Verlauf dieser Arbeit konnten bereits einzelne Zwischenergebnisse publiziert werden:

- Der Selbsttest mit Rücksetzpunkten beruht auf der Idee, durch gezielte Wiederholungen zwischen permanenten und nicht-permanenten Fehlern zu unterscheiden. Hierzu wird der Test in Sitzungen aufgeteilt und nach jeder Sitzung Soll- und Ist-Signatur miteinander verglichen. Für die Implementierung im Selbsttest ist es deshalb notwendig, die Referenzsignaturen auf dem Chip vorzuhalten. Aber auch als eingebetteter Test hat das Verfahren Nachteile: Nach jeder Sitzung muss die komplette Signatur über den TAM an die Schaltung angelegt werden. Um diesen Hardware- oder Kommunikationsaufwand zu minimieren, wurde in [ISH10b] [ISH10a] [Ind13] die zweistufige Kompaktierung aus Kapitel 3 eingeführt, bei der die Signatur durch eine zusätzliche Paritätslogik räumlich kompaktiert wurde. Hieraus ließen sich, ohne Aliasing-Verluste, Kompaktierungsraten von 4X- 12X erreichen, im Vergleich zu einer rein räumlichen Kompaktierung mit einem MISR.

- In [CHIW11b] [CHIW11b] wurden einige Anpassungen entwickelt, um den obigen Test mit einem integrierten Diagnoseverfahren zu kombinieren: Beispielsweise wurde das Schattenregister in ein Schatten-MISR umgewandelt, in dem nur einige Bits der Signatur mit Referenzbits verglichen wurden. Das Problem der Fehlermaskierung bei Betrachtung nur einiger weniger Bits wurde bisher mit der Paritätsberechnung über die gesamte Signatur umgangen und wird hier durch die Zustandsüberführungen gelöst. Für die Diagnose wird ein zusätzlicher Fehlerspeicher benötigt und durch Auswertungen der fehlerhaften Signaturen lässt sich der Fehler genau lokalisieren. Es konnte gezeigt werden, dass die Fehlerabdeckung nahezu identisch ist mit der Fehlerabdeckung eines Ansatzes ohne Kompaktierung und dass trotzdem eine hohe diagnostische Auflösung erreicht wird.

- Ein kombinierter Ansatz zur Klassifikation wurde in [CGH+13] [GCI+14] beschrieben, um die nicht-permanenten Fehler zu unterscheiden: Zunächst wurde der

Fehlerspeicher modifiziert, so dass neben der fehlerhaften Signatur und der dazugehörigen Sitzungsnummer auch noch das adaptive Verhalten abgelegt werden konnte. Gleichzeitig konnten mit Hilfe Bayesscher Netze diejenigen Fehler, die nicht eindeutig kritisch waren, weiter klassifiziert werden. Hierzu wurde ein Modell beschrieben, welches so skalierbar ist, dass der Fokus entweder auf höherer Produktqualität oder auf geringerer Ausbeuteverluste liegt. Durch die Klassifikation transienter Fehler ließen sich weitere Ausbeuteverluste minimieren.

- In [HIK+14] wurde erstmals ein Selbsttest vorgestellt, der auch kleine versteckte Verzögerungsfehler durch Erhöhung der Test-Taktfrequenz erkennen kann. Das Problem, dass unbekannte Werte an Pfaden, die nicht für diese höhere Frequenz ausgelegt sind, auftreten, wurde mit einem X-toleranten Kompaktierer gelöst. Hierzu werden Zwischenzustände des MISRs in einen eingebetteten Speicher ablegt und die X-freien Bits berechnet, welche mögliche Fehlerinformationen enthalten. Hieraus lassen sich 50% zusätzliche Fehlerabdeckung erreichen. In der Masterarbeit [Kam14] wurde die für diese Arbeit wichtige FAST-Gruppenerzeugung implementiert, welche die kleinen Verzögerungsfehler analysiert und in möglichst wenige FAST-Gruppen mit minimaler FAST-Taktfrequenz einsortiert.

7.2 Ausblick

In dieser Dissertation wurden Verfahren entwickelt, um hochzuverlässige Schaltungen effizient mit hoher Produktqualität bei gleichzeitiger Minimierung der Ausbeuteverluste effizient zu testen. Aufbauend auf den beschriebenen Verfahren lassen sich Untersuchungen in verschiedene Richtungen weiter verfolgen, beispielsweise bei der Identifizierung und Erkennung von Schwachstellen eines Systems. So lassen sich gerade im jungen Forschungsgebiet des FAST-BIST, z.B. durch geeignete Gruppeneinteilungen oder auch verbesserte Kompaktierer, Lösungen finden, die den Hardware- und Simulationsaufwand verbessern. Hierzu wären neben genauen Timing-Analysen und Algorithmen zur Gruppeneinteilung auch umfangreiche Simulationen erforderlich, um die deterministischen Bits besser über die Menge der Testantworten zu verteilen. Erste Simulationen deuten darauf hin, dass auch ein mehrstufiger Kompaktierer geeignet ist, den Speicher der Testantworten zu minimieren.

Im Bereich der Klassifikation der intermittierenden Fehler könnten auch andere Fehlereffekte betrachtet werden, die bei noch kleineren Strukturgrößen auftreten können. Neben der empirischen Untersuchung solcher Effekte könnten die Bayesschen Modelle optimiert werden, wobei ein vorgeschalteter Prozess die *a priori* Wahrscheinlichkeiten bestimmt.

Literaturverzeichnis

[AA01] AMINIAN, F. ; AMINIAN, M.: Fault Diagnosis of Analog Circuits Using Bayesian Neural Networks with Wavelet Transform as Preprocessor. In: *Journal of Electronic Testing - Theory and Applications* 17 (2001), Nr. 1, S. 29–36.

[ABS+05] ARNIM, K. von ; BORINSKI, E. ; SEEGEBRECHT, P. ; FIEDLER, H. ; BREDERLOW, R. ; THEWES, R. ; BERTHOLD, J. ; PACHA, C.: Efficiency of Body Biasing in 90-nm CMOS for Low-Power Digital Circuits. In: *IEEE Journal of Solid-State Circuits* 40 (2005), Nr. 7, S. 1549–1556.

[AC05] AMODEO, M. ; CORY, B.: Defining faster-than-at-speed delay tests. In: *Cadence Nanometer Test Quarterly* 2 (2005), Nr. 2.

[AG04] AGOSTA, J. M. ; GARDOS, T.: Bayes Network "Smart Diagnostics". In: *Intel Technology Journal "Toward The Proactive Enterprise"* 8 (2004), Nr. 4, S. 361–372.

[AHHW08a] AMGALAN, U. ; HACHMANN, C. ; HELLEBRAND, S. ; WUNDERLICH, H.-J.: Signature Rollback - A Technique for Testing Robust Circuits. In: *Proceedings IEEE VLSI Test Symposium*, 2008, S. 125–130.

[AHHW08b] AMGALAN, U. ; HACHMANN, C. ; HELLEBRAND, S. ; WUNDERLICH, H.-J.: Testen mit Rücksetzpunkten - ein Ansatz zur Verbesserung der Ausbeute bei robusten Schaltungen. In: *20. GI/ITG/GMM Workshop „Testmethoden und Zuverlässigkeit von Schaltungen und Systemen"*. Wien, Österreich, 2008.

[AKS93] AGRAWAL, V. D. ; KIME, C. R. ; SALUJA, K. K.: A Tutorial on Built-In Self-Test, Part 2: Applications. In: *IEEE Design Test of Computers* 10 (1993), Nr. 2, S. 69–77.

[ANV06] AMYEEN, M.E. ; NAYAK, D. ; VENKATARAMAN, S.: Improving Precision Using Mixed-level Fault Fiagnosis. In: *Proceedings IEEE International Test Conference*, 2006, S. 1–10.

[ATJ06] AHMED, N. ; TEHRANIPOOR, M. ; JAYARAM, V.: A Novel Framework for Faster-than-at-Speed Delay Test Considering IR-drop Effects. In: *IEEE/ACM International Conference on Computer-Aided Design*, 2006, S. 198–203.

[AYMM03] AL-YAMANI, A. A. ; MITRA, S. ; MCCLUSKEY, E. J.: BIST reseeding with very few seeds. In: *Proceedings IEEE VLSI Test Symposium*, 2003, S. 69–74.

[BA00] BUSHNELL, M. ; AGRAWAL, V.: *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Springer Netherlands, 2000.

[Bar12] BARBER, D.: *Bayesian Reasoning and Machine Learning*. New York: Cambridge University Press, 2012.

[BG07] BEN-GAL, I.: Bayesian Networks. In: RUGGERI, F. (Hrsg.) ; FALTIN, F. (Hrsg.) ; KENETT, R. (Hrsg.): *Encyclopedia of Statistics in Quality & Reliability*. Wiley & Sons, 2007.

[BHHS01] BARTENSTEIN, T. ; HEABERLIN, D. ; HUISMAN, L. ; SLIWINSKI, D.: Diagnosing Combinational Logic Designs Using the Single Location At-a-Time (SLAT) Paradigm. In: *Proceedings IEEE International Test Conference*, 2001, S. 287–296.

[BKK04] BARFORD, L. ; KANEVSKY, V. ; KAMAS, L.: Bayesian Fault Diagnosis in Large-Ccale Measurement Systems. In: *Proceedings IEEE Instrumentation and Measurement Technology Conference* Bd. 2, 2004, S. 1234–1239.

[BM82] BARDELL, P. H. ; MCANNEY, W. H.: Self-Testing of Multichip Logic Modules. In: *Proceedings IEEE International Test Conference*, 1982, S. 200–204.

[BO02] BAYRAKTAROGLU, I. ; ORAILOGLU, A.: Gate Level Fault Diagnosis in Scan-Based BIST. In: *Proceedings IEEE Design, Automation and Test in Europe Conference & Exhibition*, 2002, S. 376–381.

[CB95] CHANDRAKASAN, A. P. ; BRODERSEN, R. W.: Minimizing Power Consumption in Digital CMOS Circuits. In: *Proceedings of the IEEE* 83 (1995), Nr. 4, S. 498–523.

[CEWA11] COOK, A. ; ELM, M. ; WUNDERLICH, H.-J. ; ABELEIN, U.: Structural In-Field Diagnosis for Random Logic Circuits. In: *Proceedings 16th IEEE European Test Symposium*, 2011, S. 111–116.

[CGH+13] COOK, A. ; GOMEZ, L. R. ; HELLEBRAND, S. ; INDLEKOFER, T. ; WUNDERLICH, H.-J.: Adaptive Test and Diagnosis of Intermittent Faults. In: *14th Latin American Test Workshop*. Cordoba, Argentina, 2013.

[CHE+08] CZUTRO, A. ; HOUARCHE, N. ; ENGELKE, P. ; POLIAN, I. ; COMTE, M. ; RENOVELL, M. ; BECKER, B.: A Simulator of Small-Delay Faults Caused by Resistive-Open Defects. In: *13th IEEE European Test Symposium*, 2008, S. 113–118.

[CHIW11a] COOK, A. ; HELLEBRAND, S. ; INDLEKOFER, T. ; WUNDERLICH, H.-J.: Diagnostic Test of Robust Circuits. In: *Proceedings 20th IEEE Asian Test Symposium*, 2011, S. 285–290.

[CHIW11b] COOK, A. ; HELLEBRAND, S. ; INDLEKOFER, T. ; WUNDERLICH, H.-J.: Robuster Selbsttest mit Diagnose. In: *5. GMM/GI/ITG Fachtagung Zuverlässigkeit und Entwurf*. Hamburg, 2011.

[Con07] CONSTANTINESCU, C.: Impact of Intermittent Faults on Nanocomputing Devices. In: *Proceedings Workshop Dependable and Secure Nanocomputing*. Edingburgh, UK, 2007.

[CSR+06] CHENG, W.-T. ; SHARMA, M. ; RINDERKNECHT, T. ; LAI, L. ; HILL, C.: Signature Based Diagnosis for Logic BIST. In: *Proceedings IEEE International Test Conference*, 2006, S. 1–9.

[Cul97] CULLEN, C.G.: *Linear Algebra with Applications (2nd edition)*. Pearson, 1997.

[D+06] DAS, S. u. a.: A Self-Tuning DVS Processor Using Delay-Error Detection and Correction. In: *IEEE Journal of Solid-State Circuits*, 2006, S. 792–804.

[DPB06] DESINENI, R. ; POKU, O. ; BLANTON, R. D.: A Logic Diagnosis Methodology for Improved Localization and Extraction of Accurate Defect Behavior. In: *Proceedings IEEE International Test Conference*, 2006, S. 1–10.

[E+04] ERNST, D. u. a.: Razor: Circuit-Level Correction of Timing Errors for Low-Power Operation. In: *IEEE Micro* 24 (2004), Nr. 6, S. 10–20.

[EW10] ELM, M. ; WUNDERLICH, H.-J.: BISD: Scan-Based Built-In Self-Diagnosis. In: *Proceedings IEEE Design, Automation and Test in Europe Conference & Exhibition*, 2010, S. 1243 – 1248.

[Fec08] FECHNER, B.: A Dynamic Fault Classification Scheme. In: *Proceedings European Conference on Safety and Reliability*, 2008, S. 147–153.

[FGGL99] FAGOT, C. ; GASCUEL, O. ; GIRARD, P. ; LANDRAULT, C.: On Calculating Efficient LFSR Seeds for Built-In Self Test. In: *Proceedings European Test Workshop*, 1999, S. 7–14.

[FLL12] FU, X. ; LI, H. ; LI, X.: Testable Path Selection and Grouping for Faster Than At-Speed Testing. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 20 (2012), Nr. 2, S. 236–247.

[GCI+14] GÓMEZ, L. R. ; COOK, A. ; INDLEKOFER, T. ; HELLEBRAND, S. ; WUNDERLICH, H.-J.: Adaptive Bayesian Diagnosis of Intermittent Faults. In: *Journal of Electronic Testing* 30 (2014), Nr. 5, S. 527–540.

[Gom11] GOMEZ, L. R.: *Simulation Framework for Built-In diagnosis of Self-Checking Circuits*, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Diplomarbeit Nr. 3069, 2011.

[GPT08] GARG, R. ; PUTMAN, R. ; TOUBA, N.A.: Increasing Output Compaction in Presence of Unknowns Using an X-Canceling MISR with Deterministic Observation. In: *Proceedings 26th IEEE VLSI Test Symposium*, 2008, S. 35–42.

[GSB+08] GRACIA, J. ; SAIZ, L.J. ; BARAZA, J.C. ; GIL, D. ; GIL, P.J.: Analysis of the influence of intermittent faults in a microcontroller. In: *11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*, 2008, S. 1–6.

[GWV+04] GHERMAN, V. ; WUNDERLICH, H.-J. ; VRANKEN, H. ; HAPKE, F. ; WITTKE, M. ; GARBERS, M.: Efficient pattern mapping for deterministic logic BIST. In: *Proceedings IEEE International Test Conference*, 2004, S. 48–56.

[HHLL05] HAN, Y. ; HU, Y. ; LI, H. ; LI, X.: Theoretic Analysis and Enhanced X-Tolerance of Test Response Compact based on Convolutional code. In: *Proceedings of the Asia and South Pacific Design Automation Conference* Bd. 1, 2005, S. 53–58.

[HHW+09] HAKMI, A.-W. ; HOLST, S. ; WUNDERLICH, H.-J. ; SCHLÖFFEL, J. ; HAPKE, F. ; GLOWATZ, A.: Restrict Encoding for Mixed-Mode BIST. In: *27th IEEE VLSI Test Symposium*, 2009, S. 179–184.

[HIK+14] HELLEBRAND, S. ; INDLEKOFER, T. ; KAMPMANN, M. ; KOCHTE, M. ; LIU, C. ; WUNDERLICH, H.-J.: FAST-BIST: Faster-than-At-Speed BIST Targeting Hidden Delay Defects. In: *Proceedings IEEE International Test Conference*, 2014, S. 1–8.

[Hil04] HILLERINGMANN, U.: *Silizium-Halbleiterrechnologie*. Bd. 5. Teubner Verlag, 2004.

[HM93] HAO, H. ; MCCLUSKEY, E.J.: Very-low-voltage testing for weak CMOS logic ICs. In: *Proceedings International Test Conference*, 1993, S. 275–284.

[HRT+95] HELLEBRAND, S. ; RAJSKI, J. ; TARNICK, S. ; VENKATARAMAN, S. ; COURTOIS, B.: Built-In Test for Circuits with Scan Based on Reseeding of Multiple-Polynomial Linear Feedback Shift Registers. In: *IEEE Transactions on Computers* 44 (1995), Nr. 2, S. 223–233.

[HW07] HOLST, S. ; WUNDERLICH, H.-J.: Adaptive Debug and Diagnosis Without Fault Dictionaries. In: *Proceedings 12th IEEE European Test Symposium*, 2007, S. 7–12.

[HW09] HOLST, S. ; WUNDERLICH, H.-J.: Adaptive Debug and Diagnosis Without Fault Dictionaries. In: *Journal of Electronic Testing* 25 (2009), Nr. 4-5, S. 259–268.

[Ind13] INDLEKOFER, T.: Signature rollback with extreme compaction - A technique for testing robust VLSI circuits with reduced hardware overhead. In: *ANNALES Universitatis Scientiarum Budapestinensis de Rolando Eötvös Nominatae, Sectio Computatorica* 39 (2013), S. 161–180.

[inta] *Informationen der Intel Prozessoren*. `http://www.intel.com/pressroom/kits/events/moores_law_40th/`, Abruf: 28.08.2014.

[intb] *Intel Pressemittelung*. `http://newsroom.intel.com/community/intel_newsroom/blog/2011/01/31/intel-identifies-chipset-design-error-implementing-solution`, Abruf: 12.01.2015.

[ISH10a] INDLEKOFER, T. ; SCHNITTGER, M. ; HELLEBRAND, S.: Efficient Test Response Compaction for Robust BIST using Parity Sequences. In: *Proceedings IEEE International Conference on Computer Design*, 2010, S. 480–485.

[ISH10b] INDLEKOFER, T. ; SCHNITTGER, M. ; HELLEBRAND, S.: Robuster Selbsttest mit extremer Kompaktierung. In: *4. GMM/GI/ITG-Fachtagung Zuverlässigkeit und Entwurf*. Wildbach Kreuth, 2010, S. 17–24.

[ITR] *International Roadmap for Semiconductors, ITRS Report 2012*. `http://public.itrs.net/`, Abruf: 06.12.2013.

[Juh03] JUHNKE, T.: *Die Soft-Error-Rate von Submikrometer-CMOS-Logikschaltungen*, Technische Universität Berlin, Dissertation, 2003.

[Kam14] KAMPMANN, M.: *Ein Verfahren zur Testmustererzeugung für kleine Verzögerungsfehler*, Universität Paderborn, Institut für Elektrotechnik und Informationstechnik, Masterarbeit, 2014.

[Kar72] KARP, R.M.: Reducibility among Combinatorial Problems. In: MILLER, R. (Hrsg.) ; THATCHER, J. (Hrsg.) ; BOHLINGER, J. (Hrsg.): *Complexity of Computer Computations*. Springer US, 1972 (The IBM Research Symposia Series), S. 85–103.

[KBK+01] KOENEMANN, B. ; BARNHART, C. ; KELLER, B. ; SNETHEN, T. ; FARNSWORTH, O. ; WHEATER, D.: A SmartBIST Variant with Guaranteed Encoding. In: *Proceedings 10th IEEE Asian Test Symposium*, 2001, S. 325–330.

[KDBK10] KRISHNAN, S. ; DOORNBOS, K.D. ; BRAND, R. ; KERKHOFF, H. G.: Block-Level Bayesian Diagnosis of Analogue Electronic Circuits. In: *Proceedings IEEE Design, Automation and Test in Europe Conference & Exhibition*, 2010, S. 1767–1772.

[KJT01] KRISHNA, C.V. ; JAS, A. ; TOUBA, N.A.: Test Vector Encoding Using Partial LFSR Reseeding. In: *Proceedings IEEE International Test Conference*, 2001, S. 885–893.

[KK05] KUMAR, R. ; KURSUN, V.: Voltage Optimization for Temperature Variation Insensitive CMOS Circuits. In: *Proceedings 48th IEEE Midwest Symposium on Circuits and Systems* Bd. 1, 2005, S. 476–479.

[KK07] KOREN, I. ; KRISHNA, C.M.: *Fault-Tolerant Systems*. Morgan Kaufmann Publishers, 2007.

[KKK+10] KIM, Y.M. ; KAMEDA, Y. ; KIM, H. ; MIZUNO, M. ; MITRA, S.: Low-Cost Gate-Oxide Early-Life Failure Detection in Robust Systems. In: *IEEE Symposium on VLSI Circuits*, 2010, S. 125–126.

[Kle09] KLEER, J. D.: Diagnosing Multiple Persistent and Intermittent Faults. In: *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, 2009, S. 733–738.

[Koe91] KOENEMANN, B.: LFSR-Coded Test Patterns for Scan Design. In: *Proceedings European Test Conference*. München, Deutschland, 1991, S. 237–242.

[KP11] KRAUSE, P.K. ; POLIAN, I.: Adaptive Voltage Over-Scaling for Resilient Applications. In: *Proceedings IEEE Design, Automation Test in Europe Conference*, 2011, S. 1–6.

[KR05] KAUFMAN, L. ; ROUSSEEUW, P.J.: *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, Inc., 2005.

[KT04] KRISHNA, C.V. ; TOUBA, N.A.: 3-Stage Variable Length Continuous-Flow Scan Vector Decompression Scheme. In: *Proceedings 22nd IEEE VLSI Test Symposium*, 2004, S. 79–86.

[LCG02] LIU, C. ; CHAKRABARTY, K. ; GOESSEL, M.: An Interval-based Diagnosis Scheme for Identifying Failing Vectors in a Scan-BIST Environment. In: *Proceedings IEEE Design, Automation and Test in Europe Conference & Exhibition*, 2002, S. 382–386.

[LGM04] LEININGER, A. ; GOESSEL, M. ; MUHMENTHALER, P.: Diagnosis of Scan-Chains by Use of a Configurable Signature Register and Error-Correcting Codes. In: *Proceedings IEEE Design, Automation and Test in Europe Conference & Exhibition* Bd. 2, 2004, S. 1302–1307.

[LM08] LEE, J. ; MCCLUSKEY, E.J.: Failing Frequency Signature Analysis. In: *Proceedings IEEE International Test Conference*, 2008, S. 1–8.

[LN96] LIDL, R. ; NIEDERREITER, H.: *Finite Fields*. Zweite Edition. Cambridge University Press, 1996.

[LNO06] LIU, F. ; NIKOLOV, P.K. ; OZEV, S.: Parametric Fault diagnosis for Analog Circuits Using a Bayesian Framework. In: *Proceedings 24th IEEE VLSI Test Symposium*, 2006, S. 272–277.

[LSC10] L-SHI ; CAI, X.: An Exact Fast Algorithm for Minimum Hitting Set. In: *Third IEEE International Joint Conference on Computational Science and Optimization*, 2010, S. 64–67.

[MF00] MCLAURIN, T.L. ; FREDERICK, F.: The testability features of the MCF5407 containing the 4th generation ColdFire®microprocessor core. In: *Proceedings IEEE International Test Conference*, 2000, S. 151–159.

[MK04] MITRA, S. ; KIM, K.S.: X-Compact: An Efficient Response Compaction Technique. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 23 (2004), Nr. 3, S. 421–432.

[Moo65] MOORE, G.E.: Cramming More Components onto Integrated Circuits. In: *Electronics Magazine* 38 (1965), Nr. 8.

[MR10] MUSTAFA, N.H. ; RAY, S.: Improved Results on Geometric Hitting Set Problems. In: *Discrete & Computational Geometry* 44 (2010), Nr. 4, S. 883–895.

[Nic07] NICOLAIDIS, M.: GRAAL: A New Fault Tolerant Design Paradigm for Mitigating the Flaws of Deep Nanometric Technologies. In: *Proceedings IEEE International Test Conference*, 2007, S. 1–10.

[NPRK03] NARUSE, M. ; POMERANZ, I. ; REDDY, S.M. ; KUNDU, S.: On-chip Compression of Output Responses with Unknown Values Using LFSR Reseeding. In: *Proceedings IEEE International Test Conference* Bd. 1, 2003, S. 1060–1068.

[OMCE05] O'FARRILL, C. ; MOAKIL-CHBANY, M. ; EKLOW, B.: Optimized Reasoning-Based Diagnosis for Non-Random, Board-Level, Production Defects. In: *Proceedings IEEE International Test Conference*, 2005, S. 173–179.

[PB06] PRESS, R. ; BOYER, J.: Easily Implement PLL Clock Switching for At-Speed Test. In: *Chip Design* (2006). `http://chipdesignmag.com/display.php?articleId=376`, Abruf: 18.08.2015.

[PCKB07] POLIAN, I. ; CZUTRO, A. ; KUNDU, S. ; BECKER, B.: Power Droop Testing. 24 (2007), Nr. 3, S. 276–284.

[Pea88] PEARL, J.: *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, Inc., 1988.

[PJ72] PETERSON, W.W. ; JR., E.J. W.: *Error-Correcting Codes*. MIT Press, 1972.

[PLR03] PATEL, J.H. ; LUMETTA, S.S. ; REDDY, S.M.: Application of Saluja-Karpovsky Compactors to Test Responses with Many Unknowns. In: *Proceedings 21st IEEE VLSI Test Symposium*, 2003, S. 107–112.

[PT00] PRZYTULA, K.W. ; THOMPSON, D.: Construction of Bayesian Networks for Diagnostics. In: *Proceedings IEEE Aerospace Conference* Bd. 5, 2000, S. 193–200.

[RBO03] RAO, W. ; BAYRAKTAROGLU, I. ; ORAILOGLU, A.: Test Application Time and Volume Compression through Seed Overlapping. In: *Proceedings IEEE Design Automation Conference*, 2003, S. 732–737.

[RTKM04] RAJSKI, J. ; TYSZER, J. ; KASSAB, M. ; MUKHERJEE, N.: Embedded Deterministic Test. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 23 (2004), Nr. 5, S. 776–792.

[RTWR05] RAJSKI, J. ; TYSZER, J. ; WANG, C. ; REDDY, S.M.: Finite Memory Test Response Compactors for Embedded Test Applications. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 24 (2005), Nr. 4, S. 622–634.

[RTZ98] RAJSKI, J. ; TYSZER, J. ; ZACHARIA, N.: Test Data Decompression for Multiple Scan Designs with Boundary Scan. In: *IEEE Transactions on Computers* 47 (1998), Nr. 11, S. 1188–1200.

[SAK08] SANYAL, A. ; ALAM, S.M. ; KUNDU, S.: A Built-In Self-Test Scheme for Soft Error Rate Characterization. In: *Proceedings 14th IEEE International On-Line Testing Symposium*, 2008, S. 65–70.

[SAK10] SANYAL, A. ; ALAM, S.M. ; KUNDU, S.: BIST to Detect and Characterize Transient and Parametric Failures. In: *IEEE Design & Test of Computers* 27 (2010), Nr. 5, S. 50–59.

[SC05] SHARMA, M. ; CHENG, W.-T.: X-filter: Filtering unknowns from compacted test responses. In: *Proceedings IEEE International Test Conference*, 2005, S. 1090–1098.

[Shi] SHIMPI, A.L: *The Source of Intel's Cougar Point SATA Bug*. `http://www.anandtech.com/show/4143/the-source-of-intels-cougar-point-sata-bug`, Abruf: 12.01.2015.

[SHQ10] SINGH, A. ; HAN, C. ; QIAN, X.: An Output Compression Scheme for Handling X-states from Over-Clocked Delay Tests. In: *Proceedings IEEE VLSI Test Symposium*, 2010, S. 57–62.

[TA08] TAYADE, R. ; ABRAHAM, J.A.: On-chip Programmable Capture for Accurate Path Delay Test and Characterization. In: *Proceedings IEEE International Test Conference*, 2008, S. 1–10.

[TM96] TOUBA, N.A. ; MCCLUSKEY, E.J.: Altering a pseudo-random bit sequence for scan-based BIST. In: *Proceedings IEEE International Test Conference*, 1996, S. 167–175.

[Tou07] TOUBA, N.A.: X-Canceling MISR – An X-Tolerant Methodology for Compacting Output Responses with Unknowns Using a MISR. In: *Proceedings IEEE International Test Conference*, 2007, S. 1–10.

[TPC12] TEHRANIPOOR, M. ; PENG, K. ; CHAKRABARTY, K.: *Test and Diagnosis for Small Delay Defects*. Springer New York, 2012.

[TWV+04] TANG, Y. ; WUNDERLICH, H.-J. ; VRANKEN, H. ; HAPKE, F. ; WITTKE, M. ; ENGELKE, P. ; POLIAN, I. ; BECKER, B.: X-Masking During Logic BIST and Its Impact on Defect Coverage. In: *Proceedings IEEE International Test Conference*, 2004, S. 442–451.

[V+06] VRANKEN, H. u. a.: Fault Detection and Diagnosis with Parity Trees for Space Compaction of Test Responses. In: *Proceedings 43rd ACM/IEEE Design Automation Conference*, 2006, S. 1095–1098.

[VM03] VOLKERINK, E.H. ; MITRA, S.: Efficient Seed Utilization for Reseeding based Compression. In: *Proceedings 21st IEEE VLSI Test Symposium*, 2003, S. 232–237.

[VM05] VOLKERINK, E.H. ; MITRA, S.: Response Compaction with any Number of Unknowns using a new LFSR Architecture. In: *Proceedings 42nd IEEE Design Automation Conference*, 2005, S. 117–122.

[WB81] WILLIAMS, T.W. ; BROWN, N.C.: Defect Level as a Function of Fault Coverage. In: *IEEE Transactions on Computers* C-30 (1981), S. 987–988.

[WK96] WUNDERLICH, H.-J. ; KIEFER, G.: Bit-Flipping BIST. In: *Proceedings IEEE/ACM International Conference on Computer-Aided Design*, 1996, S. 337–343.

[WMF96] WIDMANN, D. ; MADER, H. ; FRIEDRICH, H.: *Technologie hochintegrierter Schaltungen*. 2. Auflage. Springer-Verlag, 1996 (19).

[WR00] WU, J. ; RUDNICK, E.M.: Bridge Fault Diagnosis Using Stuck-At Fault Simulation. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 19 (2000), Nr. 4, S. 489–495.

[Wun91] WUNDERLICH, H.-J.: *Hochintegrierte Schaltungen: Prüfgerechter Entwurf und Test*. Springer-Verlag, 1991.

[WWP04] WOHL, P. ; WAICUKAUSKI, J.A. ; PATEL, S.: Scalable Selector Architecture for X-Tolerant Deterministic BIST. In: *Proceedings 41st Design Automation Conference*, 2004, S. 934–939.

[WWPM02] WOHL, P. ; WAICUKAUSKI, J.A. ; PATEL, S. ; MASTON, G.: Effective Diagnostics through Interval Unloads in a BIST Environment. In: *Proceedings 39th Design Automation Conference*, 2002, S. 249–254.

[WWW06] WANG, L.-T. ; WU, C.-W. ; WEN, X.: *VLSI Test Principles and Architectures: Design for Testability*. Morgan Kaufmann Publishers, 2006.

[YHIF11] YONEDA, T. ; HORI, K. ; INOUE, I. ; FUJIWARA, H.: Faster-Than-At-Speed Test for Increased Test Quality and In-Field Reliability. In: *Proceedings IEEE International Test Conference*, 2011, S. 1–9.

[ZWGC10] ZHANG, Z. ; WANG, Z. ; GU, X. ; CHAKRABARTY, K.: Board-Level Fault Diagnosis using Bayesian Inference. In: *Proceedings 28th IEEE VLSI Test Symposium*, 2010, S. 244–249.

Abbildungsverzeichnis

Tabellenverzeichnis

A Intel Prozessoren

Jahr	Prozessortyp	Anzahl Transistoren	Technologie-knoten	Chip-größe
1971	Intel 4004	2.300	$10\mu m$	$12mm^2$
1987	Intel 8008	3.500	$10\mu m$	$14mm^2$
1974	Intel 8080	4.500	$6\mu m$	$20mm^2$
1978	Intel 8086	29.000	$3\mu m$	$33mm^2$
1982	Intel 80286	134.000	$1,5\mu m$	$49mm^2$
1985	Intel 80386	275.000	$1,5\mu m$	$104mm^2$
1989	Intel 80486	1.180.235	$1\mu m$	$173mm^2$
1993	Pentium	3.100.000	$0,8\mu m$	$294mm^2$
1995	Pentium Pro	5.500.000	$0,5\mu m$	$307mm^2$
1997	Pentium II Klamath	7.500.000	$0,35\mu m$	$195mm^2$
2001	Pentium III Tualatin	45.000.000	$130nm$	$81mm^2$
2005	Pentium 4 Prescott-2M	169.000.000	$90nm$	$143mm^2$
2007	Core 2 Duo	411.000.000	$10\mu m$	$12mm^2$
2008	Core i7 (Quad)	731.000.000	$45nm$	$346mm^2$
2012	Dual-Core Itanium 2	1.700.000.000	$90nm$	$596mm^2$
2014	62-Core Xeon Phi	5.000.000.000	$22nm$	n.b.

Tabelle A.1: Übersicht Intel Prozessoren [inta]