UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

FAKULTÄT FÜR
ELEKTROTECHNIK,
INFORMATIK UND
MATHEMATIK

Eingebauter Selbsttest für kleine Verzögerungsfehler

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

zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften (Dr.-Ing.)

genehmigte Dissertation

von

M. Sc. Matthias Kampmann

Erster Gutachter:	Prof. Dr. Sybille Hellebrand
Zweiter Gutachter:	Prof. Dr. Bernd Becker

Tag der mündlichen Prüfung: 20.02.2020

Paderborn 2020

Diss. EIM-E/353

Danksagung

Das Entwickeln und Schreiben einer Dissertation ist mit großem Aufwand verbunden und sicherlich nicht ohne die Unterstützung weiterer Personen möglich, denen ich an dieser Stelle danken möchte. Zuallererst möchte ich natürlich meiner Doktormutter, Frau Prof. Dr. Sybille Hellebrand, danken. Ohne ihre Unterstützung und zahlreichen Denkanstöße rund um meine Forschungsarbeiten wäre die vorliegende Arbeit sicherlich nicht in dieser Qualität ausgefallen. Mein Dank gebührt auch Herrn Prof. Dr. Bernd Becker von der Universität Freiburg, der sich als Zweitgutachter zur Verfügung gestellt und ebenfalls einige Vorschläge zur Verbesserung beigetragen hat. Weiterhin möchte ich Herrn Prof. Dr. Hans-Joachim Wunderlich von der Universität Stuttgart danken, denn auch von ihm bekam ich viele hilfreiche Tipps mitgegeben. Herr Dr. Thomas Indlekofer brachte mich dazu, überhaupt die Arbeit als Doktorant aufzunehmen und gab mir am Anfang viele Tipps zum Thema, Herrn Dr. Michael Kochte danke ich für weitere Denkanstöße und interessante Diskussionen, Herrn Dr. Eric Schneider für Hilfestellungen bei der Benutzung des GPU-basierten Simulators und „interessante“ Abende bei unseren Projekttreffen. Mein Dank gebührt natürlich auch dem gesamten Team des Fachgebiets Datentechnik, denn ohne dessen Unterstützung hätte ich weder die nötige Software noch die nötigen Freiheiten, um meine Forschungsarbeiten effektiv voranzutreiben.

Es gibt aber auch in meinem privaten Umfeld Personen, denen mein Dank gebührt. Frau Kornelia Bökmann half mir bei den grammatikalischen Korrekturen zu meiner Arbeit und sorgte dafür, dass die schlimmsten Rechtschreibfehler behoben wurden – alle möglicherweise noch vorhandenen Fehler sind nicht ihrem Übersehen, sondern mir selbst verschuldet. Meinen Eltern, die mich auf vielfältige Weise unterstützt haben, möchte ich selbstverständlich auch meinen Dank aussprechen. Und zuletzt möchte ich meiner lieben Helen danken, dass sie mir die Kraft gab, insbesondere zum Ende der Arbeit hin nicht den Mut zu verlieren.

Ostinghausen, im März 2020 *Matthias Kampmann*

Kurzfassung

Die steigende Integrationsdichte moderner Fertigungsprozesse erlaubt es, immer komplexere Funktionen auf immer kleinerer Fläche unterzubringen. Gleichzeitig birgt sie aber auch das Risiko, „schwache“ Schaltungsstrukturen zu erzeugen. Diese verursachen zu Beginn der Lebensdauer unter Betriebsbedingungen kein messbares Fehlverhalten, können sich jedoch schnell zu einem echten Defekt weiterentwickeln, was katastrophale Auswirkungen haben kann. Beobachtbar werden solche Strukturen dennoch über subtile Abweichungen im zeitlichen Verhalten, welche als kleine Verzögerungsfehler modelliert werden können. Zur Erkennung dieser Fehler kann der Hochgeschwindigkeitstest verwendet werden. Hierbei wird die zu testende Schaltung übertaktet, um so selbst sehr kleine Abweichungen von der Spezifikation sichtbar zu machen.

Der Hochgeschwindigkeitstest alleine reicht jedoch nicht aus, um einige der Herausforderungen moderner Fertigungsprozesse beim Test zu meistern. In dieser Arbeit wird daher ein Konzept für einen eingebauten Selbsttest vorgestellt, bei welchem sämtliche für den Hochgeschwindigkeitstest benötigten Infrastrukturen auf der Schaltung selbst untergebracht werden. Dadurch werden periodische Tests ermöglicht, welche die Veränderungen der Schaltung zu einem möglichen Defekt verfolgen und rechtzeitig erkennen können. Dazu werden in dieser Arbeit zweierlei Herausforderungen adressiert und Lösungen vorgestellt. Zum einen wird die Menge an benötigten Testfrequenzen minimiert, zum anderen werden Randbedingungen für die Schaltungssynthese erzeugt, um spezialisierte Prüfpfade zu bilden. Diese, in Kombination mit einem sehr einfachen Maskierungssystem, mindern die Auswirkungen des Übertaktens und erlauben einfachere Verfahren zur X-toleranten Kompaktierung. Ausführliche Simulationen zeigen, dass für vollständige Fehlereffizienz mehr Testfrequenzen benötigt werden als bislang in der Literatur verwendet wurden. Zudem belegen sie die Effektivität der spezialisierten Prüfpfade und zeigen, dass sich der Hochgeschwindigkeitstest als eingebauter Selbsttest kostengünstig realisieren lässt.

Abstract

The continuous downscaling of feature sizes in modern process technologies allows to integrate increasingly complex functionality onto a decreasing chip area. While this enables to build highly advanced applications, the risk of producing marginal hardware is increasing as well. Marginal hardware does not result in faulty behavior at the beginning of the product lifecycle, but can degenerate into an actual defect quickly. This leads to an early life failure, which can result in catastrophic failures. It has been shown that some marginalities influence the affected element by increasing its switching delay by a small amount. Therefore, they can be modeled as a small delay fault. These faults can be detected by using Faster–than–At-Speed Test (FAST), which overclocks the circuit under test in order to make small deviations from specified timing behavior visible.

FAST alone, however, is not sufficient to deal with the challenges that modern process technologies pose to testing. In this work, a concept for a built-in FAST is presented, which integrates all required test infrastructures onto the chip itself. This allows for instance to periodically re-test the device, such that changes in the timing behavior can be observed, that can hint to an early life failure. Two challenges for built-in FAST are addressed and solved in the work at hand. Firstly, the number of required test frequencies for FAST is minimized. Secondly, a method is presented to generate constraints for a synthesis tool, such that the tool generates specialized scan-chains for FAST. These, in combination with a simple masking system, allow to reduce unwanted side effects of FAST and support X-tolerant compaction. Extensive simulations show that on the one hand, more frequencies are required for full fault efficiency than previous works have used. On the other hand, they demonstrate the effectiveness of the specialized scan-chains and show, that built-in FAST can indeed be realized with a lower cost than one would expect.

Inhaltsverzeichnis

1 Einleitung

Die stetig voranschreitende Weiterentwicklung moderner Fertigungs- und Prozesstechnologien in der Halbleiterindustrie hat dazu geführt, dass Mikro- und Nanoelektronik mittlerweile in nahezu jedem Bereich des Lebens eingesetzt wird. Im Automobilbereich etwa belegt dies der „PwC Semiconductor Report“ eindrucksvoll anhand des Anteils an Elektronikkosten am gesamten Fahrzeugpreis (Abbildung 1.1), welcher für das Jahr 2030 auf 50% geschätzt wird [75].

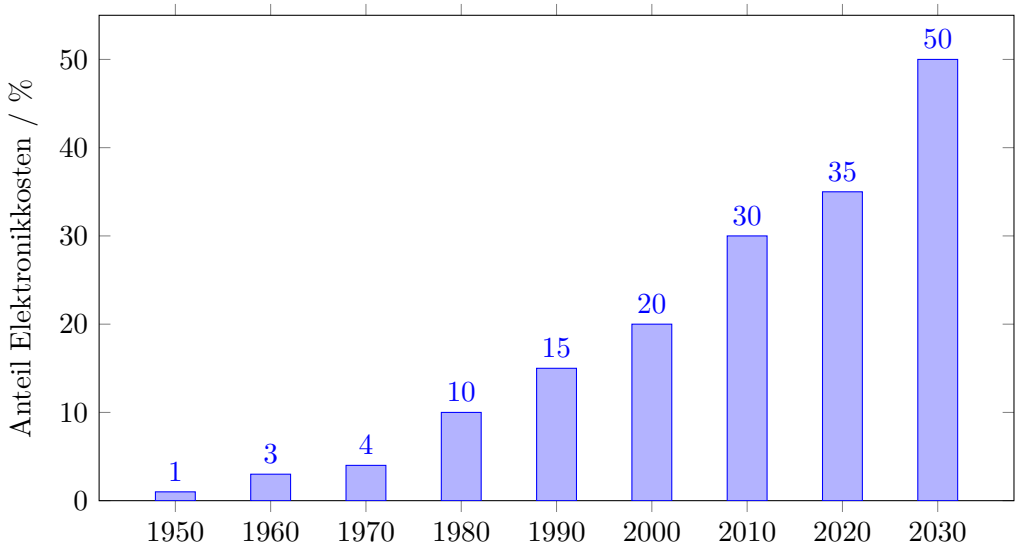


Abbildung 1.1: Anteil der Elektronikkosten am gesamten Fahrzeugpreis [75]

Stetig sinkende Strukturgrößen – derzeitige FinFET Technologien rangieren bei 14 nm [110], AMD hat bereits einen 7 nm Prozessor auf dem Markt [93, 54] – erlauben einen anhaltenden Trend zur steigenden Integrationsdichte. Zwar können die Strukturgrößen nicht endlos verkleinert werden, aber unter dem Stichwort „More than Moore“ werden dennoch Technologien erforscht, um diesen Trend weiterhin aufrecht zu erhalten [99]. Dies ermöglicht es, insbesondere in Anwendungen mit hohen Zuverlässigkeitsanforderungen wie selbstfahrende Fahrzeuge,

Luft- und Raumfahrt, Nukleartechnologie usw. mechanische oder menschliche Einflüsse zu minimieren und durch elektronische Steuergeräte zu ersetzen.

Diese Entwicklung hat neben all ihren Vorteilen allerdings auch ihren Preis: Die Komplexität moderner Schaltungen und Fertigungstechnologien in Kombination mit vielschichtigen Softwaresystemen lässt zwar ein leistungsfähiges, aber auch sehr anfälliges System entstehen. Deshalb steigen mit dem Fortschritt der Technologie auch stark die Ansprüche, denen der Test hochintegrierter Schaltungen genügen muss. Waren früher Tests auf Defekte wie unterbrochene Leiterbahnen oder unbeabsichtigte Brücken ausreichend, müssen moderne Schaltungen auch auf sehr subtile Abweichungen von der Spezifikation überprüft werden. Beispielsweise können Leiterbahnen zu nahe beieinander aufgebracht worden sein, was zu kapazitiven Kopplungen und Leitungsübersprechen führen kann. Auch kann es eine direkte Verbindung zwischen Gate- und Sourceanschlüssen eines Transistors geben, welche allerdings einen hohen Widerstandswert aufweist. Ein weiteres Beispiel ist eine ohmsche Unterbrechung zwischen zwei Anschlüssen, wie in Abbildung 1.2 gezeigt [52].

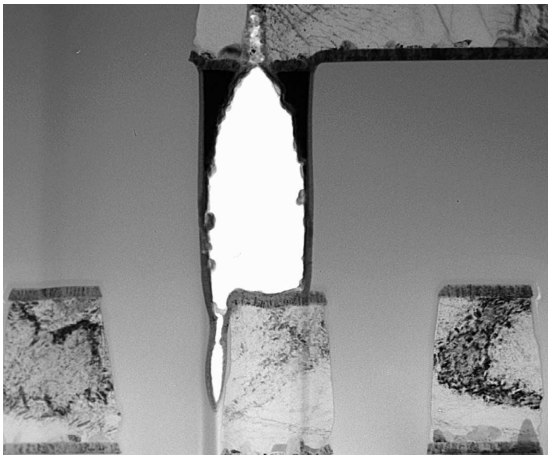


Abbildung 1.2: Ohmsche Unterbrechung in einer Schaltung [52]

Die weiße Fläche in der Abbildung stellt einen Bereich dar, in welchem Wolfram eingebracht werden sollte. Dies ist aber nicht geschehen, es besteht deshalb nur eine sehr geringfügige Verbindung zwischen den Anschlüssen. Solche „schwachen“ Schaltungsstrukturen resultieren oft in einer Veränderung des zeitlichen Verhaltens des betroffenen Bauteils, d. h. die Propagierung einer Signalflanke dauert länger als in der Spezifikation gefordert. Ist dies der Fall, spricht man von einem *kleinen Verzögerungsdefekt (engl. Small Delay Defect, SDD)*. Ein anderes Beispiel für

einen SDD in einer modernen Prozesstechnik sind zerstörte oder verbogene Finnen in einem FinFET [106, 129]. Diese senken die Treiberstärke des Gates und resultieren in einer erhöhten Schaltzeit des Transistors [108, 51, 103]. Auch wenn die zeitliche Abweichung gering ausfallen kann, ist sie möglicherweise ausreichend, um beim Betrieb der Schaltung zu einem Fehlverhalten zu führen. Insbesondere bei kritischen Anwendungen kann das katastrophale Folgen haben. Zusätzlich können schwache Schaltungsstrukturen sich bereits kurz nach der Herstellung durch Effekte wie Elektronenmigration stark verändern: Die zuvor erwähnte ohmsche Unterbrechung z. B. kann vollends durchbrechen und sich zu einer echten Unterbrechung wandeln. Daher besteht bei SDDs das Risiko eines frühzeitigen Systemausfalls (engl. Early Life Failure, ELF), was auch explizit im ELF-Experiment der Universität Stanford [39] nachgewiesen wurde. Seit einiger Zeit wird von Wissenschaft und Industrie gleichermaßen prognostiziert, dass diese Art von Defekten weiter zunehmen wird [57, 65], sodass Testverfahren entwickelt werden müssen, mit denen solche Defekte möglichst früh und kostengünstig erkannt werden, idealerweise bereits beim normalen Produktionstest.

1.1 Grenzen moderner Testverfahren

Um Testverfahren für SDDs zu entwickeln, werden die Defekte typischerweise zu einem gemeinsamen Fehlermodell, dem *kleinen Verzögerungsfehler (engl. Small Delay Fault, SDF)*, abstrahiert. Ein SDF bschreibt eine unerwünschte zusätzliche Verzögerung, welche auf die Schaltzeit eines Bauelements addiert werden muss. Die Erkennung von SDFs ist abhängig von der gewählten Testfrequenz: ein Testmuster kann bei einer niedrigen Frequenz ein fehlerfreies Ergebnis liefern, aber bei einer hohen Frequenz zum Fehlschlag führen [38]. Daher nutzt die Industrie zunehmend den Betriebsfrequenztest, bei dem die Testmuster unter Betriebsbedingungen angelegt werden. Oft kommen dabei Programme zur automatisierten Testmustererzeugung (engl. Automatic Test Pattern Generation, ATPG) zum Einsatz, welche Pfade mit besonders hoher Verzögerung zu sensibilisieren versuchen [49, 78, 48]. Aber auch der Betriebsfrequenztest kann nicht die Erkennung sehr kleiner SDFs garantieren. Wenn alle Pfade, durch welche der SDF propagiert werden kann, nur eine geringe Verzögerungszeit aufweisen, ist es unmöglich, ein

Fehlverhalten unter Betriebsbedingungen zu provozieren. In einem solchen Fall spricht man von einem *versteckten Verzögerungsfehler (engl. Hidden Delay Fault, HDF)*.

Beispiel 1.1. Gegeben sei der Volladdierer aus Abbildung 1.3.

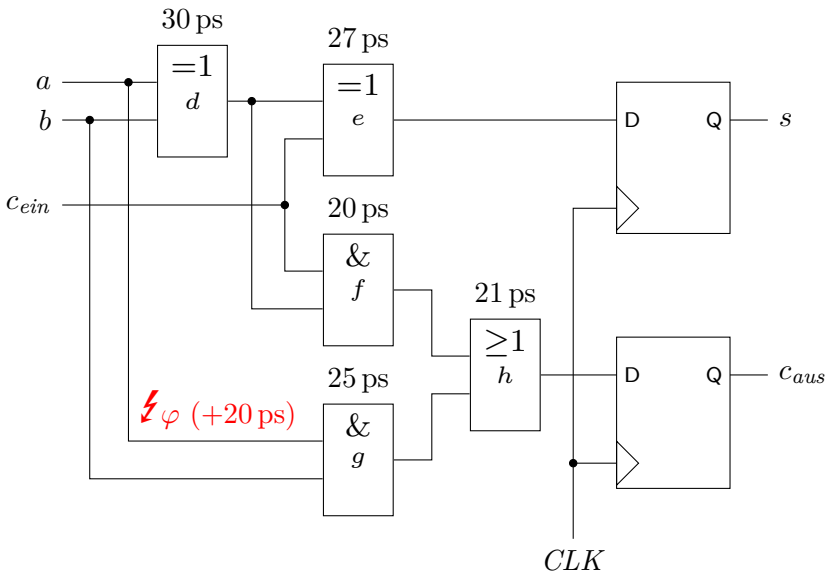


Abbildung 1.3: Ein Volladdierer mit Flipflops und einem Fehler

Die Ausgänge des Volladdierers sind mit Flipflops verbunden, die mit einer Taktperiode von 75 ps getaktet sind. Der eingezeichnete Blitz bedeutet, dass die betroffene Leitung von einem SDF φ befallen ist, welcher eine Signalflanke um zusätzliche 20 ps verzögert. Der einzige Pfad, über welchen φ zu einem der Flipflops propagiert werden kann, ist über die Gatter g und h, die gesamte Verzögerung (inklusive Fehlereffekt) beträgt $t = 20\,\text{ps} + 25\,\text{ps} + 21\,\text{ps} = 66\,\text{ps}$. Diese Verzögerung ist allerdings zu gering, um φ unter Betriebsbedingungen zu erkennen, φ ist damit ein HDF.

Tests auf ELFs erfolgen typischerweise als Hochtemperatur- bzw. Hochspannungstests, welche die Schaltung einem beschleunigten Alterungsprozess aussetzen, um sie aus dem ELF-Bereich in der Badewannenkurve (Abbildung 1.4) zu heben. Diese Tests sind jedoch sehr teuer, da spezielle Testausrüstung benötigt wird und der Test sehr zeitaufwändig ist. Außerdem wirken sie zerstörerisch: Eine mit einem Hochspannungs- oder Hochtemperaturtest überprüfte Schaltung weist eine geringere Lebensdauer auf als eine frisch gefertigte Schaltung. Daher versucht

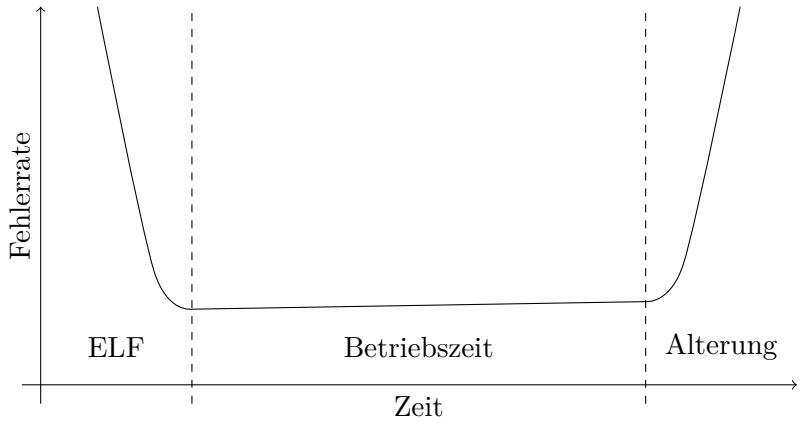


Abbildung 1.4: Die klassische Badewannenkurve [42]

die Industrie seit einiger Zeit, die Anwendung solcher Tests auf ein Minimum zu reduzieren, z. B. durch adaptive Testverfahren, bei denen die Funktionalität einer Schaltung basierend auf den Ergebnissen von benachbarten Schaltungen auf dem Wafer abgeschätzt wird [56].

1.2 Hochgeschwindigkeitstest als kostengünstige Alternative

Eine Alternative zu Hochtemperatur- und Hochspannungstests bietet der *Hochgeschwindigkeitstest (engl. Faster-than-at-Speed Test, FAST)* [141, 3, 7, 43]. Statt mit hoher Temperatur oder Spannung zu arbeiten, verwendet FAST eine hohe Testfrequenz, welche über der Betriebsfrequenz der Schaltung liegt. Durch dieses Übertakten können selbst kleinste SDFs und HDFs sichtbar gemacht werden. Kehrt man zurück zum Volladdierer in Beispiel 1.1, so kann der HDF φ sichtbar gemacht werden, wenn der Test nicht mit einer Taktperiode von 75 ps, sondern mit einer verringerten Taktperiode von 50 ps durchgeführt wird – dies entspricht dem 1,5-fachen der Betriebsfrequenz. Durch FAST werden zum einen die Testqualität und Verlässlichkeit gesteigert und zum anderen die Testkosten im Vergleich zu den bisherigen Verfahren reduziert, da beispielsweise keine Spezialöfen wie beim Hochtemperaturtest benötigt werden. Zudem ist FAST nicht zerstörerisch, erhält also die Qualität der getesteten Schaltung. Die Effektivität von FAST wurde bereits mit gefertigten Schaltungen validiert [13, 119], der bisher in der

Literatur betrachtete Einsatz von FAST als (einmaliger) Herstellungstest kann in seiner Effektivität allerdings deutlich gesteigert werden, wenn eine Schaltung auf mögliche ELFs untersucht werden soll. Das ELF-Experiment aus Stanford zeigt, dass ELFs besonders gut über periodisch wiederholte Tests der Schaltung entdeckt werden können. Dadurch, dass die betroffenen Bauelemente sich schnell in ihrer Verzögerungszeit verändern, kann man über eine deutliche Veränderung in den Ergebnissen der Tests auf einen drohenden ELF schließen. In bisherigen Publikationen über FAST wird für periodische Tests selbst mit integrierten Taktgeneratoren allerdings noch immer ein externes Testequipment (engl. Automatic Test Equipment, ATE) für den Test benötigt. Ein Kunde müsste beispielsweise sein Fahrzeug in der ersten Zeit nach dem Kauf regelmäßig zur Werkstatt zur Überprüfung bringen. Es liegt auf der Hand, dass eine solche Art des Tests unpraktisch und teuer ist.

Die Lösung dieses Problems liegt darin, FAST vollständig als eingebauten Selbsttest (engl. Built-In Self-Test, BIST) in die zu testende Schaltung zu integrieren, d. h. einen *eingebauten Hochgeschwindigkeitstest* zu implementieren. BIST löst auf elegante Art das Problem der periodischen Tests, denn diese können einfach zu Ruhezeiten des Systems oder – wie beispielsweise im Fahrzeug – beim Starten oder Beenden durchgeführt werden [62]. Inzwischen wird sogar vermehrt versucht, BIST während des Betriebs durchzuführen [68]. Ein weiterer Vorteil des Selbsttests ist, dass die benötigten Testinfrastrukturen für FAST ebenfalls effektiv zur Erkennung von Alterungseffekten am Ende der Lebenszeit eingesetzt werden können [33].

Die praktische Realisierung eines solchen Selbsttests ist allerdings mit einigen Herausforderungen verbunden. Zum einen muss die benötigte Testfrequenz auf der zu testenden Schaltung selbst erzeugt werden, da beim Selbsttest kein ATE verfügbar ist, welches diese bereitstellen könnte. Zum anderen ist die Schaltung nicht für die hohen Testfrequenzen ausgelegt. Daher kann die Versorgungsspannung einbrechen, denn die Schaltaktivität ist deutlich erhöht [3, 4]. Für diese Herausforderungen gibt es bereits Lösungen aus Industrie und Forschung, welche in dieser Arbeit kurz vorgestellt werden, und eine Grundlage für die folgenden Erweiterungen darstellen.

1.3 Beitrag dieser Arbeit

Im Rahmen eines Forschungsprojekts der Deutschen Forschungsgemeinschaft (DFG) entwickelt die Universität Paderborn in Kooperation mit der Universität Stuttgart das theoretische Fundament eines eingebauten Hochgeschwindigkeitstests [156, 153, 152]. Die vorliegende Arbeit bietet dazu ein umfassendes Konzept des Selbsttests, Abbildung 1.5 zeigt schematisch den Aufbau des Systems, wie es in dieser Arbeit verwendet wird.

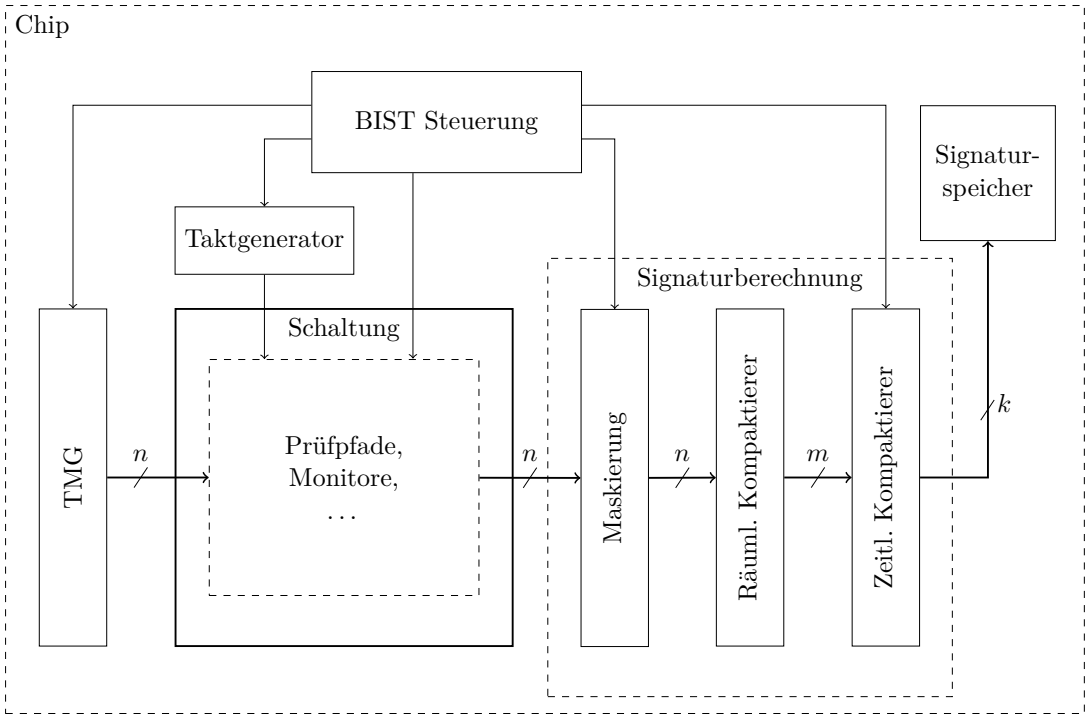


Abbildung 1.5: Komponenten eines eingebauten Hochgeschwindigkeitstests

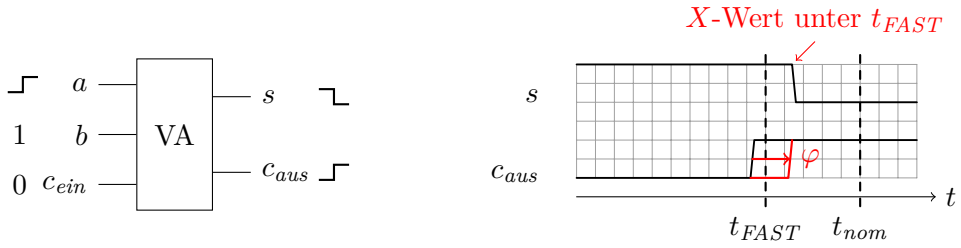
Das System basiert auf dem klassischen „Self-Test Using MISR and Parallel SRSG (STUMPS)“ [9], ist also kompatibel zu vielen in der Industrie gebräuchlichen Selbsttestprodukten. Es werden jedoch einige zusätzliche Komponenten verwendet. Die zu testende Schaltung kann modifiziert werden, um die Effektivität des BIST zu steigern. Beispielsweise können neben der Prüfpfadarchitektur auch Monitore zur Überwachung der Alterungsprozesse in die Schaltung integriert werden [158, 33]. Für die Testdatenauswertung wird eine zweistufige Kompaktierung verwendet, bei welcher zunächst die einzelnen Testantworten in ihrer Größe verringert werden (von n auf $m < n$ Bits), bevor sie zur Berechnung einer Signatur verwendet werden. Das System verfügt über einen Signaturspeicher, um die berechneten

Zwischensignaturen der Tests abzuspeichern, damit diese nach Abschluss des Tests ausgewertet werden können. Dies kann zwar auch in Hardware geschehen [116], ist jedoch sehr aufwändig, weshalb eine Softwarelösung bevorzugt wird.

Zur Erzeugung der benötigten Testfrequenzen wird ein integrierter Taktgenerator benötigt. Oftmals wird bereits beim Herstellungstest unter Betriebsbedingungen ein solcher Taktgenerator eingesetzt, da kostengünstiges ATE möglicherweise nicht die benötigte Frequenz zur Verfügung stellen kann [74]. Das bedeutet, dass lediglich ein optimierter Taktgenerator verwendet werden muss, um die zusätzlichen Frequenzen für FAST zu erzeugen. Der Beitrag dieser Arbeit besteht hier in einem Verfahren, um die Anzahl der benötigten Testfrequenzen zu minimieren, während gleichzeitig maximale Fehlereffizienz für einen Testsatz garantiert wird. Der Unterschied zu bestehenden Ansätzen in der Literatur liegt darin, dass explizit Frequenzen für einen Testsatz berechnet werden, während in bestehenden Ansätzen typischerweise mit einer vorgegebenen Menge an Frequenzen gearbeitet wird. Zusätzlich wird für die bestimmten Frequenzen eine Fehler- und Musterpartitionierung durchgeführt, um die Testzeit (d. h. Anzahl der angelegten Testmuster) und Menge an zu speichernden Testmustern zu minimieren. Besonders bei der Reduktion der Anzahl der angelegten Testmuster konnte in den Experimenten ein Faktor von mehr als 100 gegenüber dem mehrfachen Anlegen aller Testmuster erzielt werden.

Eine zentrale Herausforderung von FAST ist die Tatsache, dass je nach gewählter Frequenz Teile der Testantworten nicht mehr vorhersagbar werden. Es kann passieren, dass einige Ausgänge ihre Berechnungen zum gewählten Beobachtungszeitpunkt (entspricht der Taktperiode der jeweiligen Testfrequenz) noch nicht abgeschlossen haben.

Beispiel 1.2. Kehrt man zurück zum Volladdierer aus Beispiel 1.1, so kann man den HDF φ erkennen, indem an a eine steigende Signalflanke erzeugt wird und b und c_{ein} die Logikwerte 1 bzw. 0 annehmen. Dadurch wandert aber ebenfalls eine Signalflanke von a über die Gatter d und e zum oberen Flipflop, mit einer Verzögerungszeit von $t' = 30\,\text{ps} + 27\,\text{ps} = 57\,\text{ps}$. Dieser Zusammenhang ist vereinfacht in Abbildung 1.6 dargestellt. Abbildung 1.6a zeigt zunächst abstrakt den Volladdierer mit den gegebenen Eingangsbelegungen, Abbildung 1.6b dann die resultierenden Signalverläufe. Bei einer verringerten Taktperiode von $t_{FAST} = 50\,\text{ps}$



(a) Eingaben des Volladdierers (VA) (b) Verläufe der Ausgangssignale

Abbildung 1.6: Entstehung von unbekannten Logikwerten durch FAST

ist demnach das von s getriebene Flipflop noch nicht stabil, ein gespeicherter Logikwert kann nicht deterministisch bestimmt werden.

Solche *unbekannten Logikwerte (X-Werte)* können während der Testantwortkompaktierung die berechneten Signaturen unbrauchbar machen. Wird FAST mit Hilfe eines ATE durchgeführt, lässt sich dieses Problem über spezielle Maskierungsverfahren eindämmen [80]. Im Selbsttest jedoch müssen einfachere Verfahren zum Einsatz kommen, da sehr feingranulare Ausgangsmaskierung zuviel zusätzlichen Steuerungsaufwand benötigt. Zwar können existierende Ansätze für BIST eine gewisse Anzahl an X-Werten tolerieren [116, 100, 61, 83], typischerweise müssen diese aber für eine feste Rate an X-Werten optimiert werden. FAST erzeugt allerdings je nach gewählter Testfrequenz ganz verschiedene X-Raten, es wird also ein adaptives Verfahren benötigt (erste Entwicklungen sind z. B. [107, 46, 104]).

Um die Entwicklung einer solchen adaptiven, X-toleranten Signaturberechnung für einen BIST zu vereinfachen, wird in dieser Arbeit als zweiter Schwerpunkt eine Methode des prüfgerechten Entwurfs für den eingebauten Hochgeschwindigkeitstest vorgestellt. Mit einem speziell für FAST entwickelten, aber generell bei variierenden X-Raten einsetzbaren Verfahren, werden Prüfzellen so gruppiert, dass die resultierende Prüfpfadarchitektur die X-Werte in wenigen Prüfpfaden konzentriert. Dadurch lässt sich ein besonders einfaches Maskierungsschema implementieren: In dieser Arbeit wird ein neuartiges Schema vorgestellt, welches lediglich ein Schieberegister benötigt, um die Masken zu aktualisieren. Dies ist möglich, da die Maske während des Tests bei einer Frequenz konstant gehalten wird. Lediglich beim Wechsel der Frequenz muss auch die Maske angepasst werden. Die Simulationsergebnisse zeigen hier, dass die für FAST optimierten

Prüfpfade mit dem Maskierungsschema deutlich höhere X-Reduktionsraten bei gleichen (oder teils höheren) Fehlereffizienzen gegenüber „klassischen“ Prüfpaden haben, sodass der Aufwand bei der X-toleranten Signaturberechnung verringert wird. Als Alternative wird in dieser Arbeit auch die Grenzwertmaskierung [153, 154] betrachtet. Bei dieser kann die Maske über eine variable Anzahl Schiebezyklen konstant gehalten werden. Damit kann die Effektivität der Maskierung noch weiter gesteigert werden, allerdings auf Kosten des Implementierungs- und Steuerungsaufwandes.

Einige Ergebnisse wurden bereits vorab publiziert [156, 153, 154, 155, 157, 152]. Bei der Frequenzauswahl findet hier zusätzlich eine ausführliche simulationsbasierte Analyse und Diskussion der Verfahren statt, insbesondere wird untersucht, wie sie sich unter einer erhöhten Maximalfrequenz verhalten. Es wird weiterhin, wie bereits erwähnt, eine neue Variante der Ausgangsmaskierung vorgestellt, welche mit einem Schieberegister arbeitet und dadurch besonders wenig Steuerungsaufwand und Testdaten benötigt. Auch bei der Analyse des prüfgerechten Entwurfs für FAST wird untersucht, wie sich eine erhöhte Maximalfrequenz auswirkt. Dadurch wird aufgezeigt, wie sich die Verfahren verhalten, wenn noch höhere X-Raten als bisher auftreten.

Die weiteren Teile der Arbeit gliedern sich wie folgt. In Kapitel 2 werden zunächst erforderliche Grundlagen über Übergangsfehler, Testverfahren, prüfgerechter Entwurf sowie selbsttestende Schaltungen beschrieben. Kapitel 3 beschreibt dann detailliert das Modell des kleinen Verzögerungsfehlers, den Stand der Technik beim Test auf SDFs und Grundlagen über den Hochgeschwindigkeitstest. Der eigenene Anteil der Arbeit zum Thema eingebauter Hochgeschwindigkeitstest ist in zwei Kapitel aufgeteilt. Kapitel 4 erläutert zunächst die Minimierung der Testfrequenzen und Fehler- sowie Musterpartitionierung, Kapitel 5 beschreibt dann den prüfgerechten Entwurf für den eingebauten Hochgeschwindigkeitstest. Abschließend fasst Kapitel 6 diese Arbeit zusammen und gibt einen Ausblick auf weitere Forschungsmöglichkeiten zum Thema.

2 Verzögerungs- und Selbsttest hochintegrierter Schaltungen

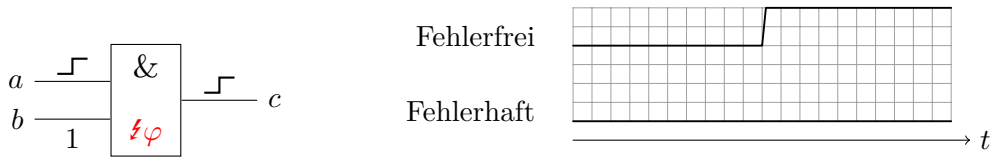
Um die in dieser Arbeit entwickelten und vorgestellten Konzepte zum Hochgeschwindigkeitstest besser verstehen zu können, werden in diesem Kapitel zunächst einige Grundlagen vorgestellt und erläutert. Diese beziehen sich auf das klassische Modell des Übergangsfehlers (Abschnitt 2.1), gefolgt von einem kurzen Überblick über Testverfahren und Metriken für solche Fehler (Abschnitt 2.2). Anschließend stellt Abschnitt 2.3 mit der Prüfpfadarchitektur eine wichtige Technik des prüfgerechten Entwurfs vor, welche ebenfalls ein zentrales Thema dieser Arbeit darstellt. Übergangsfehler benötigen bei einer Prüfpfadarchitektur spezielle Testmethoden, die drei gebräuchlichsten werden in Abschnitt 2.4 präsentiert. Dann wird gezeigt, wie zusätzliche Testinfrastruktur in die Schaltung integriert werden kann, um die Anforderungen an das Testequipment zu reduzieren (Abschnitt 2.5). Integriert man die gesamte Testinfrastruktur auf dem Chip selbst, so kann sich die Schaltung selbst testen. Abschnitt 2.6 stellt den Selbsttest vor, der Fokus liegt hierbei auf der integrierten Testantwortevaluation. Als Basis für die hier vorgestellten Konzepte dient – soweit nicht anders angegeben – das Buch „Essentials of Electronic Testing for Digital, Memory & Mixed-Signal VLSI Circuits“ von Bushnell und Agrawal [11].

2.1 Übergangsfehler

Es gibt eine ganze Reihe von physikalischen Defektmechanismen, welche das zeitliche Verhalten des betroffenen Bauelements beeinflussen, beispielsweise die

in der Einleitung beschriebene ohmsche Unterbrechung. Die Ursachen und Arten solcher Defekte können sehr verschieden sein, die Auswirkungen lassen sich jedoch oft gut zusammenfassen, sodass typischerweise diese Defekte abstrahiert und in einem Fehlermodell zusammengefasst werden. Ein Fehlermodell betrachtet lediglich die Auswirkungen auf die Logik und das zeitliche Verhalten der Schaltung, was die Entwicklung von Testmethoden ungemein vereinfacht. Der klassische *Übergangsfehler* modelliert Defekte, welche die Signalpropagierung durch ein Bauteil (z. B. Gatter) so stark verzögern, dass sie aus praktischen Gesichtspunkten „verhindert“ wird.

Beispiel 2.1. Gegeben sei das UND-Gatter aus Abbildung 2.1a, welches von einem Übergangsfehler φ befallen ist.



(a) Ein UND-Gatter mit Übergangsfehler

(b) Verlauf von c im fehlerfreien und fehlerhaften Fall

Abbildung 2.1: Auswirkungen eines Übergangsfehlers

Wenn nun beispielsweise eine steigende Signalflanke über den Eingang a propagiert wird (Eingang b ist konstant logisch 1), dann sorgt φ dafür, dass diese Flanke am Ausgang nicht mehr beobachtet wird, wie in Abbildung 2.1b gezeigt.

Der Zusammenhang mit den in dieser Arbeit betrachteten kleinen Verzögerungsfehlern (SDFs) kann hergestellt werden, indem ein Übergangsfehler als SDF mit „unendlicher“ Verzögerung modelliert wird. Aus diesem Grunde sind die grundsätzlichen Testverfahren und insbesondere auch praktischen Herausforderungen zum Großteil für Übergangsfehler und SDFs gleich. Deshalb werden in diesem Kapitel zunächst lediglich Übergangsfehler betrachtet um die Teststrategien zu beschreiben, Kapitel 3 beschreibt dann notwendige Anpassungen und Erweiterungen, welche speziell für kleine Verzögerungsfehler erforderlich sind.

In der Literatur gibt es zwei Varianten des Übergangsfehlers, welche sich im Ort des Auftretens unterscheiden. Bei einem *Gatterverzögerungsfehler* [125] ist die unerwünschte Verzögerung in einem einzelnen Gatter konzentriert, wie etwa im

UND-Gatter in Beispiel 2.1. Der Rest der Schaltung wird als fehlerfrei betrachtet, dieses Fehlermodell ist also gut geeignet, um Punktdefekte wie einzelne defekte Transistoren darzustellen. Gatterverzögerungsfehler werden entsprechend der betroffenen Signalflanke in Anstiegs- und Abfallsfehler unterteilt. Die Attraktivität dieses Fehlermodells liegt in seiner guten Skalierbarkeit – die Anzahl möglicher Fehlerorte in einer Schaltung ist linear in der Anzahl der Gatter. Zusätzlich ist die Testmustererzeugung vereinfacht: Es muss lediglich am betreffenenden Fehlerort eine Signalflanke erzeugt und zu einem der Schaltungsausgänge propagiert werden. Wird diese dann beim Test nicht beobachtet, so gilt der Fehler als erkannt.

Ein anderes Modell des Übergangsfehlers ist der *Pfadverzögerungsfehler* [105]. Hier wird angenommen, dass die zusätzliche Verzögerung anders als beim Gatterverzögerungsfehler über mehrere Gatter hinweg verteilt ist. Ein Fehlverhalten tritt erst dann auf, wenn die Summe dieser einzelnen Verzögerungen eine gewisse Grenze – etwa die nominelle Taktperiode – überschreitet. Dadurch lassen sich Defekte modellieren, welche beispielsweise durch unerwünschte Beugungseffekte an Fertigungsmasken auftreten und sich flächig auf der Schaltung auswirken. Um einen Pfadverzögerungsfehler erkennen zu können, muss eine Signalflanke durch eine Reihe von Gattern, als *Pfad* bezeichnet, propagiert werden. Man unterscheidet anhand der Signalflanke, welche an einem Eingang der Schaltung erzeugt wird, zwischen steigenden und fallenden Pfadverzögerungsfehlern. Ob ein Pfadverzögerungsfehler durch einen Test tatsächlich erkannt wird, hängt von verschiedenen Faktoren ab, insbesondere von den Verzögerungszeiten der restlichen Bauelemente in der Schaltung, welche aufgrund von Prozessvariationen von der Spezifikation abweichen können. Ein Test auf Pfadverzögerungsfehler wird als *robust* bezeichnet, wenn der Fehler unabhängig von anderen Verzögerungen in der Schaltung erkannt werden kann [47]. Robuste Tests stellen besondere Anforderungen an die Testmustererzeugung, deshalb kann unter Umständen nicht jeder Pfadverzögerungsfehler robust getestet werden.

Beispiel 2.2. Ein UND-Gatter mit zwei Eingängen a und b sei Teil eines zu testenden Pfades, bei dem eine Signalflanke über den Eingang a propagiert werden soll. Ist die Signalflanke steigend, so kann b konstant logisch 1 bleiben oder selbst eine steigende Signalflanke propagieren. In beiden Fällen wäre der Test robust: Wenn b eine Signalflanke propagiert, welche ebenfalls einen Fehler beinhaltet, so wird dieser nicht die Erkennung eines Fehlers behindern (es kann aber schwierig

sein, den Fehlerort zu bestimmen). Ist die Signalflanke an a jedoch fallend, so muss b konstant auf logisch 1 verbleiben, damit die Signalflanke von a robust propagiert werden kann. Andernfalls, wenn b eine fallende Signalflanke propagiert, kann diese die Erkennung des Fehlers behindern, da das UND-Gatter auf logisch 0 schaltet, sobald einer der Eingänge auf logisch 0 schaltet.

Da die Anzahl an Pfaden (und damit Pfadverzögerungsfehlern) exponentiell in der Anzahl der Gatter einer Schaltung ist, ist ein vollständiger Test aller Fehler nicht in realistischer Zeit durchführbar. In der Praxis muss man sich deshalb auf einige wenige, kritische Pfade mit besonders hohen Verzögerungszeiten beschränken. Aufgrund dieser Einschränkungen hat sich dieses Fehlermodell nur eingeschränkt in der Industrie durchgesetzt.

2.2 Testmethoden für Übergangsfehler

Der klassische Herstellungstest benötigt einen externen Tester (ATE), um die Schaltung zu testen. Der Aufbau ist schematisch in Abbildung 2.2 dargestellt.

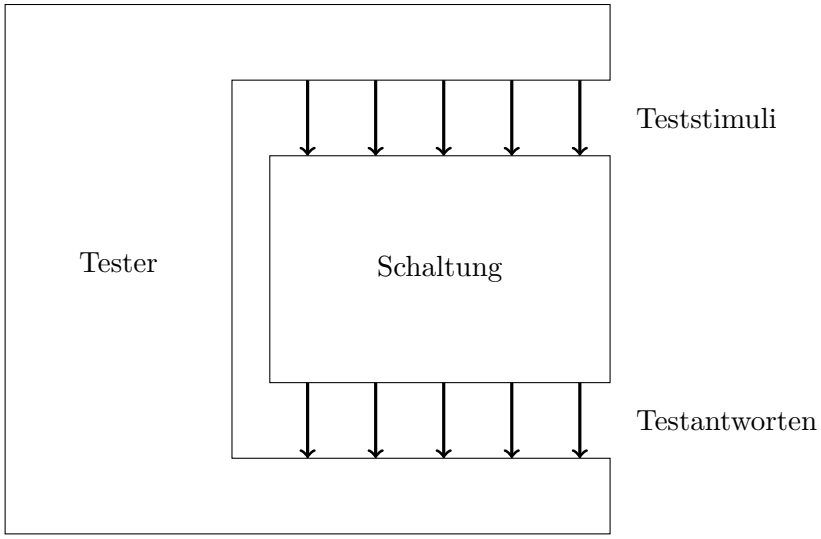


Abbildung 2.2: Herstellungstest mit externem Tester

Um einen Test durchzuführen, werden die Eingänge einer Schaltung vom Tester mit Teststimuli belegt, anschließend werden die Logikwerte der Schaltungsausgänge als Testantworten aufgezeichnet. Diese Testantworten werden dann mit

zuvor berechneten Sollantworten verglichen. Übergangsfehler benötigen dabei mindestens zwei aufeinander folgende Teststimuli, um Signalflanken an den Schaltungseingängen zu erzeugen und durch die Schaltung zu propagieren.

Beispiel 2.3. Abbildung 2.3 zeigt beispielhaft den Verzögerungstest mit einer kombinatorischen Schaltung.

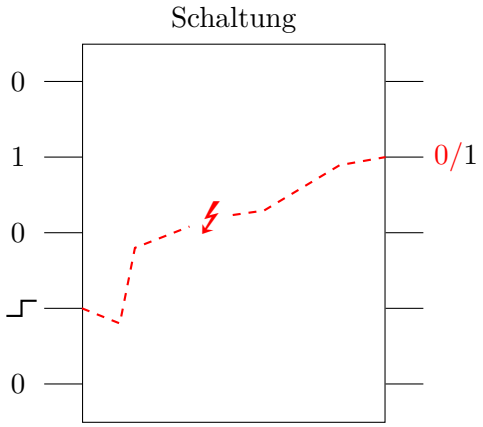


Abbildung 2.3: Beispiel eines Verzögerungstests

An jedem Eingang werden insgesamt zwei Logikwerte angelegt, der vierte Eingang vollzieht dabei einen Pegelwechsel, welcher entlang der gestrichelten Linie durch die Schaltung (inklusive Fehlerort) zum zweiten Schaltungsausgang propagiert wird. Dort wird dann aufgrund des Fehlers ein Logikwert von 0 anstatt der erwarteten 1 beobachtet.

Basierend auf dem vorherigen Beispiel soll im Folgenden der Verzögerungstest formal definiert werden.

Definition 2.1. *Eine Belegung von n Schaltungseingängen mit jeweils einem einzelnen Logikwert bezeichnet man als* Testvektor $\mathbf{v} = \begin{pmatrix} v_1 & \cdots & v_n \end{pmatrix}$, $v_i \in \{0, 1\}$. *Ein* Testmuster $P = (\mathbf{v_1}, \ldots, \mathbf{v_m})$ *ist ein Tupel aus m Testvektoren, welche für einen einzelnen Test benötigt werden. Die Menge aller Testmuster bezeichnet man als* Testsatz $\mathscr{P}$.

Wie bereits in Beispiel 2.3 angedeutet, benötigt ein Verzögerungstest für eine kombinatorische Schaltung $m = 2$ Testvektoren. Der erste Vektor dient dazu, die Schaltung zu einem definierten Zustand zu initialisieren, der zweite Vektor erzeugt

dann Signalflanken, welche durch die Schaltung zu den Ausgängen wandern. Bei sequentiellen Schaltungen können mehr Testvektoren benötigt werden, um den Fehlereffekt über die Speicherelemente hinweg zu einem der Schaltungsausgänge zu propagieren. Hier ist es wichtig, dass die Testvektoren unmittelbar aufeinander folgend angelegt werden, d. h. im ersten Takt wird $\mathbf{v_1}$ angelegt, im zweiten Takt $\mathbf{v_2}$, usw. Die Güte eines Tests kann man anhand verschiedener Metriken quantitativ feststellen. Am gebräuchlichsten sind Fehlerabdeckung und Fehlereffizienz.

Definition 2.2. *Seien Φ die Menge aller möglichen Fehler in einer Schaltung und $\mathscr{P}$ ein Testsatz. Weiterhin bezeichnen $\Phi_{unerkennbar} \subseteq \Phi$ die Menge an unerkennbaren Fehlern sowie $\Phi_{erkannt}(\mathscr{P}) \subseteq \Phi$ die Menge an Fehlern, welche mit $\mathscr{P}$ erkennbar sind. Die* Fehlerabdeckung *ist dann definiert als*

$$FA(\mathscr{P}) = \frac{|\Phi_{erkannt}(\mathscr{P})|}{|\Phi|}. \tag{2.1}$$

Die Fehlereffizienz *ist definiert als*

$$FE(\mathscr{P}) = \frac{|\Phi_{erkannt}(\mathscr{P})|}{|\Phi \setminus \Phi_{unerkennbar}|}. \tag{2.2}$$

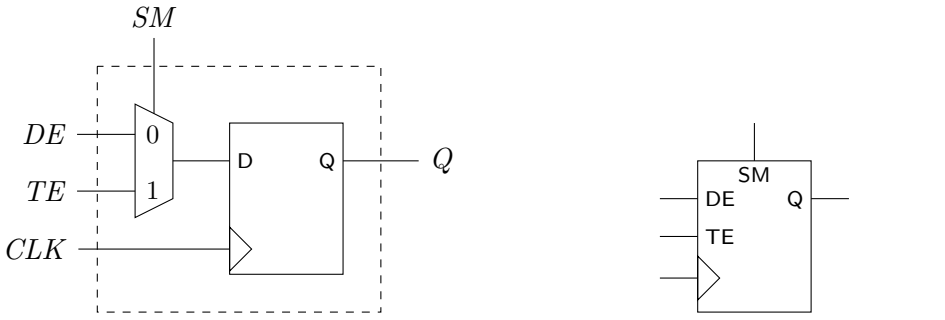
Der Unterschied zwischen Fehlerabdeckung und Fehlereffizienz liegt darin, dass die Fehlerabdeckung auch solche Fehler berücksichtigt, für welche beweisbar keine Testmuster erzeugt werden können – also Fehler, welche sich beweisbar nicht auf die Schaltungsfunktion auswirken können. In dieser Arbeit werden Metriken genutzt, welche sich unmittelbar von der Fehlerabdeckung und Fehlereffizienz ableiten. Zur Bewertung von Testsätzen für SDFs gibt es allerdings auch andere, komplexere Metriken, wie beispielsweise das „Statistical Delay Quality Level (SDQL)“, auf welches in Kapitel 3 genauer eingegangen wird.

2.3 Prüfgerechter Entwurf: Prüfpfadarchitektur

Unter dem Begriff *prüfgerechter Entwurf (engl. Design for Test, DFT)*, auch als testfreundlicher Entwurf bezeichnet, werden eine Reihe von Methoden zusammengefasst, um die Testbarkeit einer Schaltung bereits beim Entwurf zu

berücksichtigen und damit zu erhöhen. Unter Testbarkeit versteht man die Steuer- und Beobachtbarkeit einer Schaltung. Steuerbarkeit ist ein Maß dafür, wie leicht es ist, ein Element der Schaltung mit einem gewissen Logikwert zu belegen, analog beschreibt die Beobachtbarkeit wie leicht es ist, den Zustand eines Elements an den Schaltungsausgängen zu beobachten.

Eine gängige DFT-Technik ist die Prüfpfadarchitektur, welche die internen Speicherelemente (hier: Flipflops) einer Schaltung von außen zugänglich macht. Dazu werden die Flipflops zu sogenannten *Prüfzellen* erweitert. Es gibt verschiedene Varianten von Prüfzellen, die gebräuchlichste besteht aus einem flankengesteuerten Flipflop, welches um einen Multiplexer vor dem Dateneingang D erweitert wird. Diese Prüfzelle ist in Abbildung 2.4 dargestellt.[1]



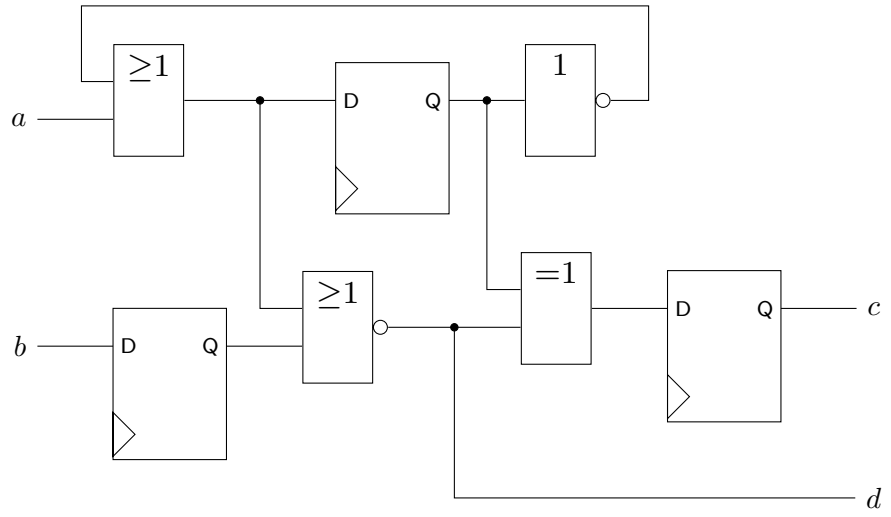
(a) Interner Aufbau (b) Schematische Darstellung

Abbildung 2.4: Eine flankengesteuerte Prüfzelle

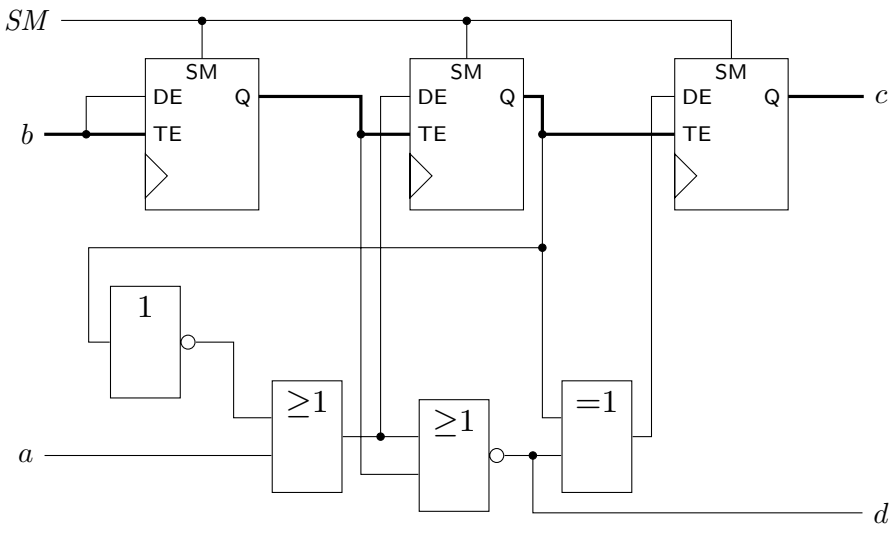
Eine Prüfzelle besitzt zwei verschiedene Eingänge: Den Dateneingang *DE* und den Testeingang *TE*, welche über das Signal *SM* (Schiebemodus) gesteuert werden können. *DE* wird von der kombinatorischen Logik der Schaltung getrieben, über *TE* können mehrere Prüfzellen zu einem Schieberegister, genannt *Prüfpfad*, in Reihe geschaltet werden. Der *SM*-Eingang aller Prüfzellen wird dann von einem globalen *SM*-Signal getrieben, welches die Schaltung in den Betriebs- ($SM = 0$) oder Schiebemodus ($SM = 1$) versetzen kann. Die Prüfpfade und zusätzlichen Steuersignale bezeichnet man auch als *Prüfpfadarchitektur*.

Beispiel 2.4. Abbildung 2.5 demonstriert die Umwandlung einer Schaltung durch das Einfügen eines Prüfpfades in eine Prüfpfadarchitektur.

[1] Aus Gründen der Übersichtlichkeit werden in den folgenden Abbildungen die Set- und Reset-Signale nicht eingezeichnet.



(a) Eine Schaltung mit Flipflops



(b) Die gleiche Schaltung mit Prüfpfadarchitektur. Prüfzellen und Kombinatorik sind hier getrennt gezeichnet, der Prüfpfad ist fett markiert.

Abbildung 2.5: Von einer Schaltung zur Prüfpfadarchitektur. Alle Flipflops und Prüfzellen werden durch dasselbe Taktsignal gesteuert, welches zur Vereinfachung nicht eingezeichnet wurde.

Die Schaltung ohne jedwede DFT-Erweiterungen ist in Abbildung 2.5a gezeigt. Die Flipflops der Schaltung wurden in Abbildung 2.5b zu Prüfzellen erweitert und durch die *TE*-Eingänge zu einem einzelnen Prüfpfad verbunden. In dieser Abbildung wurden die Prüfzellen getrennt vom kombinatorischen Teil gezeichnet, um den Prüfpfad besser darstellen zu können. Zusätzlich wurde die Schaltung um den globalen Eingang *SM* erweitert. Stellt man $SM = 1$ ein, ist der Prüfpfad wie ein Schieberegister konfiguriert, und es können über den Eingang b beliebige Werte hineingeschoben werden. Gleichzeitig kann über den Ausgang c der Inhalt des Registers ausgelesen werden kann.

Man sieht anhand des Beispiels 2.4, dass die Prüfzellen einerseits über ihre *Q*-Ausgänge die Kombinatorik treiben, andererseits aber auch von der Kombinatorik über ihre *DE*-Eingänge getrieben werden. Deshalb bezeichnet man die Ausgänge der Prüfzellen auch als pseudo-primäre Eingänge (PPE) und analog die Eingänge der Prüfzellen als pseudo-primäre Ausgänge (PPA) der Schaltung. Durch diese Betrachtungsweise vereinfacht sich beispielsweise das Problem der Testmustererzeugung (ATPG) ungemein, da nur noch für rein kombinatorische Schaltungen Testmuster berechnet werden müssen.Wenn die letzte Prüfzelle eines Prüfpfads keinen primären Ausgang der Schaltung treibt, so fügt man typischerweise der Schaltung einen zusätzlichen Ausgang *TA* (Testausgang) hinzu, welcher mit dem Ausgang der letzten Prüfzelle im Prüfpfad verbunden ist. Analog muss ein zusätzlicher Eingang *TE* hinzugefügt werden, falls die erste Prüfzelle im Pfad nicht direkt von einem Schaltungseingang getrieben wird. Deshalb kann sich die Anzahl der primären Ein- und Ausgänge (also physikalische Pins der Schaltung) gegenüber der normalen Schaltung erhöhen.

Der Test mit einem Prüfpfad läuft nun wie folgt ab. Zunächst wird die Schaltung über $SM = 1$ in den Testmodus versetzt, um den Prüfpfad als Schieberegister zu konfigurieren. Daraufhin wird das Testmuster seriell in den Prüfpfad geladen. Sobald das Muster geladen wurde, wird der Betriebsmodus der Schaltung eingestellt ($SM = 0$), die Testantwort wird dadurch in den Prüfzellen aufgezeichnet. Abschließend versetzt man die Schaltung abermals in den Testmodus, um die Testantwort aus dem Prüfpfad herauszuschieben und auszuwerten. Dabei kann bereits das nächste Testmuster in den Prüfpfad geladen werden. Es zeigt sich, dass die Länge des Prüfpfads (Anzahl der Prüfzellen) die Testzeit entscheidend dominiert. Hat die Schaltung sehr viele Prüfzellen, ist es daher sinnvoll, statt

eines einzelnen, langen Prüfpfades mehrere kleinere Prüfpade zu verwenden. Im einfachsten Fall bekommt dann jeder dieser Prüfpfade einen eigenen *TE*-Eingang und *TA*-Ausgang, das *SM*-Signal wird für alle Prüfpfade geteilt. In der Praxis versucht man, die Anzahl dieser zusätzlichen Pins zu reduzieren, z. B. indem existierende Pins wiederverwendet werden (sofern möglich). Auch können Methoden des eingebetteten deterministischen Tests die Anzahl der benötigten Pins reduzieren, indem die eingeschobenen Werte auf mehrere Prüfpfade verteilt werden (vgl. Abschnitt 2.5).

2.4 Prüfpfadbasierte Testmethoden für Verzögerungsfehler

Eine Prüfpfadarchitektur, wie in Abschnitt 2.3 gezeigt, hat den großen Vorteil, dass die Komplexität einer sequentiellen Schaltung für den Test auf die Kombinatorik reduziert wird. Da die primären und pseudo-primären Eingänge zusammengefasst werden können, werden lediglich zwei Testvektoren benötigt (vgl. Abschnitt 2.2) – die Größe eines Testvektors ist dann die Summe der primären und pseudo-primären Eingänge. Die Herausforderung liegt nun darin, dass die beiden Testvektoren $\mathbf{v_1}$ und $\mathbf{v_2}$ durch unmittelbar aufeinander folgende Taktpulse an die Schaltung angelegt werden müssen, eine Prüfzelle jedoch nur ein einzelnes Bit speichern kann. Es ist aufgrund der Architektur also nicht möglich, beide Testvektoren nacheinander in die Prüfpfade zu laden und anschließend direkt anzulegen. Als Lösung dieses Problems wurden in der Vergangenheit drei Verfahren entwickelt, die im Folgenden genauer betrachtet werden sollen. Ohne Beschränkung der Allgemeinheit werden hier die primären Ein- und Ausgänge der Schaltung ignoriert und lediglich die Prüfzellen betrachtet. Die Abbildungen und Beschreibungen stammen im Wesentlichen aus [11].

Launch-on Shift

Bei Launch-on Shift (LOS) [95] wird $\mathbf{v_2}$ direkt aus $\mathbf{v_1}$ gewonnen, indem ein zusätzlicher Schiebetakt ausgeführt wird. Es müssen also pro Testmuster lediglich

die Bits des ersten Testvektors plus einem weiteren Schiebetakt gespeichert werden. Außerdem sind keine Modifikationen der Prüfpfadarchitektur erforderlich. Abbildung 2.6 zeigt die Signalverläufe von LOS.

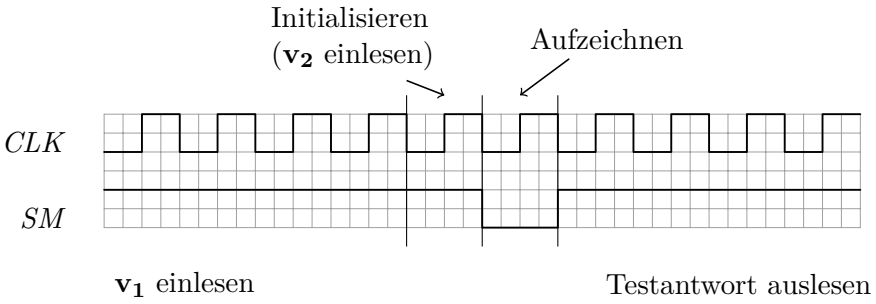


Abbildung 2.6: Signalverläufe bei LOS

Zunächst wird $\mathbf{v_1}$ in den Prüfpfad geschoben ($SM = 1$). Beim Initialisieren wird weiterhin $SM = 1$ gelassen, was dem Einlesen von $\mathbf{v_2}$ entspricht. Dann wird für einen einzelnen Takt $SM = 0$ gesetzt, daraufhin zeichnen die Prüfzellen die Schaltungsantwort auf, welche dann wieder aus dem Prüfpfad geschoben werden kann. LOS ist in der Industrie gebräuchlich, hat allerdings die Herausforderung, dass insbesondere beim Betriebsfrequenztest (vgl. Kapitel 3) das SM-Signal wie ein Taktsignal behandelt werden muss. Neuere Ansätze wie [136, 137] verwenden spezielle zusätzliche Logik innerhalb der Prüfzellen, um diesen Nachteil auszugleichen.

Launch-on Capture

Eine Alternative zu LOS bietet Launch-on Capture (LOC) [94]. Wenn $f(\mathbf{v})$ die Schaltungsantwort auf $\mathbf{v}$ bezeichnet, dann gilt bei LOC $\mathbf{v_2} = f(\mathbf{v_1})$. Dazu wird nach dem Einlesen von $\mathbf{v}_1$ bereits $SM = 0$ gesetzt. Abbildung 2.7 zeigt die Signalverläufe für LOC.

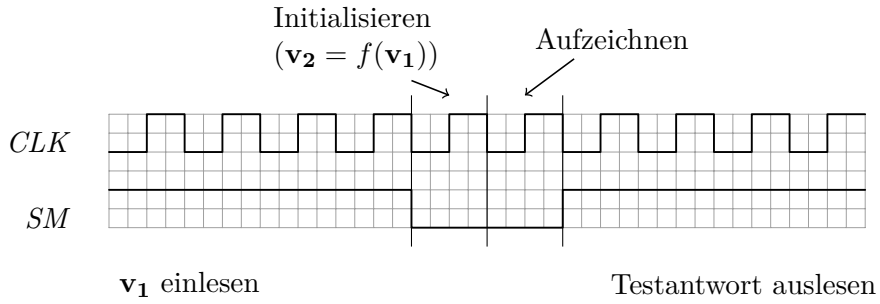


Abbildung 2.7: Signalverläufe bei LOC

LOC hat gegenüber LOS den Vorteil, dass es einfacher zu implementieren ist, da das *SM*-Signal keine harten zeitlichen Anforderungen hat. Es können zusätzliche „Tottakte“ vor dem Initialisieren eingefügt werden, um der Schaltung genügend Zeit zu geben, alle Elemente (inklusive *SM*) auf die gewünschten Werte zu stabilisieren. Aus diesem Grund wird LOC häufiger in der Industrie verwendet als LOS. Da bei LOC die Transitionen an den Eingängen nicht so leicht zu kontrollieren sind wie bei LOS, ist die Fehlerabdeckung bei LOC allerdings manchmal reduziert [113].

Erweiterte Prüfpfadarchitektur

Eine intuitive Lösung des Problems liegt darin, die Prüfzellen zu erweitern, um zwei Bit speichern zu können. In der erweiterten Prüfpfadarchitektur (engl. Enhanced Scan, ES) werden zusätzliche Haltelatches in die Architektur integriert. Außerdem wird ein weiteres Signal *HALTEN* benötigt, welches den gespeicherten Wert der Haltelatches aktualisiert. Abbildung 2.8 zeigt die Architektur mit zugehörigen Signalverläufen in Abbildung 2.9.

Durch die erweiterten Prüfzellen können die bei LOS und LOC gegebenen Abhängigkeiten zwischen $\mathbf{v_1}$ und $\mathbf{v_2}$ eliminiert werden. ES hat deshalb gegenüber LOS und LOC den Vorteil einer höheren Fehlerabdeckung bei geringeren Testsatzgrößen und vereinfachter Testmusterberechnung. Allerdings steigen die Hardwarekosten, da zusätzliche Haltelatches und ein *HALTEN*-Signal einfügt werden müssen. Deshalb wird ES nur selten in der Industrie eingesetzt.

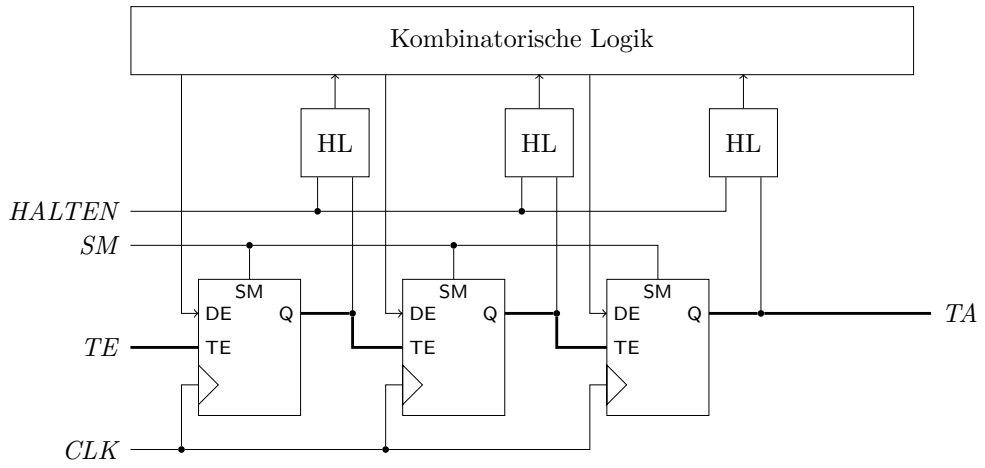


Abbildung 2.8: Erweiterte Prüfpfadarchitektur mit Haltelatches (HL)

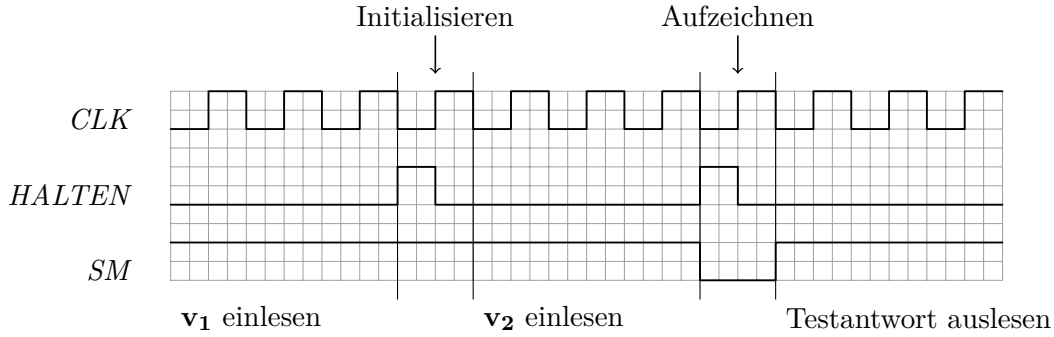


Abbildung 2.9: Signalverläufe bei ES

2.5 Eingebetteter deterministischer Test

Die steigende Integrationsdichte und Komplexität moderner Schaltungen lässt einen klassischen Herstellungstest, bei dem das ATE sämtliche Testinfrastruktur (abgesehen von DFT-Erweiterungen) bereitstellt, zunehmend teurer und inpraktikabel werden. Um Testkosten zu senken, welche z. B. durch große Mengen an Testmusterbits, Testantwortbits und eine hohe Anzahl an benötigten Testerkanälen entstehen, wird inzwischen vermehrt Testinfrastruktur auf der Schaltung selbst integriert. Ein typisches Beispiel zur Reduktion dieser Datenmengen und Testerkanäle, die vom ATE verwaltet werden müssen, sind Testmusterdekompressoren und Testantwortkompaktoren. Solche Erweiterungen werden etwa im *eingebetteten deterministischen Test (engl. Embedded Deterministic Test, EDT)* zusammengefasst [81] (Abbildung 2.10).

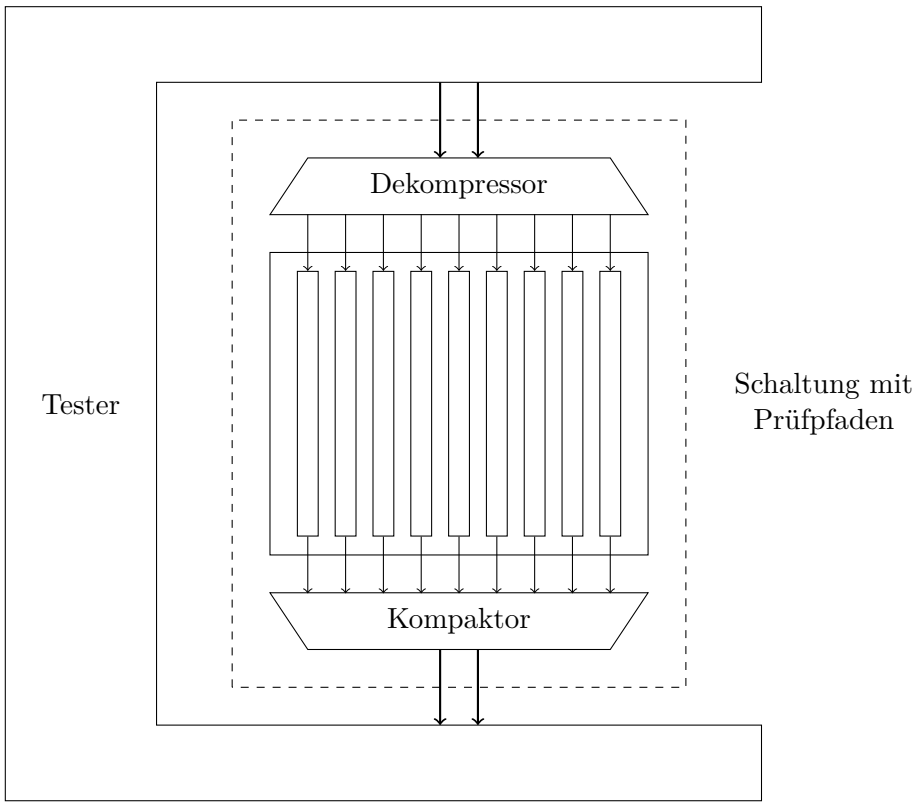


Abbildung 2.10: Eingebetteter deterministischer Test

Der Dekompressor erlaubt es, auf dem ATE komprimierte Testsätze zu speichern, sodass weniger Daten an die zu testende Schaltung übertragen werden müssen. Zusätzlich kann ein Dekompressor die komprimierten Testmuster auf sehr viele Prüfpfade verteilen. Er erlaubt es also, sehr viele interne Prüfpfade zu nutzen, ohne dass zusätzliche physikalische Eingänge erforderlich sind. Dadurch kann die Testdauer erheblich gesenkt werden, da weniger Schiebezyklen pro Muster erforderlich sind. Verlustfreie Testmusterkompression ist möglich, da die vom ATPG berechneten Testsätze nur wenige Eingänge mit festen Werten belegen. Oftmals ist der Großteil der Eingänge unspezifiziert, da diese nicht zur Fehlererkennung beitragen. Deshalb kann ein Testmuster auf dem ATE in verkleinerter Form gespeichert werden, die komprimierten Daten werden über den Dekompressor auf dem Chip selbst dekomprimiert und in die echten Testmuster umgewandelt. Eine typische Variante eines Testmusterdekompressors ist ein linearer Dekompressor, oft als linear rückgekoppeltes Schieberegister (engl. Linear Feedback Shift Register, LFSR) implementiert. Das ATE überträgt einen Startzustand an das LFSR, sodass dieses beim Durchlaufen seiner Folgezustände genau die geforderten

Eingangsbelegungen produziert. Testmusterdekompressoren sind nicht im Fokus dieser Arbeit, der geneigte Leser sei hier an die Literatur verwiesen. Eine gute Übersicht über weitere Testmusterdekompressoren liefert beispielsweise [115].

Der Kompaktor ist das Gegenstück des Dekompressors und dient dazu, die Testantworten der Schaltung zu verkleinern, bevor diese wieder zum ATE übertragen werden. Anders als bei der Dekompression muss hier nicht verlustfrei komprimiert werden, etwas Informationsverlust ist erlaubt, solange dieser nicht die Fehlereffizienz zu stark beeinflusst. Im EDT kommen sogenannte räumliche Kompaktoren zum Einsatz. Diese wandeln eine Testantwort $\mathbf{a}$ in eine verkleinerte Testantwort $\mathbf{a}'$ um. Es müssen also noch immer alle Testantworten verglichen werden, aber pro Testantwort fallen weniger Bits an. Typische Verfahren nutzen XOR-Bäume und beruhen auf Paritätsberechnung [122, 12] oder fehlerkorrigierenden Codes [130, 60, 134]. Erwähnenswert ist hier die „extreme Kompaktierung“, welche alle Ausgänge zu einem einzelnen Paritätsbit zusammenfasst.

Ein Beispiel für eine industrielle Kompaktierung ist das X-Press Verfahren [82], welches aus zwei Stufen an räumlichen Kompaktierern besteht. In der ersten Stufe werden die Daten von sämtlichen Schaltungsausgängen mittels extremer Kompaktierung auf ein Bit reduziert. Dieses Bit wird dann in ein spezielles Register geschoben, welches eine zuvor vom Anwender festgelegte Länge hat. Ist das Register voll, wird dessen Inhalt in der zweiten Stufe erneut kompaktiert und erst dann werden die Daten an den Tester weitergegeben. In Abhängigkeit von der Länge des Registers können so die Testantwortdaten um bis zu einem Faktor von 1000 reduziert werden.

2.6 Selbsttest

Entfernt man die beim EDT gegebenen Abhängigkeiten zum ATE durch Integration weiterer Testkomponenten auf dem Chip, kann sich die Schaltung im Extremfall vollständig selbst testen. Ein solcher Selbsttest (BIST) ist aus verschiedenen Gründen attraktiv. Zum einen übernimmt BIST die Vorteile des EDT, z. B. die Möglichkeit, viele kurze Prüfpfade zu nutzen um die Testdauer zu minimieren.

Zum anderen erlaubt BIST ohne großen Aufwand ein periodisches Testen der Schaltung, beispielsweise während der Ruhephasen des Systems. Obwohl die Fertigungskosten unter BIST geringfügig erhöht sind, reduziert die Fähigkeit zum Selbsttest spätere Kosten im Betrieb, da eine Werkstatt beispielsweise weniger komplexe Testausrüstung benötigt, um eine Schaltung zu untersuchen [1]. Hybride Varianten des BIST erlauben nicht nur den vollständig autonomen Selbsttest, sondern unterstützen auch den Test mit ATE oder vereinfachen die Diagnose einer Schaltung. Aus diesen Gründen wird BIST mittlerweile vielfach in Anwendungen mit hohen Verlässlichkeitsanforderungen eingesetzt, wie beispielsweise in der Automobilindustrie [62, 50, 53, 68, 88]

In diesem Abschnitt werden kurz die Kernkomponenten und Herausforderungen eines BIST vorgestellt, basierend auf der STUMPS Architektur (Abbildung 2.11) [9]. STUMPS bildet die Grundlage für viele in der Industrie gebräuchlichen BIST Produkte kommerzieller Anbieter [30, 2] und dient ebenfalls als Basis für den eingebauten Hochgeschwindigkeitstest (vgl. Abbildung 1.5).

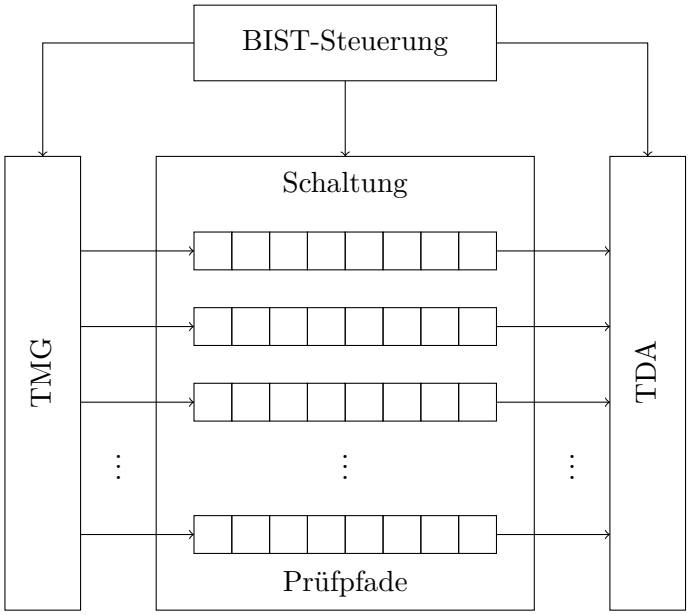


Abbildung 2.11: Selbsttest nach STUMPS-Prinzip

Das System besteht aus der zu testenden Schaltung, welche mit einer Prüfpfadarchitektur ausgestattet ist, einem Testmustergenerator (TMG) sowie einem Testdatenauswerter (TDA). Der Selbsttest wird durch die BIST-Steuerung verwaltet, die unter anderem das *SM*-Signal der Prüfpfade steuert, passende Takte

wählt usw. Bei STUMPS werden Pseudozufallsmuster im TMG mit geringem Hardwareaufwand über ein LFSR erzeugt, andere Varianten wie zellulare Automaten können ebenfalls zum Einsatz kommen [64, 8] Auch können kleinere Sätze an deterministischen Mustern gespeichert werden [45, 67, 28, 29]. Die Testdatenauswertung mittels TDA wird in Abschnitt 2.6.1 genauer beschrieben, da sie eine wichtige Grundlage für die in dieser Arbeit behandelten Themen ist.

Herausforderungen bei der Implementierung eines Selbsttest liegen unter anderem in strengeren Entwurfsregeln einer Schaltung, in dem Entwurf der BIST-Steuerung sowie in der Handhabung von unbekannten Logikwerten [30]. Eine Schaltung muss bestimmte Entwurfsregeln beachten, damit ein Selbsttest erfolgreich umgesetzt werden kann. Darunter fällt beispielsweise, dass *unbekannte Logikwerte (X-Werte)* minimiert werden sollten. X-Werte sind symbolische Werte, deren konkreter Logikwert im Test (0 oder 1) nicht durch eine Simulation vorhergesagt werden kann. Ursachen für X-Werte sind unter anderem nicht initialisierte Speicherelemente wie Flipflops, asynchrone Reset-Signale, Übergänge zwischen verschiedenen Taktdomänen, Ausgaben von analogen Bauelementen, Busbenutzung usw. X-Werte können die Testdatenauswertung im Selbsttest unbrauchbar machen, deshalb sollten sie so gut wie möglich verhindert werden. Gelangen trotz Einhalten der Entwurfsregeln für BIST noch immer unbekannte Logikwerte in die Testantworten, müssen diese separat behandelt werden. Ein anderes Beispiel für eine BIST-Entwurfsregel ist, dass ein eingebauter Speicher den Test nicht behindern sollte. Dieser kann beispielsweise durch einen zusätzlichen Prüfpfad (falls der Speicher nicht bereits selbsttestbar ist) „transparent“ geschaltet werden. Der Entwurf der BIST-Steuerung ist z. B. bei Schaltungen mit verschiedenen Taktdomänen erschwert, denn sie muss alle Taktsignale beim Test korrekt setzen, sodass kein Fehlverhalten künstlich herbeigeführt wird. Kommerzielle Hersteller bieten mittlerweile Lösungen zum automatisierten Einfügen der BIST Infrastruktur in eine Schaltung (z. B. [69, 114]).

2.6.1 Testdatenauswertung

Im Selbsttest ist der TDA dafür verantwortlich, die produzierten Testantworten auszuwerten und das Testergebnis (korrekt / fehlerhaft) zu berechnen. Da schon

externe Tester oft auf Kompaktoren, wie in Abschnitt 2.5 erwähnt, angewiesen sind, ist es einleuchtend, dass das Vergleichen der direkten Testantworten auf dem Chip nicht praktikabel ist. Attraktiv ist hier die *Signaturanalyse* [58, 10], bei welcher über alle Testantworten hinweg eine kleine Signatur berechnet wird, die dann mit einer vorberechneten Signatur verglichen werden kann. Die Theorie beruht dabei auf der zyklischen Redundanzprüfung, welche sich über ein Signaturregister mit mehreren Eingängen (engl. Multiple-Input Signature Register, MISR) besonders einfach implementieren lässt (vgl. Abbildung 2.12).

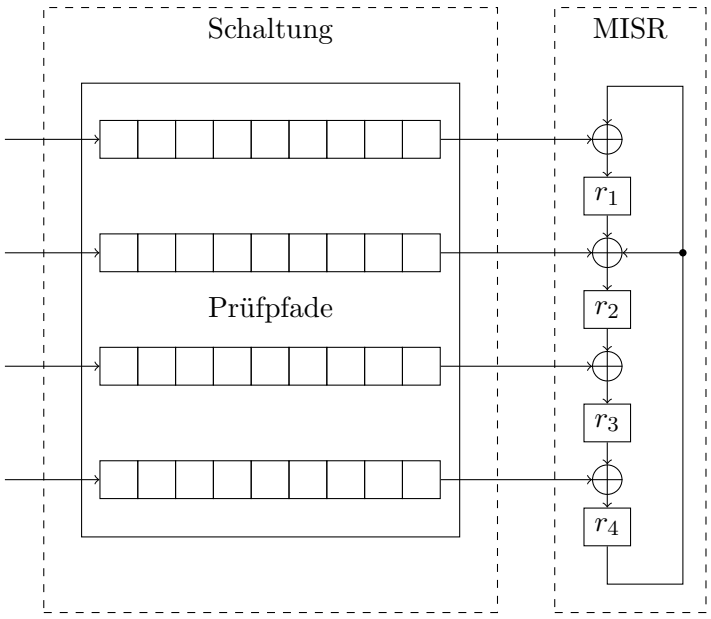


Abbildung 2.12: Ein Signaturregister mit mehreren Eingängen

Ohne Beschränkung der Allgemeinheit wird im Folgenden angenommen, dass alle Prüfpfade der Schaltung die gleiche Länge haben. Ist dies nicht der Fall, so können kürzere Prüfpfade mit logisch 0 „aufgefüllt“ werden, um gleiche Länge herzustellen.

Definition 2.3. *Gegeben sei eine Schaltung mit n Prüfpfaden, jeweils mit der Länge l. Die* $n \times l$ *Matrix* $\mathbf{A}$*,* $a_{ij} \in \{0, 1, X\}$*, definiert die* Testantwortmatrix *der Schaltung auf ein angelegtes Testmuster.*

Die Testantwortmatrix $\mathbf{A}$ beinhaltet alle Bits, welche in den Prüfpfaden als Schaltungsantwort auf ein angelegtes Testmuster aufgezeichnet werden. Beim Auslesen einer Testantwort wird $\mathbf{A}$ spaltenweise von rechts nach links durchlaufen

(der Inhalt der letzten Prüfzelle im Prüfpfad wird als erstes ausgelesen). In das MISR werden also die Spaltenvektoren $\mathbf{a_l}$, $\mathbf{a_{l-1}}$, bis $\mathbf{a_1}$ nacheinander geschoben. Nach Abschluss des Tests (einlesen aller Testantwortmatrizen) enthalten die Flipflops des MISRs dann die Signatur.

Beispiel 2.5. Gegeben seien die Schaltung und das MISR aus Abbildung 2.12. Das MISR wird durch einen Spaltenvektor $\mathbf{r} = \begin{pmatrix} r_1 & r_2 & r_3 & r_4 \end{pmatrix}^T$ beschrieben. Nun wird ein Testmuster an die Schaltung angelegt. Zur Veranschaulichung der MISR-Operationen werden zwei beispielhafte Taktzyklen des Auslesens der Testantwort betrachtet, die entsprechende Testantwortmatrix sei

$$\mathbf{A} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ 0 & 1 \end{pmatrix}.$$

Zu Beginn wird das MISR mit $\mathbf{r_0} = \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix}$ initialisiert. Wird nun der erste Schiebetakt durchgeführt, wird ein Vektor $\mathbf{a_2} = \begin{pmatrix} a_{1,2} & a_{2,2} & a_{3,2} & a_{4,2} \end{pmatrix}^T = \begin{pmatrix} 0 & 1 & 1 & 1 \end{pmatrix}^T$ aus der Schaltung geschoben, so enthält das MISR danach $\mathbf{r_1} = \mathbf{a_2}$. Nach einem zweiten Vektor $\mathbf{a_1} = \begin{pmatrix} 1 & 0 & 1 & 0 \end{pmatrix}^T$ berechnet das MISR den neuen Zustand zu

$$\mathbf{r_2} = \begin{pmatrix} r_{4,1} \oplus a_{1,1} \\ r_{1,1} \oplus r_{4,1} \oplus a_{2,1} \\ r_{2,1} \oplus a_{3,1} \\ r_{3,1} \oplus a_{4,1} \end{pmatrix} = \begin{pmatrix} 1 \oplus 1 \\ 0 \oplus 1 \oplus 0 \\ 1 \oplus 1 \\ 1 \oplus 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}.$$

In dieser Weise fährt die Berechnung fort, bis sämtliche Testantwortvektoren in das MISR geschoben wurden.

MISR sind extrem effizient und werden häufig in der Industrie eingesetzt – ein MISR der Länge k kann die gesamte Menge an Testantwortbits auf k Bits kompaktieren. Um die Signatur noch weiter zu verkleinern, kann das MISR auch mit einem vorgeschalteten räumlichen Kompaktierer (vgl. Abschnitt 2.5) kombiniert werden [134]. Es darf allerdings kein X-Wert in das MISR geschoben werden, denn bereits ein einzelner X-Wert kann die Signatur unbrauchbar machen.

Die dazu erforderliche Kenntnis über alle Testantwortbits kann jedoch nicht immer gegeben sein, selbst wenn die Schaltung die zuvor skizzierten Entwurfsregeln für BIST befolgt. Verbleiben X-Werte in den Testantworten, müssen diese besonders behandelt werden, damit die resultierende Signatur nicht korrumpiert wird. Es gibt im Wesentlichen drei, sich ergänzende Ansätze zur Handhabung von X-Werten: Blockieren, Maskieren und Tolerieren. Beim Blockieren von X-Werten wird versucht, in der Schaltung den X-Wert bereits am Entstehungsort (z. B. der Ausgang eines Flipflops) zu maskieren [63, 30]. Maskieren und Tolerieren werden im Folgenden genauer beschrieben werden, da sie insbesondere im Kapitel 5 relevant werden.

Maskieren

Eine Möglichkeit X-Werte zu behandeln besteht darin, diese an den Prüfpfadausgängen zu maskieren, sobald sie aus dem Prüfpfad geschoben werden. Damit wird ein X-Wert mit einem bekannten Logikwert überschrieben [80, 127, 82, 111, 121, 132]. Abbildung 2.13 zeigt ein prinzipielles System dieser Maskierung.

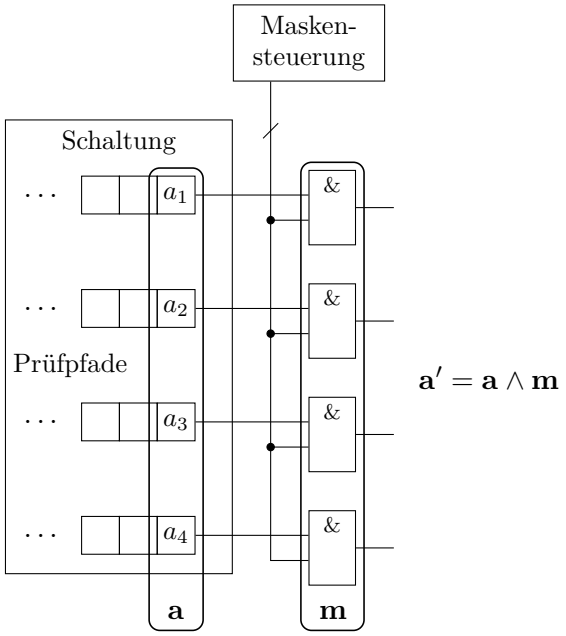


Abbildung 2.13: Maskierung von X-Werten mit UND-Gattern

Hinter jedem Prüfpfadausgang befindet sich ein UND-Gatter, dessen zweiter Eingang mit der Maskensteuerung verbunden ist. Dadurch bildet sich die maskierte

Testantwort $\mathbf{a}'$ über die bitweise logische UND-Verknüpfung: $a'_i = a_i \wedge m_i$. Im Idealfall setzt die Maskensteuerung $m_i = 0$, sobald $a_i = X$. Die Herausforderungen sind hier zum einen der hohe Steueraufwand, da es viele und lange Prüfpfade geben kann und zum anderen erhöhter Speicherbedarf, denn die Maskenbits müssen auf dem Chip gespeichert werden. Ansätze wie von Wang et al. [127] oder Tang et al. [111] versuchen diesen Mehraufwand zu minimieren, indem die Maskenbits komprimiert gespeichert und beispielsweise mit einem LFSR entpackt werden. Hier kann es allerdings passieren, dass ebenfalls Prüfpfadausgänge maskiert werden, welche keinen X-Wert haben. Das kann zur Reduzierung der Fehlereffizenz führen, da möglicherweise durch die Maskierung wichtige Fehlerinformationen verloren gehen können.

Tolerieren

X-Werte können alternativ im Kompaktor selbst toleriert werden. Hierzu gibt es verschiedene Strategien, je nach Variante des Kompaktierers. Für räumliche Kompaktierer gibt es etwa die X-Compact-Strategie von Mitra et al. [60, 61] oder das X-Filter von Sharma und Cheng [100]. Als Beispiel für eine zeitliche Kompaktierung wird hier das X-Streichende MISR von Touba et al. [116, 24] vorgestellt, da es in dieser Arbeit ebenfalls verwendet wird. Bei diesem Verfahren wird versucht, durch geschicktes Bilden von Linearkombinationen der einzelnen Bits des MISR X-Werte zu streichen. Das funktioniert, da X-Werte sich durch die lineare Rückkopplung auf mehrere Flipflops des MISR verteilen. Nötig ist dazu eine symbolische Simulation der Testantworten, bei der jeder aus der Schaltung herausgeschobene X-Wert eine eindeutige Nummer zugewiesen bekommt (also X_1, X_2, usw.). Dadurch kann genau verfolgt werden, welches Flipflop von welchen X-Werten beeinflusst wird.

Beispiel 2.6. Abbildung 2.14 zeigt erneut das MISR aus Abbildung 2.12, jetzt mit konkreten Testantwortbits, inklusive X-Werten. Die Testantwortmatrix, welche

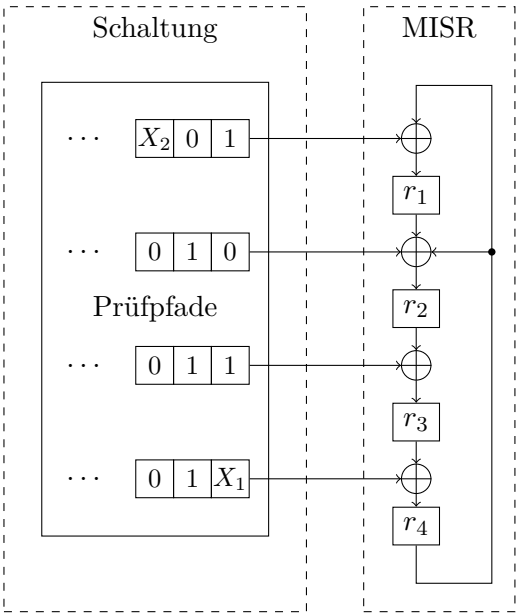


Abbildung 2.14: Beispiel der X-streichenden MISR-Operationen

den dargestellten Teil der Testantwortbits beschreibt, ist

$$\mathbf{A} = \begin{pmatrix} X_2 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & X_1 \end{pmatrix}.$$

Gefordert sei, dass mindestens 3 Bits der Signatur X-frei sein müssen, um ausreichend Fehlereffizienz zu gewährleisten. Nach dem ersten Schiebetakt gilt $\mathbf{r_1} = \begin{pmatrix} 1 & 0 & 1 & X_1 \end{pmatrix}^T$ (die letzte Spalte aus $\mathbf{A}$), es gibt genau 3 X-freie MISR-Bits ($r_{1,1}$–$r_{1,3}$). Nach dem zweiten Schiebetakt gilt für die nächste berechnete Signatur

$$\mathbf{r_2} = \begin{pmatrix} r_{4,1} \oplus a_{1,2} \\ r_{1,1} \oplus r_{4,1} \oplus a_{2,2} \\ r_{2,1} \oplus a_{3,2} \\ r_{3,1} \oplus a_{4,2} \end{pmatrix} = \begin{pmatrix} X_1 \\ X_1 \\ 1 \\ 0 \end{pmatrix}.$$

Es sind nur noch 2 MISR-Bits X-frei, aber durch Bilden der Linearkombination $r' = r_{1,2} \oplus r_{2,2} = 0$ kann X_1 gestrichen werden, und es sind insgesamt wieder 3 X-freie Bits verfügbar. Nach einem weiteren Takt enthält das MISR $\mathbf{r_3} = \begin{pmatrix} X_2 & X_1 & X_1 & 1 \end{pmatrix}^T$. Zwar lässt sich wieder X_1 aus den MISR-Bits herausrechnen ($r_{2,3} \oplus r_{3,3} = 0$), aber X_2 nicht. Da es nun weniger als drei X-freie

Linearkombinationen gibt ($r_{4,3}$ und $r_{2,3} \oplus r_{3,3}$), muss die vorherige Signatur $\mathbf{r_2}$ als *Zwischensignatur* verwendet werden und das MISR wird zurückgesetzt, bevor $\mathbf{a_1}$ hineingeschoben wird.

X-freie Linearkombinationen des MISR können gefunden werden, indem eine X-Matrix aufgestellt wird, bei welcher jede Zeile der Matrix einem Flipflop des MISR entspricht und pro vorhandenem X-Wert eine Spalte angelegt wird. Diese Matrix hat eine 1 an einer Position (i, j), wenn r_i von X_j beeinflusst wird, andernfalls ist dieser Wert 0. Sie kann mittels des Gauß-Verfahrens auf eine Dreiecksform gebracht werden, sodass Zeilen, welche ausschließlich aus Nullen bestehen, den X-freien Linearkombinationen entsprechen. In der ursprünglichen Veröffentlichung postuliert Touba, dass bereits 7 X-freie Kombinationen ausreichen, damit fehlerhafte Testantworten mit einer Wahrscheinlichkeit von mehr als 99% zu einer fehlerhaften Signatur führen [116].

3 Test auf kleine Verzögerungsfehler

Kapitel 2 nutzte den Übergangsfehler als einfaches Modell, um die grundlegenden Techniken beim Verzögerungstest zu erläutern. In diesem Kapitel wird nun der kleine Verzögerungsfehler (SDF) im Detail vorgestellt (Abschnitt 3.1). Als eine wesentliche Metrik zur Bewertung der Testqualität für SDFs wird das „Statistical Delay Quality Level (SDQL)“ in Abschnitt 3.2 beschrieben. Diese dient als Motivation der notwendigen Erweiterungen beim Verzögerungstest, um SDFs sichtbar zu machen. Hier werden in den Abschnitten 3.3 und 3.4 neue Verfahren bei der Testmustererzeugung, sowie der Betriebsfrequenztest vorgestellt, welche zu den wichtigsten Methoden beim Test auf SDFs zählen. Weiterführende Informationen zum Thema kann der interessierte Leser in der Literatur finden, beispielsweise im Buch „Test and Diagnosis for Small Delay Defects“ von Tehranipoor, Peng und Chakrabarty [113].

Der derzeitige Stand der Technik kann nicht garantieren, dass auch sehr kleine SDFs erkannt werden können, Abschnitt 3.5 zeigt die Grenzen der bisherigen Testverfahren auf. Abschnitt 3.6 definiert dann den versteckten Verzögerungsfehler. Dieser dient als Zielfehlermodell für den Hochgeschwindigkeitstest, dessen Prinzip in Abschnitt 3.7 vorgestellt wird. In diesem Abschnitt wird auch auf den Stand der Technik eingegangen, indem verschiedene Herausforderungen und deren bisherige Lösungen aufgezeigt werden. Damit bietet dieses Kapitel alle notwendigen Voraussetzungen zum Verständnis der eigenen Arbeiten in den Kapiteln 4 und 5.

3.1 Das Modell des kleinen Verzögerungsfehlers

In dieser Arbeit werden SDFs als Gatterverzögerungsfehler betrachtet (vgl. Abschnitt 2.1).

Definition 3.1. *Sei g ein Gatter mit n Eingängen g_i, $1 \leq i \leq n$ und einem Ausgang g_0. Zusätzlich bezeichne $\uparrow$ ($\downarrow$) eine steigende (fallende) Signalflanke. Dann ist $\varphi = (g_j, s, \zeta)$, $0 \leq j \leq n$, $s > 0$, $\zeta \in \{\uparrow, \downarrow\}$ ein* SDF *am Ort g_j, welcher die Signalflanke ζ um s Zeiteinheiten verzögert.*

Es werden individuelle Anstiegs- und Abfallszeiten der einzelnen Gattereingänge angenommen, sodass ein Gatter mit n Eingängen und einem Ausgang insgesamt $n + 1$ verschiedene Orte für einen SDF aufweist. Da die Größe eines SDF kontinuierlich ist, kann es praktisch unbegrenzt viele verschiedene SDFs in einer Schaltung geben. Problematisch ist dies insofern, als dass verschieden große SDFs am selben Fehlerort unterschiedliches Verhalten der Schaltung hervorrufen können. Dies spielt insbesondere bei der Frequenzauswahl beim Hochgeschwindigkeitstest (Kapitel 4) eine Rolle.

Beispiel 3.1. Gegeben sei der Volladdierer aus Beispiel 1.1, hier nochmals in Abbildung 3.1 dargestellt. Die Verzögerungszeiten der Gatter sind ebenfalls eingetragen, zum besseren Verständnis werden in diesem Beispiel alle Signalflanken mit derselben Verzögerung durch ein Gatter propagiert. Gegeben sei ein Testmuster $P_1 = (\mathbf{v_1}, \mathbf{v_2})$ bestehend aus $\mathbf{v_1} = \begin{pmatrix} a & b & c_{ein} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}$ und $\mathbf{v_2} = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}$, welches eine steigende Signalflanke an a erzeugt und über die Gatter d und e zum oberen Flipflop propagiert (in der Abbildung blau eingezeichnet). Die Verzögerung beträgt $t = 30\,\text{ps} + 27\,\text{ps} = 57\,\text{ps}$. Am oberen Gattereingang d_1 des Gatters d sei nun ein SDF $\varphi = (d_1, 15\,\text{ps}, \uparrow)$ vorhanden. Das bedeutet, dass die Signalflanke von a um zusätzliche 15 ps verzögert wird und das obere Flipflop erst zum späteren Zeitpunkt $t' = 72\,\text{ps}$ erreicht.

Beispiel 3.1 zeigt, dass SDFs sich grundsätzlich wie Übergangsfehler erkennen lassen, indem eine Signalflanke am Fehlerort erzeugt und zu einem der Schaltungsausgänge propagiert wird. Die große Herausforderung bei der Erkennung von φ liegt jedoch darin, dass der Fehler, abhängig vom Pfad über den er propagiert

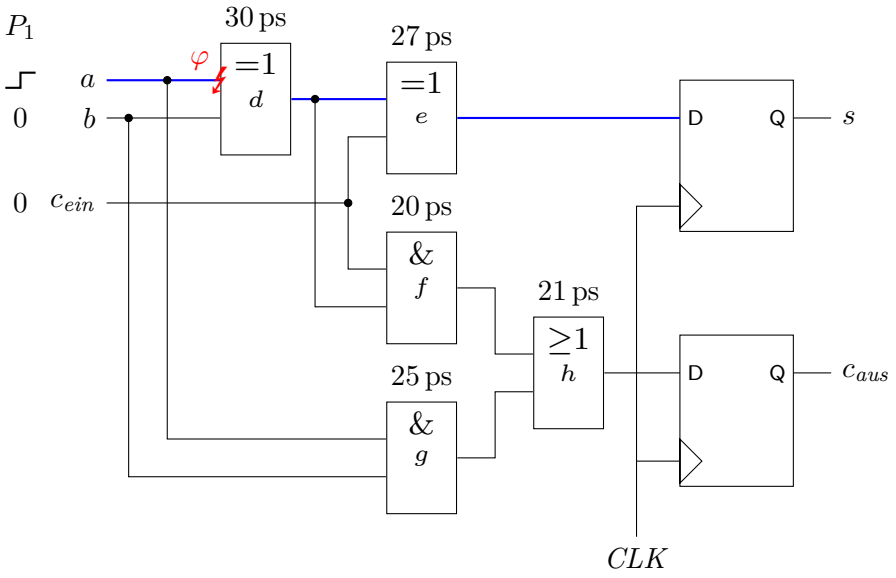


Abbildung 3.1: Ein Volladdierer mit Fehler und Eingangsbelegung

wird, möglicherweise zum Beobachtungszeitpunkt nicht erkennbar ist, obwohl der Fehler den Ausgang beeinflusst. In einem solchen Fall ist die zusätzliche Verzögerung s zu klein, um ein fehlerhaftes Verhalten zu provozieren.

Beispiel 3.2. Nimmt man an, dass die Flipflops aus Beispiel 3.1 mit einer Taktperiode von 75 ps gesteuert werden, so erkennt man, dass der Fehler nicht am oberen Flipflop erkannt wird, da die zusätzliche Verzögerung zu klein ist. Ein anderes Testmuster $P_2 = (\mathbf{v_3}, \mathbf{v_4})$ mit $\mathbf{v_3} = \begin{pmatrix}0 & 1 & 1\end{pmatrix}$ und $\mathbf{v_4} = \begin{pmatrix}1 & 1 & 1\end{pmatrix}$ würde allerdings die Signalflanke von a zusätzlich über die Gatter d, f und h zu c_{aus} propagieren. Abbildung 3.2 zeigt diesen zusätzlichen Signalpfad (rot eingezeichnet). Dieser Pfad hat eine Verzögerung von $t'' = 30\,\text{ps} + s + 20\,\text{ps} + 21\,\text{ps} = 86\,\text{ps}$. In diesem Fall würde das untere Flipflop einen falschen Logikwert aufzeichnen.

Eine Erkenntnis aus Beispiel 3.2 ist, dass Testmustererzeugung (ATPG) für SDFs ebenfalls das zeitliche Verhalten der Schaltung berücksichtigen muss. Algorithmen für Übergangsfehler, welche ausschließlich die Logik betrachten, reichen möglicherweise nicht aus, denn diese werden aus Laufzeitgründen versuchen, die fehlerhafte Signalflanke über einen möglichst kurzen Pfad zu einem der Ausgänge zu propagieren (vgl. z. B. [49]).

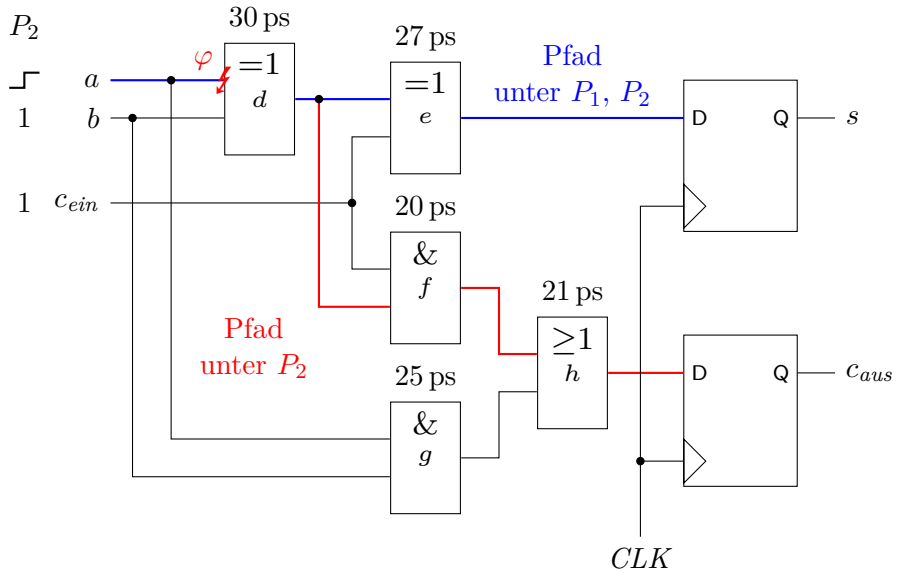


Abbildung 3.2: Der Volladdierer mit Flipflops, Fehler und Transitionspfaden für das Muster P_2

3.2 Qualität eines Verzögerungstests

In Beispiel 3.2 wurde gezeigt, dass die Verzögerungszeit des Pfades, über welchen eine fehlerhafte Signalflanke propagiert wird, entscheidend für die Erkennung eines SDFs ist. Aus diesem Grunde spielt diese auch bei modernen Metriken zur Bestimmung der Qualität eines Verzögerungstests eine zentrale Rolle. Für die folgenden Betrachtungen bezeichnen t_{Test} die Stabilisierungszeit eines Signals an einem Pfadendpunkt (Flipflop oder primärer Ausgang) und $t = 1/f$ den *Beobachtungszeitpunkt* des Pfadendpunkts, wobei f die Testfrequenz darstellt. Ein SDF mit Größe s kann nur dann erkannt werden, wenn für die Pufferzeit $t_{Puffer} = t - t_{Test} < s$ gilt. Eine quantitative Analyse der Testqualität unter diesen Gesichtspunkten bietet dabei das „Statistical Delay Quality Model (SDQM)“ [36, 120, 87, 86], welches 2005 von Sato et al. entwickelt, in Grundzügen aber bereits 1997 von Jone et al. veröffentlicht wurde [34]. In diesem Modell leitet sich der Verlust der Testqualität aus der Wahrscheinlichkeit ab, dass ein SDF φ mit einer Größe s vom Test nicht erkannt wird, aber im Normalbetrieb zu einem Fehler führen kann. Das bedeutet, es existiert ein Pfad durch die Schaltung mit ausreichender Verzögerung, der jedoch nicht vom Test genutzt wird (vgl. auch Abbildung 3.3).

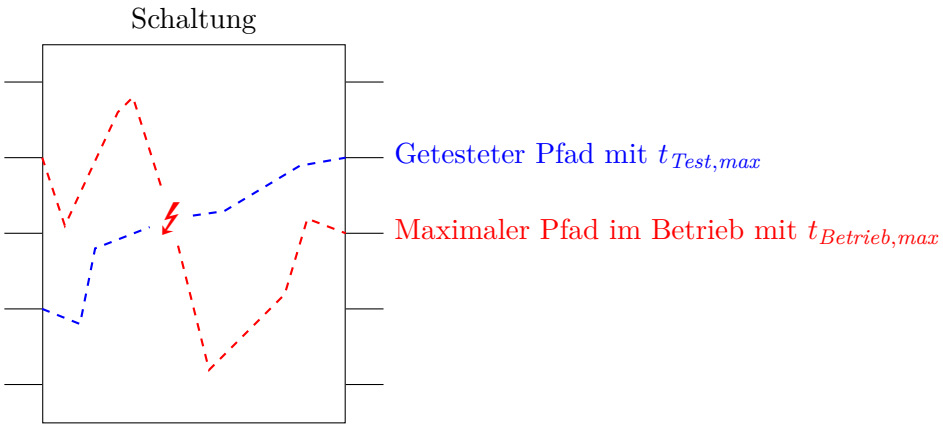


Abbildung 3.3: Zusammenhang zwischen maximaler Pfadlänge im Test und im Betrieb

Hier bezeichnet $t_{Test,max}$ ($t_{Betrieb,max}$) die Verzögerungszeit des längsten, im Test (Betrieb) sensibilisierten Pfades. Ein Problem tritt auf, wenn $t_{Test,max} \ll t_{Betrieb,max}$ gilt. Bezogen auf Beispiel 3.2 bedeutet dies etwa, dass der Test den (zu kurzen) Pfad d–e sensibilisiert, obwohl der längere Pfad d–f–h hätte genutzt werden können. Tritt ein solcher Fall häufig auf, so hat der Test eine geringe Qualität.

Die zur Berechnung des SDQM benötigte Wahrscheinlickeit, dass ein SDF mit Größe s auftritt – als $F(s)$ bezeichnet – muss dabei aus Prozessdaten gewonnen werden und wird sowohl in [34] als auch in [86] als Exponentialfunktion approximiert. Im Allgemeinen wird $t_{Test,max} \leq t_{Betrieb,max}$ angenommen, dies ist insbesondere bei einfacheren ATPG-Varianten der Fall. Zu beachten ist, dass nicht alle Pfade, die in einem Test sensibilisiert werden können, auch im Betrieb sensibilisierbar sind. Dies liegt daran, dass bestimmte Eingangskombinationen im Betrieb möglicherweise nicht vorkommen können bzw. nicht zugelassen sind. Deshalb muss diese Annahme nicht immer erfüllt sein.

Nun werden zwei Pufferzeiten definiert, um SDFs klassifizieren zu können. Die Pufferzeit des getesteten Pfades berechnet sich über $t_{Puffer,Test} = t - t_{Test,max}$. Analog sei $t_{Puffer,Betrieb} = t_{nom} - t_{Betrieb,max}$ die Pufferzeit des längsten, im Betrieb sensibilisierbaren Pfades. Für $t \geq t_{nom}$ gilt unter obiger Annahme $t_{Puffer,Test} \geq t_{Puffer,Betrieb}$. Basierend auf diesen Pufferzeiten werden SDFs in eine der folgenden Kategorien eingeteilt:

1. Alle SDFs mit Größe $s > t_{Puffer,Test}$ werden durch den Test erkannt,

2. kein SDF mit Größe $s < t_{Puffer,Betrieb}$ kann eine sichtbare Abweichung des Schaltungsverhaltens provozieren und

3. SDFs mit einer Größe $t_{Puffer,Betrieb} \leq s \leq t_{Puffer,Test}$ werden vom Test nicht erkannt, können aber zum Fehlverhalten während des Betriebs führen.

Dieser Zusammenhang ist graphisch in Abbildung 3.4 dargestellt, hierbei wurde $F(s) = 0{,}1 \exp(-0{,}1s)$ gewählt [34].

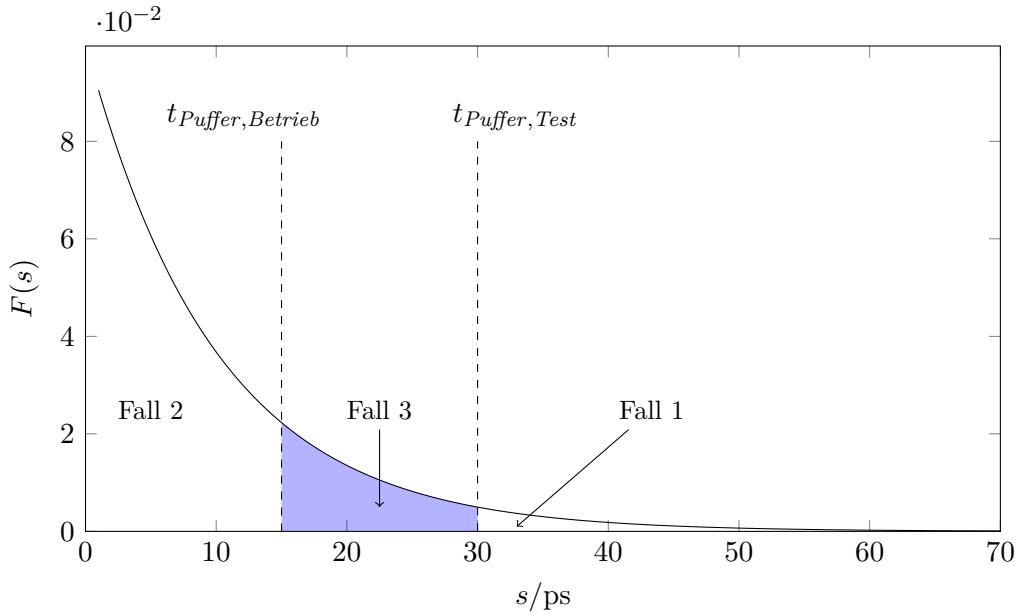


Abbildung 3.4: Beispiel für die Defektwahrscheinlichkeit

Der Qualitätsverlust eines Tests nach SDQM, als „Statistical Delay Quality Level (SDQL)“ bezeichnet, bestimmt sich über die Wahrscheinlichkeit, dass ein SDF der dritten Kategorie auftritt:

$$P[\text{SDF mit Größe } s \text{ nicht erkannt}] = \int_{t_{Puffer,Betrieb}}^{t_{Puffer,Test}} F(s)\,\mathrm{d}s. \tag{3.1}$$

Für eine optimale Testqualität muss diese Wahrscheinlichkeit minimiert werden.

3.3 Testmustererzeugung für kleine Verzögerungsfehler

Um Gleichung (3.1) zu minimieren, muss ATPG für kleine Verzögerungsfehler versuchen, die fehlerbehafteten Signalflanken über Pfade mit möglichst hoher Verzögerung zu treiben, d. h. es wird versucht, t_{Test} möglichst nah an $t_{Betrieb}$ anzunähern. Einer der älteren Ansätze berechnet die k längsten Pfade pro Gatter (KLPG) [32, 79, 77, 101], dies ist aber mit hohem Rechenaufwand und großen Testsätzen verbunden. Ein anderer Ansatz, welcher von Srivastava et al. in [108] verfolgt wird, liegt darin, die Gatter- und Pfadverzögerungsmodelle zu verbinden: Es werden Testmuster für Pfadverzögerungsfehler erzeugt, aber später werden die erkennbaren Gatterverzögerungsfehler daraus zurück berechnet. Dadurch werden ebenfalls sehr lange Pfade sensibilisiert.

In der Forschung rückt seit einiger Zeit auch verstärkt ATPG basierend auf dem Booleschen Erfüllbarkeitsproblem (engl. Satisfiability, SAT) in den Fokus [21]. Dabei wird die Schaltung in eine sogenannte SAT-Instanz transformiert, welche das Schaltverhalten formal anhand von Literalen und Klauseln beschreibt. Dort wird der Fehler „injiziert“, indem zusätzliche Bedingungen in die SAT-Instanz kodiert werden. Abschließend wird die SAT-Instanz mit Hilfe eines SAT-Lösers gelöst und aus dem Ergebnis wird das Testmuster extrahiert. Der große Vorteil von SAT-ATPG liegt darin, dass es viel schneller möglich ist, beweisbar (unter dem gegebenen Modell) zu zeigen, dass ein Fehler nicht erkannt werden kann – das ist genau dann der Fall, wenn der SAT-Löser die Instanz als unlösbar klassifiziert.

SAT-ATPG für SDFs kann ebenfalls versuchen, fehlerhafte Signalflanken über Pfade mit möglichst großer Verzögerung zu propagieren. PHAETON [91, 89], entwickelt von Sauer et al., ist eine solche Variante. Andere Methoden umfassen das „As-Robust-As-Possible“ ATPG von Eggersglüß et al. [20], welche versuchen, einen möglichst robusten Test auf Pfadverzögerungsfehler zu erzeugen. „Möglichst robust“ bedeutet in diesem Zusammenhang, dass so viele Bedingungen wie möglich erfüllt werden, aber nicht alle erfüllt werden müssen, wenn andernfalls kein Testmuster erzeugt werden kann.

Zur Erkennung von SDFs interessant ist das WaveSAT ATPG, ebenfalls entwickelt von Sauer et al. [90]. Bei diesem Verfahren wird der vollständige relevante Signalverlauf in der SAT-Instanz kodiert, sodass typische Probleme, welche bei einer reinen Logiksimulation eines Testvektorpaars auftreten – Signalpulse sind z. B. nur schwer handzuhaben, da sie den finalen Logikwert nicht beeinflussen – elegant gelöst werden. Dadurch können alle Zeitpunkte betrachtet werden, an denen ein SDF ein abweichendes Verhalten des Ausgangssignals zum fehlerfreien Signal verursacht. Da SDFs auch auf einen ELF hinweisen können, wurde auch eine spezielle SAT-ATPG-Variante für ELFs entwickelt [92]. Diese kombiniert die beiden zuvor genannten Werkzeuge PHAETON und WaveSAT, indem zunächst PHAETON benutzt wird, um einen robusten Test zu erzeugen. Ist dies nicht möglich, wird mit WaveSAT dann ein genauerer Test berechnet.

In der Industrie ist der Stand der Technik derzeit bei ATPG für Übergangsfehler, erweitert um zeitliche Informationen der Schaltung, sodass wiederum Pfade mit besonders langen Verzögerungszeiten ausgewählt werden. Ahmed et al. nutzen klassisches n-erkennendes ATPG, bei welchem versucht wird, jeden Fehler über n verschiedene Testmuster zu erkennen. Dadurch erhält man eine große Verteilung der Pfadlängen [5]. Aus dem so berechneten Testsatz werden abschließend jene Muster ausgewählt, welche die größten Verzögerungszeiten aufweisen. Lin et al. haben in [49] ein zweistufiges Verfahren vorgestellt, in welchem Aktivierungs- und Propagierungspfad getrennt voneinander berechnet werden. Bei beiden Pfaden wird durch Ausnutzen der zeitlichen Informationen über die Schaltung der längste Pfad bevorzugt. Durch die getrennte Betrachtung kann jedoch nicht garantiert werden, dass der sensibilisierte Pfad tatsächlich die maximale Verzögerungszeit besitzt, zusätzlich benötigt dieser Ansatz eine hohe Rechenzeit. Um diese zu verringern, versuchen Goel et al. in [25] zunächst, Fehlerorte zu identifizieren, für welche besondere ATPG-Ansätze erforderlich sind. Nur für diese werden dann Methoden wie z. B. [49] angewandt, für die verbleibenden wird klassisches ATPG für Übergangsfehler genutzt, welches weitaus schneller Testmuster berechnet.

3.4 Betriebsfrequenztest und Vielfrequenztest

Im vorherigen Abschnitt wurde beschrieben, wie $t_{Puffer,Test} = t - t_{Test}$ durch Maximierung von t_{Test} minimiert werden kann, was aber mitunter mit hohem Rechenaufwand verbunden ist. Alternativ dazu kann jedoch auch der Beobachtungszeitpunkt t verringert werden, um $t_{Puffer,Test}$ zu minimieren. Klassischerweise wird ein Test mit einer Frequenz durchgeführt, welche deutlich kleiner als die Betriebsfrequenz ist, um die Verlustleistung beim Test zu minimieren. In Folge ist $t_{Puffer,Test} \gg t_{Puffer,Betrieb}$ und die Testqualität entsprechend gering. Deshalb wird heutzutage unter Betriebsbedigungen ($t = t_{nom}$) getestet, wodurch $t_{Puffer,Betrieb} - t_{Puffer,Test} = t_{Test,max} - t_{Betrieb,max}$ gilt und die Testqualität deutlich erhöht wird [48, 31]. Der Preis dieses Ansatzes liegt darin, dass kostengünstiges Testequipment (ATE) möglicherweise nicht in der Lage ist, die benötigte Betriebsfrequenz der Schaltung bereitzustellen. Spezielle Hardware zur Testfrequenzerzeugung, welche auf dem Chip integriert sein muss, wird deshalb benötigt [74]. Zusätzlich steigt beim Betriebsfrequenztest die Verlustleistung an, denn oftmals wird während des Tests eine deutlich höhere Schaltaktivität erzeugt, als während des Betriebs üblich ist. In Folge ist das Versorgungsnetz der Schaltung nicht ausreichend dimensioniert und es kommt zum Absinken der Versorgungsspannung an betroffenen Transistoren [4].

Es ist jedoch nicht unbedingt erforderlich, für SDFs mit ausreichender Größe den Test mit Betriebsfrequenz durchzuführen. Stattdessen können mehrere Testfrequenzen verwendet werden, sodass die individuellen Pufferzeiten optimiert werden und gleichzeitig nicht zu hohe Frequenzen benötigt werden [34, 55]. Dieser Ansatz, der bereits 1990 von Mao und Ciletti vorgeschlagen wurde [55], hat in der Industrie keine nennenswerte Verwendung gefunden. Es zeigen sich hier allerdings bereits Parallelen zum Hochgeschwindigkeitstest, denn auch dieser benutzt typischerweise eine Reihe von Testfrequenzen. Tatsächlich ist das in dieser Arbeit entwickelte Verfahren zur Optimierung der Testfrequenzen auf einen gegebenen Testsatz (Kapitel 4) stark mit den Arbeiten in [34, 55] verwandt.

3.5 Beschränkungen der modernen Testverfahren

Die bisher in diesem Kapitel vorgestellten Verfahren eignen sich, um die Qualität von Verzögerungstests, beispielsweise bestimmt durch SDQL, zu optimieren. Eine Grenze dieser Verfahren liegt allerdings darin, dass für die minimale erkennbare Fehlergröße $s_{min} = t_{Puffer,Test}$ gilt. Bei sehr langen bzw. kritischen Pfaden einer Schaltung kann diese sehr klein sein, aber es gibt auch mögliche Fehlerorte in einer Schaltung, bei der selbst der längste sensibilisierbare Pfad eine sehr hohe Pufferzeit besitzt. Ein Extrembeispiel hierfür ist etwa eine Verbindungsleitung zwischen zwei Flipflops eines Schieberegisters. Selbst ein moderner Betriebsfrequenztest mit ATPG auf dem Stand der Technik wird an solchen Fehlerorten keine SDFs erkennen können, deren Größe unterhalb von $t_{Puffer,Test}$ liegt – es muss also ein sehr großer SDF vorliegen, damit ein Fehlverhalten erkannt werden kann. Freilich bezieht sich SDQL nur auf solche Fehler, welche ein tatsächliches Fehlverhalten während des Betriebs provozieren können. Es wird jedoch nicht berücksichtigt, dass die den SDFs möglicherweise zugrunde liegenden schwachen Schaltungsstrukturen durch Alterungseffekte degradieren können [39]. So kann sich beispielsweise aus einer ohmsche Unterbrechung mit Widerstand R eine echte Unterbrechung ($R \to \infty$) bereits kurz nach Betriebsbeginn entwickeln. Um solche ELFs erkennen zu können, sind aus diesem Grunde die in diesem Kapitel bislang beschriebenen Ansätze nur begrenzt geeignet. Im Folgenden werden daher zunächst versteckte Verzögerungsfehler beschrieben, gefolgt vom Hochgeschwindigkeitstest, dessen Aufgabe es ist, diese Fehler sichtbar zu machen.

3.6 Versteckte Verzögerungsfehler

Orientiert sich die minimale Fehlergröße nicht mehr an den Pufferzeiten sensibilisierbarer Pfade, sondern an den Eigenschaften des betroffenen Gatters, so ergibt sich eine drastisch veränderte Situation in Bezug auf die Qualität eines Verzögerungstests. Selbst ein Betriebsfrequenztest wird nun eine schlechtere Testqualität aufweisen, da alle Fehler unerkannt bleiben, die ausschließlich über sehr kurze Pfade propagiert werden können. Es können sogar Fehler unerkannt bleiben, obwohl es Testmuster gibt, welche den Fehler aktivieren und über lange Pfade zu

den Ausgängen treiben. In diesem Fall werden die Fehler auch nicht den Betrieb beeinflussen, dennoch ist diese Betrachtungsweise sinnvoll, denn die den SDFs zugrunde liegenden Defekte können ja an beliegen Orten in der Schaltung zu ELFs ausarten.

Definition 3.2. *Seien φ ein SDF mit minimaler Fehlergröße s_{min}, t_{nom} der nominelle Beobachtungszeitpunkt, $\mathscr{P}$ ein Testsatz und sei $t_{Puffer,Test} = t_{nom} - t_{Test,max}$ die Pufferzeit des längsten unter $\mathscr{P}$ sensibilisierten Pfades mit Verzögerung $t_{Test,max}$, welcher φ zu einem der Schaltungsausgänge propagiert. Wenn $s_{min} < t_{Puffer,Test}$, dann ist φ ein* versteckter Verzögerungsfehler (HDF) unter $\mathscr{P}$ und t_{nom}. *Die Menge aller HDFs unter $\mathscr{P}$ und t_{nom} wird als $\Phi(\mathscr{P}, t_{nom})$ bezeichnet. Ist eine Menge T an Beobachtungszeitpunkten gegeben, so wird die Menge aller HDFs unter $\mathscr{P}$ und T als $\Phi(\mathscr{P}, T)$ bezeichnet.*

Ein HDF unter $\mathscr{P}$ und t_{nom} muss kein HDF unter einem anderen Testsatz $\mathscr{P}'$ und t_{nom} sein, dies ist eine Frage der Qualität des Testsatzes. Es kann allerdings auch Fehlerorte geben, an denen ein SDF mit s_{min} unter t_{nom} unerkennbar bleibt, unabhängig vom angelegten Testmuster.

Definition 3.3. *Seien φ ein SDF mit minimaler Fehlergröße s_{min}, t_{nom} der nominelle Beobachtungszeitpunkt und sei $t_{Puffer,min} = t_{nom} - t_{max}$ die Pufferzeit des längsten sensibilisierbaren Pfades mit Verzögerungszeit t_{max}, welcher φ zu einem der Schaltungsausgänge propagiert. Wenn $s_{min} < t_{Puffer,min}$, dann ist φ ein* harter HDF unter t_{nom}. *Die Menge aller harten HDFs unter t_{nom} wird als $\Phi(t_{nom})$ bezeichnet.*

Harte HDFs senken in jedem Fall die Fehlerabdeckung beim Betriebsfrequenztest. Sie benötigen Beobachtungszeitpunkte kleiner als t_{nom}, um erkannt werden zu können.

Bestimmung der minimalen Fehlergröße

Moderne Fertigungstechnologien weisen eine erhöhte Sensibilität gegenüber Prozessvarationen auf. Darunter versteht man subtile, aber harmlose Abweichungen

der gefertigten Schaltungsstrukturen von der Spezifikation. Diese haben seit dem Unterschreiten der Strukturgrößen von 100 nm einen nicht mehr zu vernachlässigenden Einfluss auf die elektrischen Eigenschaften der Bauteile [6]. Beispiele dafür sind Variationen in den Längen und Weiten der Transistoren oder natürliche Schwankungen in den Dotierungskonzentrationen der p- und n-Gebiete, wodurch beispielsweise die Schwellspannung des betroffenen Transistors variiert. In Folge wird die Schaltzeit des Transistors ebenfalls schwanken, sodass die Verzögerungszeit t des resultierenden Gatters über die gefertigten Schaltungen hinweg dem Verhalten einer Zufallsvariablen entspricht. Die tatsächliche Modellierung der Prozessvariationen ist komplex und würde den Rahmen dieser Arbeit sprengen. Daher sei an dieser Stelle auf entsprechende Literatur verwiesen, beispielsweise auf das Buch „Statistical Analysis and Optimization for VLSI: Timing and Power“ von Srivastava, Sylvester und Blaauw [109]. Der Einfachheit halber sei hier angenommen, dass die Verzögerungszeit t eines Gatters einer Normalverteilung mit Erwartungswert $E[t] = \mu$ und Standardabweichung σ folgt. Abbildung 3.5 zeigt eine beispielhafte Verteilung mit $\mu = 30\,\text{ps}$ und $\sigma = 6\,\text{ps}$.

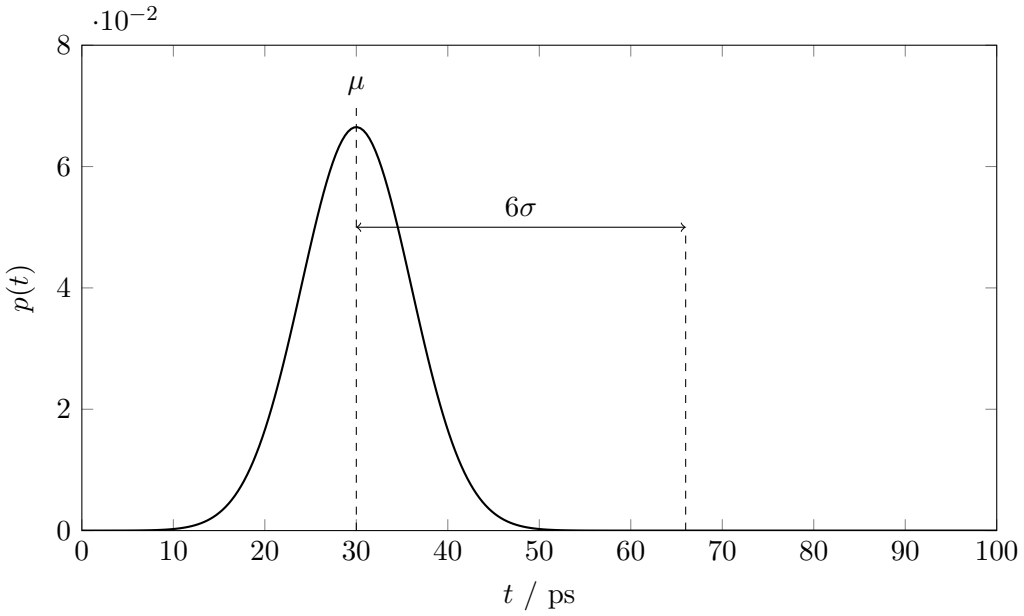


Abbildung 3.5: Normalverteilung einer Gatterverzögerungszeit

Definition 3.4. *Sei g ein Gatter in einer Schaltung, über das eine Signalflanke mit einer zufälligen Verzögerungszeit mit Standardabweichung σ propagiert wird. Dann bestimmt sich die* minimal erkennbare Fehlergröße *für diese Signalflanke zu $s_{min} = k \cdot \sigma$, für ein $k > 0$.*

In dieser Arbeit wird $k = 6$ angenommen. Ein Gatter gilt demnach als defekt, wenn dessen Verzögerungszeit in Abbildung 3.5 außerhalb der eingezeichneten Grenze $\mu + 6\sigma$ liegt. Unter der Annahme von $\sigma = 0{,}2\mu$ und $k = 6$ verzögert ein defektes Gatter eine Signalflanke also auf etwa das Doppelte der erwarteten Zeit. Das ist eine sehr kleine zusätzliche Verzögerung und führt (wie vorher gezeigt) zu schlechten Fehlerabdeckungen und möglicherweise pessimistischen Qualtitätsabschätzungen. Hat man jedoch einen Test, der auf Fehler mit Größe s_{min} optimiert ist, so ist es sehr wahrscheinlich, dass man damit auch Fehler mit Größen über s_{min} erkennen kann [156].

3.7 Der Hochgeschwindigkeitstest

Das Ziel des Hochgeschindigkeitstests (FAST) ist, alle HDFs $\Phi(\mathscr{P}, t_{nom})$ unter $\mathscr{P}$ und t_{nom} zu erkennen. Um das zu erreichen, wird die Pufferzeit des im Test sensibiliserten Pfades minimiert, indem der Beobachtungszeitpunkt t auf Werte unterhalb von t_{nom} verringert wird. Das bedeutet, dass die Schaltung übertaktet wird [141, 3, 7, 43]. Da es verschiedene $t_{Puffer,Test}$ für verschiedene HDFs geben kann, ist FAST ein Vielfrequenztest (vgl. Abschnitt 3.4). Nimmt man an, dass beliebige Beobachtungszeitpunkte gewählt werden können, so kann man mit FAST sämtliche HDFs erkennen, da immer ein t gewählt werden kann, sodass $s_{min} > t - t_{Test}$ gilt. In der Praxis lässt sich das in der Regel nicht umsetzen, sodass auch FAST nicht immer vollständige HDF-Abdeckung garantieren kann. Man muss an dieser Stelle jedoch erwähnen, dass jeder über FAST erkannte HDF insgesamt die Testqualität steigert. Das heißt, selbst bei unvollständiger Fehlerabdeckung kann man durch FAST mehr Fehler erkennen, als wenn ausschließlich bestehende Testverfahren eingesetzt werden. In diesem Abschnitt wird der Stand der Technik zum FAST vorgestellt. Er gliedert sich nach den Herausforderungen, welche für eine realistische Implementierung gelöst werden müssen.

3.7.1 Erzeugen der Testfrequenz

Das erste Problem ist, die erforderlichen Frequenzen auf der Schaltung selbst zu erzeugen. In Abschnitt 3.4 wurde erwähnt, dass bereits beim Betriebsfrequenztest auf integrierte Testtakterzeugung ausgewichen werden muss, sofern das ATE nicht in der Lage ist, die benötigte Frequenz zu erzeugen. Dieses Problem ist beim FAST verschärft, da nicht nur die Betriebsfrequenz, sondern auch mehrere Frequenzen darüber mit hoher Auflösung benötigt werden.

In der Industrie werden häufig Taktgeneratoren basierend auf Phasenregelschleifen (engl. Phase-locked Loops, PLLs) eingesetzt, welche die erforderlichen kurzen Testpulse z. B. mit spezieller Schaltlogik erzeugen [135, 74, 59]. Diese eignen sich ohne weitere Anpassungen nur bedingt für FAST, da die Menge und Auflösung der erzeugbaren Frequenzen durch die PLL begrenzt ist. Zudem muss die PLL sich jedes Mal auf die Zielfrequenz einschwingen, was eine gewisse Zeit erfordert. Ein industrieller Taktgenerator, welcher direkt für FAST eingesetzt werden kann, wurde von Kaeriyama et al. in [35] vorgestellt. Dabei handelt es sich um einen Taktgenerator, der mittels Phaseninterpolation aus einem Referenzsignal ein nahezu beliebig steuerbares Ausgangssignal erzeugt, da der Generator sowohl die steigende als auch die fallende Taktflanke verschieben kann. In der Veröffentlichung geben die Autoren an, aus einem Referenzsignal mit einer Frequenz von 1 GHz Ausgangssignale im Frequenzbereich von 125 MHz bis 2 GHz erzeugen zu können. Eine Anwendung für FAST findet sich dann auch in der Arbeit von Noguchi et al. [66].

Im wissenschaftlichen Bereich gibt es einige Arbeiten zu Taktgeneratoren, welche speziell auf die Anforderungen des FAST zugeschnitten sind. Die hier vorgestellten Verfahren von Tayade und Abraham [112] sowie Pei et al. [70, 71, 72] bieten beide die Möglichkeit, die Zielfrequenz im Testmuster zu kodieren, was insbesondere ATPG für FAST vereinfacht. Zusätzlich entstehen keine weiteren Verzögerungen, da nicht auf das Einschwingen von PLLs oder anderer analoger Hardware gewartet werden muss. Beide Ansätze basieren auf dem gleichen Prinzip: Ein Eingangssignal wird durch eine programmierbare Verzögerungsstufe geschickt, danach werden mit Hilfe des verzögerten Signals zwei kurze Taktpulse erzeugt. Beim Programmable Capture Generator (PCG) von Tayade und Abraham [112] werden dazu das

Startsignal und das verzögerte Signal verwendet, während im Launch and Capture Clock Generator (LCCG) von Pei et al. [70, 71, 72] das Signal parallel durch zwei Verzögerungsstufen (als obere und untere Stufe bezeichnet) geschickt wird. Die obere Stufe hat dabei immer eine geringere Verzögerung als die untere Stufe. Aus der Differenz dieser beiden Stufen werden dann die benötigten Testtakte gewonnen.

3.7.2 Schaltaktivität

Der Testbetrieb einer Schaltung unterscheidet sich stark vom normalen Betrieb. Ein struktureller Test wie der Verzögerungstest wird typischerweise eine hohe Schaltaktivität in der Schaltung erzeugen, da viele Eingänge (primär oder pseudo-primär) Schaltvorgänge vollziehen, um mit möglichst wenig Testmustern viele Fehlerorte abdecken zu können. Dadurch entsteht aber eine hohe Verlustleistung. Da das Versorgungsnetz der Schaltung nicht für diesen Betrieb ausgelegt wurde, entsteht ein erhöhter Abfall der Versorgungsspannung an den Bauelementen. In Folge verlangsamen sich die Bauelemente künstlich und die Testergebnisse können verfälscht werden. Dieses Problem ist bei FAST verstärkt, denn nun wird zusätzlich noch die Testfrequenz über die Betriebsfrequenz hinaus erhöht, was zu einem deutlich erhöhten Spannungsabfall führt. Ahmed und Tehranipoor haben das Problem genauer betrachtet und in [4, 3] eine erste Lösung entwickelt. Dabei wird eine zweifache Simulation durchgeführt: Zunächst wird in einer ersten Simulation die Schaltaktivität bestimmt, danach wird auf Grundlage der Schaltaktivität der individuelle Spannungsabfall der Elemente berechnet. Dieser wird beim zeitlichen Verhalten in einem zweiten Simulationsschritt berücksichtigt, sodass korrigierte zeitliche Simulationen möglich sind.

3.7.3 Prozessvariationen

Definition 3.4 bestimmt die minimale Fehlergröße s_{min} über die Standardabweichung der betroffenen nominellen Verzögerungszeit. Damit werden den Prozessvariationen bereits explizit Rechnung getragen. Eine Herausforderung ist allerdings

dennoch, dass s_{min} im Vergleich zur Verzögerungszeit des sensibilisierten Pfades sehr klein sein kann. Da alle Elemente des Pfades von Variationen betroffen sind, kann es schwierig bis unmöglich sein, auf Pfaden mit hoher Verzögerung eine fehlerhafte Testantwort einem tatsächlichen Defekt zuzuordnen – schließlich könnte auch schlicht die Kombination der Variationen aller Elemente für die hohe Verzögerung gesorgt haben. Die nun erforderliche Klassifikation einer Schaltung in gut oder defekt steht bereits seit einigen Jahren im Fokus der Forschung.

Ein wichtiger Ansatz bezieht Informationen aus benachbarten Schaltungen auf einem Wafer mit in die Klassifikation ein. Yan und Singh nehmen in [138] an, dass die Ausgänge einer Schaltung innerhalb eines gewissen Intervalls stabil werden, die Größe dieses Intervalls modelliert den Einfluss der Prozessvariationen. Durch FAST kann der konkrete Stabilisierungszeitpunkt eines Ausgangs für ein Testmuster bestimmt werden, indem das Testmuster wiederholt bei steigender Frequenz angelegt wird. Da benachbarte Schaltungen auf einem Wafer in ihren Prozessvariationen korreliert sind [109], können die Unsicherheitsintervalle, in denen ein Ausgang stabil werden kann, verkürzt werden. Hat ein Ausgang eine Schaltzeit außerhalb des erwarteten Intervalls, kann ein Defekt in der Schaltung angenommen werden. Daasch und Madge haben ein ähnliches Verfahren zur Varianzreduktion bei Ruhestrommessungen entwickelt, welches sich potentiell ebenfalls für FAST modifizieren lässt [18]. Hierbei wird aus Messungen benachbarter Gutschaltungen eine erwartete Varianz des Ruhestroms geschätzt. Diese kann dann mit der tatsächlichen Varianz verglichen werden. Die so erhaltene Fehlervarianz ist deutlich geringer als die gemessene Varianz und kann besser zur Klassifikation der Schaltung (gut oder defekt) verwendet werden.

Yilmaz und Chakrabarty haben die sogenannte Ausgangsabweichung als Metrik vorgestellt [140, 139, 128]. Basierend auf der Wahrscheinlichkeit, dass ein Gatter nicht rechtzeitig innerhalb eines bestimmten Zeitraums eine Signalflanke propagiert, kann die Wahrscheinlichkeit für den gesamten Pfad als Ausgangsabweichung über statistische Methoden berechnet werden. Mit dieser Metrik können besonders effektive Testmuster aus einem großen Mustersatz ausgewählt werden, um die Testdauer zu minimieren, während gleichzeitig die Testqualität erhalten bleibt. Die ursprüngliche Variante aus [139] wurde von Wang et al. in [126] erweitert, um Hazards zu berücksichtigen.

Ein anderer Ansatz besteht darin, nicht nur die Testfrequenz zu variieren, wie es beim reinen FAST der Fall ist. Qian et al. schlagen in [76] vor, zusätzlich die Versorgungsspannung sukzessive abzusenken. Wie bei [138] wird FAST verwendet, um die Stabilisierungszeit der Ausgänge genau zu bestimmen. Die Beobachtung hinter diesem Ansatz ist, dass bestimmte Defekte, wie z. B. eine ohmsche Brücke zwischen Gate und Source in Transistoren, sich bei Veränderung der Versorgungsspannung anders verhalten als (gutmütige) Prozessvariationen. Senkt man die Versorgungsspannung, so werden die durch diese Defekte verursachten Anomalien sichtbar.

Eine weitere Alternative besteht darin, rein über statistische Überlegungen abzuschätzen, welche Pfade zum gegebenen Beobachtungszeitpunkten besonders wahrscheinlich auch im fehlerfreien Fall einen Fehlschlag produzieren. Dies kann beispielsweise durch geschicktes Ausnutzen der statistischen Maximumoperation geschehen, wie in den Arbeiten von Wagner und Wunderlich [124, 123]. Die dort entwickelten Konzepte beziehen sich zwar auf die nominelle Taktperiode zur statistischen Bestimmung der kritischen Pfade, können aber leicht für beliebige Beobachtungszeitpunkte angepasst werden. Durch diese Berechnungen lassen sich aufwändige Monte Carlo Simulationen vermeiden.

Wird FAST zum Erkennen von ELFs eingesetzt, so können Prozessvariationen berücksichtigt werden, indem periodisch die Tests wiederholt werden, wie Kim et al. vorschlagen [39]. Ein Defekt, welcher zu einem ELF degradiert, wird sich frühzeitig stark in seiner induzierten zusätzlichen Verzögerung verändern, im Gegensatz zu den klassischen Alterungseffekten, welche erst über Jahre hinweg größere Auswirkungen aufweisen. Durch periodische Tests können demnach ELFs erkannt werden, wenn sich die Verzögerungszeiten stark verschieben – es wird also vor allem auf starke Änderung der Verzögerungszeiten geprüft.

3.7.4 Zusammenfassung

Der Hochgeschwindigkeitstest ist eine effektive Methode zum Erkennen von versteckten Verzögerungsfehlern. Durch die hohen Testfrequenzen lassen sich selbst kleinere Abweichungen der Schaltung von der Spezifikation sichtbar machen. In

diesem Abschnitt wurden drei der wesentlichen Herausforderungen bei der Umsetzung eines Hochgeschwindigkeitstests beschrieben: Erzeugen der Testfrequenz, höhere Schaltaktivität beim Test, sowie Auswirkungen der Prozessvariationen. Für diese Herausforderungen gibt es existierende Lösungen in der Literatur, die hier kurz vorgestellt wurden. Damit sind wesentliche Voraussetzungen zum Einsatz von FAST beim Herstellungstest gegeben. Dieser alleine kann aber nicht garantieren, dass alle Defekte erkannt werden, die zu einem späteren ELF führen können. Aus diesem Grunde ist es wichtig, auch zu Beginn des Produkteinsatzes im Feld regelmäßig die Schaltung mit FAST zu überprüfen. Der deshalb benötigte eingebaute Hochgeschwindigkeitstest erfordert aber die Lösung weiterer Herausforderungen, welche in der Literatur bisher nur geringfügig behandelt wurden. Die folgenden Kapitel werden daher zwei weitere Herausfordungen detailliert beschreiben und Lösungsansätze vorstellen: Die Frequenzauswahl für den Hochgeschwindigkeitstest und eine Methode zum prüfgerechten Entwurf zur Unterstützung der Testdatenauswertung im Selbsttest.

4 Frequenzauswahl für den eingebauten Hochgeschwindigkeitstest

Nachdem in den Kapiteln 2 und 3 die Grundlagen bzw. der Stand der Technik des Tests auf kleine Verzögerungsfehler (SDFs) vorgestellt wurden, stellt dieses Kapitel den ersten von zwei Teilen des eigenen Beitrages dieser Arbeit dar. In Kapitel 3 wurden bereits das Prinzip sowie einige Herausforderungen des Hochgeschwindigkeitstests (FAST) vorgestellt. Die bisherigen Lösungsansätze reichen aber noch nicht aus, wenn FAST als Selbsttest realisiert werden soll. Unter anderem ist unklar, welche Testfrequenzen für den Selbsttest gewählt werden sollen. Dieses Kapitel stellt ein Verfahren zur Berechnung einer minimalen Menge an Testfrequenzen vor, sodass alle durch einen gegebenen Testsatz beobachtbaren Fehler erkannt werden können. Weiterhin wird eine Fehlerpartitionierung mit den gewählten Frequenzen durchgeführt. Dies dient dazu, den gegebenen Testsatz in kleinere Teilsätze aufzuteilen, sodass jeder Frequenz ein möglichst kleiner Testsatz an besonders effektiven Testmustern zugeordnet wird. Dadurch kann signifikant die Testdauer verringert werden, was letztendlich die Testkosten senkt.

Dieses Kapitel gliedert sich wie folgt. Abschnitt 4.1 zeigt zunächst die Anforderungen an die Frequenzauswahl auf. Bisherige Ansätze zur Lösung werden in Abschnitt 4.2 verglichen, Abschnitt 4.3 formalisiert dann das Problem, um es auf ein bekanntes Optimierungsproblem zurückzuführen. Lösungsansätze zur Frequenzauswahl werden dann in den Abschnitten 4.4 (Heuristiken) und 4.5 (optimale Lösung) vorgestellt. Ein besonderer Lösungsansatz arbeitet mit einem zweistufigen Verfahren, um vorgegebene Frequenzen der optimalen Lösung zu

kombinieren und dadurch die Laufzeit zu verringern (Abschnitt 4.6). Die Fehlerpartitionierung und Testmusterauswahl werden dann in Abschnitt 4.7 beschrieben. Ergebnisse einer Simulationsstudie werden in Abschnitt 4.8 präsentiert, um die in diesem Kapitel beschriebenen Verfahren quantitativ bewerten zu können. Abschnitt 4.9 schließt dann das Kapitel ab und gibt einen Ausblick auf weitere Forschungsmöglichkeiten auf diesem Gebiet.

4.1 Anforderungen an die Frequenzauswahl

Für einen Selbsttest gibt es eine Reihe von Anforderungen, die für eine praktikable Frequenzauswahl erfüllt werden müssen:

1. Die gewählten Frequenzen müssen vom Taktgenerator erzeugbar sein,

2. die Anzahl der Frequenzen sollte gering sein und

3. alle potentiell erkennbaren Fehler sollten unter den gewählten Frequenzen beobachtbar sein.

Die erste Anforderung ist selbsterklärend. Die Anzahl der Frequenzen sollte minimiert werden, da man beispielsweise bei PLLs eine Weile warten muss, bis die Zielfrequenz erzeugt wird, häufiges Wechseln der Frequenz sollte daher vermieden werden. Aber auch bei anderen Generatoren wie [112, 72] ist eine geringe Zahl von Frequenzen vorteilhaft, da sie die benötigten Pufferstufen vereinfacht und insgesamt die Hardwarekosten des Generators senkt. Wird die letzte Anforderung nicht erfüllt, so verringert sich die Fehlereffizienz des gegebenen Testsatzes, da nicht mehr alle erkennbaren Fehler erkannt werden. Das kann durch Bestimmung zusätzlicher Testmuster kompensiert werden, dies erhöht jedoch die Testkosten, denn die Testdauer steigt durch die weiteren Testmuster an. Deshalb sollte immer versucht werden, so viele Fehler wie möglich mit den gegebenen Testmustern zu erkennen. Das Erfüllen aller Anforderungen gleichzeitig ist jedoch nicht immer möglich. Beispielsweise kann ein Fehler mit dem verfügbaren Testsatz nur erkennbar sein, wenn eine bestimmte Frequenz gewählt wird, welche nicht vom

Taktgenerator erzeugt werden kann. In einem solchen Fall müssen entweder ein anderer Taktgenerator oder zusätzliche Testmuster verwendet werden, oder es muss auf Fehlerabdeckung verzichtet werden. Dadurch wird bereits klar, dass die Auswahl an Testfrequenzen grundsätzlich – nicht nur für den Selbsttest – bereits ein komplexes Problem ist, welches sich nicht trivial lösen lässt.

4.2 Lösungsansätze

In der Literatur gibt es im Wesentlichen zwei gegenläufige Ansätze zur Lösung dieses Problems:

1. Erzeugung bzw. Optimierung von Testsätzen für einen gegebenen Satz an Testfrequenzen (z. B. [3, 141]),

2. Optimierung von Testfrequenzen für einen gegebenen Testsatz (z. B. [55, 34, 157]).

Der erste Ansatz erfüllt die beiden ersten Anforderungen trivialerweise. Die dritte Anforderung kann durch Überlegungen aus dem Betriebsfrequenztest erreicht werden: Minimiere die Pufferzeit $t_{Puffer,Test}$ eines sensibilisierten Pfades, indem die Pfadlänge maximiert wird. Ansatz 1 ist damit vor allem ATPG-basiert und kann daher aufgrund der gleichen Argumentation wie in Abschnitt 3.6 nicht garantieren, dass alle Fehler mit s_{min} erkannt werden können.

Der zweite Ansatz ist geeignet, um die Fehlereffizienz zu maximieren, d. h. die dritte Anforderung wird erfüllt. Für diesen Ansatz wird das Verfahren von [152, 157, 156] in diesem Kapitel genauer beschrieben. Es abstrahiert die Frequenzauswahl zu einem klassischen Optimierungsproblem und stellt Heuristiken vor, zeigt einen Algorithmus zur Berechnung der optimalen Lösung und präsentiert eine Variante, mit der sich der optimale Algorithmus auch in realistischen Szenarien einsetzen lässt. Damit die Frequenzen auch tatsächlich erzeugt werden können, liefert das Verfahren keine exakten Frequenzwerte zurück, sondern kleine Intervalle, innerhalb derer die Fehlererkennung garantiert wird. Dadurch ist die

Wahrscheinlichkeit hoch, dass der Taktgenerator Frequenzen erzeugen kann, die innerhalb der geforderten Intervalle liegen.

4.3 Formalisierung des Problems

Im Folgenden wird mit Beobachtungszeitpunkten $t \in \mathbb{N}$ argumentiert,[1] es gelte $t_{nom} = \lceil 1/f_{nom} \rceil$ für den nominellen Beobachtungszeitpunkt. Um den Beschränkungen der Taktgeneratoren Rechnung zu tragen, wird in dieser Arbeit zudem eine maximale Frequenz f_{max} definiert, für welche ein entsprechender minimaler Beobachtungszeitpunkt $t_{min} = 1/f_{max}$ existiert. Für die Simulationsstudien in dieser Arbeit wird $f_{max} = 3 \cdot f_{nom}$ angenommen, je nach verfügbaren Taktgeneratoren kann f_{max} auch anders belegt werden.

Problem 4.1 (Optimale Frequenzauswahl)**.** *Gegeben seien ein Testsatz $\mathscr{P}$, der nominelle Beobachtungszeitpunkt t_{nom}, sowie eine Menge $\Phi(\mathscr{P}, t_{nom})$ von HDF unter $\mathscr{P}$ und t_{nom}. Zusätzlich sei t_{min} der minimale Beobachtungszeitpunkt. Bestimme eine minimale Menge T an Beobachtungszeitpunkten, sodass jedem Fehler $\varphi \in \Phi(\mathscr{P}, t_{nom})$ mindestens ein Beobachtungszeitpunkt $t \in T$, $t_{min} \leq t < t_{nom}$, zugeordnet wird, für den mindestens ein Testmuster aus $\mathscr{P}$ existiert, welches eine fehlerhafte Schaltungsantwort unter t provoziert.*

Problem 4.1 nimmt implizit an, dass der Taktgenerator in der Lage ist, beliebige Frequenzen zu erzeugen, denen Beobachtungszeitpunkte zwischen t_{min} und t_{nom} entsprechen. Dies widerspricht allerdings der ersten Anforderung an die Frequenzauswahl. Deshalb liefert die hier vorgestellte Lösung wie eingangs erwähnt Intervalle an möglichen Beobachtungszeitpunkte zurück. Für jedes dieser Intervalle kann die Fehlererkennung bei einer minimalen Fehlergröße von s_{min} garantiert werden. Die Lösung des Problems 4.1 erfordert diagnostische Informationen über den Testsatz $\mathscr{P}$, speziell die Erkennungsbereiche der HDF, welche zunächst bestimmt werden müssen. Die folgenden Definitionen und Beispiele setzen keinen minimalen Beobachtungszeitpunkt t_{min} voraus, können aber leicht entsprechend angepasst werden.

[1] t sei in Pikosekunden angegeben, was in dieser Arbeit als kleinstmögliche Auflösung angesehen wird.

Definition 4.1. *Seien P ein Testmuster, t_{nom} der nominelle Beobachtungszeitpunkt sowie φ ein HDF unter P und t_{nom}. Dann ist $t \in \mathbb{N}$ ein* erkennender Beobachtungszeitpunkt*, wenn es mindestens einen Schaltungsausgang gibt, welcher zum Zeitpunkt t einen abweichenden Wert zwischen fehlerfreien und fehlerhaften Fall aufweist. Die Menge $I(\varphi, P)$ an erkennenden Beobachtungszeitpunkten bezeichnet man als* Erkennungsbereich von φ unter P.

Der vollständige Erkennungsbereich $I(\varphi, \mathscr{P})$ von φ unter einem Testsatz $\mathscr{P}$ bestimmt sich aus den Erkennungsbereichen der einzelnen Testmuster:

$$I(\varphi, \mathscr{P}) = \bigcup_{P \in \mathscr{P}} I(\varphi, P). \tag{4.1}$$

Aus Gründen der Einfachheit wird im weiteren Verlauf der Begriff Erkennungsbereich sowohl für $I(\varphi, P)$ als auch für $I(\varphi, \mathscr{P})$ verwendet. In dieser Arbeit werden nur solche erkennenden Beobachtungszeitpunkte betrachtet, bei denen das entsprechende Signal des fehlerfreien Ausgangs bereits stabil ist. Abweichungen vor dem Stabilisierungszeitpunkt sind aufgrund der Prozessvariationen hier nicht zugelassen. Definition 4.1 verwendet zudem keinen minimalen Beobachtungszeitpunkt um die Erkennungsbereiche nach unten zu begrenzen. Dies ist für Analysezwecke vorteilhaft, für die Praxis lassen sich die Erkennungsbereiche aber einfach über ein t_{min} limitieren.

Beispiel 4.1. Betrachte eine Schaltung mit einem Ausgang, an welche ein Testmuster P angelegt wird. Dabei sei die Schaltung von einem SDF φ betroffen. Um den Erkennungsbereich zu bestimmen, müssen die Signalverläufe des fehlerfreien und fehlerhaften Falls verglichen werden. Abbildung 4.1 zeigt beispielhafte Signalverläufe für beide Fälle.

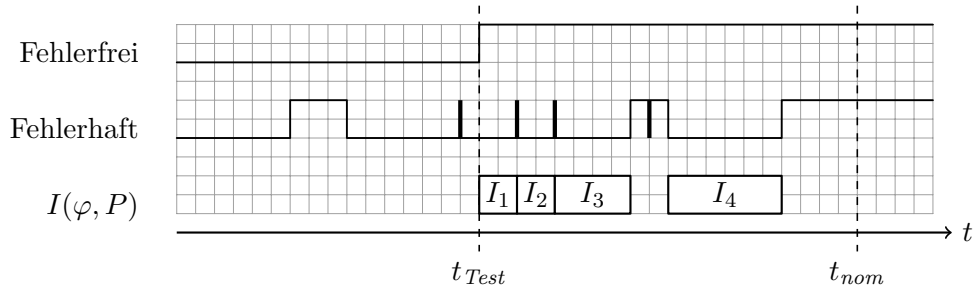


Abbildung 4.1: Bestimmung des Erkennungsbereiches für ein Testmuster

Der Erkennungsbereich bestimmt sich nun aus denjenigen Zeitbereichen zwischen t_{Test} und t_{nom}, in denen sich die Signale unterscheiden. In Abbildung 4.1 sind das die Intervalle I_1–I_4. Dabei gilt, dass kurze Signalpulse vom fehlerfreien zum fehlerhaften Wert nicht mit in die Bestimmung der Erkennungsbereiche einfließen. Andersherum trennen kurze Signalpulse vom fehlerhaften zum fehlerfreien Wert die Erkennungsbereiche auf, etwa bei I_2. Der Erkennungsbereich $I(\varphi, P)$ von φ unter P setzt sich nun aus den Teilintervallen zusammen, d. h. $I(\varphi, P) = I_1 \cup I_2 \cup I_3 \cup I_4$.

Jetzt wird ein Algorithmus zur Bestimmung des Erkennungsbereichs vorgestellt.

Definition 4.2. *Seien $\mathscr{P}$ ein Testsatz und O die Menge der Ausgänge einer Schaltung. Dann beschreibt die Funktion $x : \mathscr{P} \times O \times \mathbb{N} \to \{0, 1\}$ das* Ausgangssignal *eines Ausgangs $o \in O$ unter einem Testmuster $P \in \mathscr{P}$. Ein $t_s > 0$ mit $x(P, o, t_s - 1) \neq x(P, o, t_s)$ wird als* Schaltzeitpunkt von x *bezeichnet. Der Schaltzeitpunkt t_{Test} ist die* maximale Schaltzeit*, d. h. für alle $t \geq t_{Test}$ gilt $x(P, o, t) = x(P, o, t_{Test})$.*

Nachfolgend werden feste P und o betrachtet, aus Gründen der Einfachheit wird daher lediglich mit $x(t)$ argumentiert. Nun bezeichne $x_\varphi(t)$ das Ausgangssignal, wenn ein SDF φ angenommen wird. Zur Bestimmung des Erkennungsbereiches wird aus dem Signalen $x(t)$ und $x_\varphi(t)$ ein Differenzsignal

$$\tilde{x}(t) = x(t) \oplus x_\varphi(t) \tag{4.2}$$

gebildet. $\tilde{x}(t)$ hat den Wert 1 zu allen Zeitpunkten, an denen sich $x(t)$ und $x_\varphi(t)$ unterscheiden.

Beispiel 4.2. Abbildung 4.2 zeigt die beispielhafte Bildung von $\tilde{x}(t)$ aus den Signalen $x(t)$ und $x_\varphi(t)$ aus Beispiel 4.1. $\tilde{x}(t)$ enthält ebenfalls die Pulse aus $x_\varphi(t)$, diese müssen bei der Bestimmung der Erkennungsbereiche berücksichtigt werden.

Der Erkennungsbereich kann nun über folgende, zu Definition 4.1 äquivalente Definition bestimmt werden.

Definition 4.3. *Seien P ein Testmuster, o ein Schaltungsausgang, φ ein SDF, $x(P, o, t)$ das Ausgangssignal von o unter P, $x_\varphi(P, o, t)$ das fehlerhafte Ausgangsi-*

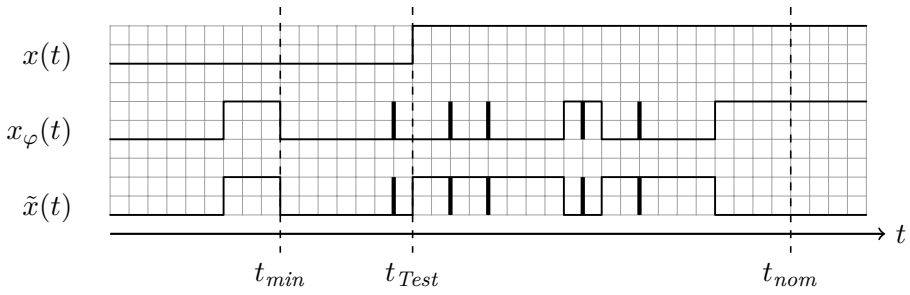


Abbildung 4.2: Bestimmung des Differenzsignals

gnal von o unter P und φ, sowie $\tilde{x}(P, o, t)$ das Differenzsignal aus $x(P, o, t)$ und $x_\varphi(P, o, t)$. Dann ist ein Zeitpunkt $t_B \in \mathbb{N}$ ein erkennender Beobachtungszeitpunkt, wenn $\tilde{x}(P, o, t_B) = 1$ gilt. Seien t_1 und t_2 zwei aufeinander folgende Schaltzeitpunkte von $\tilde{x}(P, o, t)$. Das Intervall $[t_1, t_2]$ ist ein Erkennungsbereich, wenn $\tilde{x}(P, o, t) = 1$ für alle $t_1 \leq t \leq t_2$ gilt.

Bei der Bestimmung der Erkennungsbereiche wird weiterhin eine *Filterkonstante* τ_f verwendet: Ein Intervall $[t_1, t_2]$ muss mindestens eine Größe $t_2 - t_1 \geq \tau_f$ besitzen, damit es zum Erkennungsbereich beitragen kann. τ_f sorgt dafür, dass sehr kurze Signalpulse nicht als Erkennungsbereich betrachtet (vgl. auch Beispiel 4.1) werden. Damit werden Randbedingungen der Flipflops wie Setup- und Haltezeiten oder Pulsfiltereffekte in der Logik berücksichtigt. Algorithmus 4.1 beschreibt das Finden dieser Intervalle in $\tilde{x}(t)$ als Pseudocode.

Algorithmus 4.1 Bestimme-Erkennungsbereich

Eingabe: $\tilde{x}(t)$, t_{Test}, τ_f
Ausgabe: I

```
1: I = ∅
2: t_1 = 0
3: für alle Schaltzeitpunkte t_s ≥ t_Test in x̃(t)  // Aufsteigend sortiert
4:     wenn x̃(t_s) = 1 dann
5:         t_1 = t_s
6:     sonst wenn x̃(t_s) = 0 und t_s − t_1 > τ_f dann
7:         I = I ∪ [t_1, t_s]
8: rückgabe I
```

Es werden alle Schaltzeitpunkte von $\tilde{x}(t)$, die nach t_{Test} folgen, in aufsteigender Reihenfolge durchlaufen. Wenn t_s Schaltzeitpunkt einer steigenden Flanke ist, dann beginnt dort ein neues Teilintervall des Erkennungsbereiches, in diesem Fall wird $t_1 = t_s$ als Startpunkt gespeichert. Andernfalls schließt t_s genau ein solches

Teilintervall ab. Bevor das Teilintervall zum Erkennungsbereich hinzugefügt werden kann, muss geprüft werden, ob dessen Länge die Filterkonstante τ_f überschreitet.

Mit Hilfe von Bestimme-Erkennungsbereich können nun in einer Simulationsschleife alle HDF unter t_{nom} und $\mathscr{P}$ aus einer SDF-Menge Φ sowie deren Erkennungsbereiche bestimmt werden. Algorithmus 4.2 zeigt den vereinfachten Pseudocode der Simulationsschleife.

Algorithmus 4.2 FAST-Simulation

Eingabe: Φ, $\mathscr{P}$, O, t_{nom}, τ_f
Ausgabe: $\Phi(\mathscr{P}, t_{nom})$

für alle $P \in \mathscr{P}$
  für alle $o \in O$
    $x(P, o, t) =$ GUTSIMULATION(P, o)
    $t_{Test} =$ maximale Schaltzeit von $x(P, o, t)$
    für alle $\varphi \in \Phi$
      $x_\varphi(P, o, t) =$ FEHLERSIMULATION(P, o, φ)
      $\tilde{x}(P, o, t) = x(P, o, t) \oplus x_\varphi(P, o, t)$
      wenn $\exists t \geq t_{nom} : \tilde{x}(P, o, t) = 1$ **dann**
        // φ kann mit Betriebsfrequenztest erkannt werden
        $\Phi = \Phi \setminus \{\varphi\}$
      sonst
        $I(\varphi, P) =$ BEST.-ERKENNUNGSBEREICH($\tilde{x}(P, o, t)$, t_{Test}, τ_f)
        $I(\varphi, \mathscr{P}) = I(\varphi, \mathscr{P}) \cup I(\varphi, P)$
rückgabe Φ

Der Algorithmus simuliert alle Testmuster zunächst in einer Gutsimulation, um das Referenzverhalten der fehlerfreien Schaltung als $x(t)$ für jeden Ausgang zu erhalten. Dann wird für jeden Fehler eine Fehlersimulation durchgeführt, um $x_\varphi(t)$ für die entsprechenden Ausgänge zu bestimmen. Wenn ein Fehler φ zu einem Beobachtungszeitpunkt $t \geq t_{nom}$ erkennbar ist ($\tilde{x}_\varphi(t) = 1$), dann ist φ kein HDF und kann entsprechend aus Φ entfernt werden. Andernfalls wird der Erkennungsbereich von φ mit den Informationen aus $\tilde{x}_\varphi(t)$ erweitert. Nach dem Aufruf von FAST-Simulation wurden alle SDF aus Φ entfernt, welche mit $\mathscr{P}$ und einem Betriebsfrequenztest erkannt werden können. Von daher ist die zurückgegebene Menge tatsächlich die Menge aller HDF unter $\mathscr{P}$ und t_{nom}. Hat ein Fehler φ einen leeren Erkennungsbereich $I(\varphi, \mathscr{P}) = \emptyset$, dann wurde er nicht von $\mathscr{P}$ aktiviert oder zu keinem der Schaltungsausgänge getrieben. Die mit dieser Methode be-

stimmten Erkennungsbereiche enthalten alle möglichen Beobachtungszeitpunkte, eine Begrenzung auf einen minimalen Beobachtungszeitpunkt t_{min} lässt sich aber sehr leicht zur Methode hinzufügen.

FAST-Simulation muss $\mathcal{O}(|\mathscr{P}| \cdot (|\Phi| + 1))$ Simulationsdurchläufe durchführen, ein effizientes Simulationsverfahren wird daher benötigt. Für die Experimente der vorliegenden Arbeit wurde deshalb ein in der Universität Stuttgart entwickelter, zeitlich genauer Simulator verwendet, welcher Grafikprozessoren (engl. Graphics Processing Units, GPUs) als Hardwarebeschleuniger verwendet [97, 96, 98]. GPUs eignen sich sehr gut zur kombinatorischen Schaltungssimulation, da sie den inhärenten Gatterparallelismus ausnutzen können (viele Gatter einer Schaltung können unabhängig voneinander berechnet werden). Der Simulator arbeitet nicht mit einzelnen Logikwerten, sondern mit Signalverläufen, welche die Zeitpunkte der Signalflanken abspeichern. Dadurch können auch komplexe Signale zeitlich genau abgebildet und die $x(t)$ bzw. $x_\varphi(t)$ direkt aus dem Simulator gewonnen werden. Moderne GPUs bieten zudem genügend Speicher, um musterparallele Simulation durchzuführen. Außerdem wurde der Simulator in [98] um fehlerparallele Simulation erweitert. Sie erlaubt es, topologisch unabhängige Fehler gleichzeitig zu simulieren. All diese Erweiterungen können den benötigten Zeitaufwand erheblich reduzieren, was die Bestimmung der Erkennungsbereiche in annehmbarer Zeit durchführbar werden lässt.[2] Mit den Erkennungsbereichen kann Problem 4.1 zu einem äquivalenten Problem umformuliert werden:

Problem 4.2 (Optimale Frequenzauswahl)**.** *Gegeben seien ein Testsatz $\mathscr{P}$, die nominellen und minimalen Beobachtungszeitpunkte t_{nom} bzw. t_{min} sowie eine Menge an HDF $\Phi(\mathscr{P}, t_{nom})$ unter $\mathscr{P}$ und t_{nom}. Bestimme eine minimale Menge T an Beobachtungszeitpunkten, sodass $t_{min} \leq t < t_{nom}$ für alle $t \in T$ gilt und zusätzlich für alle $\varphi \in \Phi(\mathscr{P}, t_{nom})$ mit $I(\varphi, \mathscr{P}) \neq \emptyset$ ein $t \in T$ mit $t \in I(\varphi, \mathscr{P})$ existiert.*

Nun soll gezeigt werden, dass es sich bei Problem 4.1 bzw. Problem 4.2 um ein NP-vollständiges Optimierungsproblem handelt. Dazu müssen zwei Schritte durchgeführt werden [23]:

[2] Mit dem GPU-basierten Simulator dauerte beispielsweise die Simulation aller Fehler und Testmuster bei einer kleineren Schaltung ca. 45 Minuten, mit einer rein CPU-basierten Referenzimplementierung lag die Ausführungszeit bei etwa 1 Tag.

1. Zeige, dass die optimale Frequenzauswahl ein Problem in der Komplexitätsklasse NP ist und

2. gegeben ein NP-vollständiges Problem Ψ, zeige, dass Ψ in polynomiell vielen Schritten auf die optimale Frequenzauswahl reduziert werden kann.

Als NP-vollständiges Ausgangsproblem wird das Minimum Hitting Set Problem betrachtet [37]:

Problem 4.3 (Minimum Hitting Set)**.** *Gegeben seien eine endliche Menge U und eine endliche Menge $\mathscr{S}$ an Teilmengen aus U, sodass $\bigcup_{S\in\mathscr{S}} S = U$. Finde eine Teilmenge $M \subseteq U$ minimaler Größe, sodass $M \cap S \neq \emptyset$ für alle $S \in \mathscr{S}$.*

Eine Schwierigkeit bei der Beweisführung liegt nun darin, dass Problem 4.2 ein kontinuierliches Problem ist (die Erkennungsbereiche sind kontinuierlich in den Beobachtungszeitpunkten), das Minimum Hitting Set Problem aber diskrete Mengen verwendet. Das Problem der optimalen Frequenzauswahl muss also zunächst diskretisiert werden.

Definition 4.4. *Sei $\mathscr{I}$ eine Menge an Intervallen. Ein Intervall I ist ein* atomares Intervall*, wenn es aus der Schnittmenge der Intervalle in $\mathscr{I}$ bestimmt werden kann und dabei minimal ist, d. h. es kann kein anderes Intervall $J \neq I$ geben, für welches $J \cap I \subset I$ gilt.*

Beispiel 4.3. Abbildung 4.3 zeigt zwei beispielhafte Erkennungsbereiche $I(\varphi_1, \mathscr{P})$ und $I(\varphi_2, \mathscr{P})$.

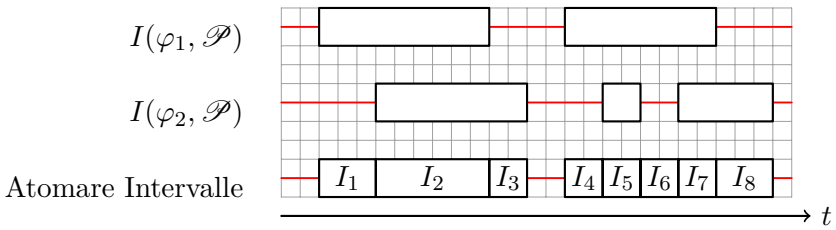


Abbildung 4.3: Bestimmung atomarer Intervalle

Die atomaren Intervalle bestimmen sich aus den Schnitten der Erkennungsbereiche. Zu beachten ist, dass die atomaren Intervalle sich „berühren“ können (wie in

der Abbildung gezeigt), aber aufgrund der Minimalitätsanforderung dennoch als getrennte Intervalle betrachtet werden müssen.

Die Bedeutung der atomaren Intervalle ist wie folgt: Ein Beobachtungszeitpunkt kann beliebig innerhalb eines atomaren Intervalles gewählt werden, die Menge an erkannten HDF ändert sich dabei nicht, da stets dieselben Erkennungsbereiche abgedeckt werden. Dies entspricht damit effektiv der Diskretisierung der Menge an Beobachtungszeitpunkten. Wie zu Beginn des Kapitels erwähnt, werden dadurch zusätzliche Freiheitsgrade für den Taktgenerator geschaffen. Insbesondere ermöglichen die atomaren Intervalle auch eine gewisse Toleranz gegenüber Prozessvariationen und Ungenauigkeiten des Taktgenerators, abhängig von der Größe des atomaren Intervalls. Damit sind alle Voraussetzungen geschaffen, um die NP-Vollständigkeit zu beweisen.[3] Der Beweis wurde bereits in [156] geführt und wird hier nochmals angegeben.

Theorem 4.1. *Die optimale Frequenzauswahl ist ein NP-vollständiges Problem.*

Beweis. Ein Problem ist in der Komplexitätsklasse NP, wenn eine gegebene Lösung in polynomiell vielen Schritten überprüft werden kann. Ob eine Menge T an Beobachtungszeitpunkten eine Lösung der optimalen Frequenzauswahl ist, lässt sich wie folgt überprüfen: Für alle Fehler $\varphi \in \Phi(\mathscr{P}, t_{nom})$, prüfe ob es ein t in T gibt, sodass $t \in I(\varphi, \mathscr{P})$. Ist dies nicht der Fall, so kann T keine Lösung des Problems sein (es werden nicht alle Fehler abgedeckt). Andernfalls fahre mit dem nächsten Fehler fort. Dies benötigt $\mathcal{O}(|T| \cdot |\Phi(\mathscr{P}, t_{nom})|)$ viele Schritte, ist also linear in der Problemgröße $|\Phi(\mathscr{P}, t_{nom})|$.

Betrachte nun eine Instanz des Minimum Hitting Set Problems, bestehend aus einer Menge $U = \{u_1, \ldots, u_k\}$ und eine endliche Menge $\mathscr{S}$ an Teilmengen aus U. Für jedes $u_i \in U$ definiere ein Intervall $I(u_i) = [i, i+1]$. Beachte, dass die Menge $\mathscr{I} = \{I(u_1), \ldots, I(u_m)\}$ immer noch eine diskrete Menge ist. Nun definiere für jede Menge $S \in \mathscr{S}$ einen Bereich $I(S)$ über

[3] Streng genommen ist der Begriff „NP-vollständig“ für Entscheidungsprobleme definiert. Für eine übersichtlichere Darstellung wird hier allerdings nicht zwischen Entscheidungs- und Optimierungsproblem unterschieden.

$$I(S) = \bigcup_{u_i \in S} I(u_i).$$

Dadurch entspricht ein $I(u_i)$ einem atomaren Intervall und $I(S)$ einem Erkennungsbereich. Löst man das Problem der optimalen Frequenzauswahl für die so definierten Erkennungsbereiche, dann erhält man eine Menge T an Beobachtungszeitpunkten, sodass für alle $t \in T$ ein $I(u_i)$ mit $t \in I(u_i)$ existiert. Zudem gilt $I(S) \cap T \neq \emptyset$ für alle $S \in \mathscr{S}$. Dadurch löst man aber gleichzeitig das Minimum Hitting Set Problem für U, wodurch gezeigt wurde, dass dieses Problem sich auf die optimale Frequenzauswahl reduzieren lässt. Die Rückführung geschieht in polynomiell vielen Schritten, denn für jedes Element u_i aus U muss lediglich ein Intervall $I(u_i)$ und für jede Menge $S \in \mathscr{S}$ muss ein Bereich $I(S)$ definiert werden. Damit ist die optimale Frequenzauswahl ein NP-vollständiges Problem. □

4.4 Heuristische Lösungsverfahren

Optimale Lösungen für NP-vollständige Problem lassen sich oft nur mit großem Rechenaufwand bestimmen. Daher ist es in vielen Fällen sinnvoll, eine Heuristik zu verwenden, um nahezu optimale Ergebnisse in vertretbarer Zeit zu erhalten. Eine simple Heuristik zur Lösung der optimalen Frequenzauswahl ist die gierige Auswahl: Wähle in jedem Schritt dasjenige atomare Intervall, welches die meisten noch nicht abgedeckten Fehler abdeckt. Fahre fort, bis alle Fehler abgedeckt sind.

Eine weitere Heuristik zur Bestimmung einer Lösung des Problems 4.2 besteht darin, stets die kleinste obere bzw. größte untere Intervallgrenze aller verbleibenden Erkennungsbereiche zu wählen. Hierfür müssen nicht einmal die atomaren Intervalle bestimmt werden. Algorithmus 4.3 zeigt den Pseudocode des Verfahrens für die kleinste obere Intervallgrenze (engl. Least Upper Bound, LUB), der Algorithmus für die größte untere Intervallgrenze (engl. Greatest Lower Bound, GLB) kann analog bestimmt werden. Die Menge L enthält alle noch nicht abgedeckten Fehler aus $\Phi(\mathscr{P}, t_{nom})$. Solange noch Fehler in L verbleiben, wird als nächster Beobachtungszeitpunkt das Minimum der oberen Intervallgrenzen

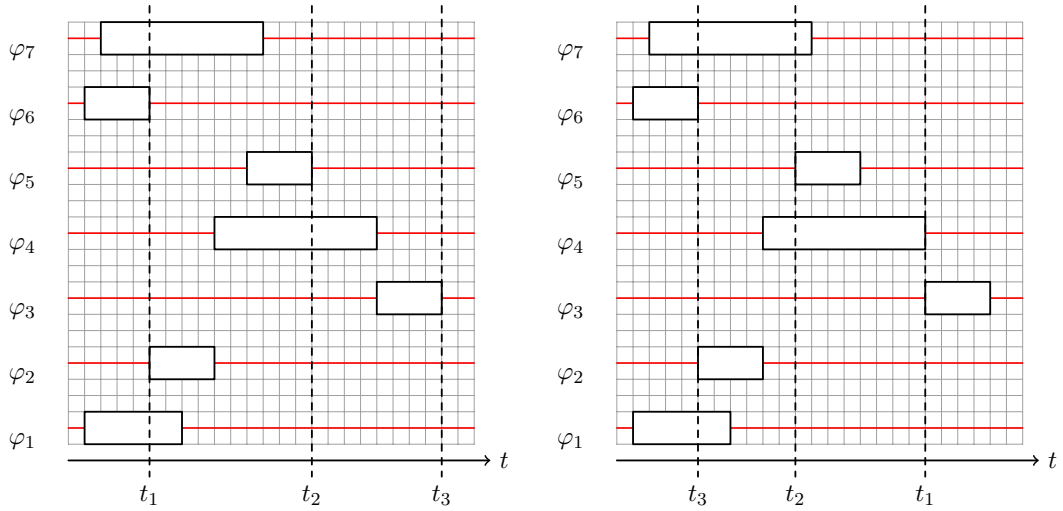
Algorithmus 4.3 MHS-LUB

Eingabe: $\Phi(\mathscr{P}, t_{nom})$
Ausgabe: T

```
1: T = ∅
2: L = Φ(𝒫, t_nom)
3: solange L ≠ ∅
4:     t = min_{φ∈L} {max{I(φ, 𝒫)}} // MHS-GLB: max_{φ∈L} {min{I(φ, 𝒫)}}
5:     für alle φ ∈ L
6:         wenn t ∈ I(φ, 𝒫) dann
7:             L = L \ {φ}
8:     T = T ∪ {t}
9: rückgabe T
```

aller verbleibenden Erkennungsbereiche gewählt. Dann werden alle durch diesen Beobachtungszeitpunkt erkennbaren Fehler aus L entfernt, und der nächste Beobachtungszeitpunkt wird ausgewählt.

Beispiel 4.4. Das folgende Beispiel ist direkt aus [152] entnommen, um MHS-LUB mit MHS-GLB (Auswahl der größten unteren Intervallgrenzen) zu vergleichen. Gegeben seien die in Abbildung 4.4 gegebenen Fehler mit den eingezeichneten Erkennungsbereichen.



(a) Ergebnis für MHS-LUB (b) Ergebnis für MHS-GLB

Abbildung 4.4: Beispiel für die Frequenzauswahl [152]

MHS-LUB (dargestellt in Abbildung 4.4a) geht wie folgt vor: Die kleinste obere Intervallgrenze aller Erkennungsbereiche ist die obere Grenze für Fehler φ_6 und

wird als t_1 gespeichert. Über t_1 lassen sich folgende Fehler erkennen: φ_1, φ_2, φ_6 und φ_7. Diese Fehler werden entsprechend markiert und spielen bei der Bestimmung des nächsten Beobachtungszeitpunktes keine Rolle mehr. Unter den verbleibenden Fehlern (φ_3–φ_5) hat φ_5 wiederum die kleinste obere Intervallgrenze, sodass diese als t_2 bestimmt wird. Mit t_2 lassen sich φ_4 und φ_5 abdecken, sodass letztendlich φ_3 verbleibt. Dessen obere Intervallgrenze wird als t_3 gewählt und alle Fehler sind abgedeckt, woraufhin der Algorithmus endet. Abbildung 4.4b zeigt das Ergebnis, wenn MHS-GLB ausgeführt wird. Da dieser Algorithmus analog zu MHS-LUB verfährt, wird hier nicht im Detail der Ablauf beschrieben. Es ist allerdings erwähnenswert, dass MHS-GLB kleinere Beobachtungszeitpunkte (und damit höhere Frequenzen) bevorzugt, wie in der Abbildung ersichtlich wird.

Sowohl MHS-GLB als auch MHS-LUB können sehr schnell eine Lösung berechnen. Ein signifikanter Nachteil des Verfahrens ist allerdings, dass von einigen Fehlern offensichtlich die Grenze der Erkennungsbereiche gewählt wird. Dadurch ist dieses Verfahren nicht besonders robust gegenüber Variationen in der zu testenden Schaltung oder Ungenauigkeiten im Taktgenerator. Kleinste Abweichungen der Erkennungsbereiche bzw. Beobachtungszeitpunkte können bereits dazu führen, dass bestimmte Fehler nicht mehr erkennbar sind. Zudem liefern die Algorithmen echte Zeitpunkte und keine (wie anfangs erwähnt) Intervalle zurück. Durch diese Nachteile sind die Verfahren vorrangig aus theoretischer Sichtweise interessant.

4.5 Ein optimales Verfahren

Im vorangehenden Abschnitt wurde erwähnt, dass eine optimale Lösung eines NP-vollständigen Problems sich oft nur mit großem Rechenaufwand bestimmen lässt. Nichtsdestotrotz soll an dieser Stelle ein optimaler Algorithmus beschrieben werden. Für sich genommen ist das hier beschriebene Verfahren möglicherweise nicht effizient, da aufgrund der exponentiellen Laufzeit sehr große Fehlermengen zu langer Rechendauer führen können. In Abschnitt 4.6 wird allerdings ein zweistufiges Verfahren beschrieben, bei dem in einem ersten Schritt die Problemgröße deutlich reduziert wird. Dadurch wird auch eine optimale Lösung des Problems interessant. Der hier vorgestellte Algorithmus wurde von Shi und Cai in [102] be-

schrieben und in [157] für Problem 4.2 implementiert. Die zentrale Datenstruktur, welche hierbei verwendet wird, ist der Hypergraph.

Definition 4.5. *Ein* Hypergraph $H = (V, \mathscr{E})$ *ist ein Tupel bestehend aus einer Menge* V *an Knoten und einer Menge* $\mathscr{E} \subseteq \mathrm{Pot}(V)$ *an Hyperkanten.*

Im Gegensatz zu einer Kante in einem „klassischen“ Graphen kann eine Hyperkante $K \in \mathscr{E}$ beliebig viele Knoten aus V verbinden. Die Kardinalität von K wird auch als Grad der Hyperkante bezeichnet. Der Grad eines Knoten $v \in V$ ist die Anzahl der Hyperkanten, die v enthalten und wird mit $\deg(v)$ gekennzeichnet. Mit einem Hypergraphen lässt sich das Problem 4.3 umformulieren zu folgendem äquivalenten Problem.

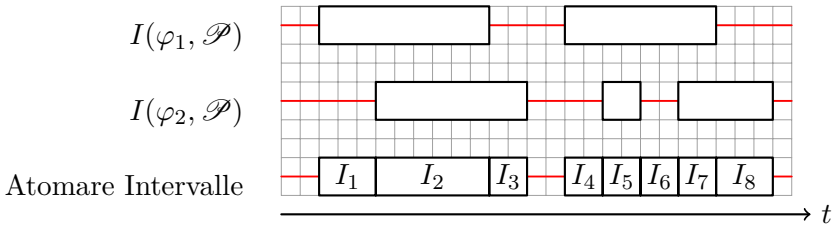
Problem 4.4 (Hypergraph Minimum Hitting Set)**.** *Gegeben sei ein Hypergraph* $H = (V, \mathscr{E})$*, sodass jeder Knoten* $v \in V$ *mit mindestens einer Hyperkante* $K \in \mathscr{E}$ *verbunden ist. Bestimme eine minimale Teilmenge* $M \subseteq V$*, sodass* $M \cap K \neq \emptyset$ *für alle* $K \in \mathscr{E}$*.*

4.5.1 Erzeugen des Hypergraphen

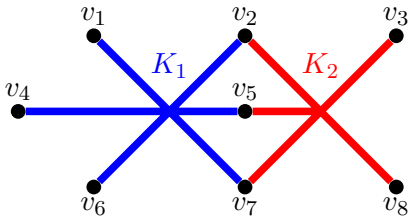
Der Hypergraph H wird wie folgt aufgebaut: Bestimme als erstes die atomaren Intervalle für die gegebenen Erkennungsbereiche. Dann erzeuge einen Knoten v für jedes atomare Intervall und lege ihn in V ab. Abschließend erzeuge eine Hyperkante K für jeden Erkennungsbereich und füge zu K alle Knoten $v \in V$ hinzu, wenn deren entsprechendes atomares Intervall Teil des durch K beschriebenen Erkennungsbereiches ist.

Beispiel 4.5. Es soll der Hypergraph für die Erkennungsbereiche aus Abbildung 4.3 (Beispiel 4.3) erzeugt werden. Abbildung 4.5 zeigt zunächst in Abbildung 4.5a erneut die atomaren Intervalle, sie sind bereits mit I_1–I_8 gekennzeichnet. Für jedes atomare Intervall I_i wird ein Knoten v_i erzeugt, sodass $V = \{v_1, \ldots, v_8\}$. Für jeden Erkennungsbereich wird eine Hyperkante erzeugt und mit den zugehörigen Knoten verbunden. Es ergeben sich somit die zwei Hyper-

kanten $K_1 = \{v_1, v_2, v_4, v_5, v_6, v_7\}$ und $K_2 = \{v_2, v_3, v_5, v_7, v_8\}$. Abbildung 4.5b zeigt den resultierenden Hypergraphen. K_1 ist blau, K_2 rot eingezeichnet.



(a) Atomare Intervalle (vgl. Beispiel 4.3)



(b) Resultierender Hypergraph

Abbildung 4.5: Erzeugen eines Hypergraphen aus atomaren Intervallen

4.5.2 Das Lösungsverfahren

Der Algorithmus arbeitet in zwei Stufen. In der ersten Stufe wird versucht, durch Anwenden verschiedener Reduktionsregeln H soweit zu reduzieren, dass der maximale Grad aller Knoten 2 ist. In der zweiten Stufe kann dann für einen solchen Hypergraphen eine Lösung direkt bestimmt werden. Um den Graphen zu reduzieren, werden vier verschiedene Schritte angewandt.

Essentieller Knoten Existiert eine Hyperkante K vom Grad 1, so ist der einzige mit K verbundene Knoten v ein essentieller Knoten und muss zur Lösung hinzugefügt werden. Dann können alle mit v verbundenen Hyperkanten aus H entfernt werden.

Knotendominanz Existieren zwei Knoten v_1 und v_2, sodass die Menge aller mit v_1 verbundenen Kanten eine Obermenge der mit v_2 verbundenen Kanten ist, dann wird v_2 von v_1 dominiert und kann aus H entfernt werden.

Kantendominanz Existieren zwei Kanten K_1 und K_2, sodass $K_1 \subseteq K_2$, dann wird K_1 von K_2 dominiert. K_2 kann aus H entfernt werden.

Die vierte Regel, als Nachbarschaftsregel bezeichnet, ist eine Neuerung von [102] und soll an dieser Stelle ausführlicher beschrieben werden.

Definition 4.6. *Sei $H = (V, \mathscr{E})$ ein Hypergraph und $v \in V$ ein Knoten. Dann ist die* 2-Nachbarschaft von v *definiert als* $B_2(v) = \{u \mid u \in V, \{u, v\} \in \mathscr{E}\}$.

Nachbarschaftsregel Gegeben ein Knoten v, sei m die Anzahl aller Hyperkanten, welche mit Knoten aus der 2-Nachbarschaft von v, aber nicht mit v selbst verbunden sind. Gilt $m < |B_2(v)|$, dann entferne alle mit v verbundenen Hyperkanten aus H und füge v zur Lösung hinzu.

Beispiel 4.6. Gegeben sei der Hypergraph $H = (V, \mathscr{E})$ aus Abbildung 4.6.

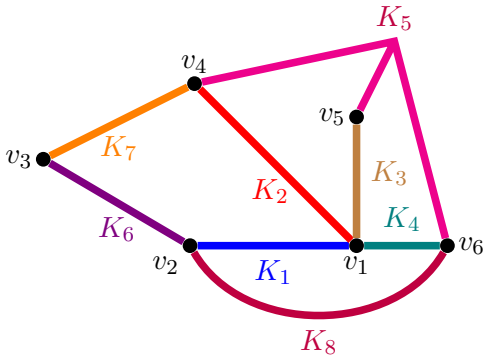


Abbildung 4.6: Ein Hypergraph

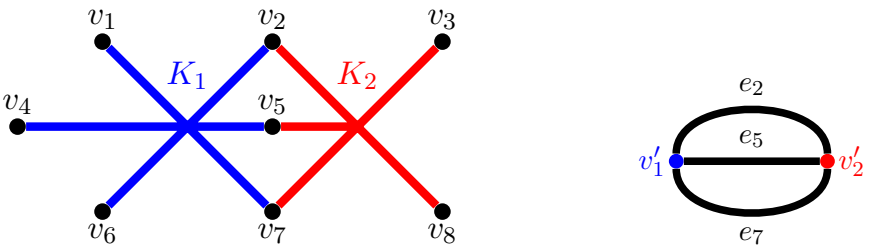
Die 2-Nachbarschaft des Knoten v_1 besteht aus allen Knoten, welche mit v_1 über eine Hyperkante mit Grad 2 verbunden sind, d. h. $B_2(v_1) = \{v_2, v_4, v_5, v_6\}$. Die Nachbarschaftsregel entscheidet nun darüber, ob v_1 zur Lösung hinzugefügt werden soll. Wählt man v_1 zur Lösung, so sind alle Hyperkanten, welche mit v_1 verbunden sind, abgedeckt. In diesem Beispiel sind das K_1, K_2, K_3 und K_4, eine Lösung bestände dann z. B. aus den Knoten v_1, v_3 und v_5. Andernfalls müssen alle Knoten aus der 2-Nachbarschaft in die Lösung aufgenommen werden, um die Kanten K_1–K_4 abzudecken. Dann allerdings werden noch zusätzliche Kanten abgedeckt, hier K_5, K_6, K_7. Eine Lösung bestände in diesem Fall aus den Knoten v_2, v_4, v_5 und v_6. Man sieht bereits, dass diese Lösung mehr Knoten benötigt, als

die Lösung mit v_1. Mathematisch betrachtet gilt $|B_2(v_1)| = 4$ und es werden, falls v_1 nicht zur Lösung gewählt wird, 3 weitere Kanten abgedeckt. Da $|B_2(v_1)| > 3$, wird v_1 zur Lösung gewählt.

Nun kann es passieren, dass nach Anwenden der vier Regeln in der ersten Phase ein Hypergraph verbleibt, dessen Knoten einen höheren Grad als 2 besitzen, d. h. die zweite Phase ist noch nicht anwendbar. In diesem Fall wird ein Entscheidungsbaum aufgebaut. Dazu wird der Knoten $v \in V$ mit maximalem Grad bestimmt und H wird in zwei Teilgraphen H_1 und H_2 aufgeteilt. Diese entsprechen dem resultierenden Hypergraphen, wenn v zur Lösung hinzugefügt wird (H_1) bzw. wenn v nicht zur Lösung hinzugefügt wird (H_2). Dann wird jeweils die Lösung für H_1 und H_2 berechnet, verglichen und die kleinere Lösung wird akzeptiert.

Ist der Hypergraph soweit reduziert, dass alle Knoten einen Grad von maximal 2 besitzen, so kann eine Lösung direkt bestimmt werden. Dazu wird der Hypergraph in einen klassischen Graphen $G = (V', E')$ konvertiert, indem aus jeder Kante $K \in \mathscr{E}$ ein Knoten $v' \in V'$ im klassischen Graphen geformt wird. Analog wird aus jedem Knoten $v \in V$ eine Kante $e' \in E'$ im klassischen Graphen gebildet, wenn v den Grad zwei hat.

Beispiel 4.7. Der Hypergraph aus Beispiel 4.5 hat einen maximalen Knotengrad von zwei. Abbildung 4.7 zeigt, wie aus dem Hypergraph ein klassischer Graph erzeugt wird.



(a) Der Hypergraph (b) Konvertierter klassischer Graph

Abbildung 4.7: Konvertierung des Hypergraphen

Abbildung 4.7a zeigt nochmals den zu konvertierenden Hypergraphen, Abbildung 4.7b das Ergebnis, wenn aus den Hyperkanten K_1 und K_2 zwei Knoten v'_1 bzw. v'_2, sowie aus den Knoten v_2, v_5 und v_7 die konvertierten Kanten e_2, e_5 und e_7

erzeugt wurden. Da diese Kanten zusammenfallen, ist es nicht notwendig, sie als eigene Kanten darzustellen, man kann auch beispielsweise lediglich e_2 betrachten.

Die Lösung des Problems 4.4 liegt nun darin, für den konvertierten Graphen $G = (V', E')$ eine minimale Kantenabdeckung zu finden.

Definition 4.7. *Sei $G = (V, E)$ ein Graph. Eine Teilmenge $C \subseteq E$ heißt* Kantenabdeckung*, wenn alle Knoten aus V mit mindestens einer Kante aus C verbunden sind. C ist eine* minimale Kantenabdeckung*, wenn C unter allen möglichen Kantenabdeckungen minimale Kardinalität besitzt.*

Eine Kantenabdeckung in G' entspricht im Hypergraphen H einer Menge an Knoten, sodass jede Hyperkante mit mindestens einem Knoten verbunden ist. Ist die Kantenabdeckung von G minimal, so ist auch die Menge an benötigten Knoten in H minimal, was genau der Lösung des Problems 4.4 entspricht. Eine minimale Kantenabdeckung lässt sich in polynomieller Zeit bestimmen [23], indem zuerst eine maximale Paarung bestimmt wird und diese dann (falls notwendig) mit einem gierigen Algorithmus zu einer Kantenabdeckung erweitert.

Definition 4.8. *Sei $G = (V, E)$ ein Graph. Eine Menge $M \subseteq E$ heißt* Paarung*, wenn keine zwei Kanten aus M mit dem gleichen Knoten verbunden sind. M ist eine* maximale Paarung*, wenn M unter allen möglichen Paarungen maximale Kardinalität besitzt. M ist eine* perfekte Paarung*, wenn alle Knoten aus V mit einer Kante aus M verbunden sind.*

Eine perfekte Paarung ist bereits eine minimale Kantenabdeckung, in diesem Fall ist keine Erweiterung mehr notwendig. Da aber nicht immer für einen beliebigen Graphen eine perfekte Paarung existiert, wird in dieser Arbeit nach einer maximalen Paarung gesucht und dann geprüft, ob diese ebenfalls eine perfekte Paarung ist. Nur wenn die Paarung nicht perfekt ist, wird diese mit einem gierigen Verfahren zur Kantenabdeckung erweitert. Es existieren deterministische Algorithmen, die in polynomieller Zeit eine maximale Paarung finden können, beispielsweise der Algorithmus von Edmonds [19]. Eine sehr gute Beschreibung des Algorithmus mit Hilfe von selbst erstellbaren Beispielen befindet sich im Internet [22].

Shi und Cai konnten in [102] zeigen, dass die Nachbarschaftsregel nicht die Optimalität des Algorithmus beeinflusst. Alle anderen Regeln haben ebenfalls keinen Einfluss darauf, der Algorithmus berechnet daher tatsächlich eine optimale Lösung. Algorithmus 4.4 zeigt nun den Pseudocode der optimalen Lösung des Problems 4.2. Dabei bezeichne $\mathscr{E}(v)$ die Menge aller Hyperkanten, welche mit dem Knoten v verbunden sind.

Algorithmus 4.4 MHS-Hypergraph

Eingabe: Hypergraph $H = (V, \mathscr{E})$

1: **wenn** $\max_{v \in V}\{\deg(v)\} = 2$ **dann**
2: Bestimme $G = (V', E')$ aus H
3: **rückgabe** Berechne Lösung über minimale Kantenabdeckung in G
4: **sonst wenn** $\exists v \in V, K \in \mathscr{E} : \ K = \{v\}$ **dann** // Essentieller Knoten
5: $H' = (V \setminus \{v\}, \{K \mid K \in \mathscr{E}, v \notin K\})$
6: **rückgabe** MHS-Hypergraph$(H') \cup \{v\}$
7: **sonst wenn** $\exists u, v \in V : \ \mathscr{E}(u) \subseteq \mathscr{E}(v)$ **dann** // Knotendominanz
8: $H' = (V \setminus \{u\}, \{K \backslash u \mid K \in \mathscr{E}\})$
9: **rückgabe** MHS-Hypergraph(H')
10: **sonst wenn** $\exists K_1, K_2 \in \mathscr{E} : \ K_1 \subseteq K_2$ **dann** // Kantendominanz
11: $H' = (V, \mathscr{E} \setminus \{K_2\})$
12: **rückgabe** MHS-Hypergraph(H')
13: **sonst wenn** $\exists v \in V : \ |B_2(v)| > |\{K \mid K \in \mathscr{E}, \ B_2(v) \cap K \neq \emptyset, \ v \notin K\}|$ **dann** // Nachbarschaftsregel
14: $H' = (V \setminus \{v\}, \{K \mid K \in \mathscr{E}, v \notin K\})$
15: **rückgabe** MHS-Hypergraph$(H') \cup \{v\}$
16: **sonst** // Erzeuge Entscheidungsbaum
17: Wähle Knoten $v \in V$ mit maximalem Grad
18: $L_1 =$ MHS-Hypergraph$((V \setminus \{v\}, \{K \mid K \in \mathscr{E}, v \notin K\})) \cup \{v\}$
19: $L_2 =$ MHS-Hypergraph$((V \setminus \{v\}, \{K \setminus \{v\} \mid K \in \mathscr{E}\}))$
20: **rückgabe** $\min\{L_1, L_2\}$

In jedem Aufruf wird zunächst überprüft, ob die Lösung über eine minimale Kantenabdeckung im konvertierten Graphen bestimmt werden kann. Ist dies nicht der Fall, so wird der Hypergraph sukzessive über die vier Regeln verkleinert (Zeilen 4–15). Erst wenn dies nicht mehr möglich ist, wird der Entscheidungsbaum aufgebaut (Zeilen 16–20). Die konkrete Implementierung von MHS-Hypergraph weicht ein wenig von Algorithmus 4.4 ab. Dort wird lediglich beim Aufbau des Entscheidungsbaumes eine Rekursion durchgeführt, die Minimierungsregeln werden iterativ überprüft. Dies ist notwendig, da die bei der Frequenzauswahl typischen großen Datenmengen (viele Erkennungsbereiche und daher viele atomare

Intervalle) schnell sehr große Rekursionstiefen verursachen können, was selbst moderne Rechner aufgrund des erforderlichen Kontextspeichers vor Probleme stellen kann.

4.6 Zweistufige optimierte Frequenzauswahl

Nach [102] liegt die Komplexität des Algorithmus bei $\mathcal{O}(1{,}238\,01^n)$, $n = |V| + |\mathscr{E}|$, ist also exponentiell in der Problemgröße. Wenn $\Phi(\mathscr{P}, t_{nom})$ sehr groß ist, kann die Laufzeit von MHS-Hypergraph stark ansteigen. Das ist tatsächlich häufig der Fall, größere Schaltungen können schnell Fehlermengen mit hundertausenden bis millionen Fehlern aufweisen. Um diese Komplexität zu verringern, wurde in [157] ein zweistufiges Verfahren entwickelt. Dieses nutzt aus, dass mit festen Beobachtungszeitpunkten zwar keine vollständige, aber dennoch oftmals bereits sehr hohe Fehlereffizienz erreicht werden kann. Dadurch kann die Problemgröße in der ersten Stufe stark verringert und die Berechnung einer optimalen Lösung auf der verbleibenden Fehlermenge in der zweiten Stufe in praktikabler Zeit durchgeführt werden. Zusätzlich wurden in [157] optimierte Testmuster für die Zielfehler der zweiten Stufe erzeugt, da diese besonders schwer zu erkennen sind. Der Ablauf des Verfahrens ist als Flussdiagramm in Abbildung 4.8 dargestellt.

$\mathscr{P}_{Initial}$ bezeichnet die Testmustermenge, mit welcher FAST-Simulation ursprünglich durchgeführt wird. Zunächst wird eine Menge $T_{Fix} = \{t_0, \ldots, t_{m-1}\}$ an festen Beobachtungszeitpunkten über folgende Gleichung bestimmt:

$$t_i = t_{min} + i \cdot \frac{t_{nom} - t_{min}}{m},\ 0 \leq i < m. \tag{4.3}$$

Mit T_{Fix} können die ersten HDF aus $\Phi(\mathscr{P}_{Initial}, t_{nom})$ erkannt werden, sodass eine kleinere Menge $\Phi(\mathscr{P}_{Initial}, T_{Fix}) \subset \Phi(\mathscr{P}_{Initial}, t_{nom})$ verbleibt. Für diese Menge kann direkt die optimale Lösung mittels MHS-Hypergraph bestimmt werden. Da sie aber vergleichsweise klein ist, bietet es sich an, zusätzliche, optimierte Testmuster für SDFs zu berechnen, um die Erkennungsbereiche dieser Fehler zu verbessern. Mit dem so erzeugten Testsatz $\mathscr{P}_{Extra}$ wird der finale

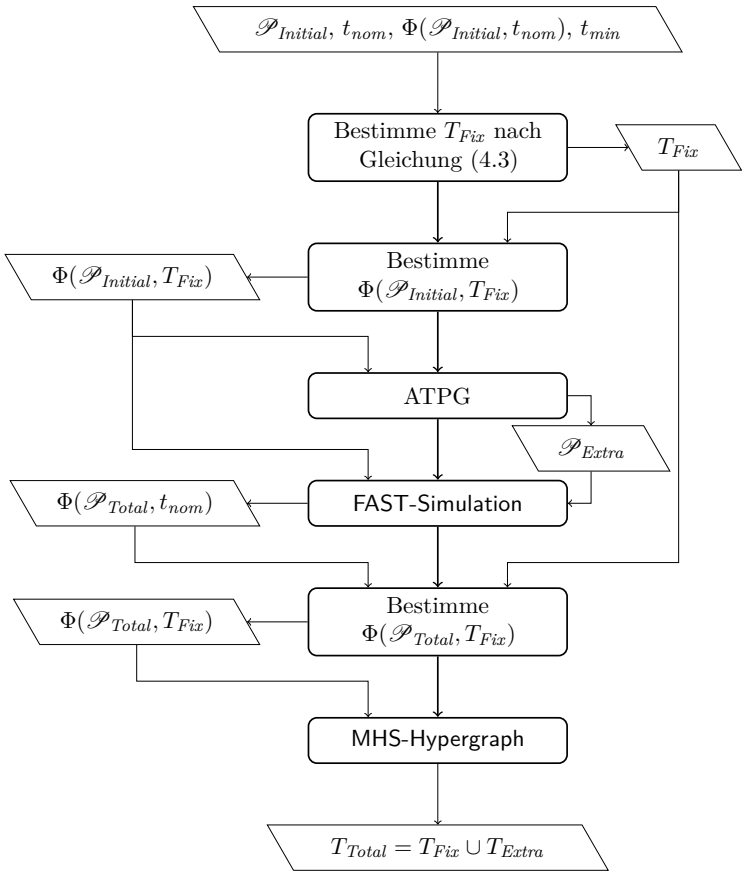


Abbildung 4.8: Ablauf der zweistufigen Frequenzauswahl

Testsatz $\mathscr{P}_{Total} = \mathscr{P}_{Initial} \cup \mathscr{P}_{Extra}$ gebildet. Danach wird FAST-Simulation mit $\mathscr{P}_{Extra}$ und $\Phi(\mathscr{P}_{Initial}, T_{Fix})$ durchgeführt. Als Ergebnis bestimmt sich die Menge $\Phi(\mathscr{P}_{Total}, t_{nom})$, aus welcher erneut die unter T_{Fix} erkennbaren HDF aussortiert werden. Für die verbleibende HDF-Menge $\Phi(\mathscr{P}_{Total}, T_{Fix})$ wird MHS-Hypergraph aufgerufen, welcher weitere Beobachtungszeitpunkte T_{Extra} erzeugt, sodass sich als finale Menge an Beobachtungszeitpunkten $T_{Total} = T_{Fix} \cup T_{Extra}$ ergibt.

Dieses Verfahren ist nicht optimal im Sinne des Problems 4.2, kann also in mehr als minimal benötigten Beobachtungszeitpunkten resultieren. Es ist aber dennoch attraktiv, nicht nur wegen der Laufzeitminimierung, sondern auch, da für die (typischweise kleine Anzahl an) schwer zu erkennenden Fehler $\Phi(\mathscr{P}_{Initial}, T_{Fix})$ sehr spezialisierte Algorithmen zur Testmustererzeugung zum Einsatz kommen können. In [157] wurde dazu ein kommerzielles Werkzeug verwendet, prinzipiell lassen sich aber beliebige Verfahren nutzen, als Beispiele seien hier PHAETON [89] oder WaveSAT [90] (vgl. auch Abschnitt 3.3) genannt. Dadurch können hoch

optimierte zusätzliche Testmuster erzeugt werden, welche die Menge an benötigten Frequenzen verringern und gleichzeitig „leicht“ zu erzeugende Frequenzen bevorzugen. Mit gut gewählten Testmustern lässt sich zudem auch signifikant die Testdauer (in Anzahl angelegter Testmuster) reduzieren, was im folgenden Abschnitt genauer beschrieben wird.

4.7 Fehlerpartitionierung und Testmusterauswahl

Nachdem die Beobachtungszeitpunkte T für die HDF bestimmt wurden, kann der Test durchgeführt werden. Es ist jedoch sehr zeitintensiv, für jedes $t \in T$ den vollständigen Testsatz $\mathscr{P}$ anzulegen, insbesondere dann, wenn $\mathscr{P}$ absichtlich sehr groß gewählt wurde, z. B. ein n-erkennender Testsatz. Es ist demnach wünschenswert, nur einen Teilsatz $\mathscr{P}_t \subseteq \mathscr{P}$ an besonders effektiven Testmustern für jeden Beobachtungszeitpunkt t zu bestimmen, um die Testdauer zu minimieren. In der Literatur gibt es dazu verschiedene Verfahren, beispielsweise partitionieren Ahmed und Tehranipoor in [3, 4] den Testsatz basierend auf der geringsten Pufferzeit der einzelnen Testmuster, ähnlich gehen auch Yoneda et al. in [141] vor. Hier wird jetzt ein Verfahren vorgestellt, welches zunächst eine Fehlerpartitionierung durchführt und dann für die einzelnen Partitionen minimale Teilsätze aus $\mathscr{P}$ bestimmt. Es ist also in gewisser Weise ähnlich zu den Verfahren aus [141, 3], aber wesentlich aggressiver in der Musterauswahl. Da auch solche Testmuster ausgewählt werden, für welche die längsten sensibilisierten Pfade eine höhere Verzögerungszeit als den gewählten Beobachtungspunkt haben, werden diese entsprechend einen X-Wert in der Simulation erzeugen. Freilich kann man versuchen, die Testmuster so auszuwählen, dass möglichst wenig X-Werte produziert werden [142]. Dies kann aber zum einen die Testdauer erhöhen, da möglicherweise mehr Testmuster benötigt werden. Zum anderen kann die erhöhte Anzahl an X-Werten durch die in Kapitel 5 vorgestellten Techniken gut gehandhabt werden. Daher wird in dieser Arbeit versucht, vor allem die Anzahl an benötigten Testmustern zu minimieren. Zunächst wird allerdings die Fehlerpartitionierung vorgestellt.

Definition 4.9. *Seien $\mathscr{P}$ ein Testsatz, t_{nom} und t_{min} der nominelle bzw. minimale Beobachtungszeitpunkt und $t_{min} \leq t < t_{nom}$ ein Beobachtungszeitpunkt. Zudem*

sei $\Phi(\mathscr{P}, t_{nom})$ eine Menge an HDF mit Erkennungsbereichen $I(\varphi, \mathscr{P})$. Eine Teilmenge $\Omega(t) \subseteq \Phi(\mathscr{P}, t_{nom})$ wird als FAST-Gruppe für t *bezeichnet, wenn $t \in I(\varphi, \mathscr{P})$ für alle $\varphi \in \Omega(t)$ gilt.*

Eine FAST-Gruppe für einen Beobachtungszeitpunkt t muss nicht alle Fehler, welche unter t erkennbar sind, enthalten. Da ein Fehler möglicherweise unter mehreren der gewählten Beobachtungszeitpunkte erkennbar ist, reicht es aus, ihn lediglich einer einzelnen FAST-Gruppe zuzuweisen, um seine Erkennbarkeit zu garantieren. Das vereinfacht später die Musterauswahl für die FAST-Gruppen. Um $\Phi(\mathscr{P}, t_{nom})$ mittels T in FAST-Gruppen zu partitionieren, wird ein besonders einfaches Verfahren eingesetzt. Es werden alle Beobachtungszeitpunkte in absteigender Reihenfolge durchlaufen und für jedes $t \in T$ wird eine FAST-Gruppe $\Omega(t)$ erzeugt, welche alle noch nicht zugeordneten HDF beinhaltet, die unter t erkennbar sind. Dieses Verfahren sorgt dafür, dass jedem HDF der größtmögliche Beobachtungszeitpunkt (und damit kleinstmögliche Frequenz) aus T zugewiesen wird, was unter praktischen Aspekten vorteilhaft ist. Der Pseudocode des Verfahrens ist in Algorithmus 4.5 gegeben.

Algorithmus 4.5 Partitioniere-Fehler

Eingabe: $\Phi(\mathscr{P}, t_{nom})$, T
Ausgabe: $\Omega(t)$ für alle $t \in T$

```
1: L = Φ(P, t_nom)
2: für alle t ∈ T  // Absteigend sortiert
3:     Ω(t) = ∅
4:     für alle φ ∈ L
5:         wenn t ∈ I(φ, P) dann
6:             Ω(t) = Ω(t) ∪ {φ}
7:             L = L\{φ}
```

Sobald die FAST-Gruppen erzeugt wurden, muss für jede FAST-Gruppe ein Testsatz bestimmt werden, welcher alle Fehler in der FAST-Gruppe erkennt.

Problem 4.5 (Optimale Musterauswahl)**.** *Gegeben seien ein Testsatz $\mathscr{P}$ und eine Menge Φ an kleinen Verzögerungsfehlern, sodass jeder Fehler $\varphi \in \Phi$ von mindestens einem Testmuster $P \in \mathscr{P}$ erkannt wird. Bestimme eine minimale Teilmenge $\mathscr{P}' \subseteq \mathscr{P}$, sodass für alle $\varphi \in \Phi$ mindestens ein Testmuster $P' \in \mathscr{P}'$ existiert, welches φ erkennt.*

Im Folgenden soll gezeigt werden, dass die optimale Musterauswahl ein NP-vollständiges Optimierungsproblem ist.[4] Dazu wird das NP-vollständige Mengenüberdeckungsproblem [37] betrachtet.

Problem 4.6 (Minimale Mengenüberdeckung)**.** *Sei U eine endliche Menge und $\mathscr{S} \subseteq \mathrm{Pot}(U)$ eine Menge an Teilmengen aus U mit $\cup_{S \in \mathscr{S}} S = U$. Finde eine minimale Teilmenge $\mathscr{S}' \subseteq \mathscr{S}$, sodass $\cup_{S' \in \mathscr{S}'} S' = U$.*

Theorem 4.2. *Die optimale Musterauswahl ist NP-vollständig.*

Beweis. Um eine Lösung $\mathscr{P}'$ der optimalen Musterauswahl für eine Fehlermenge Φ zu validieren, muss für jeden Fehler $\varphi \in \Phi$ geprüft werden, ob es ein $P' \in \mathscr{P}'$ gibt, welches φ erkennt. Dies kann z. B. über Fehlersimulation geschehen, deren Laufzeit linear in der Schaltungsgröße ist. Die Überprüfung der Lösung erfordert also $\mathcal{O}(|\Phi| \cdot |\mathscr{P}'| \cdot c_{Simulation})$ viele Schritte, demnach ist die optimale Musterauswahl ein Problem in NP.

Betrachte nun eine Instanz des Mengenüberdeckungsproblems, gegeben durch eine Menge $U = \{u_1, \ldots, u_n\}$ und eine Menge $\mathscr{S} = \{S_1, \ldots, S_m\}$ an Teilmengen von U. Zur Umformung des Problems auf die optimale Musterauswahl, betrachte eine Schaltung bestehend aus n Ein- und Ausgängen, die jeweils über einen Puffer verbunden sind (vgl. Abbildung 4.9).

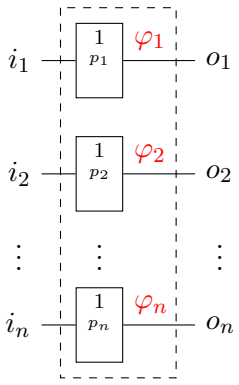


Abbildung 4.9: Eine Trivialschaltung

Konstruiere eine Fehlermenge $\Phi = \{\varphi_1, \ldots, \varphi_n\}$ aus der Menge U, indem eine bijektive Abbildung $r : U \to \Phi$ konstruiert wird, die jedes Element $u_j \in U$

[4] Auch hier gilt wieder Fußnote 3 zu beachten.

auf einen Fehler $\varphi_j \in \Phi$ abbildet, welcher den Puffer p_j befallen hat. Weiterhin konstruiere für jede Menge $S_k \in \mathscr{S}$ ein Testmuster P_k wie folgt: Für jedes Element $u_j \in S_k$ erzeuge einen Pegelwechsel an denjenigem Eingang i_j, dessen zugehöriger Puffer p_j vom Fehler $\varphi_j = r(u_j)$ befallen ist. Das bedeutet, dass ein Testmuster P_k alle von S_k über r abgebildeten Fehler erkennt. Die optimale Musterauswahl wählt nun eine minimale Teilmenge $\mathscr{P}'$ von $\mathscr{P} = \{P_1, \ldots, P_m\}$, sodass alle Fehler in Φ erkannt werden. Durch Rücktransformation erhält man eine minimale Teilmenge $\mathscr{S}' \in \mathscr{S}$, für die $\cup_{S' \in \mathscr{S}'} S' = U$ gilt, was genau der Lösung der minimalen Mengenüberdeckung entspricht. Demnach ist die optimale Musterauswahl ein NP-vollständiges Problem. □

Da die optimale Musterauswahl NP-vollständig ist, kann sie auch zum Minimum Hitting Set Problem umgeformt werden. Damit können die gleichen Algorithmen zum Einsatz kommen, welche bereits zur optimalen Frequenzauswahl vorgestellt wurden. Um den zeitlichen Aufwand der Auswahl gering zu halten – die Testmusterauswahl muss für jede FAST-Gruppe einzeln durchgeführt werden –, wird in dieser Arbeit ein gieriges Verfahren genutzt. Dabei wird als erstes dasjenige Testmuster ausgewählt, welches die meisten Fehler der FAST-Gruppe abdeckt. Danach wird dasjenige Testmuster gewählt, welches die meisten der verbleibenden Fehler abdeckt usw., bis alle Fehler abgedeckt wurden.

4.8 Simulationsergebnisse

In diesem Abschnitt werden die Ergebnisse einer simulationsbasierten Studie aufgeführt, um die optimale Frequenzauswahl zu evaluieren. Es wurden dabei eine Reihe von ITC99 Benchmarkschaltungen [16] untersucht, welche mit einer 28 nm Complementary Metal Oxide Semiconductor (CMOS) Standardzellbibliothek für Forschungs- und Lehrzwecke synthetisiert wurden [85]. Der Syntheseablauf ist im Kapitel 5.3 beschrieben, der prüfgerechte Entwurf für FAST wurde an dieser Stelle jedoch außen vor gelassen, um die Frequenzauswahl nicht zu beinflussen. Zudem standen industrielle Schaltungen zur Verfügung, welche allerdings bereits mit einer 90 nm Standardzellbibliothek (ebenfalls für Forschungs- und Lehrzwecke) vorsynthetisiert waren. Die Anzahl der kombinatorischen Gatter in den Schal-

tungen rangierten von 7018 Gattern für b15 bis hin zu 132 044 Gatter für b19 (beides ITC99 Benchmarks). Der b14-Benchmark weist mit 215 die geringste Zahl an Flipflops auf, der b19-Benchmark mit 6130 Flipflops die größte Zahl. Für die ITC99-Schaltungen wurden Launch-on Capture (LOC) Testmuster erzeugt, für die industriellen Schaltungen konnten aufgrund fehlender Daten[5] lediglich Testätze unter Betrachtung des kombinatorischen Teils erzeugt werden, was Enhanced Scan (ES) entspricht (vgl. Kapitel 2.4). Die genauen Statistiken der Schaltungen sind in Tabelle A.1a (Anhang A.1), Daten über die erzeugten Testsätze sind in Tabelle A.3a (Anhang A.2) zu finden. An dieser Stelle sei angemerkt, dass die Qualität des Testsatzes wesentlich die Ergebnisse (HDF-Abdeckung, Anzahl benötigter Frequenzen, Musterpartitionierung) beeinflusst. Durch die Nutzung von optimierten Testsätzen können die Ergebnisse wahrscheinlich noch weiter verbessert werden, dies ist aber nicht Fokus dieser Arbeit.

Als mögliche Fehlerorte wurden alle Fehlerorte für Übergangsfehler in Betracht gezogen, nachdem Äquivalenzregeln angewandt wurden, um die Menge zu verringern (dies wurde automatisch vom ATPG-Programm erledigt). Die Größe eines SDF wird auf die minimale zusätzliche Verzögerung $s_{min} = 6\sigma$ festgelegt (vgl. Abschnitt 3.6).[6] In dieser Arbeit wurde $\sigma = 0{,}2\mu$ angenommen, da keine echten Prozessdaten vorlagen. Dieser (oder ein vergleichbarer) Wert wird auch in der Literatur verwendet, z. B. in [123]. Die Anzahl an SDF bewegt sich zwischen 29 756 (b15) und 559 759 (b19), genaue Daten sind in Tabelle A.4a (Anhang A.3) nachzulesen. Die maximale Frequenz für FAST wurde mit $f_{max} = 3f_{nom}$ festgelegt, damit gilt entsprechend $t_{min} = t_{nom}/3$. Abschnitt 4.8.3 zeigt Ergebnisse, wenn f_{max} auf $6f_{nom}$ erhöht wird.

4.8.1 Ergebnisse der Frequenzauswahl

In dieser Arbeit wurden vier verschiedene Varianten der Frequenzauswahl untersucht:

[5] Es fehlten Testprotokolle, welche vom ATPG-Programm benötigt wurden und als Teil des Syntheseprozesses erzeugt werden.

[6] Weitere Fehlergrößen wurden in [156] analysiert.

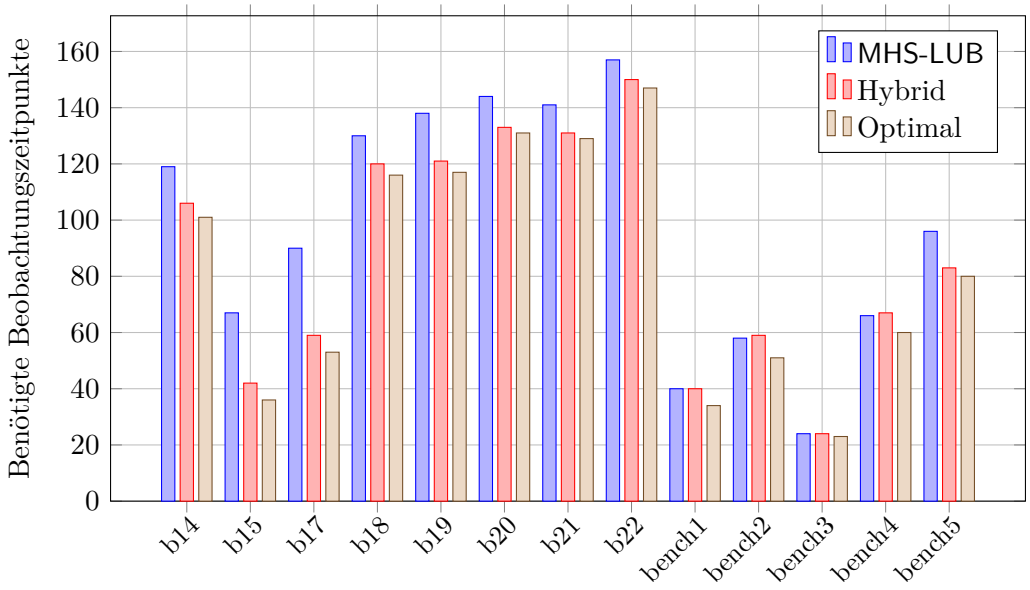


Abbildung 4.10: Ergebnisse der Frequenzauswahl

1. T_{Fix}, berechnet nach Gleichung (4.3), mit 10 Beobachtungszeitpunkten,

2. einfache Frequenzauswahl mittels kleinster oberer Schranken (MHS-LUB),

3. optimale Frequenzauswahl über den MHS-Hypergraph sowie

4. das hybride Verfahren, zusammengesetzt aus den Punkten 1 und 3.

Zunächst zeigt Abbildung 4.10 die Anzahl an benötigten Frequenzen für die letzten drei Verfahren. Die genauen Daten für diese Abbildung befinden sich in Tabelle A.6 (Anhang A.3, Bereich „Standardentwurf“). Es zeigt sich über alle Schaltungen hinweg, dass mit dem hybriden Verfahren eine gute Annäherung an das optimale Ergebnis erreicht werden kann. Es sei hier nochmals angemerkt, dass MHS-LUB kein realistisches Verfahren zur Frequenzauswahl darstellt, sondern lediglich einen theoretischen Ansatz zum Vergleich.

4.8.2 Detailanalyse

Bei der Betrachtung der Anzahl an Beobachtungspunkten für vollständige Fehlereffizienz fällt auf, dass diese über fast alle untersuchten Schaltungen hinweg recht hoch ist. Werden Taktgeneratoren wie PCG [112] oder LCCG [72] verwendet (vgl. Abschnitt 3.7.1), ist dies freilich ein geringeres Problem, da die Zielfrequenz programmierbar ist. Es können aber die Hardwarekosten des Generators steigen, da möglicherweise „schwer“ zu erzeugende Frequenzen benötigt werden, welche sehr feine Pufferstufen in den Taktgeneratoren voraussetzen. Bisherige Veröffentlichungen setzen auf deutlich weniger Frequenzen. Beispielsweise werden in [3] nur 5 Frequenzen genutzt, veröffentlichte Testdaten aus [13] wurden mit einer einzelnen erhöhten Testfrequenz erzeugt. Turakhia et al. nutzen in [119] ein anderes Verfahren, in welchem die Testfrequenz kontinuierlich erhöht wird. Dies erfolgt allerdings in sehr großen Schritten, insgesamt werden ebenfalls nur wenige Frequenzen benötigt. Aus diesen Gründen ist eine detaillierte Analyse der Ergebnisse erforderlich.

Topologische Voranalyse

Es wurde bereits in Abschnitt 4.3 erwähnt, dass FAST-Simulation eine hohe Komplexität hat, da für jedes Testmuster alle Fehler simuliert werden müssen. Das erfordert effiziente Simulationsprogramme mit entsprechender Hardwarebeschleunigung. Aus diesem Grunde empfiehlt es sich, die Menge an SDF, welche simuliert werden sollen, bereits vor der Simulation so stark wie möglich einzugrenzen. In [157] wurde dazu eine topologische Voranalyse entwickelt, welche anhand der zeitlichen Informationen der Schaltung über die Schaltungstopologie bestimmt, ob ein SDF simuliert werden sollte oder nicht. Dazu werden für jeden Fehlerort die topologischen Pfade mit den kürzesten und längsten Verzögerungen bestimmt.[7] Im Folgenden bezeichne $t_{top,min}$ ($t_{top,max}$) die Verzögerung des kürzesten (längsten) Pfades durch den Fehlerort, an dem sich der SDF φ mit Größe s_{min} befindet. Zusätzlich seien t_{min} und t_{nom} der minimal erlaubte bzw. nominelle Beobachtungszeitpunkt. Dann können folgende Fälle unterschieden werden:

[7] Dies kann z. B. über statische Analyse des Schaltverhaltens geschehen [109].

1. $t_{top,max} + s_{min} < t_{min}$: In diesem Fall ist φ ein *unerkennbarer* Fehler, da man Beobachtungszeitpunkte unterhalb von t_{min} benötigt, um φ sichtbar zu machen.

2. $t_{top,min} + s_{min} > t_{nom}$: Ein Test, welcher den Effekt von φ erfolgreich zu einem Ausgang propagiert, wird φ auch unter t_{nom} erkennen können. φ kann daher mit einem Betriebsfrequenztest erkannt werden und ist kein HDF.

3. Andernfalls kann nicht gesagt werden, ob φ ein HDF ist oder nicht, folglich muss φ simuliert werden, um den Erkennungsbereich zu bestimmen.

Es sei angemerkt, dass die Pfade mit $t_{top,min}$ bzw. $t_{top,max}$ möglicherweise nicht sensibilisierbar sind. Da die Verzögerungen jedoch als obere bzw. untere Schranken verwendet werden, ist eine Sensibilisierbarkeit unerheblich. Abbildung 4.11 zeigt die Einteilung der SDF in die verschiedenen Klassen nach der topologischen Voranalyse (nicht maßstabsgetreu).

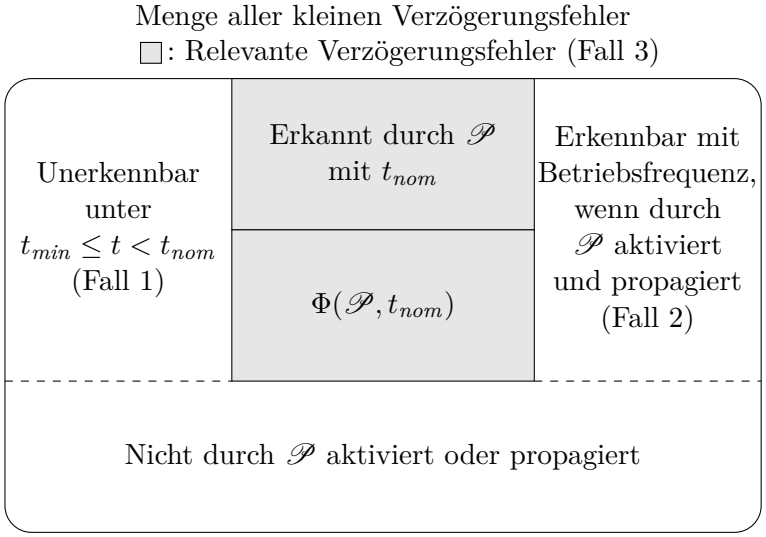


Abbildung 4.11: Einteilung SDF für die FAST-Simulation

Zunächst sind im unteren Teil der Menge alle Fehler, welche nicht durch $\mathscr{P}$ aktiviert oder propagiert werden. Solche Fehler können weder durch einen Betriebsfrequenztest noch durch FAST sichtbar gemacht werden, da es keine von $\mathscr{P}$ erzeugte Signalflanke gibt, welche durch den Fehlerort läuft und zu einem der Schaltungsausgänge propagiert wird. Diese Fehler tragen konsequenterweise zu

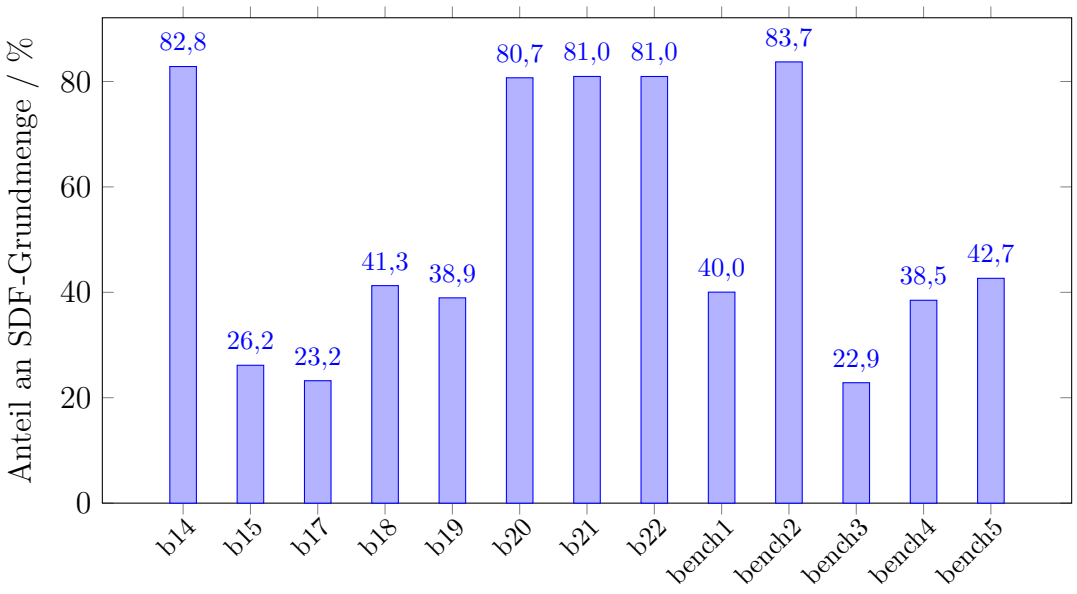


Abbildung 4.12: Anteil an relevanten SDF nach der Voranalyse

einer niedrigen Fehlerabdeckung bei. Oben links sind dann die unerkennbaren Fehler (Fall 1) dargestellt. Für diese Fehler spielt es keine Rolle, ob sie überhaupt von $\mathscr{P}$ aktiviert und propagiert werden, da selbst FAST eine Auswirkung der Fehler nicht sichtbar machen kann. Sie tragen ebenfalls zu einer geringen Fehlerabdeckung bei. Analog sind oben rechts die mit Betriebsfrequenztest sichtbaren Fehler (Fall 2) dargestellt. Verbleibend sind die grau unterlegten Fehlermengen, welche als *relevante Verzögerungsfehler* oder HDF-Kandidaten bezeichnet werden und die Ausgangsmenge für FAST-Simulation bilden. Diese unterteilen sich dann in jene Fehler, welche mit t_{nom} erkannt wurden und die HDF unter $\mathscr{P}$ und t_{nom}. Für die betrachteten Schaltungen wurde die topologische Voranalyse durchgeführt. Der Anteil der relevanten SDF an der Ausgangsmenge ist in Abbildung 4.12 dargestellt, die zugrunde liegenden Daten befinden sich in Tabelle A.4a (Anhang A.3).

Die Menge an relevanten SDF bestimmt sich über die nominelle Taktperiode der jeweiligen Schaltung, welche sich wiederum nach der Verzögerung des kritischen Pfades richtet. Ein geringer Anteil an relevanten SDF bedeutet, dass der kritische Pfad im Vergleich zu den längsten topologischen Pfaden durch die Fehlerorte eine deutliche größere Verzögerungszeit aufweist. Die Daten in Tabelle A.4a zeigen, dass kein einziger SDF als Fall 2 klassifiziert wird, d. h. alle Fehler, welche in der Voranalyse aus der Grundmenge entfernt wurden, sind mit f_{max} bzw. t_{min}

nicht erkennbar. Dazu sei aber auch angemerkt, dass mit $s = 6\sigma$ eine sehr kleine Fehlergröße betrachtet wird, diese liegt üblicherweise im Rahmen einer einzelnen Gatterverzögerung.

Fehlereffizienz bei unvollständiger Frequenzauswahl

Nun wird untersucht, inwiefern die individuellen Frequenzen zur Erhöhung der Fehlereffizienz beitragen. Dazu wird für jede Frequenz die Menge an beobachtbaren HDF berechnet und die Frequenzen werden nach absteigender Anzahl sortiert. Die Frequenz, welche die meisten Fehler beobachtbar werden lässt, wird ausgewählt. Danach startet das Verfahren mit den verbleibenden Frequenzen neu. Abbildung 4.13 zeigt die so bestimmte partielle HDF-Effizienz für das hybride Verfahren. Dabei werden repräsentativ nur die Schaltungen b19, b22 und bench5 betrachtet, um das Diagramm nicht zu überfrachten. Diese Schaltungen wurden ausgewählt, da sie unter den ITC99- und industriellen Schaltungen die meisten Beobachtungszeitpunkte für vollständige Fehlereffizienz benötigen. Es ist zu beachten, dass für die Schaltungen unterschiedliche Anzahlen an Frequenzen ausgewählt wurden. Deshalb gehen nicht alle Linien bis zum Maximalwert, sondern hören an dem Punkt auf, an dem 100% Fehlereffizienz erreicht wird.

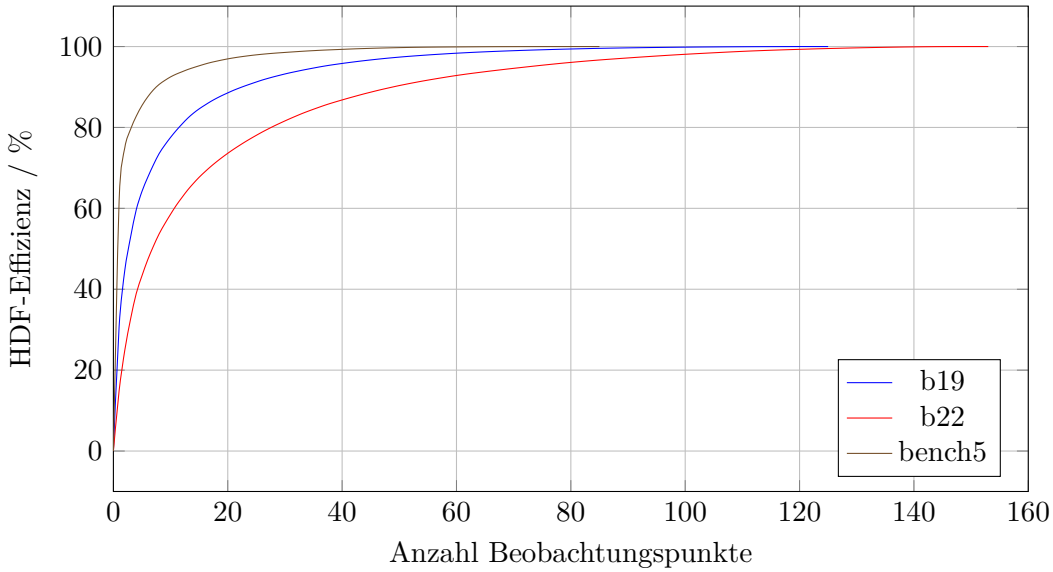


Abbildung 4.13: Partielle HDF-Effizienz bei unvollständiger Frequenzauswahl

Man sieht sehr deutlich, dass bereits bei einer geringen Anzahl an Frequenzen der Großteil der HDF erkannt werden kann, lediglich für einige wenige Fehler müssen viele verschiedene Frequenzen benutzt werden. Dies gilt insbesondere für bench5. Diese Schaltung weist insgesamt eine recht gute Fehlerabdeckung auf, was auch für die meisten anderen industriellen Schaltungen gilt.

In der Praxis kann es passieren, dass auch mit der hybriden Frequenzauswahl für den gegebenen Testsatz Frequenzen bestimmt werden, welche nicht mit dem verfügbaren Frequenzgenerator erzeugt werden können. Obwohl das Verfahren bereits Intervalle an möglichen Beobachtungszeitpunkten zurückgibt, kann es nicht garantieren, das z. B. eine PLL in der Lage ist, eine entsprechende Frequenz zu erzeugen. Typischerweise haben die Generatoren eine gewisse Auflösung an Frequenzen, welche einem Mindestabstand der Beobachtungszeitpunkte entspricht. Eine Alternative zu den bisher vorgestellten Verfahren wäre also, dass aus dem Interval $[t_{min}, t_{nom})$ gleichmäßig (mit der Auflösung des Generators) verteilte Beobachtungszeitpunkte gewählt werden, ganz so wie es weiter oben beschrieben wurde. Sortiert man nun wieder die besten Frequenzen wie gerade beschrieben, ergibt sich die Abbildung 4.14, wenn beispielsweise 100 verschiedene Beobachtungszeitpunkte gewählt werden.

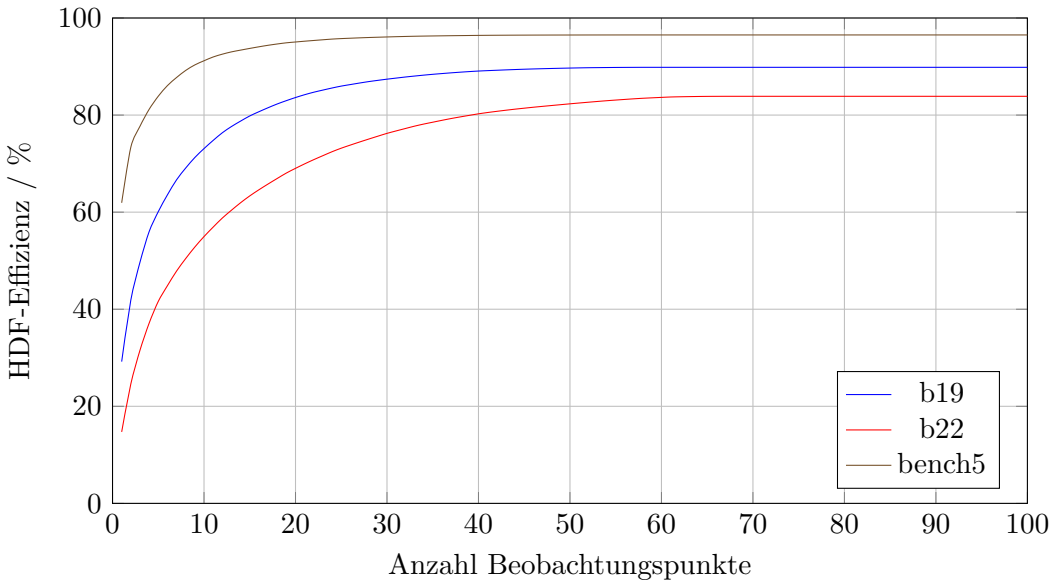


Abbildung 4.14: Partielle HDF-Effizienz bei 100 festen Frequenzen

Hier zeigt sich, dass die Fehlereffzienz gegen einen gewissen Maximalwert (z. B. 89,8% für b19, erreicht bereits ab 58 Frequenzen) strebt. Aber wie zuvor auch

ergibt sich die Situation, dass bereits eine geringe Anzahl an Frequenzen für eine hohe Fehlereffizienz sorgt, während weitere Frequenzen immer weniger zur Steigerung beitragen. Das ist eine sehr ähnliche Situation wie beispielsweise beim eingebetteten Pseudozufallstest, hier spricht man von zufallsmusterresistenten Fehlern, welche sich nur schwer durch pseudozufällige Musterfolgen erkennen lassen. Analog kann man bei FAST von „frequenzresistenten“ Fehlern sprechen. Für diese werden entweder sehr spezielle Frequenzen oder – falls möglich – sehr spezielle Testmuster benötigt, um sie sichtbar zu machen. Die Erkennung solcher Fehler in der Praxis stellt damit eine große Herausforderung dar, welche noch durch weitere Forschungsarbeiten zu lösen ist.

Qualität des Testsatzes

In diesem Abschnitt soll die Qualität des einfachen Testsatzes für Übergangsfehler hinsichtlich der Erkennung von HDF analysiert werden. Um die Ergebnisse von FAST-Simulation quantitativ darstellen zu können, werden zunächst verschiedene Fehlerabdeckungen definiert.

Definition 4.10. *Seien $\mathscr{P}$ ein Testsatz, t_{nom} der nominelle Beobachtungszeitpunkt und $\Phi(\mathscr{P}, t_{nom})$ eine Menge an HDF unter $\mathscr{P}$ und t_{nom}. $I(\varphi, \mathscr{P})$ bezeichne den Erkennungsbereich eines Fehlers $\varphi \in \Phi$ unter $\mathscr{P}$, zusätzlich sei $t \in \mathbb{N}$ ein Beobachtungszeitpunkt. Dann bezeichnet*

$$\mathrm{FA}(\Phi(\mathscr{P}, t_{nom}), t) = \frac{|\{\varphi \mid \varphi \in \Phi(\mathscr{P}, t_{nom}), t \in I(\varphi, \mathscr{P})\}|}{|\Phi(\mathscr{P}, t_{nom})|} \tag{4.4}$$

die HDF-Abdeckung von $\mathscr{P}$ unter t. *Sei T eine Menge an Beobachtungszeitpunkten. Dann bezeichnet*

$$\mathrm{FA}(\Phi(\mathscr{P}, t_{nom}), T) = \frac{|\{\varphi \mid \varphi \in \Phi(\mathscr{P}, t_{nom}), \exists t \in T : t \in I(\varphi, \mathscr{P})\}|}{|\Phi(\mathscr{P}, t_{nom})|} \tag{4.5}$$

die HDF-Abdeckung von $\mathscr{P}$ unter T.

Eine spezielle HDF-Abdeckung ist die *potentielle HDF-Abdeckung unter FAST*, $\mathrm{FA}(\Phi(\mathscr{P}, t_{nom}), I_{FAST})$, mit $I_{FAST} = [t_{min}, t_{nom})$. Zudem lohnt sich ein Blick auf

die *ideale HDF-Abdeckung von* $\mathscr{P}$

$$\mathrm{FA}(\Phi(\mathscr{P}, t_{nom})) = \frac{|\{\varphi \mid \varphi \in \Phi(\mathscr{P}, t_{nom}), \exists t > 0 : t \in I(\varphi, \mathscr{P})\}|}{|\Phi(\mathscr{P}, t_{nom})|}, \tag{4.6}$$

um die Qualität des Testsatzes zu bewerten. Abschließend sei Φ eine Menge an SDF und $\Phi(\mathscr{P}, t_{nom})$ die dazugehörige Menge an HDF unter $\mathscr{P}$ und t_{nom}. Dann ist die *Betriebsfrequenzabdeckung* $\mathrm{FA}(\Phi, t_{nom})$ der Anteil an SDFs, welcher unter t_{nom} erkannt werden kann.

Für die gegebenen Schaltungen und Testsätze zeigte sich, dass mit der gegebenen Fehlergröße von 6σ kein einziger SDF unter Betriebsfrequenz erkennbar ist, selbst mit dem erweiterten Testsatz $\mathscr{P}_{Total}$. Das ist aufgrund der extrem geringen Fehlergröße (in der Größenordnung einer einzelnen zusätzlichen Gatterverzögerung) nicht überraschend. Im Folgenden werden daher die Werte der Betriebsfrequenzabdeckung ausgelassen (werden aber in den entsprechenden Tabellen im Anhang angegeben). Abbildung 4.15 zeigt die ideale sowie potentielle HDF-Abdeckung unter FAST nach der ersten bzw. zweiten Simulation des hybriden Verfahrens (vgl. Abschnitt 4.6). Die Daten für dieses Diagramm befinden sich in Tabelle A.5a (Anhang A.3).

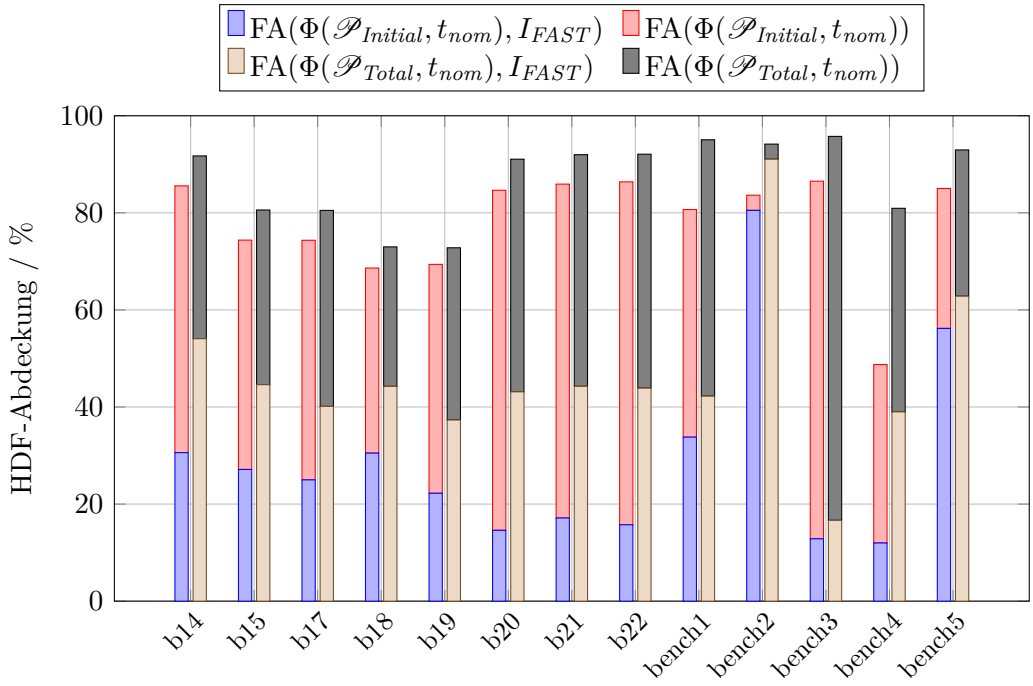


Abbildung 4.15: Potentielle und ideale HDF-Abdeckung im hybriden Verfahren

In der Abbildung ist für jede Schaltung im linken Balken die ideale und potentielle Abdeckung für die erste Simulation gestapelt dargestellt, analog zeigt der rechte Balken die Abdeckungen für die zweite Simulation. Für alle untersuchten Schaltungen ist die ideale HDF-Abdeckung bereits unter $\mathscr{P}_{Initial}$ recht hoch, aber dennoch geringer als die vom ATPG-Programm berichtete Fehlerabdeckung für Übergangsfehler. Dies erklärt sich durch die Nichtbeachtung des zeitlichen Verhaltens bei der Berechnung der Abdeckung, denn Effekte wie Signalpulse können die Erkennung eines SDF zunichte machen. Die ideale HDF-Abdeckung kann durch $\mathscr{P}_{Total}$ nochmals gesteigert werden, da die zusätzlichen Testmuster auch weitere Fehler beobachtbar machen, welche unter $\mathscr{P}_{Initial}$ nicht oder nur unzureichend beobachtbar waren. Die potentielle HDF-Abdeckung unter FAST ist deutlich geringer. Nutzt man lediglich $\mathscr{P}_{Initial}$, so liegt diese im Mittel bei 29,1%, sie kann aber mit $\mathscr{P}_{Total}$ auf 47% gesteigert werden. Diese Zahlen sind vergleichsweise gering und verdeutlichen, dass die gewählten Testsätze nicht für die Erkennung so kleiner SDF optimiert sind. Freilich liegt die Qualität der Testsatzerzeugung nicht im Fokus dieser Arbeit und hat – wie bereits zuvor erwähnt – keine Auswirkungen auf die beschriebenen Verfahren zur Frequenzauswahl.

Abdeckung mit T_{Fix}

Abbildung 4.16 zeigt zunächst den Anteil an erkannten HDFs, wenn lediglich 10 Beobachtungszeitpunkte, als T_{Fix} bezeichnet, benutzt werden (Fall 1 aus Abschnitt 4.8.1). Im Mittel werden mit den festen Frequenzen bereits 41,7% der HDFs abgedeckt, allerdings streut die Abdeckung deutlich (der Median liegt bei 37,7%). Hat eine Schaltung nur geringe Fehlerabdeckung unter T_{Fix}, so werden in Folge mehr Muster für $\mathscr{P}_{Extra}$ erzeugt und die zweite Simulation dauert länger. Allerdings ist (wie später gezeigt) in diesen Fällen der Reduktionsfaktor der angelegten Testmuster höher als bei anderen Schaltungen. Dies hängt vermutlich damit zusammen, dass die für $\mathscr{P}_{Extra}$ erzeugten Testmuster eine hohe Güte haben, sodass sie auch für viele andere HDFs benutzt werden können.

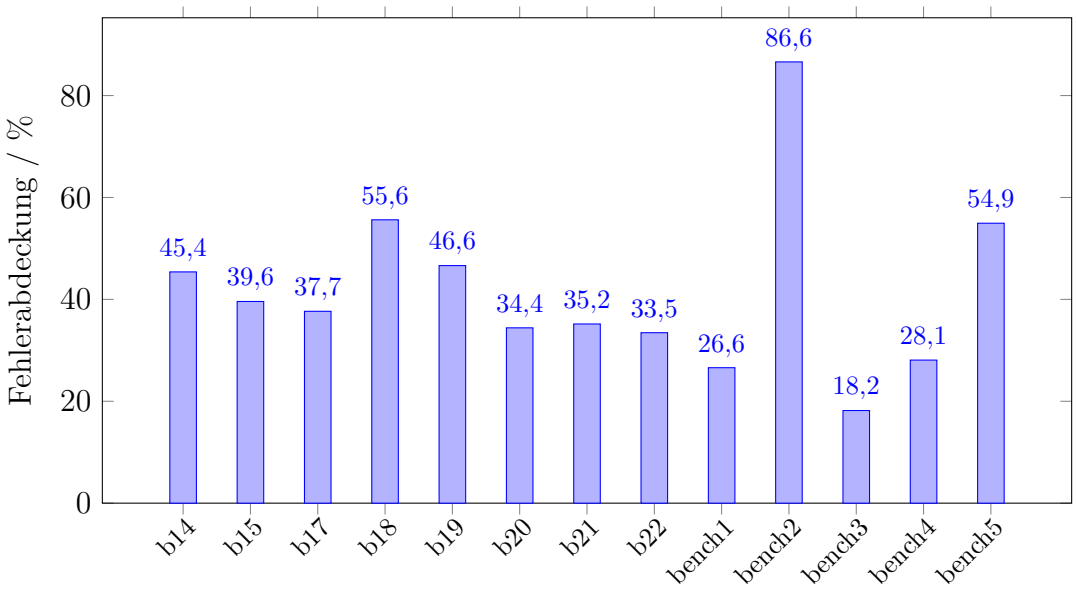


Abbildung 4.16: Fehlerabdeckung mit T_{Fix}

Abdeckung mit beliebigen Beobachtungszeitpunkten

Abbildung 4.17 zeigt den Anteil der idealen HDF-Abdeckung, wenn ein beliebiger minimaler Beobachtungszeitpunkt gewählt werden kann. Die Beobachtungszeitpunkte auf der horizontalen Achse sind dabei auf den jeweiligen nominellen Beobachtungszeitpunkt der Schaltung normiert, d. h. die horizontale Achse gibt relative Beobachtungszeitpunkte an. Diese Abbildung darf nicht mit Abbildung 4.15 verwechselt werden, da dort die absolute HDF-Abdeckung angegeben ist, während hier nur ein *Anteil* an der idealen HDF-Abdeckung gezeigt wird.

Diese Abbildung verdeutlicht, dass viele HDFs mit $\mathscr{P}_{Total}$ und t_{min} (in der Abbildung gestrichelt eingezeichnet) erkennbar sind, diese Zahl aber noch deutlich gesteigert werden kann. Eine vergleichbare Analyse diente bereits in [158] als Motivation für die Entwicklung spezieller Monitore, um einige Ausgänge mit einem künstlich verringerten Beobachtungszeitpunkt von $t'_{min} = t_{nom}/6$ aufzuzeichnen. Daher soll auch im folgenden Abschnitt eine solche Analyse durchgeführt werden.

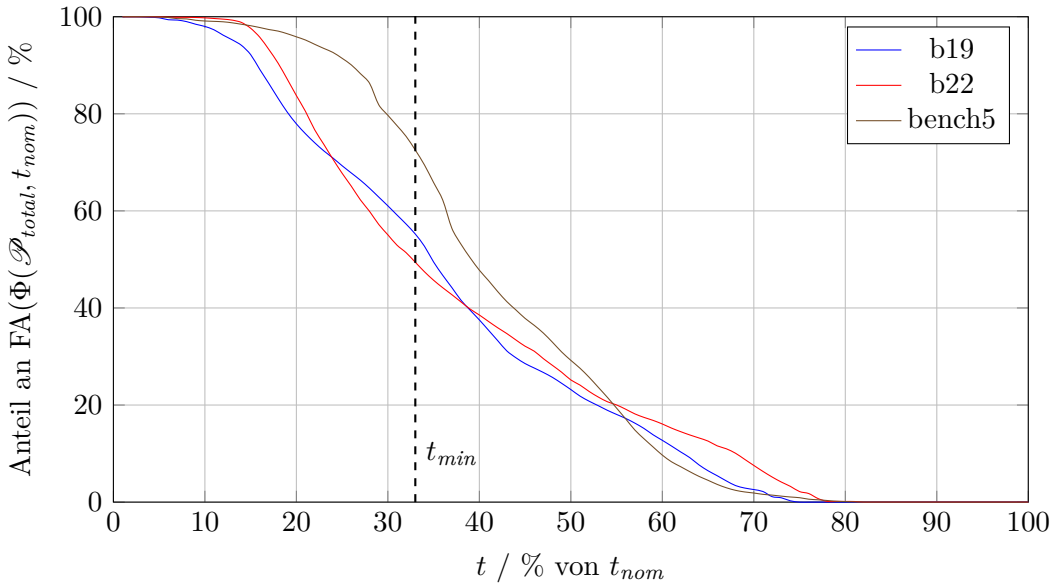


Abbildung 4.17: HDF-Effizienz unter beliebigen Beobachtungszeitpunkten

4.8.3 Simulation mit erhöhter Maximalfrequenz

Bei den Ergebnissen im vorangehenden Abschnitt zeigte sich, dass bei den meisten Schaltungen die HDF-Abdeckung unter FAST (mit $f_{max} = 3f_{nom}$) unter 60% liegt. In diesem Abschnitt sollen kurz Ergebnisse einer zweiten Simulation vorgestellt werden, bei welcher die maximale Frequenz auf $f_{max} = 6f_{nom}$ erhöht wurde. Aus Gründen der Einfachheit wurde keine erneute topologische Voranalyse durchgeführt, sondern es wurden direkt die Ergebnisse der ersten Simulation wiederverwendet. Es muss aber erneut eine Testmustererzeugung und die zweite Simulation durchgeführt werden, da sich T_{fix} nach Gleichung (4.3) geändert hat. Abbildung 4.18 zeigt die so erhaltene HDF-Abdeckung unter FAST für die erhöhte Maximalfrequenz (blau) im Vergleich zur vorherigen HDF-Abdeckung unter FAST (rot).

Die Abbildung zeigt, dass mit einer höheren Maximalfrequenz eine deutlich höhere HDF-Abdeckung unter FAST erzielt werden kann, was im Einklang mit Abbildung 4.17 steht. Des Weiteren zeigt Abbildung 4.19 die Anzahl der benötigten Frequenzen des hybriden Verfahrens. Zwei Beobachtungen sind bemerkenswert. Obgleich der Bereich, in welchem die Beobachtungszeitpunkte gewählt werden können, bei erhöhter Maximalfrequenz (entspricht einem geringeren minimalen

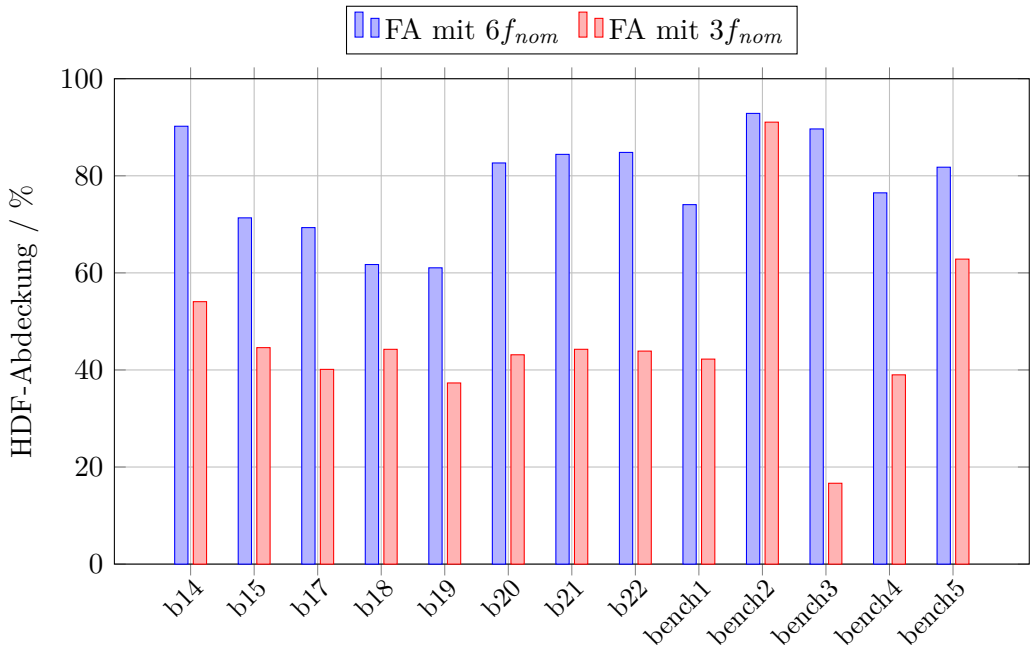


Abbildung 4.18: HDF-Abdeckung unter FAST mit $6f_{nom}$

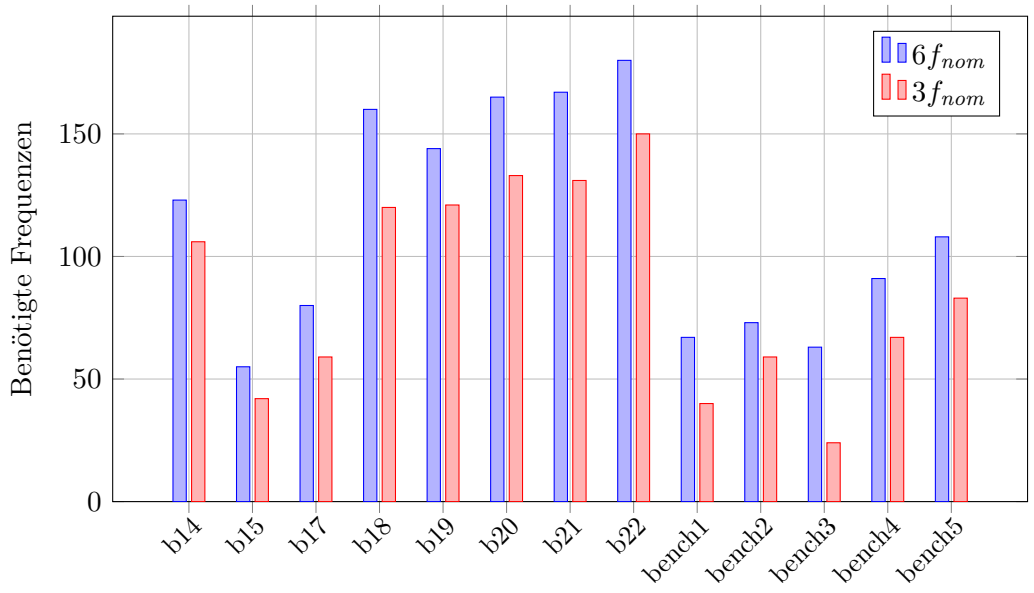


Abbildung 4.19: Vergleich der benötigten Frequenzen (hybride Auswahl)

Beobachtungszeitpunkt) deutlich größer ist, steigt die Anzahl der Frequenzen für vollständige Fehlereffizienz deutlich an. Eine mögliche Erklärung liegt in der Bestimmung der atomaren Intervalle. Schneiden sich viele Erkennungsbereiche, so werden die resultierenden atomaren Intervalle immer kleiner. In Folge müssen mehr dieser Intervalle ausgewählt werden, um die einzelnen Erkennungsbereiche abzudecken. Die zweite Beobachtung bezieht sich auf die Schaltung b17. Obwohl die Menge der Fehler, welche bei der hybriden Auswahl dem optimalen Verfahren übergeben wird, recht klein ist, dauerte die Berechnung von MHS-Hypergraph mehr als sieben Tage. Nach anfänglicher Fehlersuche stellte sich heraus, dass die Erkennungsbereiche so angeordnet sind, dass schnell keine Reduktion des Hypergraphen mehr möglich ist. So musste ein Entscheidungsbaum für etwa 160 Knoten des Hypergraphen aufgebaut werden. Hier zeigte sich deutlich, dass die Berechnung exponentiell in der Problemgröße ist. Aus diesem Grunde zeigt Abbildung 4.19 für die Schaltung b17 das Ergebnis mit einer gierigen Frequenzauswahl, nachdem T_{fix} verwendet wurde. Freilich muss dazu angemerkt werden, dass diese Schaltung die einzige mit diesem Problem war. Größere Schaltungen mit deutlich größeren Fehlermengen ließen sich problemlos in akzeptabler Laufzeit (wenige Stunden) berechnen (vgl. Abschnitte 4.8.5 und A.6).

4.8.4 Fehlerpartitionierung und Musterauswahl

Nachdem die Frequenzen für den Test ausgewählt wurden, wird eine Fehlerpartitionierung mit anschließender Musterauswahl, wie in Abschnitt 4.7 beschrieben, durchgeführt. Als Resultat ergeben sich die FAST-Gruppen, welche jeder Testfrequenz eine eigene Mustermenge (als Teilmenge von $\mathscr{P}_{Total}$) zuordnet. Sei T die Menge an Beobachtungszeitpunkten für FAST und $\mathscr{P}_t$ der Testsatz für einen Beobachtungszeitpunkt $t \in T$, dann berechnet sich der Reduktionsfaktor der angelegten Testmuster zu

$$R_{Muster} = \frac{|T| \cdot |\mathscr{P}_{Total}|}{\sum_{t \in T} |\mathscr{P}_t|}. \tag{4.7}$$

Abbildung 4.20 zeigt den so erreichbaren Reduktionsfaktor in der Anzahl der angelegten Testmuster für das hybride Verfahren, die Daten dazu befinden sich in Tabelle A.8a (Anhang A.4).

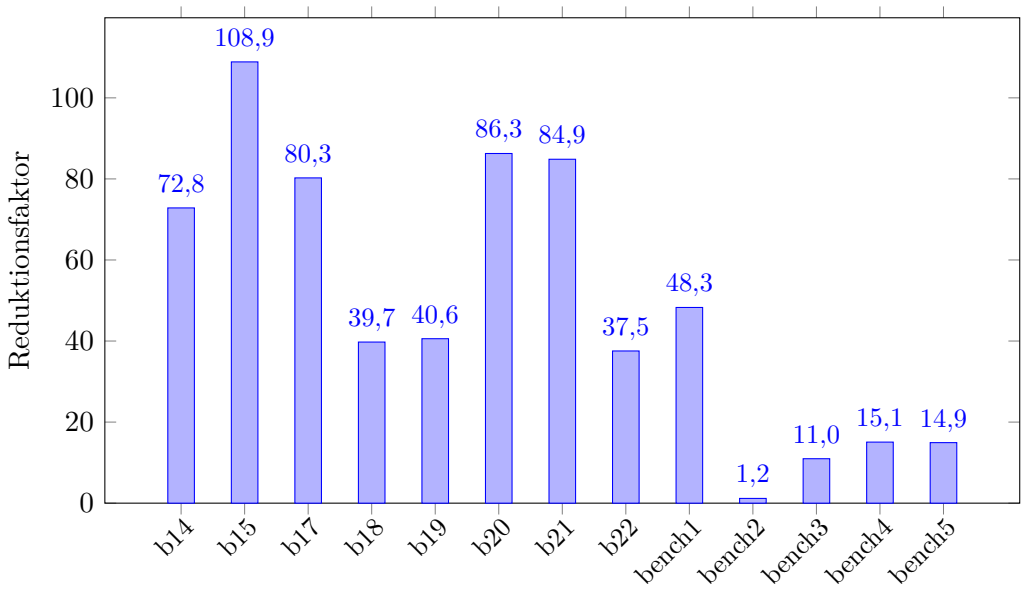


Abbildung 4.20: Reduktionsfaktor der angelegten Testmuster beim hybriden Verfahren

Es zeigt sich, dass durch diese Partitionierung bei fast allen Schaltungen eine sehr hohe Reduktion erreicht wird. Eine Ausnahme bildet bench2. Bei dieser Schaltung war die zugrunde liegende Testmustermenge so gering, dass für jede Frequenz fast alle Muster verwendet werden mussten.

Da die Anzahl der angelegten Testmuster direkt proportional zur Testdauer ist, gilt ein ähnlicher Faktor ebenfalls für die Reduktion der Testdauer und damit letztendlich der Testkosten. Es stellt sich heraus, dass typischerweise nicht alle Testmuster aus $\mathscr{P}_{Total}$ verwendet werden. Abbildung 4.21 zeigt den prozentualen Anteil der verwendeten Testmuster für das hybride Verfahren, die Daten hierzu befinden sich ebenfalls in Tabelle A.8a.

Hier sieht man, dass für die meisten Schaltungen nur ein geringer Teil der Testmuster aus $\mathscr{P}_{Total}$ verwendet werden muss (im Mittel 54,4%). Die Schaltung bench2 zeigt, in Übereinstimmung mit Abbildung 4.20, als einzige Schaltung einen Anteil von 100%, d. h. alle Muster der Grundmenge werden bei mindestens einer Frequenz angelegt. Dieses Ergebnis zeigt insgesamt, dass mit geschickter Steuerung Musterspeicher eingespart werden kann, was zusätzlich die Testkosten reduziert.

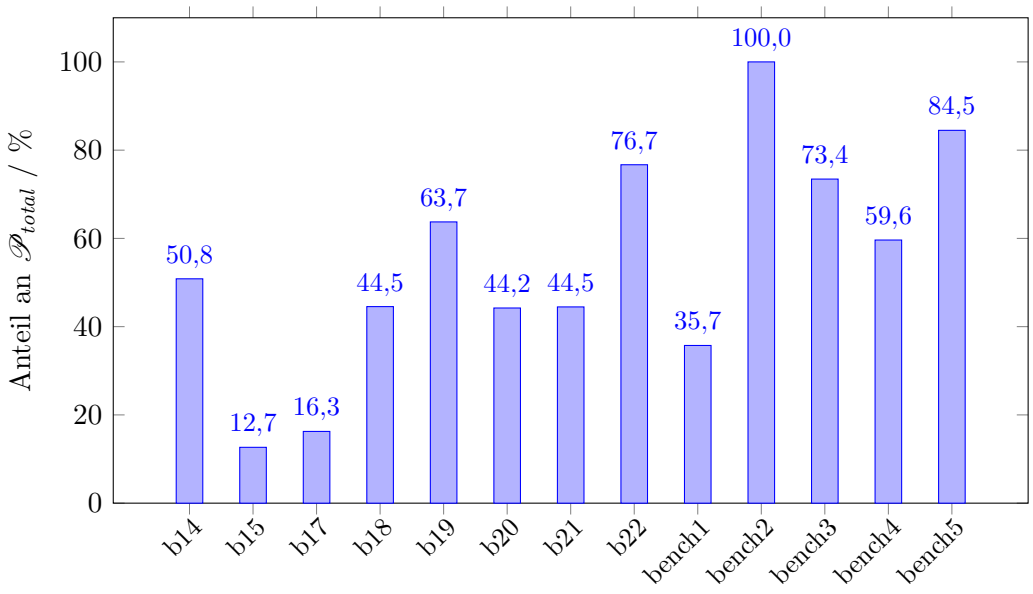


Abbildung 4.21: Anteil der gespeicherten Testmuster beim hybriden Verfahren

4.8.5 Laufzeitanalyse

Zum Abschluss der Ergebnisse der Simulationsstudie soll hier noch kurz auf die Laufzeiten der einzelnen Programme eingegangen werden. Die Berechnungen wurden dabei auf dem Paderborn Center for Parallel Computing (PC^2) durchgeführt, die Spezifikationen der verwendeten Rechner sind in Anhang A.6 gegeben. Die Laufzeiten der Synthese, Simulation und Frequenz- sowie Musterauswahl sind im Anhang A.6 gegeben. Sie lassen sich nur schwer in aussagekräftigen Diagrammen darstellen, da sie weit streuen. Beispielsweise bewegen sich die Laufzeiten von FAST-Simulation bei der ersten Simulation von 740 s (b15) bis 3,6 d (b19). Mit Ausnahme von b20, b21 und b22 ist die zweite Simulation meist deutlich schneller, da $\mathscr{P}_{Extra}$ im Vergleich zu $\mathscr{P}_{Initial}$ relativ klein ist. Ähnliches gilt auch für die Frequenz- und Musterauswahl. Das MHS-LUB-Verfahren ist typischweise sehr schnell (wenige Sekunden für alle Schaltungen mit Ausnahme von bench2), während die optimale Frequenzauswahl bis zu 2,5 h dauern kann (b22). Hier sei mit dem Verweis auf die Schaltung b17 nochmals angemerkt, dass MHS-Hypergraph durchaus sehr lange brauchen kann, um die optimale Lösung zu berechnen (vgl. Abschnitt 4.8.3).

Interessant ist, inwieweit das hybride Verfahren die Laufzeit der Frequenzauswahl beschleunigen kann. Abbildung 4.22 zeigt den erreichten Beschleunigungsfaktor der hybriden gegenüber der optimalen Auswahl. Grundlage dieser Abbildung sind die Tabellen A.16a und A.16b (beide Anhang A.6, Abschnitt „Standardentwurf“). Zu beachten ist hier, dass zusätzliches ATPG und FAST-Simulation nicht einbezogen wurden, da zum einen diese Schritte optional sind und zum anderen auch bei der optimalen Frequenzauswahl gemacht wurden. Die hybride Frequenzauswahl erreicht im Mittel bei den ITC-Schaltungen höhere Beschleunigungsfaktoren als bei den industriellen Schaltungen. Besonders hohe Faktoren konnten mit der b17-Schaltung, sowie der bench2-Schaltung erzielt werden. Bei bench1 und bench3 ist der Faktor allerdings kleiner als 1. Der Grund ist hier, dass die optimale Frequenzauswahl bereits so schnell ist, dass der Aufwand der hybriden Frequenzauswahl (Überprüfen mit T_{Fix}, Entfernen aller erkannten Fehler, Berechnung der Lösung mit den verbleibenden Fehlern) größer ist. Außerdem weisen diese Schaltungen die niedrigste HDF-Abdeckung unter T_{Fix} auf (vgl. Abbildung 4.16). Es sei außerdem angemerkt, dass lediglich 10 feste Beobachtungszeitpunkte für T_{Fix} verwendet wurden. In Anbetracht der Tatsache, dass die Gesamtzahl an benötigten Frequenzen typischerweise deutlich höher ist, kann man auch mit einem größeren T_{Fix} starten. Es ist wahrscheinlich, dass dann noch größere Beschleunigungsfaktoren erreicht werden, da die Problemgröße für die optimale Frequenzauswahl weiter verringert

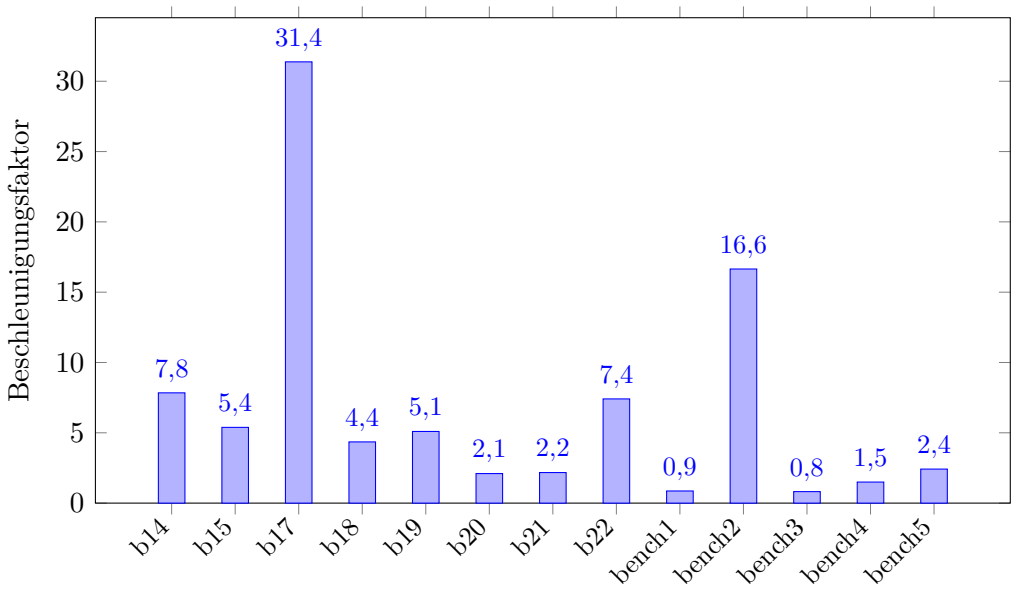


Abbildung 4.22: Beschleunigungsfaktor der hybriden Frequenzauswahl gegenüber der optimalen Auswahl

wird. Dadurch kann bei heutigen Schaltungen mit hunderten Millionen Fehlern der Beschleunigungsfaktor der hybriden Frequenzauswahl durchaus bedeutend groß werden.

Abbildung 4.23 zeigt abschließend die Aufteilung der gesamten Laufzeit auf die einzelnen Teilschritte (Synthese, ATPG, Simulation, Frequenz- und Musterauswahl). Hierbei wurde das hybride Verfahren zur Frequenzauswahl betrachtet.

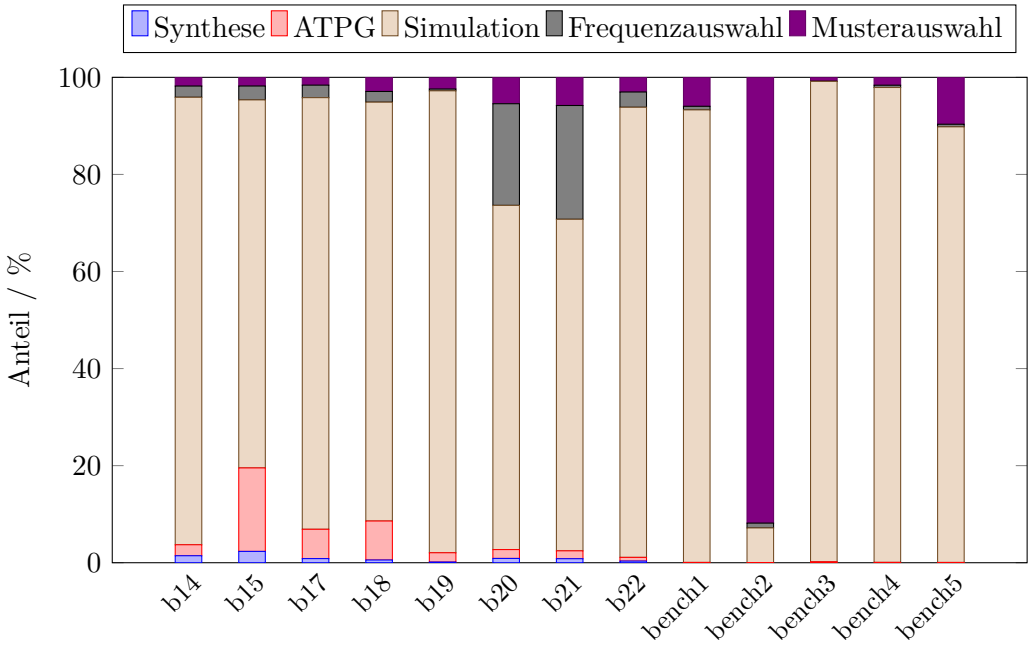


Abbildung 4.23: Anteile der Laufzeiten

Es zeigt sich, dass bei fast allen Schaltungen die Fehlersimulation den größten Anteil der Laufzeit ausmacht, was in Anbetracht des enormen Simulationsaufwandes wenig verwundert. Eine bemerkenswerte Ausnahme bildet wieder die Schaltung bench2. Bei dieser Schaltung dominiert die Auswahl der Testmuster für die FAST-Gruppen die Laufzeit. Die Ursache dafür ist die sehr kleine Basismenge an Testmustern, welche dafür sorgt, dass fast alle Muster bei fast allen Frequenzen verwendet werden, wie zuvor gezeigt wurde. Daher benötigt selbst die gierige Musterauswahl relativ lange, um passende Testsätze zu finden. Interessant sind auch die Schaltungen b20 und b21, bei denen die Frequenzauswahl einen signifikanten Anteil der Laufzeit ausmacht. Diese Schaltungen weisen im Verhältnis zu den anderen Schaltungen eine deutlich kürzere Simulationsdauer auf, weshalb der Anteil der Frequenzauswahl angestiegen ist.

4.9 Zusammenfassung und Ausblick

In diesem Kapitel wurde die Frequenzauswahl für den eingebauten Hochgeschwindigkeitstest beschrieben. Insbesondere wurde gezeigt, dass einfache Verfahren, wie etwa eine bestimmte Anzahl fest ausgewählter Frequenzen, möglicherweise zu Verlusten der Fehlereffizienz führen können. Als Ausweg wurden mehrere Varianten vorgestellt, um für einen gegebenen Testsatz Frequenzen so auszuwählen, dass jeder potentiell erkennbare Fehler tatsächlich erkannt werden kann. Mit dem optimalen Algorithms MHS-Hypergraph existiert eine Möglichkeit, die minimal notwendige Menge an Frequenzen zu bestimmen. Zudem wurde aber auch ein hybrides Verfahren vorgestellt, welches die Laufzeit der Frequenzauswahl verringert, ohne dabei stark von der optimalen Lösung abzuweichen.

Die Ergebnisse zeigen, dass für eine vollständige Fehlereffizenz viele verschiedene Frequenzen notwendig sind. Das mag zunächst wie ein schlechtes Ergebnis erscheinen, aber die Detailanalyse zeigt: Nur wenige Fehler sind besonders schwer zu erkennen. Daher kann man beispielsweise versuchen, mit dem bisherigen Verfahren zu starten, um danach die Anzahl an Frequenzen auf ein praktisches Maß zu begrenzen. Beschränkt man sich auf beispielsweise 20 verschiedene Frequenzen, so kann bei vielen Schaltungen (insbesondere den industriellen Schaltungen) bereits eine gute Fehlereffizienz erreicht werden. Für die verbleibenden wenigen Fehler kann man dann stark optimiertes ATPG verwenden, welches über das in dieser Arbeit verwendete kommerzielle ATPG hinausgeht. Denkbar sind hier wieder SAT-basierte Verfahren aufbauend auf PHAETON oder WaveSAT, um Testmuster für eine vorgegebene Zielfrequenz zu berechnen. Dadurch ist es vorstellbar, dass die Gesamtzahl an benötigten Frequenzen nicht weiter ansteigt und damit in einem praktikablen Rahmen bleibt.

Aber auch andere Verbesserungen sind vorstellbar. Beispielsweise kann die Musterauswahl für die FAST-Gruppen weiter optimiert werden. In dieser Arbeit wurde lediglich versucht, die Anzahl der Testmuster pro FAST-Gruppe zu minimieren. Es kommen aber auch andere Optimierungsziele in Betracht, etwa die Minimierung der Anzahl an X-Werten pro FAST-Gruppe [142]. Für die Testantwortkompaktierung ist das ein sinnvolles Ziel, denn X-Werte sind dort sehr problematisch, wie im nächsten Kapitel gezeigt wird.

5 Prüfgerechter Entwurf für den eingebauten Hochgeschwindigkeitstest

Dieses Kapitel stellt den zweiten Teil dieser Arbeit über den eingebauten Hochgeschwindigkeitstest (FAST) dar. Es stellt ein Verfahren vor, bei welchem die Prüfpfade einer Schaltung für FAST optimiert werden, also eine Methode des prüfgerechten Entwurfs (vgl. Abschnitt 2.3) [153, 154, 155]. Hintergrund ist hierbei die Testdatenauswertung im Selbsttest (vgl. Abschnitt 2.6.1), denn diese ist unter FAST aufgrund der X-Werte sehr herausfordernd. Mit den optimierten Prüfpfaden werden Lösungen unterstützt, denn die Prüfpfade werden so angeordnet, dass die X-Werte sich in wenigen Prüfpfaden konzentrieren, wodurch eine sehr einfache und effektive X-Maskierung möglich wird. Hierzu wird in diesem Kapitel eine neuartige, inkrementelle Maskierung vorgestellt, welche lediglich ein Schieberegister benötigt, um die Maske zu aktualisieren. Simulationsergebnisse zeigen, dass mit Prüfpfaden für FAST im Vergleich zu konventionellen Prüfpfaden deutlich höhere X-Reduktionsraten erreicht werden können. Wichtig ist, dass das hier vorgestellte Verfahren keine vollständige Prüfpfadkonfiguration darstellt. Die Algorithmen erzeugen lediglich zusätzliche Randbedingungen für die Synthesewerkzeuge, sodass diese dann die konkreten Prüfpfade zusammensetzen können. Dadurch lässt die Methode Spielraum für weitere Optimierungen.

Dieses Kapitel gliedert sich wie folgt. Abschnitt 5.1 stellt zunächst detailliert den Hintergrund und die Motivation des Verfahrens dar. Abschnitt 5.2 beschreibt dann, wie die optimierte Prüfpfade erzeugt werden können. Mit diesen Prüfpfaden lassen sich einfache Maskierungsalgorithmen entwickeln. Zwei Varianten, die

inkrementelle und die Grenzwertmaskierung, werden im Abschnitt 5.4 vorgestellt. Ein automatisierter Syntheseprozess wird in Abschnitt 5.3 präsentiert, Ergebnisse einer simulationsbasierten Studie werden daraufhin im Abschnitt 5.5 gezeigt. Den Abschluss des Kapitels bildet eine Zusammenfassung in Abschnitt 5.6.

5.1 Motivation

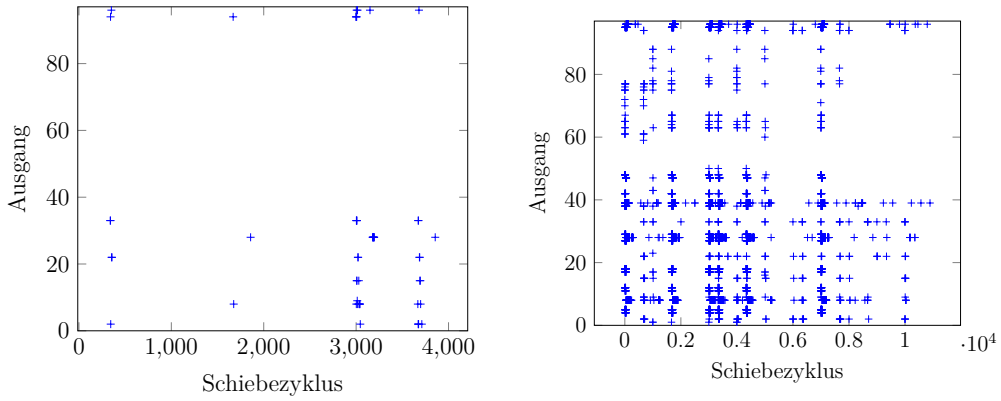
Die Testdatenauswertung ist unter FAST erschwert, denn wenn die Testsätze wie im vorangegangenen Kapitel beschrieben gewählt werden, kann es passieren, dass zu einem gewählten Beobachtungszeitpunkt nicht alle Schaltungsausgänge stabil sind. Das gilt wohlgemerkt auch im fehlerfreien Fall, da selbst hier die Verzögerungszeit des entsprechenden Pfades zu groß ist. Eine Simulation wird in diesem Fall den Ausgang symbolisch mit einem *unbekannten Logikwert (X-Wert)* belegen. In Kapitel 2.6.1 wurde bereits erwähnt, dass X-Werte problematisch für die Testantwortkompaktierung sind, besonders wenn eine Signaturberechnung, z. B. mit einem MISR, durchgeführt wird. Die Herausforderung ist hierbei dreifach:

1. Die X-Werte, welche durch FAST erzeugt werden, addieren sich zusätzlich auf andere, bereits existierende X-Quellen wie nicht initialisierte Speicherelemente, analoge Bauelemente, usw.

2. Es kann eine große Menge an X-Werten anfallen, wenn die Testfrequenz sehr aggressiv gewählt wird.

3. Die *Rate* an X-Werten kann je nach Frequenz und Testsatz stark variieren.

Diese Problematik soll anhand der industriellen Schaltung bench1 verdeutlicht werden. Abbildung 5.1 zeigt für diese Schaltung wie sich die X-Werte in den simulierten Testantworten verteilen, jeweils für doppelte (Abbildung 5.1a) und dreifache Betriebsfrequenz (Abbildung 5.1b). Für beide Frequenzen wurden mit dem im vorangegangenen Kapitel beschriebenen Verfahren geeignete Testsätze bestimmt.



(a) X-Rate bei $2f_{nom}$

(b) X-Rate bei $3f_{nom}$

Abbildung 5.1: X-Raten des bench1 für zwei Frequenzen (Prüfpfadausgänge)

In beiden Abbildungen stellt die horizontale Achse die Schiebezyklen beim Herausschieben der Testantworten der angelegten Testmuster dar (vergleichbar mit einer Zeitachse), während die vertikale Achse die Testausgänge der Prüfpfade aufführt. Ein blaues + in der Abbildung bedeutet, dass die Simulation bei einem Schiebezyklus und Testausgang einen X-Wert aufzeichnet. Es zeigt sich ein starker Unterschied in der Anzahl der herausgeschobenen X-Werte bei den verschiedenen Frequenzen.

Diese Varianz in den *X-Raten* ist problematisch für bisherige Kompaktierverfahren wie beispielsweise [116, 61]. Wenn diese Rate – wie bei FAST – variieren kann, so muss die maximale X-Rate betrachtet werden, nach Abbildung 5.1 müsste das Verfahren demnach für die dreifache Frequenz optimiert werden. Dies kann aber zu unnötigem Verlust an Fehlerinformationen führen, da bei geringeren Frequenzen unnötig viele Ausgänge ausgeblendet werden. Es gibt bereits Lösungen, die hohe und variierende X-Raten tolerieren können, beispielsweise der Ansatz von Singh et al. [104]. Dort werden Multiplexer eingesetzt, um ganze Mengen von X-behafteten Ausgängen auszublenden, ähnlich gehen auch Wohl et al. in [133] vor. Ein vergleichbarer Ansatz wird von Lien und Lee verfolgt [46, 44], bei welchem Zähler eingesetzt werden, sodass zu jedem Zeitpunkt nur einige wenige, X-freie Ausgänge betrachtet werden. Diese Verfahren leiden aber ebenfalls unter geringen Fehlereffizienzen, welche nur durch vergrößerte Testsätze wiederhergestellt werden können. Wird FAST als Herstellungstest (z. B. EDT, Kapitel 2.5) umgesetzt, ist das Problem etwas einfacher lösbar, dort können komplexe Maskierungsverfahren eingesetzt werden, um effizient mit X-Werten umgehen zu können [80, 15]. Zu-

sätzlich können auch X-behaftete Testantworten übertragen und später auf dem Testequipment selbst behandelt werden.

Im Selbsttest (oder bei einfachem Testequipment) müssen die X-Werte jedoch vollständig behandelt werden, damit die Testantworten erfolgreich ausgewertet werden können. Hier müssen in der Regel verschiedene Verfahren kombiniert werden, um optimale Fehlereffizienz bei minimaler Testdauer gewährleisten zu können, insbesondere, wenn mit hohen X-Raten gerechnet werden muss. In dieser Arbeit wird ein System bestehend aus drei Stufen betrachtet (vgl. Abbildung 5.2):

1. Eine Ausgangsmaskierung, um X-Werte direkt an den Prüfpfadausgängen auszublenden,

2. eine räumliche Kompaktierung, um die Dimension der Testantwortvektoren zu reduzieren und weitere X-Werte zu streichen und

3. eine zeitliche Kompaktierung, welche mit den verringerten X-Raten hohe Kompaktierraten erzielen kann.

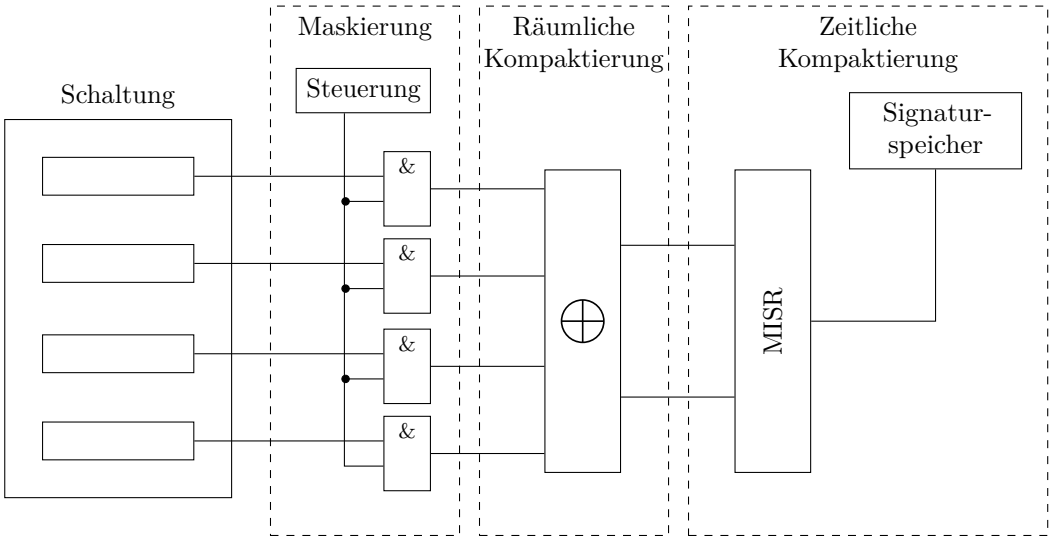
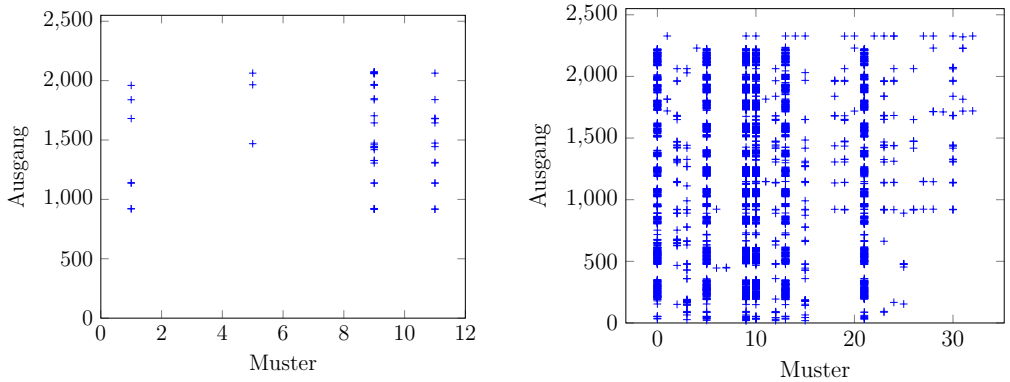


Abbildung 5.2: Testdatenauswertung im eingebauten Hochgeschwindigkeitstest

Für die X-Maskierung können einfache UND-Gatter, wie in Kapitel 2.6.1 beschrieben, verwendet werden. Die Kosten liegen hier vor allem im Steuerungsaufwand der Masken. Räumliche Kompaktierung bei variierenden X-Raten kann z. B. mit einem stochastischen Verfahren [107, 61] realisiert werden. Dabei werden die

XOR-Verknüpfungen der Ausgänge zufällig gewählt, Mitra hat in [61] eine optimale Wahrscheinlichkeit berechnet, um maximale X-Reduktion zu gewährleisten. Diese ist jedoch nur für eine feste X-Rate gültig, Sprenger und Hellebrand haben in [107] das Verfahren für FAST erweitert. Zum Schluss kann ein X-streichendes MISR [116] (vgl. Abschnitt 2.6.1) eingesetzt werden. In dieser Arbeit werden die Linearkombinationen der MISR-Bits allerdings nicht in Hardware berechnet, sondern die Signaturen werden in einem speziellen Signaturspeicher abgelegt, welcher nach Durchführung des Tests z. B. in Software ausgewertet werden kann.

Dieses Kapitel befasst sich mit der X-Maskierung. Diese kann theoretisch bereits sämtliche X-Werte aus den Testantworten streichen, ohne Informationsverlust hinnehmen zu müssen. Die praktische Realisierung einer solchen idealen Maskierung ist allerdings mit hohem Aufwand verbunden, da zu jedem Schiebetakt die Masken vorberechnet und abgespeichert werden müssen (vgl. z. B. [80, 15]). Da jedoch die verbleibenden zwei Stufen der Testdatenauswertung X-tolerant sind, ist das Maskieren aller X-Werte gar nicht notwendig. Daher kann eine stark vereinfachte, suboptimale Maskierung implementiert werden, welche versucht, so viele X-Werte wie möglich auszublenden. Hilfreich ist hierzu ein Rückblick auf das Beispiel der eingangs erwähnten Schaltung bench1 (Abbildung 5.1). Diese Abbildung kann auch anders dargestellt werden, indem statt der Prüfpfadausgänge alle pseudo-primären Ausgänge betrachtet werden. Dann ergibt sich Abbildung 5.3. Zu beachten ist bei dieser Abbildung, dass die horizontale Achse die Testmuster anzeigt.



(a) X-Rate bei $2f_{nom}$

(b) X-Rate bei $3f_{nom}$

Abbildung 5.3: X-Raten des b18 für zwei Frequenzen (pseudo-primäre Ausgänge)

Bei einer genaueren Analyse zeigt sich, dass die X-Werte nicht zufällig über die Ausgänge verteilt sind. Es lassen sich Muster erkennen – bestimmte Ausgänge zeichnen keine oder kaum X-Werte auf, während andere Ausgänge beinahe ausschließlich X-Werte aufzeichnen. Insbesondere scheinen dieselben Ausgänge oftmals auch bei denselben Mustern einen X-Wert aufzuzeichnen. Diese Beobachtung wurde unter anderem auch von Czysz et al. in [17] gemacht. Die Autoren nutzten die Korrelationen aus, um ein effizientes Maskierungs- und Kompaktierungssystem zu entwerfen. Diese Arbeit unterscheidet sich von [17] allerdings darin, dass nicht nur die Kompaktierung diese Zusammenhänge ausnutzt, sondern dass die Verteilungen explizit abgeschätzt werden und als Metrik zur Optimierung der Prüfpfade benutzt werden. Das hier vorgestellte Verfahren ist vergleichbar mit dem Ansatz von Wohl et al. [131] und Chandra et al. [14], welche sogenannte „X-Pfade“ bestimmen. Aber der in dieser Arbeit vorgestellte Ansatz geht noch über diese Arbeiten hinaus, da sämtliche Prüfzellen der Schaltung betrachtet werden statt nur einige wenige „X-Zellen“, wie in [131]. Zudem betrachten die Arbeiten keine X-Werte, welche durch Verletzung des zeitlichen Verhaltens entstehen. Die Abschätzung von „klassischen“ X-Werten, die durch nicht intialisierte Speicherelemente, Tristate-Elemente usw. entstehen, ist wesentlich einfacher, da kommerzielle Synthesewerkzeuge mitunter Methoden zur Verfügung stellen, um solche Quellen zu identifizieren [14]. Die in dieser Arbeit betrachteten X-Werte hängen vom Testsatz und der jeweiligen Frequenz ab, sind also grundsätzlich schwer vorherzusagen. Weiterhin betrachten [14, 131] lediglich eine feste X-Rate unter Betriebsbedingungen, daher ist es vergleichsweise einfach, eine optimierte Prüfpfadkonfiguration zu finden. FAST verwendet allerdings mehrere Frequenzen mit eigenen Testsätzen, die jeweils eigene X-Raten induzieren, daher sind die bisherigen Lösungen nicht für FAST geeignet. Aus diesen Gründen ist das in diesem Kapitel betrachtete Problem ungleich schwerer zu lösen als vergleichbare Probleme aus der Literatur und der hier beschiebene Lösungsansatz stellt in der Tat eine Neuheit dar.

5.2 Optimierte Prüfpfade für den Hochgeschwindigkeitstest

In diesem Abschnitt wird eine Methode zur Erzeugung von Randbedingungen für Synthesewerkzeuge beschrieben, um die besagten optimierten Prüfpfade zu erzeugen [153, 155]. Dabei wird weder eine existierende Prüfpfadkonfiguration verändert, noch wird die Anordnung der Prüfzellen in den Prüfpfaden vorgegeben. Die Randbedingungen geben lediglich eine Partitionierung der Prüfzellen an, welche zusammen in einen Prüfpfad konfiguriert werden sollen. Das Verfahren ist allgemein genug, um bei beliebigen Situationen, in denen korrelierte X-Verteilungen auftreten, eingesetzt zu werden, beispielsweise Niedrigfrequenztests oder Tests mit variierender Temperatur.

5.2.1 Übersicht und Formalisierung des Problems

Grundlage des Verfahrens bildet das *X-Profil* einer Prüfzelle, welches die X-Verteilung beschreibt. Korrelationen zwischen Prüfzellen können als Differenzen der X-Profile aufgefasst werden – eine geringe Differenz der X-Profile impliziert eine hohe Korrelation der X-Verteilungen. Die Differenz der X-Profile wird auch als Distanz bezeichnet. Im Folgenden werden lediglich pseudo-primäre Schaltungsausgänge betrachtet, zur Vereinfachung wird allerdings lediglich allgemein von Schaltungsausgängen gesprochen.

Definition 5.1. *Seien O die Ausgänge einer Schaltung und T eine Menge an Beobachtungszeitpunkten. Die Funktion $d : O \times O \times T \rightarrow [0, 1]$ wird als* Distanzfunktion der X-Profile *bezeichnet.*

In dieser Arbeit werden die Prüfzellen in Partitionen unterteilt, aus welchen das Synthesewerkzeug später die konkreten Prüfpfade bildet. Der Operator $\mathrm{Pot}(U)$ bezeichne im Folgenden die Potenzmenge einer Menge U.

Definition 5.2. *Sei O eine Menge an Schaltungsausgängen. Dann ist $\mathscr{S} \subset \mathrm{Pot}(O)$ eine* Partition *von O, wenn $S_i \cap S_j = \emptyset$ für alle $S_i \neq S_j$, $S_i, S_j \in \mathscr{S}$ gilt.*

$\mathscr{S}'$ ist eine vollständige Partition *von O, wenn $\mathscr{S}'$ eine Partition von O ist und zusätzlich $\bigcup_{S' \in \mathscr{S}'} S' = O$ gilt.*

Eine Partition besteht aus paarweise disjunkten Teilmengen von O, eine vollständige Partition enthält zusätzlich alle Elemente aus O. Mit dieser Definition kann das Problem der Prüfzellengruppierung als Partitionierungsproblem definiert werden.

Problem 5.1 (Optimale Prüfzellengruppierung)**.** *Seien O die Menge an Schaltungsausgängen, T eine Menge an Beobachtungszeitpunkten, d eine Distanzfunktion und $k \in \mathbb{N}$. Bestimme eine vollständige Partition $\mathscr{S} = \{S_1, \ldots, S_k\}$ von O, sodass $\max_{i,j}\{|S_i| - |S_j|\} = 1$ gilt und zusätzlich*

$$c = \sum_{S \in \mathscr{S}} \sum_{t \in T} \sum_{\substack{o_1, o_2 \in S \\ o_1 \neq o_2}} d(o_1, o_2, t) \tag{5.1}$$

minimal wird.

Die Bedingung $\max_{i,j}\{|S_i| - |S_j|\} = 1$ fordert, dass die Prüfpfade annähernd gleiche Länge haben müssen Das ist eine in der Industrie gebräuchliche Anforderung, da der längste Prüfpfad entscheidend die Testdauer beeinflusst.

5.2.2 Bestimmung der X-Profile

Um Problem 5.1 zu lösen, muss eine Distanzfunktion nach Definition 5.1 aufgestellt werden. Dazu müssen die X-Profile der Prüfzellen bestimmt werden, im Folgenden werden dafür zwei Varianten vorgestellt. Die *topologische X-Wahrscheinlichkeit* analysiert die Schaltungstopologie und versucht darüber, Rückschlüsse auf die X-Wahrscheinlichkeiten zu ziehen. Im Gegensatz dazu arbeitet die *simulationsbasierte X-Verteilung* mit einem konkreten Testsatz und berechnet die X-Werte und deren Verteilung auf die Prüfzellen direkt. Selbstverständlich können die X-Profile auch auf andere Art bestimmt werden, es muss lediglich eine Distanzfunktion der X-Profile nach Definition 5.1 gebildet werden können. Dadurch wird der Gruppierungsalgorithmus unabhängig von der eigentlichen Distanzfunktion, und

es können leicht anders definierte X-Profile entworfen und eingesetzt werden, was eine hohe Flexibilität des Verfahrens sicherstellt.

Topologische X-Wahrscheinlichkeit

Die Annahme der topologischen X-Wahrscheinlichkeit ist, dass Prüfzellen, an denen viele lange Pfade enden, eine erhöhte Wahrscheinlichkeit für X-Werte aufweisen.

Definition 5.3. *Seien o ein Ausgang einer Schaltung, t_{nom} der nominelle Beobachtungszeitpunkt. Zudem seien $n(o)$ die Anzahl aller topologischen Pfade, die an o enden und $n(o,t)$ die Anzahl aller topologischen Pfade, die an o enden und eine Verzögerungszeit gleich t besitzen. Die* topologische X-Wahrscheinlichkeit *von o zum Beobachtungszeitpunkt $t < t_{nom}$ bestimmt sich über*

$$P_{top}[X\text{-}Wert\ an\ o \mid t] = \frac{\sum_{j=t}^{t_{nom}} n(o,j)}{n(o)}. \tag{5.2}$$

Die topologische X-Distanz *zweier Ausgänge o_1 und o_2 zu einem Beobachtungszeitpunkt t ist definiert als*

$$d_{top}(o_1, o_2, t) = |P_{top}[X\text{-}Wert\ an\ o_1 \mid t] - P_{top}[X\text{-}Wert\ an\ o_2 \mid t]| \tag{5.3}$$

Gleichung (5.3) definiert eine zu Definition 5.1 konforme Distanzfunktion, es sei jedoch angemerkt, dass Gleichung (5.2) keine echte X-Wahrscheinlichkeit definiert. Eine solche würde eine vollständige statistische Analyse des zeitlichen Verhaltens der Schaltung erfordern [109], was mit enormem Aufwand verbunden ist. Die topologische X-Wahrscheinlichkeit lässt sich einfacher berechnen, ist aber entsprechend lediglich eine Schätzung der echten X-Wahrscheinlichkeit. Um $d_{top}(o_1, o_2, t)$ berechnen zu können, müssen die Verzögerungszeiten aller topologischen Pfade, die an einem Ausgang o enden, bekannt sein.

Definition 5.4. *Seien t_{nom} der nominelle Beobachtungszeitpunkt einer Schaltung, o ein Gatterausgang und $\tau > 0$. Das Tupel $B(o) = (b_0(o), \ldots, b_{k-1}(o))$ mit $k = \lceil t_{nom}/\tau \rceil$ wird als* Verzögerungshistogramm für o *bezeichnet. Eine Klasse*

$b_i(o)$ in $B(o)$ beschreibt ein zeitliches Intervall $[i\tau, (i+1)\tau)$, ihr Wert berechnet sich über

$$b_i(o) = \sum_{t=i\tau}^{(i+1)\tau-1} n(o,t). \tag{5.4}$$

Gleichung (5.2) kann über die Verzögerungshistogramme umgeschrieben werden zu

$$P_{top}[\text{X-Wert an } o \mid t] \leq \frac{\sum_{i=\lfloor t/\tau \rfloor}^{k-1} b_i(o)}{\sum_{i=0}^{k-1} b_i(o)}. \tag{5.5}$$

Dies ist nur noch eine Ungleichung, da auch solche Pfade mitgezählt werden, deren Verzögerungszeit kleiner als t ist, die aber dennoch in die gleiche Klasse einsortiert werden wie die Pfade mit Verzögerungszeit t. Wählt man τ klein, so kann dieser Fehler minimiert werden. Die Berechnung der Verzögerungshistogramme $\mathscr{B} = (B(o_1), \ldots, B(o_k))$ für k Ausgänge ist als Pseudocode in Algorithmus 5.1 dargestellt.

Algorithmus 5.1 Propagiere-Histogramme

Eingabe: Gatter G, t_{nom}, τ
Ausgabe: $\mathscr{B}$

```
1: Initialisiere alle Gatter mit 0-Histogrammen
2: für alle Gatter g ∈ G  // In topologischer Ordnung
3:     d = Gatterverzögerung von g
4:     z = ⌈d/τ⌉ // Gatterverzögerung als Verschiebung in Klassen
5:     für alle Gattereingänge i von g
6:         B(i) = Verzögerungshistogramm von i
7:         für j = 0 bis ⌊t_nom/τ⌋
8:             b_{j+z}(g) = b_{j+z}(g) + b_j(i)
9: B = Verzögerungshistogramme aller Ausgänge
10: rückgabe B
```

Dieser Algorithmus benötigt die Menge G von Gattern und für jedes Gatter die Signalverzögerungen. Im Pseudocode wird aus Gründen der Einfachheit ein Einheitsverzögerungsmodell benutzt, komplexere Verzögerungen wie individuelle Signalflankenverzögerungen lassen sich aber leicht integrieren. Alle Gatter werden topologisch sortiert durchlaufen, d. h. ein Gatter kann erst dann bearbeitet werden, wenn zuvor alle treibenden Gatter bearbeitet wurden. Dies setzt implizit voraus, dass keine kombinatorischen Rückkopplungen in der Schaltung existieren.

Beispiel 5.1. Sei g ein UND-Gatter mit zwei Eingängen g_1 und g_2 und einer Verzögerungszeit von 5 ps. Es sei $\tau = 1\,\text{ps}$, wodurch $z = 5$ gilt (Zeile 4 in Algorithmus 5.1). Die Verzögerungshistogramme besitzen 10 Klassen und die Histogramme der Eingänge seien

$$B(g_1) = (0, 4, 1, 2, 0, 0, 0, 0, 0, 0),$$
$$B(g_2) = (0, 1, 0, 1, 2, 2, 0, 0, 0, 0).$$

Für das resultierende Verzögerungshistogramm $B(g)$ gilt $b_{i+z}(g) = b_i(g_1) + b_i(g_2)$, $i = 0, \ldots, 9$, sodass sich ein Ergebnis von $B(g) = (0, 0, 0, 0, 0, 5, 1, 3, 2, 2)$ ergibt.

Algorithmus 5.1 bearbeitet alle Gatter in topologischer Ordnung. In diesem Fall kann man außerdem jedem Gatter g eine topologische Tiefe zuweisen, welche der maximalen Anzahl an Gattern entspricht, über welche g von einem Schaltungseingang erreicht werden kann. Dadurch lässt sich die Netzliste im Speicher effizient als Matrix darstellen. Eine schnellere – und auch in dieser Arbeit verwendete – Methode zur Berechnung der Verzögerungshistogramme nutzt diese Darstellung aus, da nun gatterparallel gearbeitet werden kann. Alle Gatter derselben topologischen Tiefe können nebenläufig aktualisiert werden, da sie nicht voneinander abhängen können.

Simulationsbasierte X-Verteilung

Die Alternative zur topologischen X-Wahrscheinlichkeit liegt darin, einen konkreten Testsatz zu simulieren und die entstandene Verteilung der X-Werte zu betrachten. In diesem Sinne entspricht dieser Ansatz der direkten Auswertung der X-Verteilung aus Abbildung 5.3. Dieser Ansatz ist ebenfalls eine Heuristik, da zum einen zum Zeitpunkt der Simulation der Syntheseprozess noch nicht abgeschlossen ist – es existieren z. B. noch keine Prüfpfade –, zum anderen existiert zu diesem frühen Zeitpunkt im Syntheseablauf typischweise noch kein Testsatz, sodass auch keine direkte Optimierung möglich ist, da man sich beispielsweise mit Zufallsmustern behelfen muss. Es zeigte sich jedoch in [155], dass auch mit Zufallsmustern die X-Profile zu vergleichbaren Ergebnissen wie bei der topologischen X-Wahrscheinlichkeit führen. Die Stärke dieses Ansatzes könnte aber

zutage treten, wenn ein bestehender Entwurf optimiert werden muss. Eine solche Situation ist denkbar, schließlich können während des Entwurfprozesses immer wieder Änderungen an der Schaltung vorgenommen werden. In diesem Fall können bereits Testsätze existieren, sodass die Prüfpfade nachträglich noch optimiert werden können.

Definition 5.5. *Sei o ein Schaltungsausgang, $\mathscr{P} = \{P_1, \ldots, P_k\}$ ein Testsatz und t ein Beobachtungszeitpunkt. Zudem bezeichne $a_{o,i} \in \{0, 1, X\}$ das Testantwortbit von o unter P_i und t, $1 \leq i \leq k$. Der X-*Vektor $\mathbf{s_o} = \begin{pmatrix} s_{o,1} & \cdots & s_{o,k} \end{pmatrix}$ *von o unter $\mathscr{P}$ und t ist definiert über*

$$s_{o,i} = \begin{cases} 1, & \textit{wenn } a_{o,i} = X, \\ 0, & \textit{sonst}, \end{cases} \tag{5.6}$$

für $1 \leq i \leq k$.

Definition 5.6. *Seien o_1 und o_2 zwei Schaltungsausgänge, $\mathscr{P}$ ein Testsatz, t ein Beobachtungszeitpunkt, sowie $\mathbf{s_1}$ und $\mathbf{s_2}$ die X-Vektoren von o_1 bzw. o_2 unter $\mathscr{P}$ und t. Dann bestimmt sich die* X-Vektordistanz *zwischen o_1 und o_2 unter $\mathscr{P}$ und t zu*

$$d_x(o_1, o_2, t) = \frac{|w(\mathbf{s_1}) - w(\mathbf{s_2})|}{|\mathscr{P}|}, \tag{5.7}$$

wobei $w(\mathbf{s})$ das Hamminggewicht von $\mathbf{s}$ bezeichne.

Die Berechnung von $d_x(o_1, o_2, t)$ erfordert zeitlich genaue Simulation eines Testsatzes, beispielsweise mit dem GPU-basierten Simulator aus [98]. Die Berechnung ist daher ähnlich aufwändig wie die Berechnung der topologischen X-Distanz.

5.2.3 Graph-basierte Prüfzellengruppierung

In [155] wurde ein Algorithmus zur Lösung des Problems 5.1 präsentiert, welcher das Problem in ein Graphproblem konvertiert und mit effizienten Algorithmen löst. Die wesentliche Datenstruktur bildet dabei der Prüfzellengraph.

Definition 5.7. *Seien O die Ausgänge einer Schaltung, T eine Menge an Beobachtungszeitpunkten, d eine Distanzfunktion, sowie $\lambda_{Konnektivität} \in [0, 1]$. $G_t = (V, E_t)$ bezeichnet einen* Prüfzellengraphen *für O unter t, bei dem jeder Ausgang $o \in O$ einem Knoten $v \in V$ entspricht und (v_1, v_2) eine Kante in E_t ist, wenn $d(o_1, o_2, t) \leq \lambda_{Konnektivität}$ für zwei Ausgänge $o_1, o_2 \in O$ gilt.*

Aus Gründen der Einfachheit wird nachfolgend nicht mehr zwischen Knoten eines Prüfzellengraphs und den entsprechenden Schaltungsausgängen unterschieden. $\lambda_{Konnektivität}$ wird als *Konnektivitätsparameter* der Prüfzellengruppierung bezeichnet.

Beispiel 5.2. Gegeben seien fünf Ausgänge $o_1, \ldots, o_5$. Die paarweisen Distanzen der entsprechenden X-Profile sind für einen Beobachtungszeitpunkt t in folgender

Distanzmatrix aufgelistet:

$$\mathbf{\Delta_t} = \begin{array}{c} \\ o_1 \\ o_2 \\ o_3 \\ o_4 \\ o_5 \end{array} \begin{array}{c} \begin{array}{ccccc} o_1 & o_2 & o_3 & o_4 & o_5 \end{array} \\ \begin{pmatrix} 0 & 0{,}25 & 0{,}7 & 0{,}15 & 0{,}6 \\ 0{,}25 & 0 & 0{,}1 & 0{,}65 & 0{,}2 \\ 0{,}7 & 0{,}1 & 0 & 0{,}4 & 0{,}2 \\ 0{,}15 & 0{,}65 & 0{,}4 & 0 & 0{,}5 \\ 0{,}6 & 0{,}2 & 0{,}2 & 0{,}5 & 0 \end{pmatrix} \end{array}.$$

Diese Matrix ist symmetrisch, da $d(o_i, o_j, t) = d(o_j, o_i, t)$ für beliebige i und j gilt. Setzt man nun $\lambda_{Konnektivität} = 0{,}3$, so ergibt sich der Prüfzellengraph aus Abbildung 5.4.

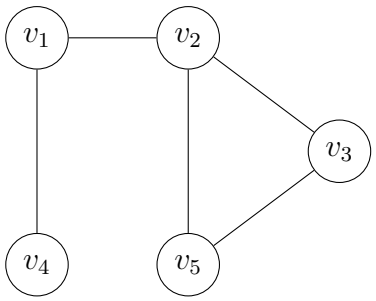


Abbildung 5.4: Beispiel eines Prüfzellengraphs

Mit dem Prüfzellengraph wandelt sich das Problem 5.1 in die Suche nach vollständig verbundenen Teilgraphen bzw. Cliquen im Prüfzellengraph um.

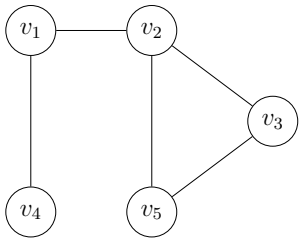
Definition 5.8. *Sei $G = (V, E)$ ein Graph. $C \subseteq V$ ist eine* Clique *von G, wenn der durch C induzierte Teilgraph von G vollständig ist, d. h. für alle Knotenpaare $u, v \in C$ existiert eine Kante $(u, v) \in E$.*

Eine Clique in einem Prüfzellengraph ist der ideale Kandidat für einen Prüfpfad, denn alle enthaltenen Ausgänge besitzen X-Profile mit paarweiser Distanz geringer als $\lambda_{Konnektivität}$. Es ist jedoch hilfreich, nicht direkt mit den einzelnen Prüfzellengraphen für die gegebenen Beobachtungszeitpunkte zu starten, sondern zunächst den übergeordneten konjunktiven Prüfzellengraphen zu analysieren.

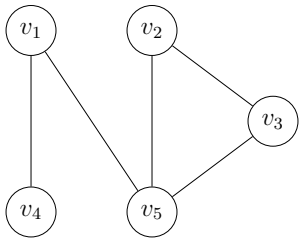
Definition 5.9. *Seien O die Ausgänge einer Schaltung, T eine Menge an Beobachtungszeitpunkten und $G_t = (V, E_t)$ der Prüfzellengraph für einen Beobach-*

tungszeitpunkt $t \in T$. $G_T = (V, E_T)$ bezeichnet den konjunktiven Prüfzellengraph, *in dem für jeden Ausgang $o \in O$ ein Knoten $v \in V$ existiert und $(v_1, v_2) \in E_T$, wenn $(v_1, v_2) \in E_t$ für alle Prüfzellengraphen G_t gilt.*

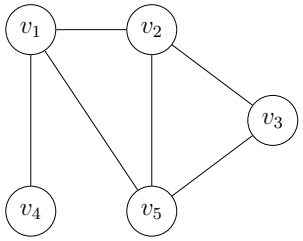
Beispiel 5.3. Gegeben sei der Prüfzellengraph aus Beispiel 5.2, für einen Beobachtungszeitpunkt t_1. Abbildung 5.5 zeigt nochmals den Graphen (Abbildung 5.5a), zusätzlich sind zwei weitere Prüfzellengraphen für die Beobachtungszeitpunkte t_2 und t_3 in den Abbildungen 5.5b bzw. 5.5c gegeben. Der daraus resultierende konjunktive Prüfzellengraph ist in Abbildung 5.5d dargestellt.



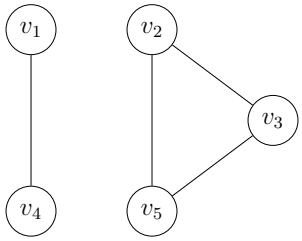
(a) Prüfzellengraph für t_1



(b) Prüfzellengraph für t_2



(c) Prüfzellengraph für t_3



(d) Konjunktiver Prüfzellengraph

Abbildung 5.5: Prüfzellengraphen für drei Beobachtungszeitpunkte

Eine Clique des konjunktiven Prüfzellengraphen ist damit auch immer eine Clique aller einzelnen Prüfzellengraphen und beschreibt somit eine Gruppe von Prüfzellen, welche über alle gegebenen Beobachtungszeitpunkte hinweg eine hohe Korrelation der X-Verteilungen (bestimmt durch $\lambda_{Konnektivität}$) aufweisen.

Übersicht der Prüfzellengruppierung

Der Ablauf der Graph-basierten Prüfzellengruppierung ist in Abbildung 5.6 dargestellt.

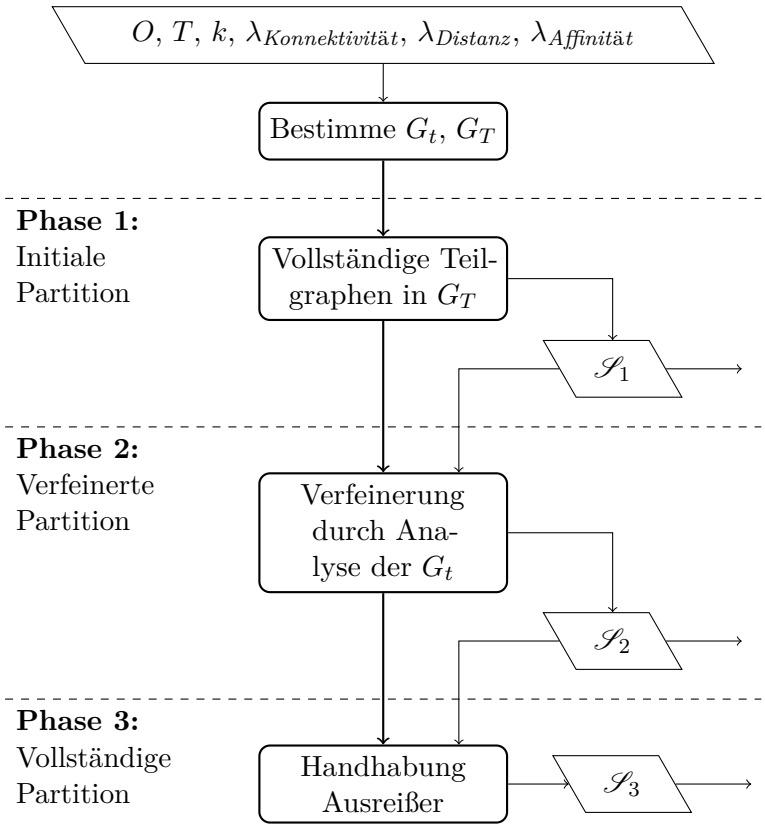


Abbildung 5.6: Ablauf der Graph-basierten Prüfzellengruppierung

Die Eingaben des Algorithmus sind die Schaltungsausgänge O, eine Menge an Beobachtungszeitpunkten T, die Anzahl zu erzeugender Partitionen k, sowie drei Parameter $\lambda_{Konnektivität}$, $\lambda_{Distanz}$ und $\lambda_{Affinität}$. $\lambda_{Konnektivität}$ ist der Konnektivitätsparameter, um Knoten in den Prüfzellengraphen zu verbinden, $\lambda_{Distanz}$ und $\lambda_{Affinität}$ sind Parameter für die Optimierung der Zuweisung von Ausgängen zu Gruppen. Der Algorithmus arbeitet grundsätzlich in drei verschiedenen Phasen. Zunächst wird im konjunktiven Prüfzellengraphen G_T nach Cliquen gesucht, welche als Ausgangsmengen für die Prüfzellengruppen dienen. Verbleiben dann noch nicht zugewiesene Prüfzellen, werden die einzelnen Prüfzellengraphen G_t analysiert, um eine optimierte Zuweisung zu finden. Sind dann immer noch einige Prüfzellen übrig, welche keiner Gruppe zugewiesen wurden, können diese z. B. genutzt werden, um die Größe der Prüfzellengruppen anzugleichen, sodass im späteren Syntheseschritt Prüfpfade gleicher Länge erzeugt werden.

Es ist nicht notwendig, alle drei Phasen des Algorithmus tatsächlich durchzuführen. Vielmehr kann der Anwender bestimmen, wie feingranular die Prüfzellengrup-

pierung sein soll. Beispielsweise kann es ausreichen, lediglich die erste Phase durchzuführen, um besonders gut zusammenpassende Prüfzellen zu finden. Alle weiteren Prüfzellen können dann normal vom Synthesewerkzeug gehandhabt, oder anhand von anderen Optimierungszielen (etwa Optimierung des Verbindungsaufwandes der Prüfpfade) zu den Prüfpfaden hinzugefügt werden. Auch kann die Gruppierung nach der zweiten Phase abgebrochen werden, sodass lediglich die Ausreißer nicht zu Prüfzellengruppen hinzugefügt wurden und beispielsweise dem Synthesewerkzeug überlassen werden. Es ist nicht notwendig, dass alle Ausgänge der Schaltung tatsächlich dem Algorithmus übergeben werden müssen, möglicherweise ist es sinnvoll, bestimmte Prüfzellen gezielt von der Gruppierung auszuschließen. So hat der Anwender gute Kontrolle darüber, wie stark der Algorithmus in die Schaltung tatsächlich eingreifen darf. Im Extremfall arbeitet dieses Verfahren dann wie die Bestimmung der „X-Pfade“aus [131] oder [14].

Phase 1: Initiale Partition In der ersten Phase wird nach n disjunkten Cliquen im konjunktiven Prüfzellengraphen $G_T = (V, E_T)$ gesucht, aus denen sich die initiale Partition $\mathscr{S}_1$ bildet.

Problem 5.2. *Seien $G = (V, E)$ ein Graph, $k > 0$ sowie $l > 1$. Finde k Cliquen $C_i, \ldots, C_k \subset V$, sodass $0 < |C_i| \leq l$ sowie $C_i \cap C_j = \emptyset$ für alle $i = 1, \ldots, k$, $j = 1, \ldots, k$, $i \neq j$ gelten.*

Eine einfache Lösung des Problems 5.2 besteht darin, Cliquen maximaler Größe zu bestimmen und in Mengen der Größe l aufzuteilen. Dies lässt sich beispielsweise mit den Verfahren von Tsukiyama et al. [118] oder Konc und Janežič [41] bewerkstelligen, was aber eine hohe Komplexität und damit verbundene Laufzeit hat, da das zugrunde liegende Problem (Bestimmung maximaler Cliquen) NP-vollständig ist [37]. In [155] wurde deshalb eine einfache Heuristik entwickelt, welche mit einem einzelnen Knoten des Graphen startet und davon ausgehend versucht, einen vollständigen Teilgraphen mit gegebener Größe „wachsen“ zu lassen. Algorithmus 5.2 zeigt den Pseudocode des Verfahrens.

Die Menge L beinhaltet alle noch nicht zugewiesenen Knoten, l gibt die Zielgröße der zu bestimmenden Teilgraphen an. Es werden n verschiedene Gruppen erzeugt, indem zunächst ein Knoten aus L ausgewählt wird. Dies kann ein zufälliger Knoten

Algorithmus 5.2 Finde-vollständige-Teilgraphen

Eingabe: $G_T = (V, E_T)$, k
Ausgabe: $\mathscr{S}_1$
1: $L = V$
2: $l = \lceil |V|/k \rceil$
3: $\mathscr{S}_1 = \emptyset$
4: **für** $i = 1$ **bis** k
5: $v =$ wähle Startknoten aus L
6: $S =$ BESTIMME-CLIQUE($\{v\}$, L, l)
7: $\mathscr{S}_1 = \mathscr{S}_1 \cup \{S\}$
8: $L = L \backslash S$
9: **rückgabe** $\mathscr{S}_1$

sein, aber vorteilhaft ist es, z. B. einen Knoten mit maximalem Grad zu wählen, welcher möglichst wenige gemeinsame Kanten mit bereits erzeugten Teilgraphen aus $\mathscr{S}_1$ hat, um eine bessere Verteilung der Gruppen auf den Graphen zu erreichen. Die Methode Bestimme-Clique berechnet dann die Clique. Hierbei ist zu beachten, dass nicht garantiert werden kann, dass das Ergebnis eine Kardinalität von l hat, denn möglicherweise kann keine so große Clique mehr gefunden werden. Der Pseudocode von Bestimme-Clique ist in Algorithmus 5.3 gegeben. Nachdem die Gruppe bestimmt wurde, wird sie $\mathscr{S}_1$ hinzugefügt und alle ausgewählten Knoten werden aus L entfernt.

Algorithmus 5.3 Bestimme-Clique

Eingabe: S, L, l
Ausgabe: S
1: **solange** $|S| < l$
2: $U =$ Bestimme Clique-Nachbarschaft von S
3: **wenn** $U = \emptyset$ **dann**
4: **rückgabe** S
5: $u =$ Knoten aus U mit maximalem Grad
6: $S = S \cup \{u\}$
7: **rückgabe** S

Bestimme-Clique versucht iterativ S zu vergrößern, bis die Zielgröße l erreicht wurde oder kein weiterer Knoten hinzugefügt werden kann. In jedem Schritt wird die *Clique-Nachbarschaft* U von S bestimmt. Das ist eine Teilmenge der Knoten von L, welche mit allen Knoten aus S verbunden sind (Die Knoten aus U müssen jedoch nicht untereinander verbunden sein). Ein Hinzufügen eines Knotens $u \in U$ zu S

garantiert, dass S weiterhin eine Clique im konjunktiven Prüfzellengraph bleibt. Es wird aus U der Knoten mit maximalem Grad ausgewählt, zu S hinzugefügt und der Prozess beginnt von vorne.

Phase 2: Verfeinerte Partition Verbleiben nach dem Bestimmen von $\mathscr{S}_1$ noch Knoten in V, die keiner Gruppe zugewiesen wurden, so wird in dieser Phase versucht, eine Zuweisung der verbleibenden Knoten zu Gruppen aus $\mathscr{S}_1$ zu finden, ohne die Kosten zu stark zu erhöhen.

Definition 5.10. *Seien t ein Beobachtunszeitpunkt, $G_t = (V, E_t)$ ein Prüfzellengraph für t, $\mathscr{S}$ eine Partition von V und d eine Distanzfunktion. Die* Affinität *eines Knotens v zu einer Menge $S \in \mathscr{S}$ zum Beobachtungszeitpunkt t ist definiert als*

$$a_t(v, S) = \frac{|\{(u, v) \mid u \in S,\ (u, v) \in E_t\}|}{|S|}. \tag{5.8}$$

Beispiel 5.4. Gegeben sei der Prüfzellengraph aus Beispiel 5.2 (Abbildung 5.4 bzw. Abbildung 5.5a). Die Affinität von v_1 zur Menge $S = \{v_2, v_3, v_5\}$ bestimmt sich zu

$$a_t(v_1, S) = \frac{|\{(v_1, v_2)\}|}{|\{v_2, v_3, v_5\}|} = \frac{1}{3},$$

da v_1 genau eine Kante zu einem der Knoten in S besitzt. Analog gilt $a_t(v_4, S) = 0$.

Gleichung (5.8) berücksichtigt einen Sonderfall nicht. Es kann passieren, dass innerhalb der Menge S ein Knoten u vorhanden ist, für den $d(v, u, t) \gg \lambda_{Konnektivität}$ gilt, d. h. u und v haben eine große Distanz, was im Prüfzellengraph jedoch nicht modelliert ist. In einem solchen Fall sollte v nicht zu S hinzugefügt werden. Dies kann über den *Distanzparameter* $\lambda_{Distanz}$ gesteuert werden, die Affinität berechnet sich dann über

$$a_t(v, S, \lambda_{Distanz}) = \begin{cases} -1, & \exists u \in S, :\ d(u, v, t) > \lambda_{Distanz}, \\ a_t(v, S), & \text{sonst.} \end{cases} \tag{5.9}$$

Beispiel 5.5. Gegeben sei der Prüfzellengraph $G_t = (V, E_t)$ für einen Beobachtungszeitpunkt t aus Abbildung 5.7.

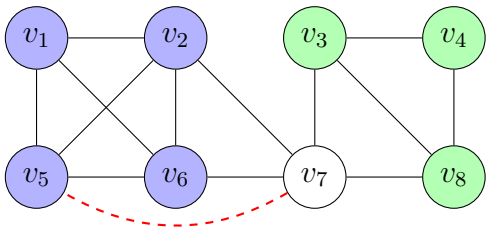


Abbildung 5.7: Beispiel zur Bestimmung der Affinität

In der Abbildung sind bereits zwei Gruppen S_1 (blau) und S_2 (grün) eingezeichnet, es sollen nun $a_t(v_7, S_1, \lambda_{Distanz})$ und $a_t(v_7, S_2, \lambda_{Distanz})$ bestimmt werden. Die rot markierte „Kante“ gibt an, dass die Distanz zwischen v_5 und v_7 deutlich größer als $\lambda_{Distanz}$ ist. Folglich gilt $a_t(v_7, S_1, \lambda_{Distanz}) = -1$. Eine solche „Kante“ gibt es nach S_2 nicht, sodass sich $a_t(v_7, S_2, \lambda_{Distanz}) = 2/3$ ergibt.

Definition 5.11. *Seien T eine Menge an Beobachtunszeitpunkten mit zugehörigen Prüfzellengraphen G_t für alle $t \in T$, $\mathscr{S}$ eine Partition von V, d eine Distanzfunktion und $\lambda_{Distanz} \in [0, 1]$. Die* konjunktive Affinität *eines Knotens v zu einer Menge $S \in \mathscr{S}$ ist definiert als*

$$a_T(v, S, \lambda_{Distanz}) = \sum_{t \in T} a_t(v, S, \lambda_{Distanz}). \tag{5.10}$$

Mit Hilfe der konjunktiven Affinität und dem *Affinitätsparameter* $\lambda_{Affinität}$ kann ein einfaches Zuweisungsschema von Knoten an Elemente der Partition erstellt werden. Jeder noch nicht zugewiesene Knoten v wird derjenigen Gruppe $S \in \mathscr{S}$ zugewiesen, für die $|S| < l$ gilt und $a_T(v, S, \lambda_{Distanz}) \geq \lambda_{Affinität}$ maximal ist. Kann keine solche Gruppe gefunden werden, so verbleibt v als nicht zugewiesener Knoten. Das Ergebnis dieser Phase ist die Partition $\mathscr{S}_2$.

Phase 3: Vollständige Partition Verbleiben nach Phase 2 noch weitere Knoten, welche keinem Element aus $\mathscr{S}_2$ zugewiesen wurden, so werden diese als Ausreißer bezeichnet. Sollen diese nicht dem Synthesewerkzeug überlassen werden, so wird in dieser Phase aus $\mathscr{S}_2$ eine vollständige Partition $\mathscr{S}_3$ gebildet, indem das gleiche Verfahren wie in Phase 2 angewandt wird, dieses Mal jedoch ohne Verwendung von $\lambda_{Affinität}$, sodass jeder Knoten einem Element der Partition zugewiesen wird.

Der in diesem Abschnitt beschriebene Algorithmus zur Prüfzellengruppierung wurde für diese Arbeit komplett neu implementiert. Gegenüber der älteren Version aus [155] nutzt die neue Implementierung Bitvektoren aus, um die Prüfzellengraphen mit Adjazenzlisten effizient darstellen zu können. Das lässt insbesondere die Berechnung des konjunktiven Prüfzellengraphen sehr effizient werden, da lediglich Bitweise UND-Verknüpfungen erfolgen, welche extrem schnell in einem Prozessor berechnet werden können.

5.3 Einbettung in den Syntheseablauf

Moderne Schaltungen werden typischerweise unter Zuhilfenahme von kommerziellen Werkzeugen zur Automatisierung des Entwurfsprozesses entwickelt und umgesetzt. Deshalb ist es erforderlich, dass die für den eingebauten Hochgeschwindigkeitstest erforderlichen Erweiterungen und Anpassungen der Schaltung einfach in den typischen Syntheseablauf integriert werden können. Bereits jetzt können integrierte Taktgeneratoren entweder als Teil der Schaltung oder als zusätzliche Einheit auf dem Chip eingefügt werden, da diese auch bei bestehenden Verzögerungstests zum Einsatz kommen. Zusätzlich sind viele für den Selbsttest erforderliche Erweiterungen, wie Testmustergenerator und Testdatenauswerter ebenfalls bereits in spezielle kommerzielle Werkzeuge integriert. An dieser Stelle soll deshalb vor allem auf die optimierten Prüfpfade aus dem vorherigen Abschnitt eingegangen werden.

Wie bereits erwähnt, berechnen die in Abschnitt 5.2 vorgestellten Verfahren keine kompletten Prüfpfade. Vielmehr stellen sie Randbedingungen für ein kommerzielles Werkzeug bereit. Diese sind typischerweise in der Lage, solche Randbedingungen als spezielle Befehle umzusetzen, welche praktischerweise in einem externen Skript gebündelt werden können. Dadurch lässt sich der vorgestellte Ansatz sehr leicht in bestehende Syntheseabläufe integrieren. Die wesentliche Erweiterung besteht darin, dass vor dem Einfügen der Testinfrastrukturen (in diesem Fall die Prüfpfade), die X-Profile der Prüfzellen bestimmt und dadurch Prüfzellengruppen ermittelt werden. Danach kann wieder zum eigentlichen Syntheseablauf zurückgekehrt werden, es müssen keine zusätzlichen Schritte mehr unternommen werden.

Abbildung 5.8 zeigt diesen in [153] vorgestellten, erweiterten Syntheseablauf.

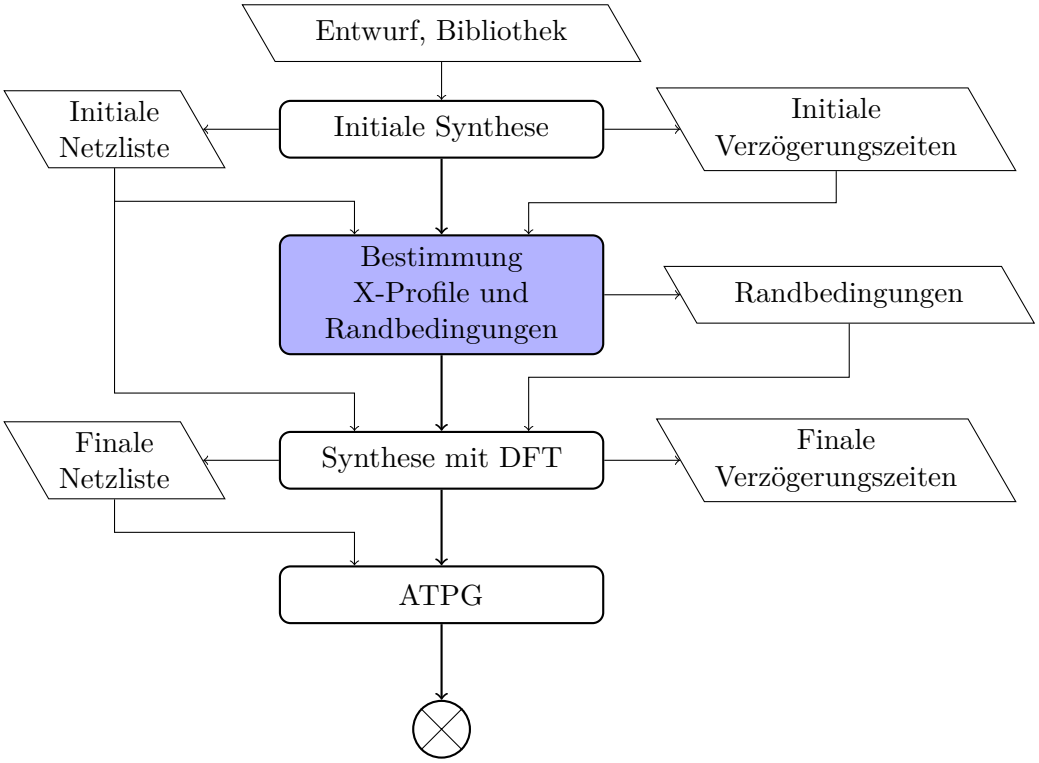


Abbildung 5.8: Erweiterter Syntheseablauf für FAST

Dort sind die im Rahmen der Arbeit entwickelten Arbeitschritte bläulich eingefärbt, die verbleibenden Arbeitsschritte werden mit kommerziellen Werkzeugen ausgeführt. Zunächst wird aus der gegebenen Verhaltensbeschreibung der Schaltung in einer Beschreibungssprache und der Standardzellbibliothek, welche die Zieltechnologie der Schaltung beschreibt, eine initiale Synthese durchgeführt. Aus dieser wird eine initiale Netzliste mit zugehörigen Verzögerungszeiten erzeugt, während dieser Synthese werden keine DFT-Strukturen in die Schaltung integriert. Die Netzliste wird in einem separaten Schritt mit Hilfe der zeitlichen Informationen analysiert, die Prüfzellengruppen werden bestimmt und als Randbedingungen in einem gesonderten Skript ausgegeben. Diese können dann wieder in das kommerzielle Werkzeug eingelesen werden, sodass die Synthese abgeschlossen werden kann, wobei nun auch DFT-Elemente wie Prüfpfade auf Grundlage der Randbedingungen erzeugt und integriert werden. Das Ergebnis dieses Schrittes ist die finale Netzliste mit aktualisierten Verzögerungszeiten, für welche nun Testmuster erzeugt werden. Damit ist der grundsätzliche Syntheseablauf abgeschlossen. Im Anschluss werden die im Kapitel 4 beschriebenen Algorithmen zur Frequenz- und

Musterauswahl durchgeführt, es stehen nun die grundlegenden Informationen für FAST bereit. Basierend auf den gewählten Frequenzen und Testsätzen werden die Ausgänge mit den im Abschnitt 5.4 genauer beschriebenen Algorithmen maskiert, um möglichst viele X-Werte aus den Testantworten herauszunehmen. Abschließend werden zur weiteren Auswertung die Frequenzen, Testsätze und Maskendaten simuliert und die maskierten Testantworten in ein MISR geschoben, sodass die Zwischensignaturen berechnet werden können.

5.4 Adaptive Maskierung für variable X-Raten

Zu Beginn des Kapitels wurde bereits erwähnt, dass X-Maskierung im Selbsttest einigen Einschränkungen unterliegt. Vor allem muss sie kostengünstig zu realisieren sein, da die verfügbaren Ressourcen im Selbsttest stark begrenzt sind. Weiterhin muss sie jedoch auch in der Lage sein, mit den variierenden X-Raten des Hochgeschwindigkeitstests umzugehen. In diesem Abschnitt werden zwei Varianten zur X-Maskierung vorgestellt, welche dieses Ziel erfüllen, indem sie sehr grobgranular arbeiten – die Maske wird über ein oder gar mehrere Muster hinweg konstant gehalten. Beide Varianten sind für variierende X-Raten ausgelegt. Der Anwendungsfall ist der Hochgeschwindigkeitstest, da dort die X-Rate von der Frequenz abhängt, es können aber auch genauso gut andere Testvarianten betrachtet werden. Wird beispielsweise nicht die Frequenz, sondern die Versorgungsspannung variiert, so stellen sich ebenfalls variierende X-Raten ein, da die Verzögerungen der Bauelemente von der Versorgungsspannung abhängen. Zusätzlich wird angenommen, dass die Testantwortkompaktierung, welche nach der Maskierung stattfindet, X-tolerant ist, d. h. die Maskierung muss nicht sämtliche X-Werte aus der Testantwort entfernen, es können verbleibende X-Werte akzeptiert werden, wenn dadurch beispielsweise die Implementierungskosten verringert werden können.

Die erste Variante, die *inkrementelle Maskierung*, ist eine Neuentwicklung dieser Arbeit. Bei diesem Verfahren bleibt ein Prüfpfad für den Rest der Testdauer maskiert, sobald er zum ersten Mal maskiert wurde. Dadurch lässt sich die Maskierung sehr einfach implementieren. Die Ergebnisse zeigen zudem, dass diese

Variante sehr gut mit den zuvor vorgestellten Prüfpfadoptimierungen zusammenarbeitet. Freilich kann die inkrementelle Maskierung in bestimmten Situationen zu aggressiv sein, sodass Fehlereffizienz verloren geht. Deshalb wird in diesem Abschnitt auch die Grenzwertmaskierung vorgestellt, welche bereits in [153, 154] publiziert wurde. Dieses Verfahren erlaubt es, Masken über eine benutzerdefinierte Dauer konstant zu halten, wodurch bessere Ergebnisse auf Kosten des Implementierungsaufwandes erzielt werden können. Bei allen Varianten werden nur die Prüfpfadausgänge maskiert, die primären Ausgänge werden in diesem Abschnitt vernachlässigt. Sie können aber prinzipiell ebenfalls berücksichtigt werden, z. B. indem sie über zusätzliche Flipflops zu einen eigenen „Prüfpfad“ konfiguriert werden.

5.4.1 Inkrementelle X-Maskierung

Das Prinzip der inkrementellen Maskierung ist einfach: Beim Test wird die Testfrequenz stetig erhöht und sobald ein Prüfpfad bei einer Frequenz maskiert werden muss, bleibt er für alle folgenden Frequenzen weiterhin maskiert. Das erlaubt es, die Maskierungslogik so weit zu vereinfachen, dass lediglich ein Schieberegister benötigt wird. Dies ist schematisch in Abbildung 5.9 dargestellt. Bei einer Maskie-

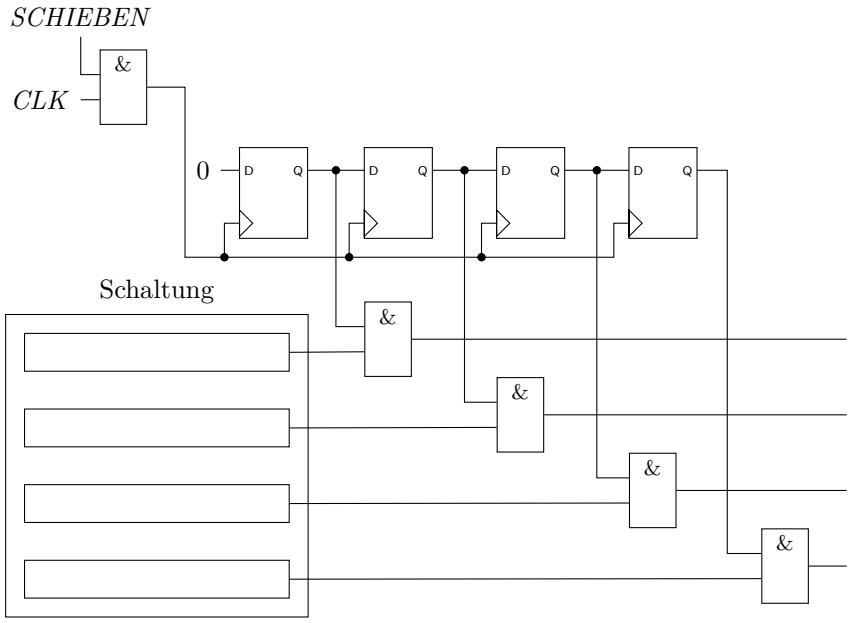


Abbildung 5.9: Architektur der inkrementellen Maskierung

rung mit UND-Gattern wird das Schieberegister initialisiert, indem alle Flipflops mit logisch-1 belegt werden. Im Laufe des Tests wird dann von der Steuerung logisch 0 in das Register geschoben, sodass ein einmal maskierter Prüfpfad weiterhin maskiert bleibt.[1] Das setzt allerdings voraus, dass die Prüfpfadausgänge entsprechend angeordnet sind, bzw. das Schieberegister muss passend geformt sein, was unter Umständen zu einem erhöhten Verbindungsaufwand führen kann. Das Signal *SCHIEBEN* aktiviert das Schieberegister und muss von der Steuerung verwaltet werden. Hier kann beispielsweise mit einem einfachen Zähler kontrolliert werden, wie viele zusätzliche Ausgänge bei der nächsten Frequenz maskiert werden sollen.

Prinzipiell kann die Maske zu einem beliebigen Zeitpunkt aktualisiert werden, in dieser Arbeit werden allerdings nur Frequenzwechsel als mögliche Zeitpunkte betrachtet. Beim Aktualisieren muss das Schieberegister gesondert zum Einlesen des ersten Testmusters neu eingestellt werden, um die Maske zu aktualisieren. Die Maske kann nicht gleichzeitig mit dem Laden des nächsten Testmusters aktualisiert werden, da beim Laden des Testmusters noch die letzte Testantwort aus den Prüfpfaden geschoben werden muss. Dieses Einstellen der Maske dauert in der Regel aber nur wenige Takte. Wird dies nur beim Wechsel der Testfrequenz durchgeführt, hält sich der zeitliche Mehraufwand in Grenzen, insbesondere wenn die Frequenzen zuvor minimiert wurden, wie in Kapitel 4 beschrieben wurde. Werden Taktgeneratoren benutzt, welche Zeit zum Einstellen der gewünschten Zielfrequenz benötigen (z. B. eine PLL), kann dies ebenfalls mit der Aktualisierung der Maske kombiniert werden.

Zur Bestimmung der neu zu maskierenden Prüfpfade wird die „X-Dichte“ der Prüfpfade für einen Test berechnet.

Definition 5.12. *Sei S ein Prüfpfad, P ein Testmuster und bezeichne $n_X(S, P)$ die Anzahl der X-Werte in dem von S aufgezeichneten Teil der Testantwort auf P. Dann ist die* X-Dichte von S für ein Testmuster P *definiert als*

$$\rho_X(S, P) = \frac{n_X(S, P)}{|S|}. \tag{5.11}$$

[1] Natürlich können auch ODER-Gatter genutzt werden. Dann wird das Schieberegister mit 0 initialisiert und es werden 1en in das Register geschoben.

Die mittlere X-Dichte von S für einen Testsatz $\mathscr{P}$ *bestimmt sich zu*

$$\rho_X(S, \mathscr{P}) = \frac{1}{|\mathscr{P}|} \sum_{P \in \mathscr{P}} \rho_X(S, P). \tag{5.12}$$

Die inkrementelle Maskierung bestimmt für jeden noch nicht maskierten Prüfpfad die mittlere X-Dichte für den Testsatz und vergleicht diese mit einem Grenzwert ρ_{max}. Ist $\rho_X(S, \mathscr{P}) \geq \rho_{max}$ für einen Prüfpfad S, so wird S für die aktuelle und alle nachfolgenden Frequenzen maskiert.

Beispiel 5.6. Gegeben sei ein Prüfpfad S mit 25 Prüfzellen und ein Testsatz $\mathscr{P}$ mit 10 Testmustern. Dabei zeichnet S beim Anlegen aller Testmuster X-Werte auf, die jeweilige Anzahl sei im Tupel $(10, 0, 3, 5, 7, 4, 10, 8, 6, 2)$ zusammengefasst. Dann gilt

$$\rho_X(S, \mathscr{P}) = \frac{1}{10} \cdot \frac{10 + 0 + 3 + 5 + 7 + 4 + 10 + 8 + 6 + 2}{25} = 0{,}22.$$

Gilt beispielsweise $\rho_{max} = 0{,}2$, so würde der Prüfpfad maskiert.

5.4.2 Grenzwertmaskierung

Die Grenzwertmaskierung stellt im Prinzip eine Verallgemeinerung der zuvor vorgestellten inkrementellen Maskierung dar, um variable Granularität der Maskierung zur Verfügung zu stellen. Zusätzlich wird es erlaubt, maskierte Prüfpfade wieder zu demaskieren, falls etwa die X-Dichte zu einem späteren Zeitpunkt wieder unter den Grenzwert fällt. Abbildung 5.10 zeigt beispielhaft, wie das Maskierungssystem implementiert werden kann.

Die aktuelle Maske wird in einem Register gespeichert, welches die Maske über q Testmuster hinweg speichern kann. Der Zähler wird verwendet, um das Maskenregister zu schreiben, er wird mit $q - 1$ initialisiert und zählt herunter. Sobald der Zähler auf Null herunter gezählt hat (oder der Test für die aktuelle Frequenz beendet ist), wird das Register mit neuen Maskenbits aus dem Speicher überschrieben.

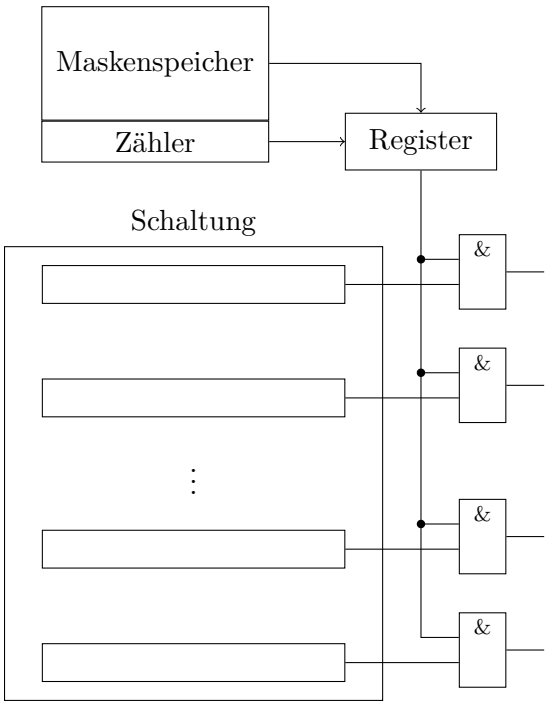


Abbildung 5.10: Architektur der Grenzwertmaskierung

Die Grenzwertmaskierung verwendet ebenfalls den Parameter ρ_{max}, um zu entscheiden, ob ein Prüfpfad maskiert werden soll. In diesem Fall wird aber immer die X-Dichte für ein einzelnes Testmuster nach Gleichung (5.11) betrachtet. Dadurch ist die kleinste Granularität der Maskierung ein einzelnes Testmuster. Sie unterscheidet sich damit deutlich von feingranulareren Ansätzen wie [17, 127, 111], welche die Maske bei jedem Schiebetakt anpassen. Es ist aber auch möglich, die Maske für mehrere Testmuster konstant zu halten. Die Menge aller Testmuster, für welche eine einzelne Maske benötigt wird, wird im Folgenden als *Fenster* mit Größe q bezeichnet. Zur Bestimmung der zu maskierenden Prüfpfade wird wieder das ρ_{max} aus dem vorangehenden Abschnitt verwendet. Das folgende Beispiel zeigt die Grenzwertmaskierung für eine Fenstergröße von $q = 1$, später folgt die Erweiterung auf andere Fenstergrößen.

Beispiel 5.7. Gegeben sei eine Schaltung mit fünf Prüfpfaden S_1–S_5. Über die entsprechenden Prüfpfadausgänge o_1–o_5 können die Inhalte der Prüfpfade ausgelesen werden. Tabelle 5.1 zeigt für vier Testmuster P_1–P_4 die X-Dichten der Prüfpfade. Wählt man $\rho_{max} = 0{,}45$, so wird im ersten Testmuster o_2 maskiert, im zweiten Testmuster werden o_2 und o_3 maskiert, im dritten Testmuster o_2 und im vierten Testmuster wird o_4 maskiert.

Ausgang	$\rho_X(S, P_i)$			
	P_1	P_2	P_3	P_4
o_1	0,23	0,05	0,27	0,0
o_2	0,5	0,47	0,62	0,3
o_3	0,43	0,49	0,42	0,53
o_4	0,0	0,0	0,04	0,0
o_5	0,1	0,04	0,0	0,0
Maske ($\rho_{max} = 0{,}45$)	o_2	o_2, o_3	o_2	o_4

Tabelle 5.1: Beispiel zur Grenzwertmaskierung

Es ist zu beachten, dass Prüfpfade mit hoher X-Dichte auch wichtige Fehlerinformationen enthalten können. Der Algorithmus könnte also den Prüfpfad maskieren, obwohl es vorteilhafter wäre, die Fehlerinformationen zu erlangen. Lässt sich ein solcher Fall nicht vermeiden, kann der entsprechende Prüfpfad als *essentiell* markiert werden, sodass er explizit von der Maskierung ausgenommen wird. Dadurch sinkt zwar die Effektivität der Maske, aber es müssen keine zusätzlichen Testmuster erzeugt und angelegt werden, um die Fehlereffizienz auf die gewünschten Werte anzuheben. In dieser Arbeit kann der Nutzer den maximalen Anteil $\gamma_{Essentiell}$ an essentiellen Prüfpfaden vorgeben. Ist $\gamma_{Essentiell} > 0$, so wird für jeden Prüfpfad und jedes Testmuster ermittelt, wie viele der Prüfzellen des Pfades Bits mit Fehlerinformationen enthalten. Solche Bits werden im Folgenden als deterministische oder D-Bits bezeichnet. Mit den D-Bits lässt sich analog zur X-Dichte eine „D-Dichte“ bestimmen. Diese wird dann genutzt, um die Prüfpfade prioritätsbasiert zu sortieren, sodass der Prüfpfad mit der höchsten Anzahl an D-Bits als erstes als essentiell markiert wird.

Beispiel 5.8. Gegeben sei wieder die Situation aus Beispiel 5.7, Tabelle 5.2 führt die X- und zusätzlich die D-Dichten auf. Für jedes Testmuster werden die Prüfpfade nun absteigend nach den D-Dichten sortiert. Die Position in der sortierten Liste entspricht der Priorität des Prüfpfads als essentieller Prüfpfad (Position 1 hat die höchste Priorität, da der Prüfpfad am meisten HDF-Anteile enthält, gefolgt von Position 2, usw.). Lässt man zu, dass ein Prüfpfad als essentiell markiert werden kann ($\gamma_{Essentiell} = 0{,}2$), so wird im ersten Testmuster o_2 als essentiell markiert und damit nicht mehr maskiert. Im zweiten Muster wird o_3 als essentiell markiert und nicht mehr maskiert. Im dritten Testmuster hat

Ausgang	Dichte							
	P_1		P_2		P_3		P_4	
	X	D	X	D	X	D	X	D
o_1	0,23	0,15	0,05	0,12	0,27	0,23	0,0	0,0
o_2	0,5	0,2	0,47	0,05	0,62	0,0	0,3	0,2
o_3	0,43	0,19	0,49	0,17	0,42	0,14	0,53	0,04
o_4	0,0	0,0	0,0	0,05	0,04	0,1	0,0	0,0
o_5	0,1	0,0	0,04	0,03	0,0	0,0	0,0	0,0
$\rho_{max} = 0{,}45$	o_2		o_2, o_3		o_2		o_3	
$\gamma_{Essentiell} = 0{,}2$	o_2		o_3		o_1		o_2	
Ergebnis	–		o_2		o_2		o_3	

Tabelle 5.2: Beispiel zur Maskierung mit essentiellen Prüfpfaden

o_1 die höchste Priorität. Da o_1 nicht maskiert wird, ändert diese Auswahl nichts an der festgelegten Maske. Schlussendlich wird beim vierten Testmuster o_2 als essentiell markiert, die Maske enthält lediglich o_3.

Wählt man $q > 1$, so muss ein zweiter Parameter $\gamma_{Fenster}$ für die Grenzwertmaskierung angegeben werden. $\gamma_{Fenster}$ definiert den Anteil der Testmuster im aktuellen Fenster, für welchen der Grenzwert ρ_{max} im Prüfpfad überschritten werden muss, damit der Prüfpfad für das komplette Fenster maskiert wird.

Beispiel 5.9. Betrachtet werden soll wieder die Situation aus Beispiel 5.7, dieses Mal jedoch erweitert auf sechs Testmuster, um den Einfluss von $\gamma_{Fenster}$ besser demonstrieren zu können. Tabelle 5.3 zeigt wieder die X-Anteile der Prüfpfade für die sechs Muster. Wählt man eine Fenstergröße von $q = 3$ und den Parameter $\gamma_{Fenster} = 2/3 \approx 0{,}67$, so wird ein Ausgang für jeweils drei Muster vollständig maskiert, wenn der entsprechende Prüfpfad mit seiner X-Dichte den Grenzwert von ρ_{max} in mindestens zwei von drei Mustern des Fensters überschreitet. In diesem Beispiel bedeutet dies, dass im ersten Fenster (P_1–P_3) nur o_2 maskiert wird, da o_3 nur einmal den Grenzwert überschreitet. Analog wird im zweiten Fenster (P_4–P_6) nur o_3 maskiert. In beiden Fällen werden die Ausgänge aber auch maskiert, wenn für ein Muster des Fensters der Grenzwert nicht überschritten wurde.

Ausgang	X-Dichte					
	P_1	P_2	P_3	P_4	P_5	P_6
o_1	0,23	0,05	0,27	0,0	0,16	0,19
o_2	0,5	0,47	0,62	0,30	0,61	0,4
o_3	0,43	0,49	0,42	0,03	0,55	0,51
o_4	0,0	0,0	0,04	0,0	0,05	0,0
o_5	0,1	0,04	0,0	0,0	0,08	0,03
$q = 1$, $\rho_{max} = 0{,}45$	o_2	o_2, o_3	o_2	–	o_2, o_3	o_3
$q = 3$, $\gamma_{Fenster} \approx 0{,}67$		o_2			o_3	

Tabelle 5.3: Beispiel zur Grenzwertmaskierung mit verschiedenen Fenstergrößen

5.5 Simulationsergebnisse

In diesem Abschnitt sollen die Ergebnisse einer Simulationsstudie vorgestellt werden, in welcher ITC99-Benchmarkschaltungen [16] (vgl. Abschnitt 4.8) mit den vorgestellten Erweiterungen des prüfgerechten Entwurfs für den eingebauten Hochgeschwindigkeitstest synthetisiert wurden. Zusätzlich wurden ebenfalls die industriellen Schaltungen betrachtet. Bei diesen ist aber eine vollständige Synthese nicht möglich, da die Verhaltensbeschreibung nicht verfügbar ist. Die spezialisierten Prüfpfade lassen sich aber in der Simulation emulieren, sodass zumindest eine Abschätzung der Effektivität gegeben werden kann. Es wurden 10 Beobachtungszeitpunkte nach Gleichung (4.3) zwischen t_{nom} und t_{min} gewählt, um die Distanzen der X-Profile berechnen zu können. In einem ersten Schritt wurde analysiert, inwiefern die Beobachtungszeitpunkte „nützlich“ für die Gruppierung sind. Bei großen Beobachtungszeitpunkten (bzw. geringen Frequenzen) sind die X-Profile der Flipflops oftmals praktisch gleich. Deshalb wurden solche Beobachtungszeitpunkte vor der Gruppierung aussortiert. Bei den analysierten Schaltungen führte dies im Mittel dazu, dass zwischen 3 und 5 Beobachtungszeitpunkte entfernt wurden.

Wie auch in Abschnitt 4.8 wurde die Anzahl der Prüfpfade für alle Schaltungen so festgelegt, dass sich maximal 25 Prüfzellen in einem Prüfpfad befinden. Für die spezialisierten Prüfpfade müssen drei Parameter ($\lambda_{Konnektivität}$, $\lambda_{Distanz}$ und $\lambda_{Affinität}$) gewählt werden (vgl. Abschnitt 5.2.3). Für alle Schaltungen wurde $\lambda_{Affinität} = 0{,}5$ gesetzt. Die ersten beiden Parameter sind abhängig von der

Schaltung und eine Suchraumexploration für die beste Konfiguration ist sehr aufwändig. Daher wurde stattdessen ein vereinfachtes Verfahren basierend auf der Medianberechnung zur automatischen Bestimmung genutzt. Dabei wird zuerst jeweils der Maximalwert der Distanzen zweier Prüfzellen über alle Beobachtungszeitpunkte bestimmt. Diese Maximalwerte wurden dann in einer Liste sortiert und die Parameter wurden so gewählt, dass 30% der Maximalwerte kleiner als $\lambda_{Konnektivität}$ und 30% der Maximalwerte größer als $\lambda_{Distanz}$ sind. Tabelle A.2 zeigt die so bestimmten Werte, Tabelle A.1b die Eigenschaften der so synthetisierten Schaltungen (beide im Anhang A.1). Diese Schaltungen werden im Folgenden auch als *FAST-Entwurf* bezeichnet.

Ziel dieses Abschnitts ist, die Auswirkungen der Prüfzellengruppierung auf die Schaltungen zu analysieren. Dazu werden die synthetisierten Schaltungen mit Varianten verglichen, in denen keine Prüfzellengruppierung durchgeführt wurde (diese wurden auch in Abschnitt 4.8 verwendet). Diese „normal“ synthetisierten Schaltungen werden im Folgenden als *Standardentwurf* bezeichnet. Um den FAST-Entwurf mit dem Standardentwurf zu vergleichen, wurde derselbe Prozess zur Auswahl von Frequenzen, FAST-Gruppen und Testsätzen wie in Kapitel 4 beschrieben durchgeführt. Insgesamt zeigt sich hier nur ein geringer Unterschied zwischen beiden Entwürfen, sodass hier auf Details verzichtet wird. Genaue Daten können im Anhang A.3 nachgeschlagen werden. Die Entwürfe sollen in diesem Abschnitt anhand der Effizienz der vorgestellten Maskierungsverfahren und deren Auswirkung auf ein X-streichendes MISR bewertet werden. Es wird also explizit keine räumliche Kompaktierung zwischen Ausgangsmaskierung und MISR durchgeführt.

Die in dieser Arbeit betrachteten Metriken zur Bewertung der Testdatenauswertung sind Fehlereffizienz, X-Reduktionsrate, sowie Speicheraufwand der Maskierung. Sei Φ die Menge an HDFs, welche beim Selbsttest durch die Testmuster erkannt und in die Prüfpfade geladen werden, sodass sie beim Herausschieben der Testantwort die Maske passieren müssen. Dann ist $\Phi_{Maske} \subseteq \Phi$ die Menge aller HDFs, welche die Maske passieren. Die *Fehlereffizienz der Maskierung* bestimmt sich dann zu

$$\mathrm{FE}_{Maske} = \frac{|\Phi_{Maske}|}{|\Phi|}. \tag{5.13}$$

Analog bestimmt sich die Fehlereffizienz des MISR zu

$$\mathrm{FE}_{MISR} = \frac{|\Phi_{MISR}|}{|\Phi_{Maske}|}, \tag{5.14}$$

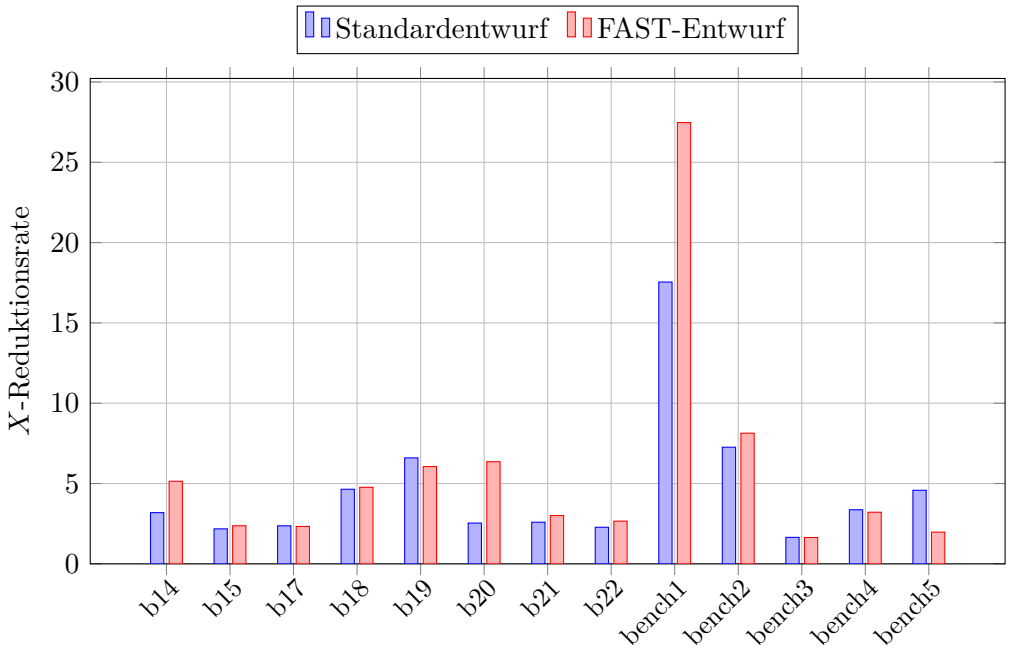
wobei Φ_{MISR} die Menge an HDFs bezeichnet, welche durch Auswertung der MISR-Signaturen erkannt werden können. Die X-Reduktionsrate bestimmt sich aus dem Kehrwert des Anteils der X-Werte, welche die Maske passieren:

$$R_X = \frac{\#\ X\text{-Werte vor Maske}}{\#\ X\text{-Werte nach Maske}}. \tag{5.15}$$

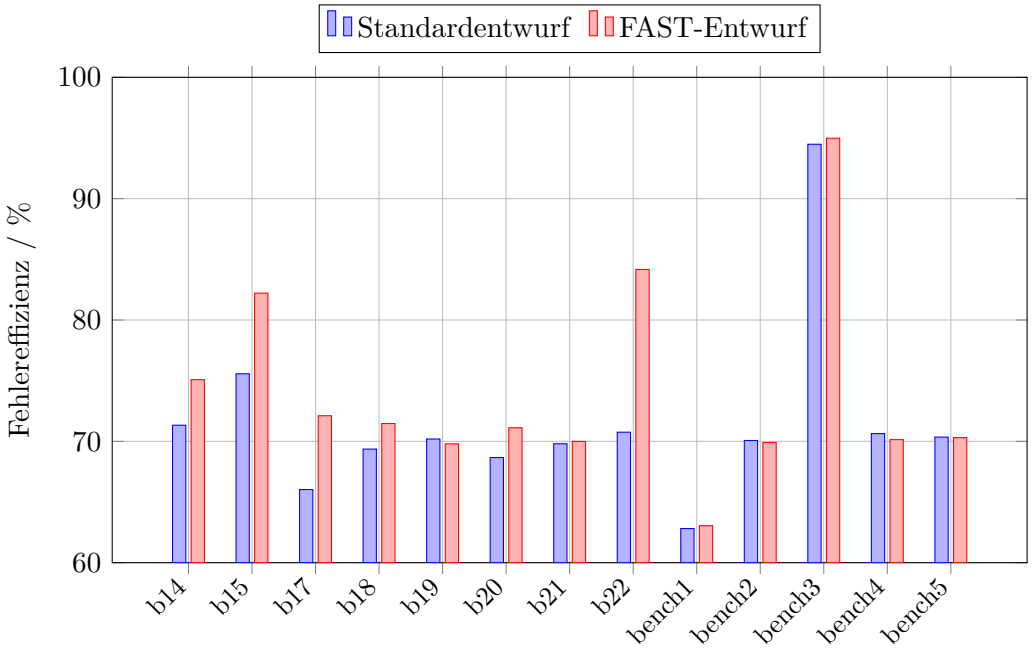
Eine X-Reduktionsrate von $R_X = 2$ bedeutet beispielsweise, dass die Anzahl der X-Werte in der Testantwort durch die Maske halbiert wird. Die Bestimmung des Speicheraufwands für die Masken wird in dieser Arbeit pessimistisch abgeschätzt über die reinen Maskenbits, die gespeichert werden müssen. In der Praxis können diese noch komprimiert werden, indem beispielsweise ein LFSR benutzt wird, um die Maskenbits aus komprimierten Startwerten zu erzeugen [127, 111, 121].

5.5.1 Ergebnisse der inkrementellen Maskierung

Als erstes sollen die Ergebnisse der inkrementellen Maskierung in Kombination mit dem X-streichenden MISR vorgestellt werden. Um die Schaltungsentwürfe zu vergleichen, wurden Grenzwerte ρ_{max} bestimmt, sodass die Maskierung bei beiden Entwürfen etwa gleiche Fehlereffizienz liefert. Abbildung 5.11 zeigt die so erzielten X-Reduktionsraten und Fehlereffizienzen. Die Daten für diese Abbildungen (inklusive gewählter ρ_{max}) befinden sich in Tabelle A.10, Anhang A.5. Es zeigt sich, dass der FAST-Entwurf bei vielen Schaltungen mit der inkrementellen Maskierung höhere X-Reduktionsraten bei gleichen, oder sogar besseren Fehlereffizienzen der Maske liefert. Bei den ITC-Schaltungen ist lediglich bei b19 der FAST-Entwurf etwas schlechter als der Standardentwurf, hier ergeben sich allerdings kaum Unterschiede bei den erzielten Werten. Bei den industriellen Schaltungen ergibt sich ein gemischtes Bild, hier zeigen bench4 und bench5 geringere X-Reduktionsraten im FAST-Entwurf. Es muss darauf hingewiesen werden, dass die Anzahl der Prüfpfade zwischen Standard- und FAST-Entwurf bei bench3, bench4 und bench5 verschieden ist, da im Standardentwurf die vorgegebenen Prüfpfade verwendet



(a) X-Reduktionsrate



(b) Fehlereffizienz

Abbildung 5.11: Ergebnisse der inkrementellen Maskierung

wurden. Die Testantworten bei bench3 enthalten generell vergleichsweise wenig X-Werte, daher ist hier die Maskierung insgesamt weniger effektiv.

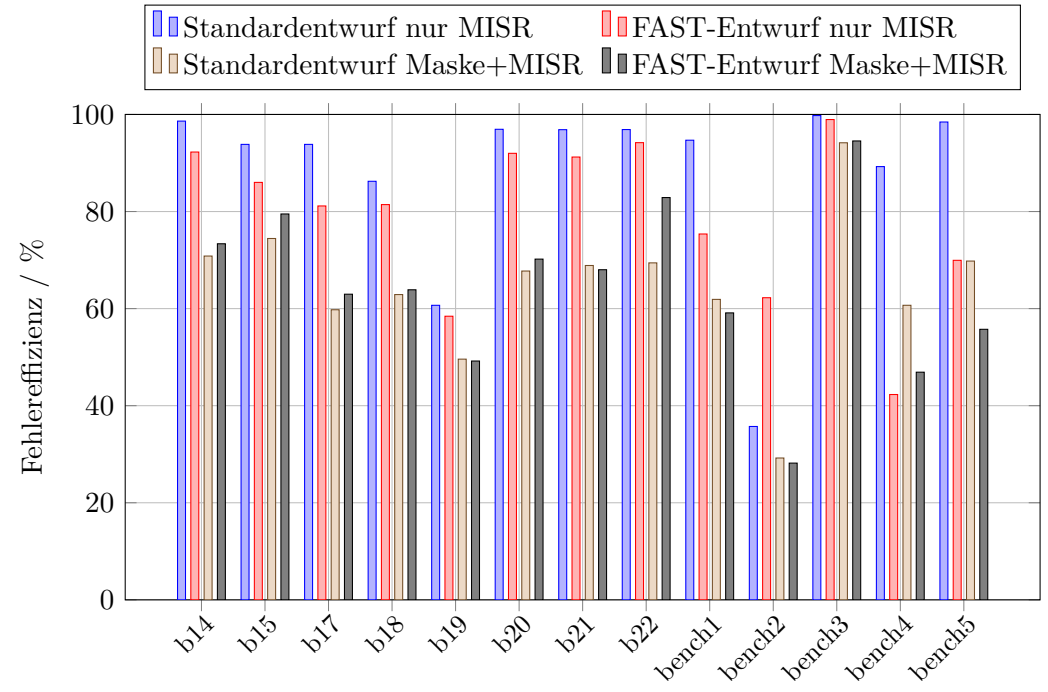
Wählt man die Werte für ρ_{max} so, dass höhere Fehlereffizienzen erzielt werden, verringert sich die X-Reduktionsrate bei beiden Entwürfen und geht gegen 1. Es verringern sich dadurch auch die Differenzen in den Entwürfen, da beide gleichermaßen weniger effektiv in der X-Reduktion sind.

Kombination mit MISR

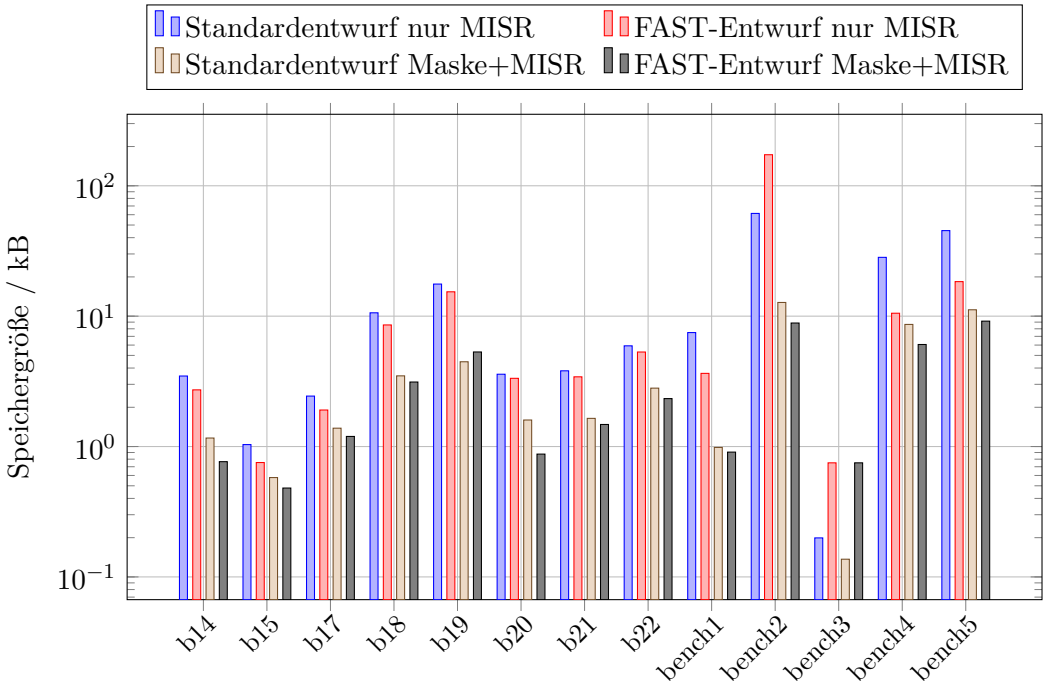
Um die Auswirkungen der inkrementellen Maskierung auf die Kompaktierung mittels X-streichendem MISR zu verdeutlichen, wird in diesem Abschnitt untersucht, wie sich Fehlereffizienz und Signaturspeichergröße verhalten. Für die Schaltungen wurden X-streichende MISR verwendet, deren Größe sich an der Anzahl der Prüfpfade orientiert. Das MISR für b14 hat beispielsweise eine Größe von 16 Bits, das MISR für b19 hat eine Größe von 256 Bits. Bei allen Schaltungen muss eine Zwischensignatur abgespeichert werden, sobald weniger als 7 X-freien Kombinationen der MISR-Bits gefunden werden (vgl. Abschnitt 2.6.1). Insgesamt vier verschiedene Fälle werden verglichen: Für Standard- und FAST-Entwurf wird jeweils der Fall ohne und mit inkrementeller Maskierung betrachtet. Abbildung 5.12 zeigt die Fehlereffizienzen und Signaturspeichergrößen für die zuvor benutzten Einstellungen der inkrementellen Maskierung. Die Daten dazu befinden sich in den Tabellen A.9 und A.10. Es wird die gesamte Fehlereffizienz betrachtet, sie berechnet über sich über

$$\mathrm{FE}_{Gesamt} = \mathrm{FE}_{Maske} \cdot \mathrm{FE}_{MISR} \,. \tag{5.16}$$

Es zeigt sich, dass das X-streichende MISR allein bei allen Schaltungen die höchste Fehlereffizienz erzielt, was bei den Ergebnissen aus Abbildung 5.11 zu erwarten war. Gleichzeitig benötigt das MISR in diesen Fällen aber auch den größten Signaturspeicher. Für die Schaltung bench2 benötigt das MISR alleine im FAST-Entwurf bereits mehr als 173 kB Speicher für die Zwischensignaturen, die Abbildung 5.12b nutzt deshalb eine logarithmische Skalierung der Vertikalachse. Hier sieht man, dass mit der inkrementellen Maskierung in beiden Entwürfen dieser Speicherbedarf deutlich gesenkt werden kann. Allerdings profitiert der FAST-



(a) Gesamte Fehlereffizienz



(b) Speicherbedarf

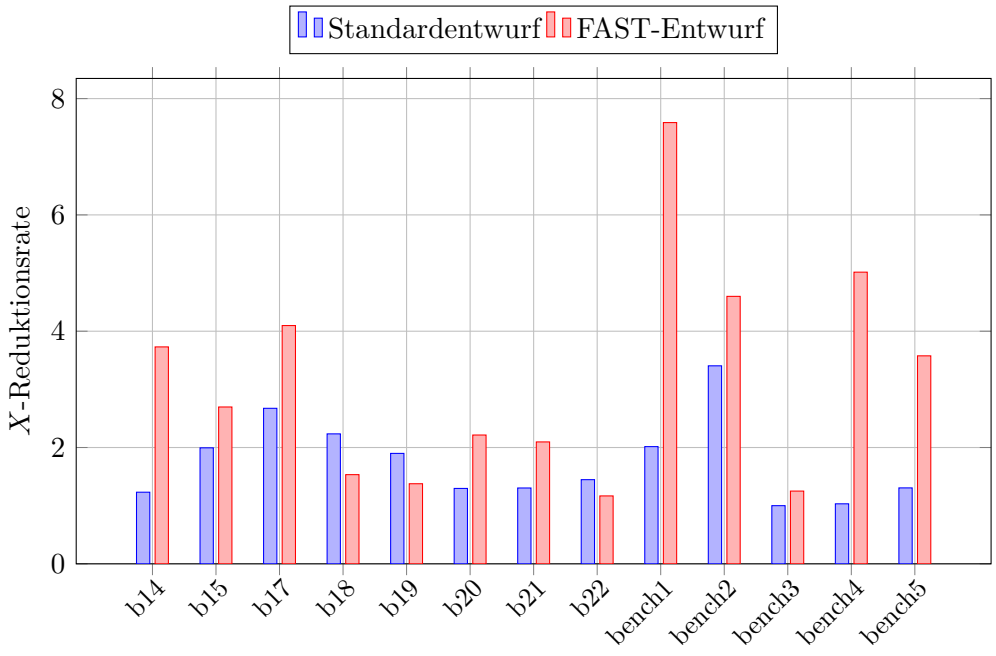
Abbildung 5.12: Ergebnisse der Kompaktierung mit inkrementeller Maskierung

Entwurf oft stärker von der Maskierung – die gesamte Fehlereffizienz ist erhöht, während gleichzeitig der Speicherbedarf verringert ist, wie man beispielsweise bei der Schaltung b22 sieht.

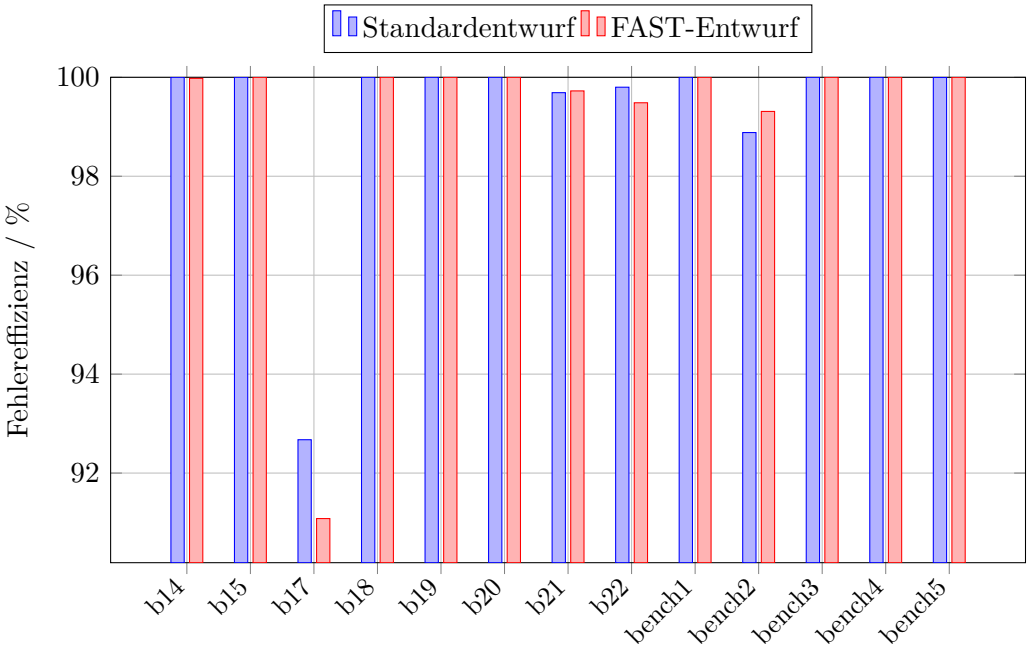
5.5.2 Ergebnisse der Grenzwertmaskierung

Die Grenzwertmaskierung erlaubt beliebige Fenstergrößen q, um die Granularität der Masken zu steuern. Diese beeinflusst die X-Reduktionsrate, die Fehlereffizienz und vor allem die Speichergröße der Masken. Da eine vollständige Analyse des Verhaltens sehr aufwändig ist, werden hier nur die Ergebnisse für eine Maskierung mit $q = 1$ vorgestellt. Wie bei der inkrementellen Maskierung zuvor auch, wurden dabei die Parameter ρ_{max} und $\gamma_{Essentiell}$ so gewählt, dass für beide Entwürfe in etwa gleiche Fehlereffizienz erzielt wurde. Abbildung 5.13 zeigt die Ergebnisse der Maskierung, unterteilt in X-Reduktionsrate (Abbildung 5.13a) und Fehlereffizienz (Abbildung 5.13b). Die Daten für diese Abbildung sind Tabelle A.11 entnommen. Durch die kleine Fenstergröße zeigt sich sofort, dass eine sehr hohe Fehlereffizienz erzielt werden kann (bei vielen Schaltungen liegt sie bei 100%). Bei der X-Reduktionsrate ergibt sich sogar ein noch eindeutigeres Bild als bei der inkrementellen Maskierung. Lediglich für die Schaltungen b18, b19 und b22 werden geringere X-Reduktionsraten im FAST-Entwurf erzielt, alle anderen Schaltungen weisen teils signifikant bessere Ergebnisse als im Standardentwurf auf (z. B. die Schaltung bench1).

Das Gesamtergebnis aus Maskierung und MISR ist für die Grenzwertmaskierung in Abbildung 5.14 dargestellt. Dabei zeigt Abbildung 5.14a die insgesamt erzielbare Fehlereffizienz und Abbildung 5.14b den gesamten Speicherbedarf, welcher sich aus dem Speicherbedarf der Maskierung und dem Signaturspeicher zusammensetzt. Die Daten dieser Abbildung entstammen wieder der Tabelle A.11. Zu beachten ist, dass der Speicherbedarf in Abbildung 5.14b logarithmisch dargestellt ist. Die Betrachtung der Fehlereffizienz fördert Erstaunliches zutage. Offenbar korrelieren X-Reduktionsrate und Fehlereffizienz der Maskierung nur bedingt mit der Fehlereffizienz der nachfolgenden Signaturberechnung im MISR. Obwohl beispielsweise die Schaltung b20 eine höhere X-Reduktionsrate im FAST-Entwurf als im Standardentwurf (bei gleicher Fehlereffizienz von 100%) hat, ist die ab-

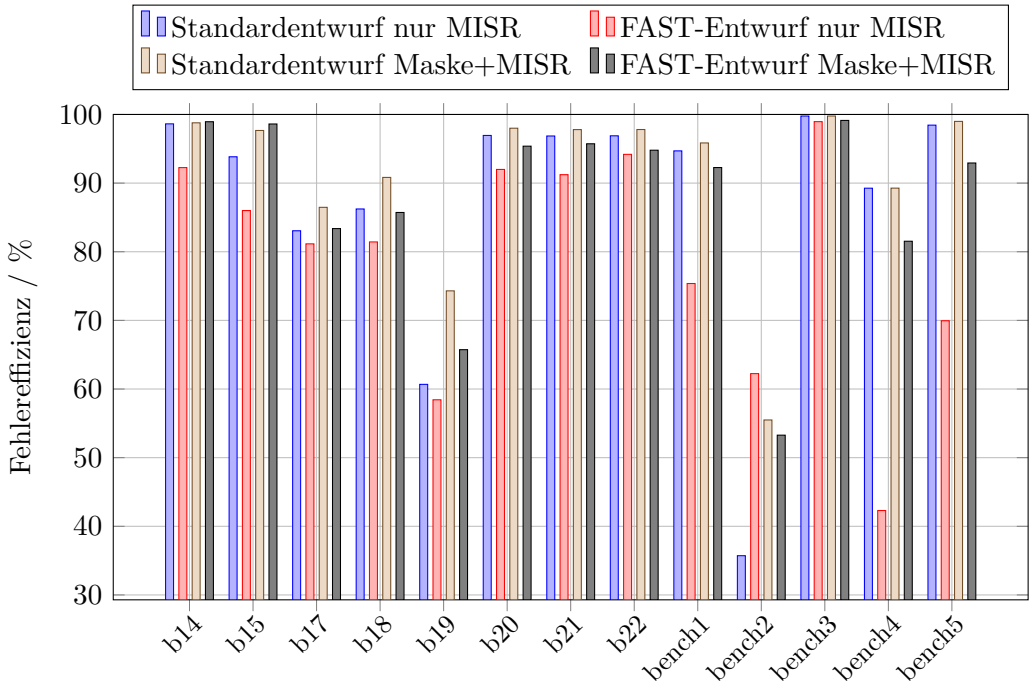


(a) X-Reduktionsrate

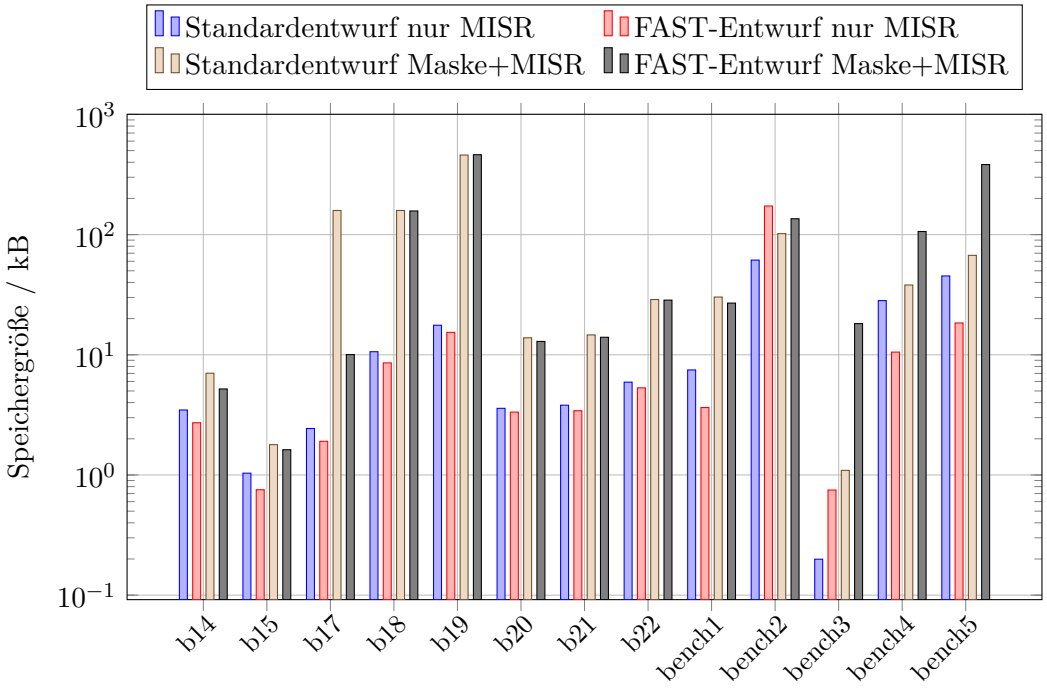


(b) Fehlereffizienz

Abbildung 5.13: Ergebnisse der Grenzwertmaskierung ($q = 1$)



(a) Gesamte Fehlereffizenz



(b) Speicherbedarf

Abbildung 5.14: Ergebnisse der Kompaktierung mit Grenzwertmaskierung ($q = 1$)

schließende Fehlereffizienz im MISR geringer. Der Grund liegt vermutlich in den Prüfpfaden selbst, denn deren Anordnung hat einigen Einfluss auf die Berechnungen des MISR. Bezüglich des Speicherbedarfs zeigt sich sofort, dass die Maskierung signifikanten Einfluss auf den benötigten Speicher hat. Wie erwähnt ist diese Abschätzung pessimistisch, denn es können Kompressionsverfahren eingesetzt werden, um ihn zu verringern. Für die industriellen Schaltungen bench3, bench4 und bench5 lassen sich die Abweichungen zwischen Standard- und FAST-Entwurf wieder über die Anzahl der Prüfpfade erklären. Da im FAST-Entwurf viel mehr Prüfpfade benutzt werden als im Standardentwurf, müssen entsprechend mehr Maskenbits gespeichert werden.

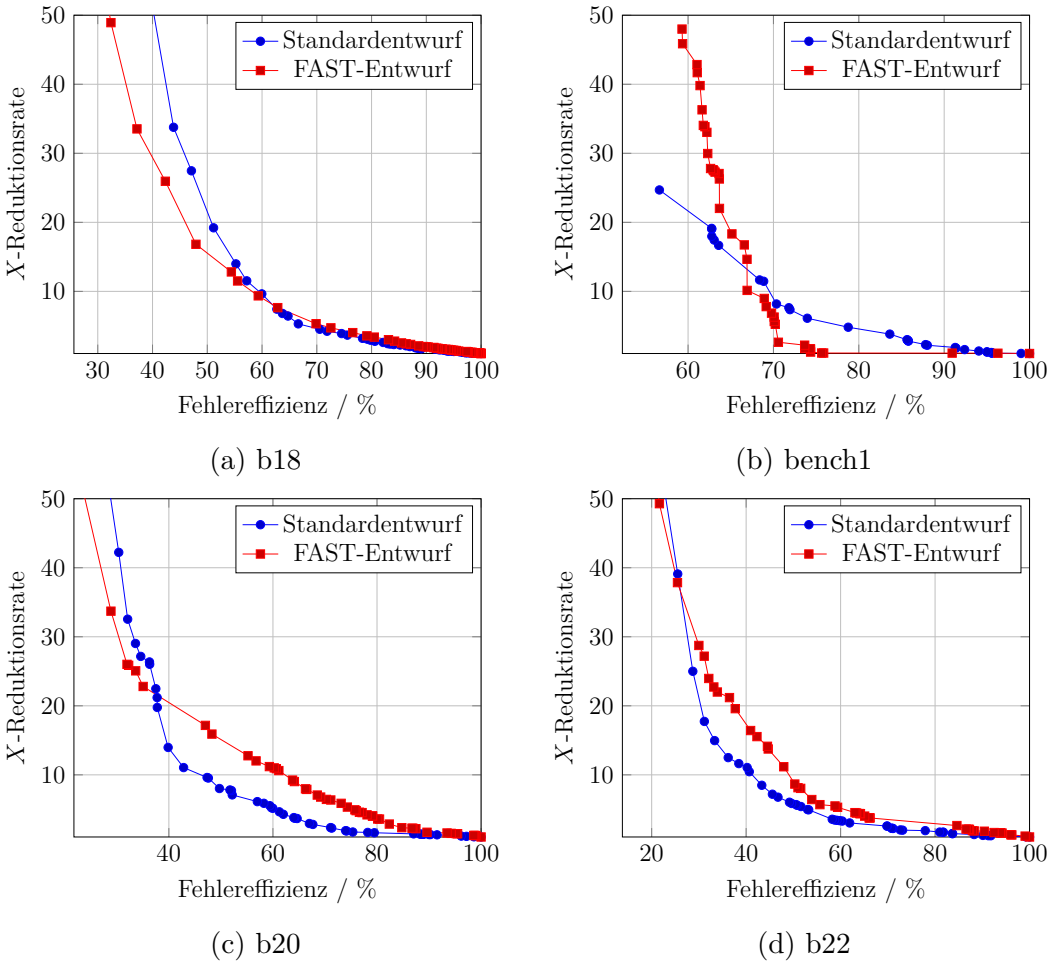
5.5.3 Detailanalyse

Im Folgenden sollen die Ergebnisse der Maskierung genauer untersucht werden. Insbesondere wird auf die Parameter der Maskierung eingegangen, sowie – ähnlich wie in Kapitel 4 – eine Simulation unter erhöhter Maximalfrequenz.

Paretooptima der Maskierung

Der wesentliche Parameter zur Steuerung der inkrementellen Maskierung ist der Grenzwert ρ_{max}. In einem kleinen Programm wurde für die analysierten Schaltungen dieser Parameter von 0,001 bis 1 in Schritten von 0,001 variiert. Für jeden Wert wurden die Masken berechnet, sowie X-Reduktionsrate und Fehlereffizienz aufgestellt. So lassen sich Paretooptima finden und darstellen, die gewählten Werte aus den vorhergehenden Betrachtungen sind ebenfalls Paretooptima aus diesem Prozess. Abbildung 5.15 zeigt beispielhaft die Paretooptima für die Schaltungen b18, b20, b22 und bench1.

Diese Schaltungen wurden ausgewählt, da sie gewisse Tendenzen in den Ergebnissen abbilden. Während die ITC99-Schaltung b18 in einigen Fällen bessere Ergebnisse mit dem Standardentwurf erzielt (höhere X-Reduktion bei gleicher Fehlereffizienz), ist die Situation bei der industriellen Schaltung bench1 umgekehrt. Hier ist der FAST-Entwurf deutlich besser, vor allem wenn höhere



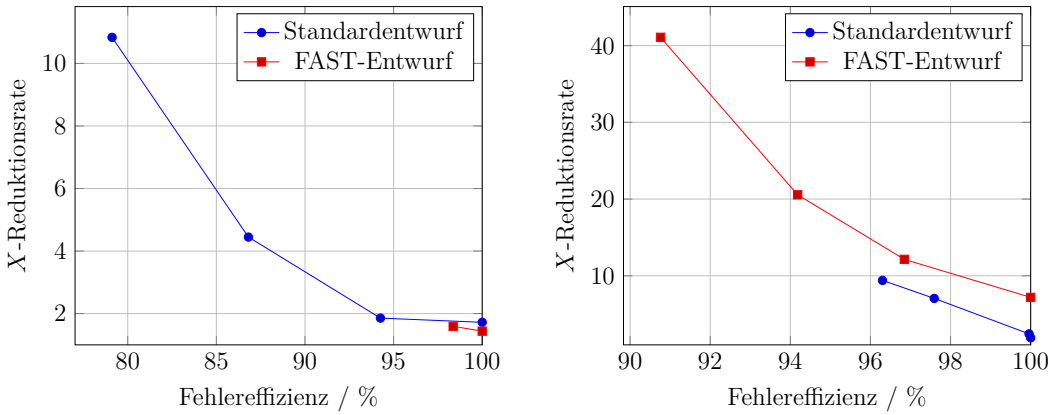
(a) b18 (b) bench1

(c) b20 (d) b22

Abbildung 5.15: Paretooptima der inkrementellen Maskierung

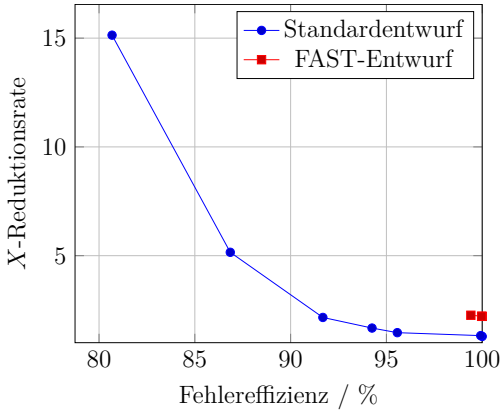
X-Reduktionsraten benötigt werden. Die Schaltungen b20 und b22 sind beim FAST-Entwurf vor allem im mittleren Bereich der Fehlereffizienz besser als beim Standardentwurf. Insgesamt zeigt sich aber auch, dass wesentliche Unterschiede erst bei steigenden X-Reduktionsraten auftreten. Anders ausgedrückt: Ist eine hohe Fehlereffizienz (mehr als 80% bei b18 oder 95% bei bench1) gewünscht, verschwinden die Unterschiede zwischen den Entwürfen zunehmend. In diesem Fall geht aber auch die X-Reduktionsrate schnell gegen 1, d. h. die Maskierung verliert generell an Effizienz.

Diese Analyse wurde auch für die Grenzwertmaskierung mit $q = 1$ durchgeführt. Dabei variierte der Parameter ρ_{max} von 0,1 bis 1 in Schritten von 0,1, gleichzeitig variierte ebenfalls $\gamma_{Essentiell}$ zwischen 0 und 1 in Schritten von 0,1. Die so erhaltenen Paretooptima sind in Abbildung 5.16 dargestellt. Mit Ausnahme der

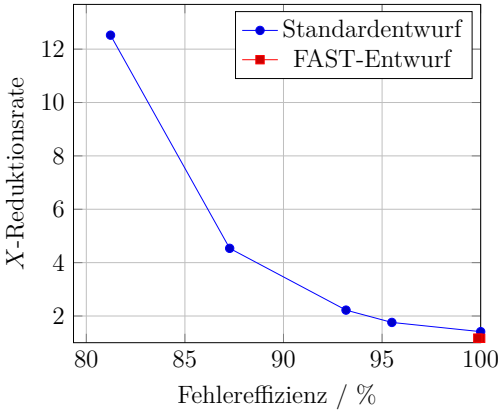


(a) b18

(b) bench1



(c) b20



(d) b22

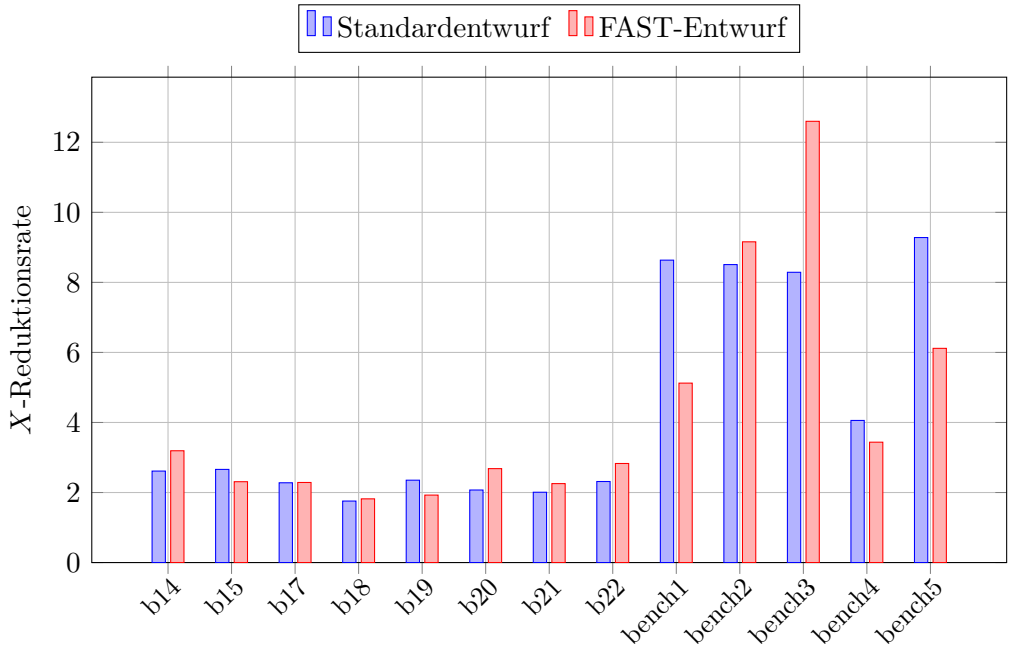
Abbildung 5.16: Paretooptima der Grenzwertmaskierung ($q = 1$)

industriellen Schaltung bench1 zeigen die Schaltungen im FAST-Entwurf nur wenige Paretooptima, welche zudem sehr nahe beieinander sind. Im Standardentwurf sind diese Optima weiter gestreut, sodass auch höhere X-Reduktionsraten (bei geringeren Fehlereffizienzen) erreicht werden können, falls dies gewünscht ist. Zudem zeigt sich für die ITC-Schaltungen b18 und b22, dass der Standardentwurf bessere Ergebnisse bei der Grenzwertmaskierung erzielt als der FAST-Entwurf. Eine Analyse ergab, dass bei einigen Schaltungen die Prüfzellen mit den höchsten X-Raten im Standardentwurf auf weniger Prüfpfade verteilt waren. Es zeigt sich, dass die Berechnung der topologischen X-Wahrscheinlichkeit lediglich die Auftrittswahrscheinlichkeit eines X-Wertes abschätzt, aber keine Aussage über Korrelationen zwischen den Prüfzellen macht. Topologisch zusammenhängende Strukturen (wie etwa ein Register) werden in dieser Variante nicht berücksichtigt, können aber im Standardentwurf zusammengefasst werden, da die entsprechenden Prüfzellen in der Schaltung nah beieinander liegen und einfach zu verbinden

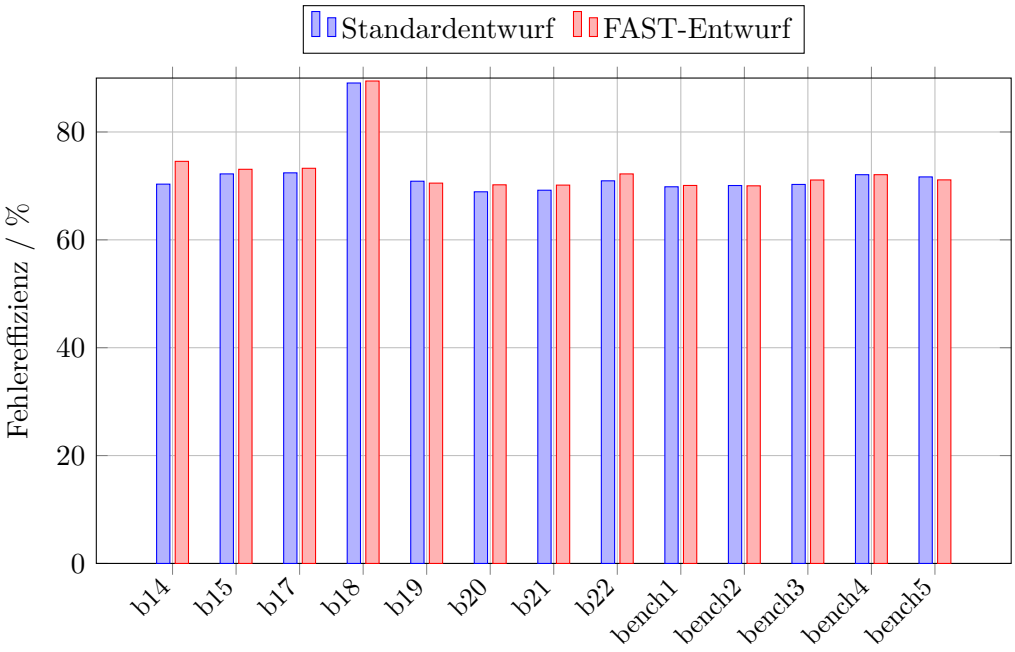
sind. Das bedeutet: Zeichnet eins der Flipflops in einer solchen Struktur einen X-Wert auf, ist es wahrscheinlich, dass die anderen Flipflops, welche von derselben Kombinatorik getrieben werden, ebenfalls einen X-Wert aufzeichnen. Dadurch kann es passieren, dass der Standardentwurf bei geringer Fenstergröße tatsächlich besser die X-Werte konzentriert als der FAST-Entwurf. Ist q daher klein, bietet sich eine genauere Analyse (wie hier gezeigt) an, um eine optimale Konfiguration zu finden.

Analyse mit erhöhter Maximalfrequenz

Wie bereits in Kapitel 4 geschehen, soll auch in diesem Abschnitt eine kleine Analyse unter einer erhöhten Maximalfrequenz durchgeführt werden. Die Prüfpfade im FAST-Entwurf wurden nicht verändert, daher können die Ergebnisse suboptimal sein. Die Motivation ist hier vor allem, dass sehr viel mehr X-Werte toleriert werden müssen, d. h. die Skalierbarkeit der Verfahren kann hier beobachtet werden. Dazu wurde, wie in Kapitel 4 beschrieben, die Frequenz- und Musterauswahl für eine Maximalfrequenz von $f_{max} = 6f_{nom}$ durchgeführt. Anschließend wurden für die so erzeugten Daten die Masken mit dem inkrementellen Verfahren berechnet und wieder verglichen. Wie zuvor auch wurden die Werte für ρ_{max} so gewählt, dass eine vergleichbare Fehlereffizienz erzielt wurde. Abbildung 5.17 zeigt die Ergebnisse der inkrementellen Maskierung unter erhöhter Maximalfrequenz, aufgeteilt in X-Reduktionsrate (Abbildung 5.17a) und Fehlereffizienz (Abbildung 5.17b). Die Ergebnisse der MISR-Simulation sind in Abbildung 5.18 zusammengefasst, Abbildung 5.18a zeigt die gesamte Fehlereffizienz nach Maske uns MISR, Abbildung 5.18b die Größe des resultierenden Speichers für die Zwischensignaturen. Die Daten dieser Abbildungen stammen aus Tabelle A.12. Man sieht, dass in diesem Fall die Ergebnisse weniger eindeutig und zugunsten des FAST-Entwurfs ausfallen. Die inkrementelle Maskierung kann beim FAST-Entwurf in einigen Fällen noch immer eine deutlich höhere X-Reduktionsrate bei praktisch gleicher Fehlereffizienz erzielen, z. B. bei der Schaltung bench3. Bei einigen Schaltungen ist die erreichte Fehlereffizienz des MISR gegenüber dem Standardentwurf allerdings reduziert, allerdings benötigt dann der MISR auch weniger Speicher für die Zwischensignaturen. Die Erklärung ist einfach: Bei den sehr aggressiven Frequenzen werden so viele X-Werte aufgezeichnet, dass es fast keine Rolle mehr

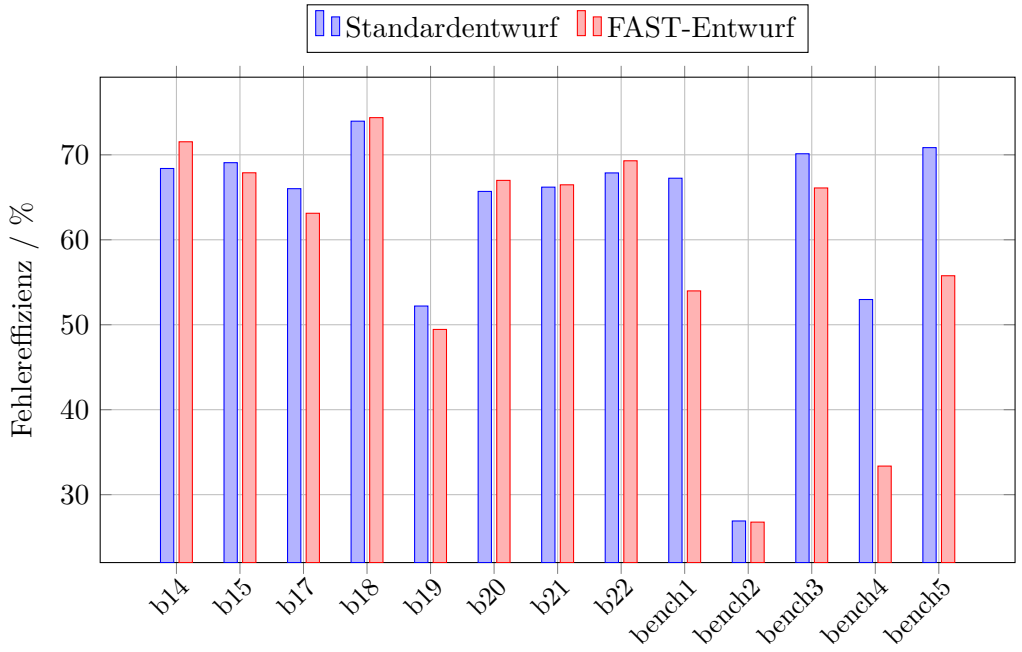


(a) X-Reduktionsrate

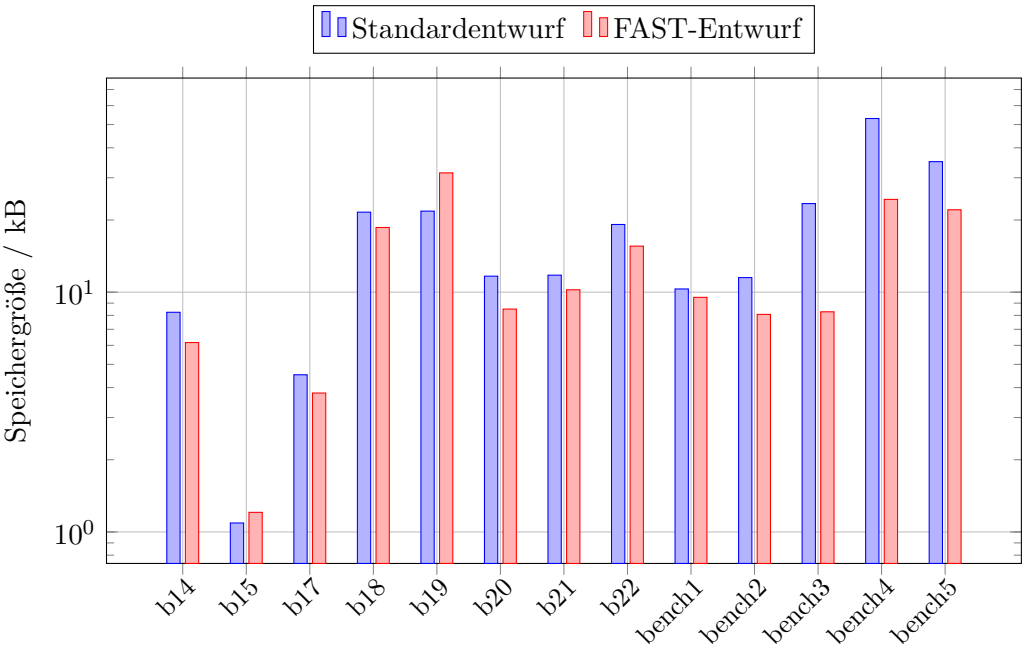


(b) Fehlereffizienz

Abbildung 5.17: Ergebnisse der inkrementellen Maskierung unter erhöhter Maximalfrequenz



(a) Gesamte Fehlereffizienz



(b) Speicherbedarf

Abbildung 5.18: Ergebnisse des MISR mit inkrementeller Maskierung unter erhöhter Maximalfrequenz

spielt, ob sie in den Prüfpfaden konzentriert werden oder nicht. Jeder Prüfpfad zeichnet in diesem Fall eine signifikante Menge an X-Werten auf, daher ist auch der Standardentwurf in der Lage, hohe X-Reduktionsraten zu erzielen.

5.5.4 Laufzeitanalyse

Zum Abschluss der Simulationsergebnisse soll noch kurz auf die Laufzeiten der Prüfzellengruppierung sowie Maskierungs- und Kompaktierungsalgorithmen eingegangen werden. Zunächst sind die Laufzeiten der Bestimmung der X-Profile (Verzögerungshistogramme) sowie der Prüfzellengruppierung für die einzelnen Schaltungen in Abbildung 5.19 dargestellt, die Daten stammen aus Tabelle A.13.

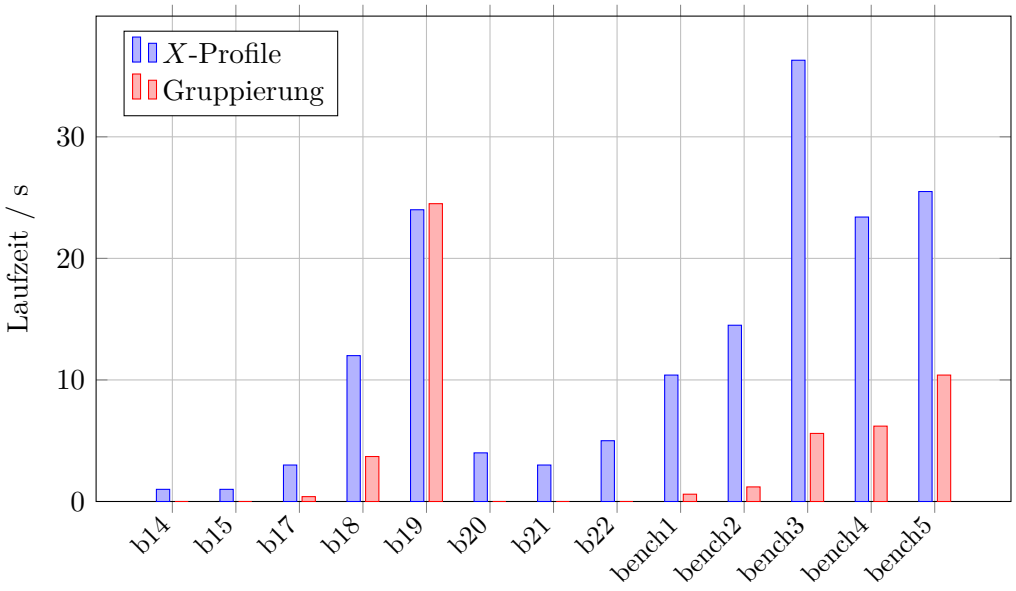


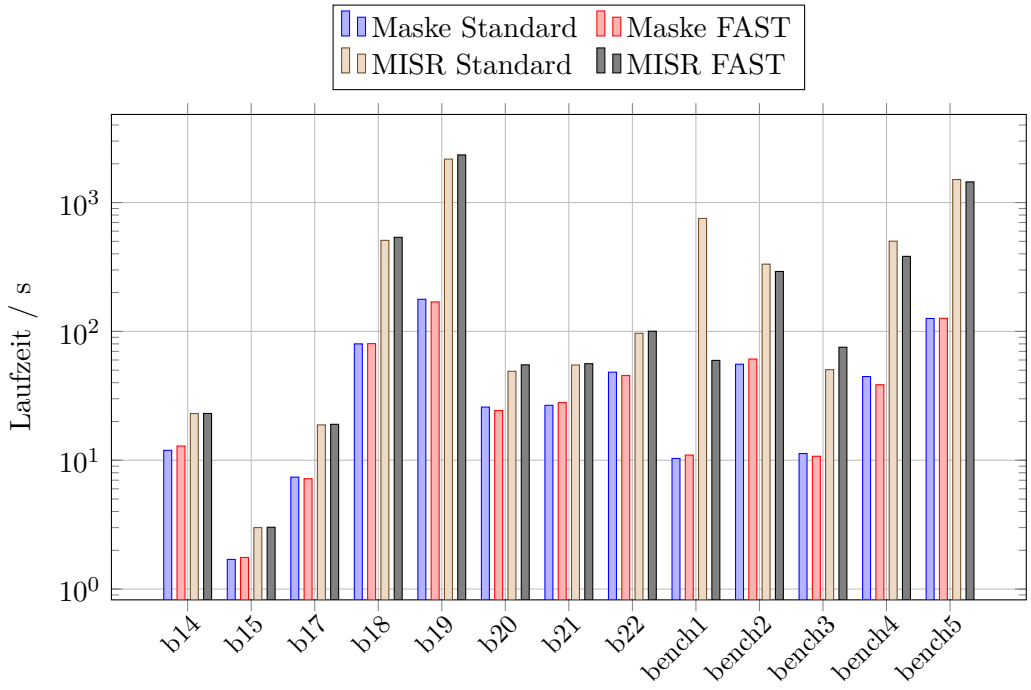
Abbildung 5.19: Laufzeit der Prüfzellengruppierung

Wie bereits in Abschnitt 5.2.3 beschrieben, wurde der Algorithmus zur Bestimmung der Prüfzellengruppen sehr effizient implementiert, was sich in den Laufzeiten widerspiegelt. Mit Ausnahme der Schaltung b19 dominiert bei allen Schaltungen die Laufzeit zur Bestimmung der X-Profile, d. h. die Berechnung der Verzögerungshistogramme. Bei vielen Schaltungen lag die Dauer der Gruppierung sogar deutlich unterhalb von einer Sekunde, daher ist diese im Diagramm praktisch nicht sichtbar.

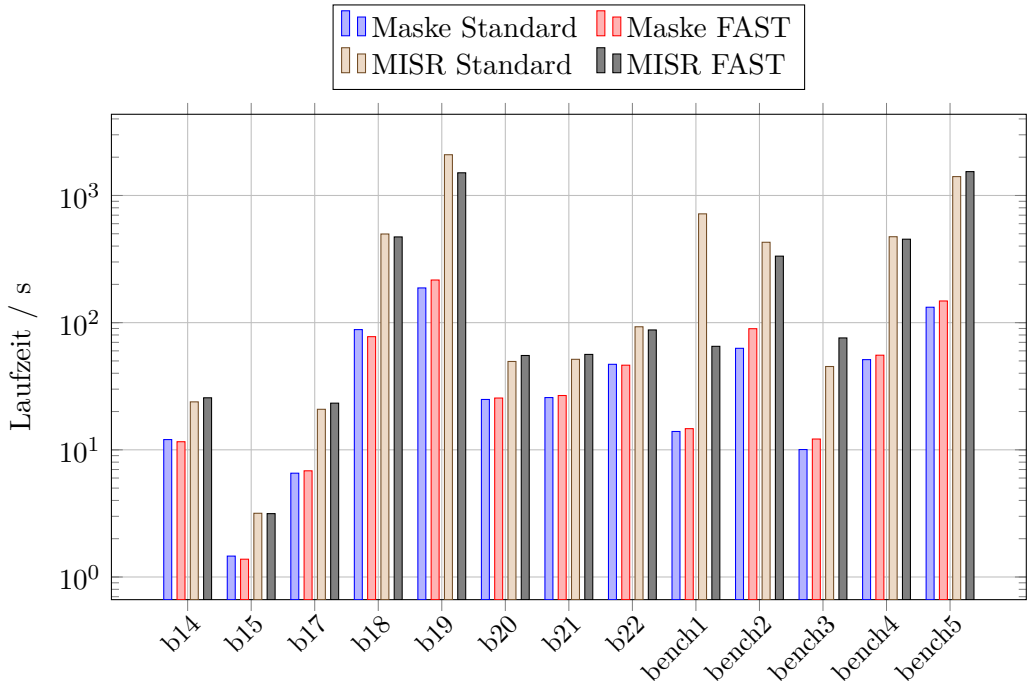
Für die Maskierung zeigt Abbildung 5.20 die Laufzeiten der Maskenberechnung und des MISR an, jeweils für die inkrementelle Maskierung (Abbildung 5.20a) und für die Grenzwertmaskierung mit $q = 1$ (Abbildung 5.20b). Die MISR-Berechnung wird dabei auf den maskierten Daten ausgeführt, Laufzeiten des MISR auf unmaskierten Daten sind in Tabelle A.17a gegeben. Zu beachten ist, dass in den Abbildungen die vertikale Achse logarithmisch dargestellt ist, um die Laufzeiten sauber anzeigen zu können. Für beide Varianten ergeben sich nur geringe Unterschiede in den jeweiligen Laufzeiten. Die Berechnung der Grenzwertmaskierung kann zusätzlich beschleunigt werden, da die Frequenzen unabhängig voneinander betrachtet werden können, was eine nebenläufige Berechnung der Masken ermöglicht. Das ist bei der inkrementellen Maskierung nicht möglich, da die Maske einer Frequenz von den Masken der vorhergehenden Frequenzen abhängt. Nichtsdestotrotz ist die Berechnung der Masken für alle Schaltungen in weniger als 100 s möglich, die Berechnung des MISR dauert je nach Schaltung deutlich länger.

5.6 Zusammenfassung

Dieses Kapitel beschäftigt sich mit einer Methode zum prüfgerechten Entwurf für den Hochgeschwindigkeitstest. Ziel ist, X-tolerante Kompaktierung beim Selbsttest zu unterstützen, da FAST variierende und teils sehr hohe X-Raten produzieren kann. Dazu wurde ein Verfahren vorgestellt, um Prüfzellen zu gruppieren. Hier wurde ausgenutzt, dass die X-Werte, welche bei FAST entstehen, nicht zufällig über die Prüfzellen verteilt sind; es können sogenannte X-Profile bestimmt werden. Diese X-Profile wurden ausgenutzt, um in einem schnellen, Graph-basierten Verfahren Gruppen an Flipflops zu erstellen, welche zu einem Prüfpfad verbunden werden sollen. Diese Gruppen stellen allerdings noch keine konkreten Prüfpfade dar, sondern können als Randbedingungen für kommerzielle Syntheseprogramme verwendet werden, welche dann die konkreten Prüfpfade erzeugen. Zusätzlich wurde ein sehr einfaches Verfahren zur Prüfpfadmaskierung vorgestellt, welches lediglich ein Schieberegister benötigt, da ein einmal maskierter Prüfpfad für den Rest des Tests weiterhin maskiert bleibt. Die Ergebnisse zeigen, dass diese Art der Maskierung besonders gut mit den speziellen Prüfpfaden zu-



(a) Inkrementelle Maskierung



(b) Grenzwertmaskierung

Abbildung 5.20: Laufzeiten der Maskierung und des MISR

sammenarbeitet und höhere X-Reduktionsraten bei gleicher (oder sogar höherer) Fehlereffizienz erzielt als mit „normalen“ Prüfpfaden. Bei der feingranulareren Grenzwertmaskierung sind die Ergebnisse sogar noch besser, allerdings muss zusätzlicher Aufwand betrieben werden, um die Maskendaten zu speichern. Obgleich in einigen wenigen Fällen der Standardentwurf bessere Ergebnisse erzielt, profitiert der überwiegende Teil der Schaltungen von den optimierten Prüfpfaden für FAST. Das bedeutet, dass der möglicherweise anfallende zusätzliche Aufwand bei der Prüfpfadkonfiguration durch das Syntheseprogramm in vielen Fällen lohnenswert ist, da die Testdatenauswertung so bessere Ergebnisse mit weniger Steuerungsaufwand erreichen kann. Da die Berechnung der spezialisierten Prüfpfade sich zudem sehr leicht in den Syntheseprozess integrieren lässt, ist diese Methode eine vielversprechende Erweiterung, die im Entwurfsprozess untersucht werden sollte.

Aus den vorangegangen Betrachtungen ergibt sich eine zusätzliche Erweiterung der Bestimmung der X-Profile um mögliche Korrelationen. Ein einfacher Ansatz ist hier, die transitiven Eingangskegel der Prüfzellen zu betrachten um topologisch zusammenhängende Strukturen zu identifizieren. Haben zwei Prüfzellen einen großen gemeinsamen Eingangskegel, so ist es wahrscheinlich, dass beide Prüfzellen beim selben Muster einen X-Wert aufzeichnen. Alternativ können zur genaueren Berechnung der topologischen X-Wahrscheinlichkeit statistische Methoden wie der Ansatz von Wagner und Wunderlich [124, 123] berücksichtigt werden. Hier zeigt sich wieder der Vorteil des Verfahrens: Es muss lediglich die Berechnung der X-Profile angepasst werden, der restliche Ablauf bleibt gleich. Auch die Grenzwertmaskierung bietet Potential zur Optimierung. Beispielsweise kann die Fenstergröße adaptiv gestaltet sein – bei geringen X-Raten können sehr hohe Fenstergrößen gewählt werden, ohne dass viele Fehlerinformationen verloren gehen, bei hohen X-Raten ist eine kleine Fenstergröße effektiver. Je nach vorhandener X-Rate kann daher die Fenstergröße pro Frequenz festgelegt werden, um Maskenspeicher zu minimieren und gleichzeitig die Effektivität zu steigern. Für $q > 1$ bietet es sich zudem an, die Testmuster optimiert anzuordnen. Beispielsweise können die Muster so angeordnet werden, dass die X-Werte in einem Fenster auf möglichst wenige Prüfpfade konzentriert sind. Dadurch kann auch ein unnötiger Verlust an Fehlerinformationen vermieden werden.

6 Zusammenfassung

Der stetige Fortschritt in den Prozesstechnologien hat eine Vielzahl von neuen Anwendungen und Einsatzgebieten für hochintegrierte Schaltungen entstehen lassen. Doch die Fertigung immer kleinerer Schaltungsstrukturen ist anfällig gegenüber Prozessvariationen und kleinen Verzögerungsdefekten (SDDs). Gefährlich sind versteckte Verzögerungsfehler, welche sich im Herstellungstest nicht bemerkbar machen und im Betrieb zunächst keine Auswirkungen haben. Sie können aber schnell ausarten und zu einem frühzeitigen Systemausfall (ELF) führen – dies muss in kritischen Anwendungen wie selbstfahrenden Fahrzeugen, Flugzeugen, usw. verhindert werden. Bestehende Testmethoden setzen auf Hochtemperatur- und Hochspannungstests („Burn-in“), diese sind aber kostenintensiv und senken die Qualität der getesteten Schaltungen.

Der Hochgeschwindigkeitstest (FAST) bietet eine kostengünstige Alternative. Hier wird die zu testende Schaltung übertaktet, um so selbst kleinste zusätzliche Verzögerungen sichtbar zu machen. Die vorliegende Arbeit beschäftigt sich mit FAST und hat als Kernthema die Erweiterung zum eingebauten Selbsttest (BIST). Der eingebaute Hochgeschwindigkeitstest ist sehr attraktiv, erlaubt er doch das periodische Testen einer Schaltung im Feld ohne zusätzliche Testausrüstung. Dadurch können mögliche Probleme, die beim Herstellungstest nicht entdeckt wurden, frühzeitig erkannt werden, bevor sie zu einem ELF führen. Um diesen Selbsttest zu realisieren, wurde in dieser Arbeit eine Selbsttestarchitektur vorgestellt, welche auf der klassischen STUMPS-Architektur basiert und somit kompatibel zu vielen anderen Selbsttestarchitekturen ist.

Zwei wesentliche Probleme beim Selbsttest wurden adressiert: Zum einen müssen Testfrequenzen ausgewählt werden, um möglichst viele versteckte Verzögerungsfehler sichtbar zu machen. Zum anderen muss die Testantwortevaluation im

Selbsttest erweitert werden, um die hohen und variierenden X-Raten unter FAST handhaben zu können. Für die Frequenzauswahl wurde ein Verfahren vorgestellt, um Testfrequenzen für einen gegebenen Testsatz so zu bestimmen, dass maximale Fehlereffizenz erreicht wird – d. h. jeder potentiell erkennbare Fehler wird unter mindestens einer Testfrequenz erkannt. Zur Minimierung der Anzahl der benötigten Frequenzen wurde zudem ein optimales Verfahren implementiert und zusätzlich ein hybrides Verfahren vorgestellt, welches den Rechenaufwand der optimalen Frequenzauswahl verringert. Weiterhin wurde eine Fehlerpartitionierung mit anschließender Musterauswahl vorgestellt, um die gesamte Testdauer weiter zu verringern. Hohe und variable X-Raten lassen sich tolerieren, indem eine spezielle Anordnung der Prüfzellen in Prüfpfaden mit einer sehr einfachen inkrementellen Maskierung kombiniert wird. Dabei werden für die Prüfzellen die besonderen X-Profile berechnet, verglichen und es wird versucht, Prüfzellen mit gleichem X-Profil zu einem Prüfpfad zu verbinden. Die so berechneten Randbedingungen können einem Standardsyntheseprogramm übergeben werden, welches daraus die konkreten Prüfpfade erzeugt. Dadurch werden X-Werte so in Prüfpfaden konzentriert, dass mit der inkrementellen Maskierung bei den meisten Schaltungen eine höhere X-Reduktionsrate bei gleicher oder sogar höherer Fehlereffizienz beobachtet werden konnte. Das unterstützt vor allem den Entwurf optimierter Verfahren zur X-toleranten Testantwortkompaktierung, sodass insgesamt der Aufwand zur Implementierung des eingebauten Hochgeschwindigkeitstests verringert werden kann.

6.1 Ausblick

Durch die in dieser Arbeit vorgestellten Lösungsansätze rückt eine konkrete Implementierung des eingebauten Hochgeschwindigkeitstests in die Nähe. Freilich verbleiben aber noch weitere Herausforderungen, die für einen erfolgreichen Einsatz gemeistert werden müssen. In den vorangehenden Kapiteln wurde bereits auf konkrete Erweiterungen der in dieser Arbeit vorgestellten Lösungsverfahren eingegangen. In diesem Abschnitt soll vor allem eine bislang noch nicht nennenswert beachtete Herausforderung vorgestellt werden: Die Erzeugung von Testmustern für den Selbsttest auf der Schaltung selbst. Die bisherigen Verfahren für FAST set-

zen auf deterministische Testsätze, welche mit einem Programm zuvor berechnet wurden. Diese können zwar hoch optimiert sein, müssen aber auf der Schaltung gespeichert werden. Obgleich es sehr effiziente Verfahren zur Testsatzkompression gibt [115], muss in jedem Fall Speicher auf der Schaltung reserviert werden, was typischerweise sehr teuer ist.

Für bisherige Selbsttests wird dabei vorrangig auf pseudo-zufällige Testmustererzeugung (engl. Pseudo-Random Pattern Generation, PRPG) zurückgegriffen. PRPG lässt sich sehr einfach in Hardware umsetzen, beispielsweise über ein LFSR in Kombination mit einer Phasenverschiebungsschaltung, was gemeinhin auch als „Logic BIST (LBIST)“ bezeichnet wird [30, 1]. Es ist aber fraglich, ob sich solch ein simples Verfahren direkt für FAST einsetzen lässt. Zwar wurde in der Vergangenheit gezeigt, dass LBIST auch für Übergangsfehler geeignet ist, aber die Eigenarten von kleinen Verzögerungsfehlern benötigen besondere Eigenschaften der erzeugten Testmuster, etwa das Sensibilisieren von Pfaden mit sehr hoher Verzögerung. PRPG lässt sich aber nur schwer kontrollieren, sodass Erweiterungen für eine Anwendung im eingebauten Hochgeschwindigkeitstest notwendig sind. Allerdings müssen auch nicht sämtliche Testmuster über PRPG erzeugt werden, das Abspeichern kleinerer deterministischer Testsätze kann einigen Qualitätsgewinn bei nur geringfügig höheren Kosten erzielen [26, 40, 27, 28].

Ein möglicher Lösungsansatz kann darin bestehen, Verfahren wie STARBIST bzw. Stellar BIST [50, 117] zu nutzen. Diese verwenden ein Basismuster, welches beispielsweise durch optimerte Testmustererzeugung berechnet wurde, um durch kleine Modifikationen weitere gute Muster abzuleiten. Einen gleichwertigen Ansatz stellt die gewichtete pseudo-zufällige Testmustererzeugung dar [64, 84, 73]. Hier kann versucht werden, durch geeignete Auswahl der Gewichte die Wahrscheinlichkeit für gute Testmuster stark zu erhöhen. Es muss noch im Detail analysiert werden, inwiefern die Ansätze für FAST verwendbar und welche zusätzlichen Erweiterungen erforderlich sind, entsprechende Forschungsarbeiten wurden aber bereits aufgenommen.

A Wertetabellen

Dieser Anhang enthält ausführliche Wertetabellen zu den experimentellen Ergebnissen dieser Arbeit. Sie stellen die Grundlage der in den Kapiteln 4 und 5 gegeben Diagramme dar.

A.1 Synthetisierte Schaltungen

Es wurde eine Reihe von ITC99-Benchmarkschaltungen [16] in zwei Synthesedurchläufen mit einer 32 nm CMOS Standardzellbibliothek der Firma Synopsys für Forschungs- und Lehrzwecke [85] erzeugt. Im ersten Durchlauf wurden die Schaltungen mit einem konventionellem Ablauf synthetisiert (Standardentwurf), im zweiten Durchlauf wurden die Schaltungen nach dem in Kapitel 5 beschriebenen prüfgerechten Entwurf für den eingebauten Hochgeschwindigkeitstest synthetisiert (FAST-Entwurf). Zusätzlich wurden einige industrielle Benchmarkschaltungen analysiert, welche mit einer 90 nm Standardzellbibliothek[1] vorsynthetisiert wurden und als fertige Netzliste und Prüfpfadkonfiguration vorliegen. Aus diesem Grund konnte für diese Schaltungen keine vollständige Neusynthese durchgeführt werden. Es war jedoch möglich, die spezialisierten Prüfpfade zu emulieren, indem die Ergebnisse des in Kapitel 5 beschriebenen Verfahrens direkt verwendet wurden, d. h. die Prüfzellengruppen wurden direkt als Prüfpfade in den weiteren Programmen betrachtet. Die Ergebnisse der Synthese befinden sich in Tabelle A.1. Tabelle A.1a gibt die Eigenschaften des Standardentwurfs, Tabelle A.1b die entsprechenden Ergebnisse des FAST-Entwurfs an. Da die industriellen Schaltungen nicht neu synthetisiert wurden, zeigt Tabelle A.1b für diese Schaltungen nur die Anzahl an

[1] Ebenfalls von Synopsys für Forschungs- und Lehrzwecke bereitgestellt. Die Bibliothek ist jedoch inzwischen nicht mehr verfügbar, es wird auf die neuere 32 nm-Bibliothek verwiesen.

Schaltung	t_{nom} / ps	PE	PA	Gatter		Prüfpade
				Kombinatorisch	Flipflop	
b14	3342	32	59	9929	215	9
b15	3298	37	82	7018	417	17
b17	3490	38	142	21 497	1317	53
b18	4518	37	146	68 897	3064	123
b19	4496	21	276	132 044	6130	246
b20	4520	32	40	19 544	430	18
b21	4519	32	40	19 361	430	18
b22	4519	32	48	29 342	645	26
bench1	3241	1408	219	22 414	2331	97
bench2	1568	171	507	46 504	2977	65
bench3	3700	152	75	78 665	3877	8
bench4	2833	331	261	56 662	4296	18
bench5	3232	167	94	53 836	5735	18

(a) Standardentwurf

Schaltung	t_{nom} / ps	PE	PA	Gatter		Prüfpade
				Kombinatorisch	Flipflop	
b14	3342	32	56	9927	215	9
b15	3298	37	74	7013	417	17
b17	3490	38	116	21 473	1317	53
b18	4498	37	146	68 894	3064	123
b19	4496	21	276	132 046	6130	246
b20	4520	32	40	19 546	430	18
b21	4519	32	40	19 363	430	18
b22	4520	32	48	29 341	645	26
bench1						97
bench2						120
bench3						156
bench4						172
bench5						230

(b) FAST-Entwurf

Tabelle A.1: Synthetisierte Schaltungen

erzeugten Prüfpfaden an (Spalte 7). Spalte 1 gibt den Namen der Schaltungen an: Zunächst die ITC99-Schaltungen (b14, ...), dann folgen die industriellen Schaltungen (bench1, ..., nur Tabelle A.1a). In der zweiten Spalte wird die nominelle Taktperiode t_{nom} der Schaltung in Pikosekunden aufgeführt. Spalten drei und vier geben die Anzahl an primären Ein- und Ausgängen der Schaltungen an. Spalten fünf und sechs führen für die Schaltungen die Anzahl an Gattern auf, unterteilt in kombinatorische Gatter (UND, ODER, ...) und Flipflops. Alle Schaltungen sind als vollständige Prüfpfadarchitekturen realisiert, Spalte 7 gibt die Anzahl an erzeugten Prüfpfaden pro Schaltung an. Dadurch haben die industriellen Schaltungen verschiedene Anzahlen an Prüfpfaden. Die Parameter für die Prüfzellengruppierung des FAST-Entwurfs sind in Tabelle A.2 aufgeführt. Spalten fünf und sechs zeigen zusätzlich, wie viele Prüfzellen nach Phase 1 bzw. Phase 2 des Algorithmus keiner Gruppe zugeordnet werden konnten. Spalte sechs gibt somit die verbleibenden Prüfzellen an, die lediglich zum Ausbalancieren der Prüfpfadlängen verwendet wurden.

Schaltung	$\lambda_{Konnektivität}$	$\lambda_{Distanz}$	$\lambda_{Affinität}$	Verbleibende Prüfzellen	
				Phase 1	Phase 2
b14	0,38	1,00	0,5	36	11
b15	$< 10^{-5}$	$< 10^{-5}$	0,5	56	56
b17	$< 10^{-5}$	$< 10^{-5}$	0,5	131	131
b18	$< 10^{-5}$	0,86	0,5	518	239
b19	$< 10^{-5}$	0,86	0,5	958	391
b20	0,29	0,99	0,5	21	3
b21	0,29	0,99	0,5	33	6
b22	0,29	0,99	0,5	26	0
bench1	$< 10^{-5}$	0,11	0,5	59	20
bench2	0,31	0,87	0,5	13	0
bench3	$< 10^{-5}$	$< 10^{-5}$	0,5	49	49
bench4	$< 10^{-5}$	0,44	0,5	207	80
bench5	$< 10^{-5}$	0,11	0,5	206	98

Tabelle A.2: Parameter und Ergebnisse der Prüfzellengruppierung

A.2 Automatisierte Testmustererzeugung

Für alle Schaltungen wurden mit einem kommerziellen ATPG-Programm Testsätze erzeugt, die Ergebnisse befinden sich in Tabelle A.3.

Schaltung	$\mathscr{P}_{Initial}$		$\mathscr{P}_{Extra}$	
	$\lvert\mathscr{P}\rvert$	FA / %	$\lvert\mathscr{P}\rvert$	FA / %
b14	1185	97,6	1425	98,2
b15	998	88,7	464	87,8
b17	1174	87,2	733	89,7
b18	1885	81,7	1478	72,3
b19	2768	80,9	2273	71,1
b20	1417	97,0	1814	98,9
b21	1483	97,3	1937	98,0
b22	1687	97,1	248	98,1
bench1	2413	100,0	181	99,9
bench2	76	100,0	72	100,0
bench3	285	100,0	133	99,9
bench4	609	100,0	476	100,0
bench5	2179	99,9	213	99,6

(a) Standardentwurf

Schaltung	$\mathscr{P}_{Initial}$		$\mathscr{P}_{Extra}$	
	$\lvert\mathscr{P}\rvert$	FA / %	$\lvert\mathscr{P}\rvert$	FA / %
b14	1176	97,6	1422	98,2
b15	947	88,6	503	87,6
b17	1168	87,2	731	89,6
b18	1885	81,7	1437	72,1
b19	2791	80,9	2362	71,1
b20	1462	97,0	1856	97,8
b21	1520	97,2	2010	98,1
b22	1683	97,1	2138	98,0

(b) FAST-Entwurf

Tabelle A.3: Ergebnisse der Testmusterzerzeugung

Tabelle A.3a zeigt die Ergebnisse für die Schaltungen im Standardentwurf, Tabelle A.3b die entsprechenden Ergebnisse für den FAST-Entwurf. Für die ITC99-Schaltungen wurden Launch-on Capture (LOC) Testsätze erzeugt. Da

das ATPG-Programm bei LOC nur den ersten Testvektor ausgibt, musste der zweite Testvektor durch eine zusätzliche Simulation bestimmt werden. Dabei wurde angenommen, dass zwischen dem letzten Schiebetakt und dem ersten Testtakt genügend Zeit vorhanden ist, dass alle Ausgangssignale stabil werden, sodass der zweite Testvektor über eine einfache Logiksimulation bestimmt werden konnte. Für die industriellen Schaltungen fehlten Testprotokolle, daher konnten die Prüfpfade bei der Testmusterberechnung nicht berücksichtigt werden. In Folge sind die erzeugten Testsätze praktisch Enhanced Scan (ES) Testsätze. Da für die industriellen Schaltungen nur eine Synthesevariante vorliegt, sind die Ergebnisse dieser Schaltungen nur in Tabelle A.3a zu finden. Es wurden zwei verschiedene Testsätze erzeugt. $\mathscr{P}_{Initial}$ bezeichnet einen einfachen Testsatz für Übergangsfehler, welcher ohne besondere Optimierungen erzeugt wurde, $\mathscr{P}_{Extra}$ einen zusätzlichen Testsatz, welcher besonders lange Pfade sensibilisiert (vgl. Abschnitt 3.3). Zu beachten ist, dass die Fehlereffizienz von $\mathscr{P}_{Extra}$ sich auf die schwer zu entdeckenden HDF bezieht, vergleiche dazu auch Abschnitt 4.6.

A.3 FAST-Simulation und Frequenzauswahl

Tabelle A.4 zeigt die Ergebnisse der Reduktion der SDF-Menge durch topologische Voranalyse. Tabelle A.4a enthält die Ergebnisse für den Standardentwurf, Tabelle A.4b die entsprechenden Ergebnisse für den FAST-Entwurf. Da bei letzterem nur die ITC-Schaltungen neu synthetisiert wurden, sind dort folglich nur Ergebnisse dieser Schaltung zu finden. Spalte 1 zeigt den Schaltungsnamen, gefolgt von der Grundmenge an SDFs, welche aus der Fehlerliste des ATPG-Programms erzeugt wurden. Spalten 3 und 4 geben dann diejenigen Teilmengen an SDFs an, welche keiner Simulation unterzogen werden müssen, weil sie entweder selbst mit FAST unerkennbar sind (Spalte 3) oder über einen konventionellen Test mit Betriebsfrequenz erkannt werden können (Spalte 4). In Spalte 5 ist dann die Anzahl der verbleibenden relevanten SDFs aufgeführt, welche auch als HDF-Kandidaten bezeichnet werden, gefolgt von deren prozentualem Anteil in Spalte 6.

Die Ergebnisse der FAST-Simulation sind in der Tabelle A.5 aufgeführt. Tabelle A.5a enthält wieder die Daten des Standardentwurfs, Tabelle A.5b entsprechend

Schaltung	Grundmenge	Unerkennbar	Betriebsfreq.	Relevant	%
b14	42 340	7266	0	35 074	82,8
b15	29 756	21 973	0	7783	26,2
b17	90 822	69 727	0	21 095	23,2
b18	293 578	172 421	0	121 157	41,3
b19	559 758	341 736	0	218 022	39,0
b20	80 738	15 579	0	65 159	80,7
b21	79 986	15 234	0	64 752	81,0
b22	121 010	23 050	0	97 960	81,0
bench1	87 936	52 730	0	35 206	40,0
bench2	208 262	33 910	0	174 352	83,7
bench3	334 868	259 122	0	76 746	22,9
bench4	225 386	138 627	0	86 759	38,5
bench5	218 234	125 144	0	93 090	42,7

(a) Standardentwurf

Schaltung	Grundmenge	Unerkennbar	Betriebsfreq.	Relevant	%
b14	42 340	7266	0	35 074	82,8
b15	29 756	21 973	0	7783	26,2
b17	90 822	69 727	0	21 095	23,2
b18	293 586	172 268	0	121 318	41,3
b19	559 756	341 759	0	217 997	39,0
b20	80 740	15 577	0	65 163	80,7
b21	79 986	15 232	0	64 754	81,0
b22	121 008	23 050	0	97 958	81,0

(b) FAST-Entwurf

Tabelle A.4: Ergebnisse der topologischen Voranalyse mit $s_{min} = 6\sigma$

die Daten des FAST-Entwurfs. Spalte 2 zeigt zunächst für jede Schaltung die Menge an relevanten SDF (Ergebnis der topologischen Voranalyse). Für die zwei Simulationen ($\mathscr{P}_{Initial}$ und $\mathscr{P}_{Extra}$) wird zuerst die Anzahl an potentiell erkennbaren HDF gelistet, d. h. HDF, welche sich zu einem beliebigen Zeitpunkt erkennen lassen (zur Berechnung der idealen HDF-Abdeckung). Dann wird wird die Anzahl an HDF gegeben, welche sich mit Beobachtungszeitpunkten in $[t_{min}, t_{nom})$ erkennen lassen.

Schaltung	SDF	Simulation 1			Simulation 2		
		t_{nom}	Ideal	FAST	t_{nom}	Ideal	FAST
b14	35 074	0	30 009	10 731	0	32 172	18 964
b15	7783	0	5789	2112	0	6272	3471
b17	21 095	0	15 684	5271	0	16 982	8463
b18	121 157	0	83 148	36 954	0	88 431	53 613
b19	218 022	0	151 271	48 463	0	158 718	81 366
b20	65 159	0	55 161	9512	0	59 329	28 100
b21	64 752	0	55 633	11 095	0	59 561	28 656
b22	97 960	0	84 621	15 416	0	90 205	42 989
bench1	35 206	0	28 404	11 896	0	33 463	14 869
bench2	174 352	0	145 811	140 377	0	164 173	158 751
bench3	76 746	0	66 405	9864	0	73 486	12 783
bench4	86 759	0	42 292	10 394	0	70 219	33 829
bench5	93 090	0	79 147	52 308	0	86 536	58 492

(a) Standardentwurf

Schaltung	SDF	Simulation 1			Simulation 2		
		t_{nom}	Ideal	FAST	t_{nom}	Ideal	FAST
b14	35 074	0	30 286	11 446	0	32 279	18 895
b15	7783	0	5655	3591	0	6281	3458
b17	21 095	0	15 703	5656	0	16 958	8498
b18	121 318	0	83 654	37 625	0	88 519	53 379
b19	217 997	0	151 235	48 472	0	158 795	81 207
b20	65 163	0	55 404	9276	0	59 419	28 706
b21	64 754	0	55 849	9908	0	59 703	28 724
b22	97 958	0	84 540	15 345	0	90 194	42 353

(b) FAST-Entwurf

Tabelle A.5: Ergebnisse FAST-Simulation

Tabelle A.6 zeigt die Ergebnisse der Frequenzauswahl für die verschiedenen Algorithmen und Entwürfe, sowohl für den Standard- als auch für den FAST-Entwurf. Für letzteren wurde lediglich die hybride Frequenzauswahl durchgeführt.

Schaltung	Standardentwurf			FAST-Entwurf
	MHS-LUB	Hybrid	Optimal	Hybrid
b14	119	106	101	104
b15	67	42	36	43
b17	90	59	53	58
b18	130	120	116	118
b19	138	121	117	123
b20	144	133	131	134
b21	141	131	129	134
b22	157	150	147	147
bench1	40	40	34	
bench2	58	59	51	
bench3	24	24	23	
bench4	66	67	60	
bench5	96	83	80	

Tabelle A.6: Ergebnisse der Frequenzauswahl

A.3.1 FAST mit erhöhter Maximalfrequenz

Für alle untersuchten Schaltungen wurde neben einer Simulation mit $f_{max} = 3f_{nom}$ auch eine Simulation mit $f_{max} = 6f_{nom}$ durchgeführt. Dabei wurde auf die Ergebnisse der ersten Simulation zurückgegriffen, sodass für die Schaltungen lediglich eine zweite Simulation durchgeführt werden musste. Die Ergebnisse sind in Tabelle A.7 gegeben, Tabelle A.7a für den Standardentwurf und Tabelle A.7b für den FAST-Entwurf. Wie zuvor auch zeigt die zweite Spalte die Menge an relevanten SDFs für die Simulation, während die Spalten drei bis fünf die Ergebnisse der zweiten Simulation bzw. Analyse der Erkennungsbereiche zeigt. Die Ergebnisse der Frequenzauswahl für die erhöhte Maximalfrequenz sind in Spalte 6 dargestellt, hier wurde allerdings lediglich die hybride Frequenzauswahl durchgeführt. Bei der Schaltung b17 dauerte die Frequenzauswahl mit MHS-Hypergraph (Algorithmus 4.4) so lange, dass das PC^2 nach sieben Tagen die Berechnung abbrechen

Schaltung	SDF	Simulation 2			Frequenzen
		t_{nom}	Potentiell	FAST	
b14	35 074	0	32 121	31 865	123
b15	7783	0	6321	5589	55
b17	21 095	0	16 970	14 778	80
b18	121 157	0	88 069	75 420	160
b19	218 022	0	15 492	134 602	144
b20	65 159	0	59 346	54 295	165
b21	64 752	0	59 554	55 038	167
b22	97 960	0	90 105	83 735	180
bench1	35 206	0	28 404	27 131	67
bench2	174 352	0	164 194	162 457	73
bench3	75 746	0	73 425	69 076	63
bench4	86 759	0	70 678	66 897	91
bench5	93 090	0	79 147	76 310	108

(a) Standardentwurf

Schaltung	SDF	Simulation 2			Frequenzen
		t_{nom}	Potentiell	FAST	
b14	35 074	0	32 201	31 946	162
b15	7783	0	6289	5592	55
b17	21 095	0	16 905	14 603	79
b18	121 318	0	88 263	75 527	156
b19	217 997	0	158 728	134 766	138
b20	65 163	0	59 273	54 165	166
b21	64 754	0	59 671	55 370	167
b22	97 958	0	90 216	83 539	174

(b) FAST-Entwurf

Tabelle A.7: Ergebnisse FAST-Simulation und Frequenzauswahl mit $6f_{nom}$

musste. Aus diesem Grunde wurde die hybride Frequenzauswahl mit einem gierigen Algorithmus durchgeführt, welcher die Frequenzen für die verbleibenden Fehler auswählte.

A.4 Fehlerpartitionierung und Musterauswahl

Tabelle A.8 zeigt die Ergebnisse der Testmusterauswahl für die FAST-Gruppen. Sie ist erneut aufgeteilt in Tabelle A.8a (Standardentwurf) und Tabelle A.8b (FAST-Entwurf). Für die Musterauswahl wurden die hybride und optimale Frequenzauswahl betrachtet, beim FAST-Entwurf lediglich die hybride Frequenzauswahl. Spalte 2 zeigt jeweils die Anzahl an Testmustern in der Ausgangsmenge ($\mathscr{P}_{Total}$). Spalten 3 und 4 geben an, wie viele Testmuster an die Schaltung insgesamt angelegt werden, wenn die ausgewählten Testmuster für die FAST-Gruppen getestet werden. Spalten 5 und 6 geben dann an, wie viele verschiedene Testmuster von $\mathscr{P}_{Total}$ verwendet wurden (ein Muster kann dabei bei verschiedenen Frequenzen wiederverwendet werden).

A.5 Testdatenauswertung

Für alle Schaltungen wurde zur Signaturberechnung ein einzelnes X-streichendes MISR verwendet. Zunächst zeigt Tabelle A.9 (aufgeteilt in Tabelle A.9a für den Standard- und Tabelle A.9b für den FAST-Entwurf) die Ergebnisse des X-streichenden MISR alleine an, d. h. ohne eine vorangehende Ausgangsmaskierung. Die zweite Spalte gibt die Größe des MISR in Anzahl benötigter Flipflops an – dies entspricht zugleich der Größe einer Zwischensignatur in Bit. Die dritte und vierte Spalte geben jeweils die Anzahl an HDF bzw. X-Werten an, die während des Tests in das MISR geschoben werden. Zeile fünf führt dann die Fehlereffizienz des MISR auf, gefolgt von der benötigten Speichergröße für die Zwischensignaturen. Letzere berechnet sich über folgende Gleichung:

$$\text{Speichergröße} = \frac{\#\text{ Zwischensignaturen } \cdot \text{ MISR-Größe in Bit}}{8 \cdot 1024}.$$

Schaltung	$\lvert\mathscr{P}_{Total}\rvert$	Gesamtzahl angelegter Muster		Eindeutige Muster	
		Hybrid	Optimal	Hybrid	Optimal
b14	2610	3798	3740	1327	1322
b15	1462	564	544	185	184
b17	1907	1402	1349	310	310
b18	3363	10 153	10 075	1498	1497
b19	5041	15 033	14 861	3213	3210
b20	3231	4981	4966	1429	1435
b21	3420	5280	5254	1521	1522
b22	1935	7730	7692	1484	1846
bench1	2594	2149	2110	927	926
bench2	148	7468	6869	148	148
bench3	418	915	888	307	307
bench4	1085	4826	4713	647	641
bench5	2392	13 295	13 273	2021	2021

(a) Standardentwurf

Schaltung	$\lvert\mathscr{P}_{Total}\rvert$	Gesamtzahl angelegter Muster	Eindeutige Muster
b14	2598	3782	1317
b15	1450	569	199
b17	1899	1418	319
b18	3322	10 016	1457
b19	5153	14 966	3257
b20	3318	5070	1450
b21	3530	5478	1573
b22	3821	7491	1856

(b) FAST-Entwurf – Nur hybride Auswahl

Tabelle A.8: Ergebnisse der Musterauswahl

Schaltung	Größe MISR	HDF	X	FE_{MISR} / %	Speicher / kB
b14	16	18 964	15 784	98,6	3,5
b15	32	3471	6993	93,8	1,0
b17	64	8463	22 592	83,1	2,4
b18	128	53 613	98 571	86,2	10,6
b19	256	81 366	307 710	60,7	17,6
b20	32	28 100	22 296	96,9	3,6
b21	32	28 656	23 586	96,9	3,8
b22	32	42 989	38 122	96,9	5,9
bench1	128	14 869	96 370	94,7	7,5
bench2	128	158 751	3 866 099	35,7	61,4
bench3	16	12 783	818	99,8	0,2
bench4	32	33 829	187 704	89,3	28,3
bench5	32	58 492	326 085	98,4	45,3

(a) Standardentwurf

Schaltung	Größe MISR	HDF	X	FE_{MISR} / %	Speicher / kB
b14	9	18 895	16 752	92,3	2,7
b15	17	3458	6203	86,0	0,8
b17	53	8498	24 264	81,1	1,9
b18	123	53 379	90 314	81,4	8,6
b19	246	81 207	306 995	58,4	15,4
b20	18	28 706	24 238	92,0	3,3
b21	18	28 724	24 938	91,2	3,4
b22	26	42 353	35 726	94,2	5,3
bench1	128	14 869	96 370	75,4	3,6
bench2	128	158 751	3 866 099	62,2	173,1
bench3	256	12 783	818	99,0	0,8
bench4	256	33 829	187 704	42,3	10,5
bench5	256	58 492	326 085	69,9	18,4

(b) FAST-Entwurf

Tabelle A.9: Ergebnisse X-streichendes MISR

Für die inkrementelle Maskierung enthält Tabelle A.10 die Ergebnisse. Wie zuvor ist die Tabelle in zwei Untertabellen, Tabellen A.10a und A.10b, aufgeteilt, um die Ergebnisse für den Standard- bzw. FAST-Entwurf darzustellen. Die zweite Spalte gibt den Grenzwert ρ_{max} der inkrementellen Maskierung an, ab welchen ein Prüfpfad maskiert werden soll. Die dritte und vierte Spalte zeigen die erreichte X-Reduktionsrate bzw. Fehlereffizienz der Maske. Die letzten beiden Spalten geben die Fehlereffizienz des Gesamtsystems (Maske und MISR) an, gefolgt vom Speicherbedarf des MISR. Für die inkrementelle Maskierung wird kein Speicherbedarf angegeben, da diese praktisch keine Daten speichern muss.

Tabelle A.11 zeigt die Ergebnisse der Grenzwertmaskierung mit $q = 1$ inklusive MISR-Simulation. Tabelle A.11a enthält die Werte des Standardentwurfs, Tabelle A.11b die des FAST-Entwurfs. Spalten zwei und drei zeigen die Parameter der Grenzwertmaskierung (aus Platzgründen wird hier das Verhältnis und kein Prozentwert dargestellt), Spalten vier und fünf die erreichten X-Reduktionsraten sowie Fehlereffizienzen der Maske. Spalte sechs gibt den benötigten Speicheraufwand der Maskenbits an, hierbei handelt es sich um umkomprimierte Datensätze. Die verbleibenden Spalten zeigen, wie bei der inkrementellen Maskierung zuvor, die Ergebnisse des MISR, wenn dieser mit den Ergebnisdaten der Grenzwertmaskierung aufgerufen wird. Spalte sieben gibt wieder die gesamte Fehlereffizienz (Maske und MISR) an, Spalte acht die Größe des Zwischensignaturspeichers.

A.5.1 Maskierung mit erhöhter Maximalfrequenz

Für die erhöhte Maximalfrequenz $f_{max} = 6f_{nom}$ wurde die inkrementelle Maskierung mit anschließender MISR-Simulation durchgeführt, Tabelle A.12 enthält die erzielten Ergebnisse. Tabelle A.12a zeigt die Daten des Standardentwurfs, Tabelle A.12b die entsprechenden Daten des FAST-Entwurfs. In beiden Tabellen gibt Spalte zwei wieder den Grenzwert für die inkrementelle Maskierung an, Spalte drei die X-Reduktionsrate der Maske und Spalte vier die Fehlereffizienz der Maske. Spalte fünf gibt die gesamte Fehlereffizienz von Maske und MISR an, gefolgt von der Größe des Zwischensignaturspeichers in Spalte sechs.

Schaltung	Inkrementelle Maskierung			MISR	
	ρ_{max} / %	R_X	FE_{Maske} / %	FE_{Gesamt} / %	Speicher / kB
b14	5,3	3,2	71,3	70,8	1,2
b15	35,8	2,2	75,6	74,4	0,6
b17	14,5	2,4	66,0	59,8	1,4
b18	1,6	4,6	69,4	62,9	3,5
b19	0,8	6,6	70,2	49,6	4,5
b20	5,0	2,5	68,7	67,7	1,6
b21	4,6	2,6	69,8	68,9	1,6
b22	3,2	2,3	70,8	69,4	2,8
bench1	2,6	17,5	62,8	61,9	1,0
bench2	7,4	7,3	70,1	29,2	12,7
bench3	0,5	1,6	94,5	94,2	0,1
bench4	1,8	3,4	70,6	60,7	8,6
bench5	1,7	4,6	70,4	69,8	11,2

(a) Standardentwurf

Schaltung	Inkrementelle Maskierung			MISR	
	ρ_{max} / %	R_X	FE_{Maske} / %	FE_{Gesamt} / %	Speicher / kB
b14	5,9	5,1	75,1	73,4	0,8
b15	26,9	2,4	82,2	79,5	0,5
b17	16,7	2,3	72,1	63,0	1,2
b18	1,4	4,8	71,5	63,9	3,1
b19	1,0	6,1	69,8	49,2	5,3
b20	2,9	6,4	71,1	70,2	0,9
b21	3,8	3,0	70,0	68,0	1,5
b22	3,3	2,7	84,2	82,9	2,3
bench1	4,1	27,5	63,1	59,1	0,9
bench2	7,1	8,1	69,9	28,2	8,9
bench3	1,9	1,6	95,0	94,5	0,8
bench4	6,2	3,2	70,1	46,9	6,1
bench5	7,3	2,0	70,3	55,7	9,2

(b) FAST-Entwurf

Tabelle A.10: Ergebnisse der inkrementellen Maskierung mit MISR

Schaltung	Maske					MISR	
	ρ_{max}	$\gamma_{Essentiell}$	R_X	FE$_{Maske}$ / %	Speicher / kB	FE$_{Gesamt}$ / %	Speicher / kB
b14	0,1	0,3	1,2	100,0	4,2	98,8	2,9
b15	0,1	0,2	2,0	100,0	1,2	97,7	0,6
b17	0,3	0,0	2,7	92,7	9,1	86,5	1,3
b18	0,1	0,1	2,2	100,0	152,4	90,8	6,3
b19	0,1	0,1	1,9	100,0	451,4	74,3	8,0
b20	0,1	0,2	1,3	100,0	10,9	98,0	2,9
b21	0,1	0,2	1,3	99,7	11,6	97,8	3,0
b22	0,1	0,1	1,4	99,8	24,5	97,8	4,3
bench1	0,1	0,2	2,0	100,0	25,4	95,9	4,8
bench2	0,3	0,1	3,4	98,9	59,3	55,5	42,7
bench3	0,1	0,0	1,0	100,0	0,9	99,8	0,2
bench4	0,1	0,2	1,0	100,0	10,6	89,3	27,5
bench5	0,1	0,4	1,3	100,0	29,2	99,0	38,0

(a) Standardentwurf

Schaltung	Maske					MISR	
	ρ_{max}	$\gamma_{Essentiell}$	R_X	FE$_{Maske}$ / %	Speicher / kB	FE$_{Gesamt}$	Speicher / kB
b14	0,1	0,2	3,7	100,0	4,2	98,9	1,0
b15	0,1	0,2	2,7	100,0	1,2	98,6	0,4
b17	0,2	0,0	4,1	91,1	9,2	83,4	0,9
b18	0,1	0,1	1,5	100,0	150,4	85,7	6,9
b19	0,1	0,1	1,4	100,0	449,4	65,7	12,7
b20	0,1	0,2	2,2	100,0	11,1	95,4	1,8
b21	0,1	0,2	2,1	99,7	12,0	95,7	2,0
b22	0,1	0,1	1,2	99,5	23,8	94,8	4,7
bench1	0,1	0,1	7,6	100,0	25,4	92,3	1,5
bench2	0,1	0,1	4,6	99,3	109,4	53,3	26,1
bench3	0,1	0,1	1,3	100,0	17,4	99,1	0,8
bench4	0,1	0,1	5,0	100,0	101,3	81,5	4,8
bench5	0,1	0,2	3,6	100,0	373,3	92,9	9,4

(b) FAST-Entwurf

Tabelle A.11: Ergebnisse der Grenzwertmaskierung ($q = 1$) mit MISR

Schaltung	Inkrementelle Maskierung			MISR	
	ρ_{max} / %	R_X	FE_{Maske} / %	FE_{Gesamt} / %	Speicher / kB
b14	11,0	2,6	70,3	68,4	8,2
b15	34,8	2,7	72,2	69,1	1,1
b17	13,5	2,3	72,4	66,0	4,5
b18	6,3	1,8	89,1	74,0	21,6
b19	1,6	2,4	70,9	52,2	21,8
b20	7,9	2,1	68,9	65,7	11,7
b21	9,4	2,0	69,2	66,2	11,8
b22	5,2	2,3	70,9	67,9	19,2
bench1	4,7	8,6	69,8	67,2	10,3
bench2	7,6	8,5	70,1	26,9	11,5
bench3	2,4	8,3	70,3	70,1	23,4
bench4	3,3	4,1	72,1	53,0	53,0
bench5	2,4	9,3	71,7	70,8	35,0

(a) Standardentwurf

Schaltung	Inkrementelle Maskierung			MISR	
	ρ_{max} / %	R_X	FE_{Maske} / %	FE_{Gesamt} / %	Speicher / kB
b14	8,4	3,2	74,6	71,5	6,2
b15	25,2	2,3	73,1	67,9	1,2
b17	14,2	2,3	73,3	63,1	3,8
b18	3,9	1,8	89,5	74,4	18,6
b19	1,4	1,9	70,5	49,4	31,4
b20	4,8	2,7	70,2	67,0	8,5
b21	5,5	2,3	70,1	66,5	10,2
b22	3,5	2,8	72,2	69,3	15,6
bench1	8,1	5,1	70,1	54,0	9,5
bench2	7,2	9,2	70,0	26,8	8,1
bench3	1,6	12,6	71,1	66,1	8,3
bench4	6,4	3,4	72,1	33,4	24,4
bench5	6,1	6,1	71,1	55,8	22,1

(b) FAST-Entwurf

Tabelle A.12: Ergebnisse der inkrementellen Maskierung mit MISR unter $6f_{nom}$

A.6 Laufzeitanalyse

Für alle in dieser Arbeit implementierten Verfahren wurden die Laufzeiten gemessen. Die Programme wurden auf Rechnern des Paderborn Center for Parallel Computing (PC^2) durchgeführt. Zur Verfügung standen Intel Xeon E5-2670 Rechner mit 2,6 GHz Taktfrequenz, 16 Rechenkernen und 64 GB Hauptspeicher. Für die GPU-basierten Simulationen wurden nVIDIA GeForce GTX1080 Ti Grafikkarten mit jeweils 11 GB Hauptspeicher eingesetzt. Alle selbst entwickelten Programme sind in C++ geschrieben, mit Ausnahme der GPU-basierten Simulation (Algorithmus 4.2), für diese musste eine vorgegebene Java-Umgebung genutzt werden, welche eine Anbindung an die GPU-Simulationsroutinen bereitstellt. Tabelle A.13 zeigt die Laufzeiten der verschiedenen Syntheseschritte sowohl für den Standard- als auch für den FAST-Entwurf, jeweils in Sekunden, gemessen als wahre Rechenzeit (CPU-Zeit).

Schaltung	Synthese	Standardentwurf	FAST-Entwurf		
		DFT	Histogramme	Randbed.	DFT
b14	131,4	16,8	1,0	0,0	17,1
b15	27,9	5,5	1,0	0,0	5,6
b17	75,0	14,0	3,0	0,4	14,1
b18	241,4	46,0	12,0	3,7	45,1
b19	504,6	125,9	24,0	24,5	134,4
b20	64,8	18,5	4,0	0,0	18,1
b21	69,9	16,8	3,0	0,0	16,4
b22	107,2	23,7	5,0	0,0	24,5
bench1			10,4	0,6	
bench2			14,5	1,2	
bench3			36,3	5,6	
bench4			23,4	6,2	
bench5			25,5	10,4	

Tabelle A.13: Laufzeiten der Standard- und FAST-Synthese (Werte sind in s angegeben)

Spalte 2 zeigt die benötigte Zeit für den ersten Syntheseschritt, um die Schaltung für den prüfgerechten Entwurf vorzubereiten. Da diese Zeit für Standard- und FAST-Entwurf praktisch gleich ist (bis hier erfolgt keine Abweichung beim

Syntheseablauf), zeigt Tabelle A.13 lediglich die Werte für den Standardentwurf. Spalte 3 zeigt dann für den Standardentwurf die benötigte Zeit zum Einfügen der Prüfpfade und Fertigstellen der finalen Netzliste. Spalten 4 bis 6 zeigen die Zeiten für den FAST-Entwurf, aufgeteilt in die Berechnung der Verzögerungshistogramme, die Graph-basierte Prüfzellengruppierung sowie das Einfügen der Prüfpfade. Tabelle A.14 stellt die Laufzeiten der kommerziellen Testmustererzeugung für $\mathscr{P}_{Initial}$ und $\mathscr{P}_{Extra}$ vor, jeweils für den Standard- und FAST-Entwurf.

Schaltung	Standardentwurf		FAST-Entwurf	
	$\mathscr{P}_{Initial}$	$\mathscr{P}_{Extra}$	$\mathscr{P}_{Initial}$	$\mathscr{P}_{Extra}$
b14	98,3	134,7	94,5	160,7
b15	139,1	107,2	127,0	107,2
b17	353,7	277,6	360,0	220,7
b18	2413,6	1704,8	2143,8	2151,3
b19	4382,3	4271,4	4380,8	4216,5
b20	46,4	125,6	53,1	158,7
b21	45,6	124,6	47,8	127,3
b22	101,1	195,7	97,8	201,9
bench1	5,3	1,0		
bench2	1,7	0,1		
bench3	10,8	0,5		
bench4	10,2	0,6		
bench5	27,0	5,8		

Tabelle A.14: Laufzeiten der Testmustererzeugung (Werte sind in s angegeben)

A.6.1 Simulation, Frequenz- und Musterauswahl

Tabelle A.15 zeigt die benötigten Laufzeiten für FAST-Simulation, unterteilt in Standard- und FAST-Entwurf. Die Werte sind auch hier in Sekunden angegeben, dabei handelt es sich aber nicht um CPU-Zeiten, da diese nicht mit dem Java-Programm ausgegeben werden konnten. Für den jeweiligen Entwurf wird die Laufzeit der Simulation für $\mathscr{P}_{Initial}$ und $\mathscr{P}_{Extra}$ angegeben, so wie es in Kapitel 4 beschrieben wird.

Schaltung	Standardentwurf		FAST-Entwurf	
	$\mathscr{P}_{Initial}$	$\mathscr{P}_{Extra}$	$\mathscr{P}_{Initial}$	$\mathscr{P}_{Extra}$
b14	4933,1	4562,9	4279,3	4441,5
b15	740,0	345,8	1012,6	373,9
b17	6155,2	3136,7	5436,4	3178,4
b18	30 664,3	13 583,3	13 734,1	13 894,2
b19	308 096,7	121 998,1	270 379,4	91 150,2
b20	3065,2	3620,1	3507,2	4204,2
b21	3239,3	3922,6	6940,0	8982,8
b22	16 152,0	19 837,0	10 017,0	13 604,3
bench1	855,4	1196,8		
bench2	766,1	452,1		
bench3	12 752,5	6834,3		
bench4	17 685,7	12 802,6		
bench5	36 375,8	2701,6		

Tabelle A.15: Laufzeiten von FAST-Simulation (Werte sind in s angegeben)

Tabelle A.16 zeigt die Laufzeiten der Frequenz- und Musterauswahl für die verschiedenen Algorithmen (MHS-LUB, Hybrid, Optimal) und Standard- bzw. FAST-Entwurf – für letzteren wurde allerdings nur die hybride Frequenzauswahl durchgeführt. Tabelle A.16a enthält die Laufzeiten der Frequenzauswahl, Tabelle A.16b die entsprechenden Laufzeiten der Musterauswahl.

A.6.2 Maskierung und MISR

Für die Maskierung listet Tabelle A.17 die Laufzeiten des MISR und der Maskierungsverfahren auf, jeweils für den Standard- und FAST-Entwurf. Tabelle A.17a zeigt dabei die Laufzeiten des reinen X-streichenden MISRs (ohne Maskierung) und der inkrementellen Maskierung. Für beide Entwürfe wird zunächst die Laufzeit des reinen MISR angegeben, dann folgt die Laufzeit zur Berechnung der Masken, die Spalte „Komb.“ gibt schließlich die Laufzeit des MISR an, wenn eine Maskierung der Eingabedaten erfolgt ist. Die Laufzeiten der Grenzwertmaskierung sind in Tabelle A.17b angegben. Hier wird lediglich die Laufzeit der Maskierung und des MISR mit maskierten Eingabedaten angegeben.

Schaltung	Standardentwurf			FAST-Entwurf
	MHS-LUB	Hybrid	Optimal	Hybrid
b14	2,1	237,8	1864,7	485,0
b15	0,1	40,9	220,4	26,7
b17	0,5	269,1	8445,2	253 528,9
b18	21,4	1128,5	4911,4	1586,0
b19	46,5	1650,5	8410,1	2340,4
b20	6,0	1974,2	4146,4	925,1
b21	6,0	2457,9	5338,7	1788,6
b22	11,1	1218,8	9030,4	1369,1
bench1	5,9	100,2	86,4	
bench2	190,3	138,4	2304,6	
bench3	24,8	31,3	25,6	
bench4	13,5	511,9	765,9	
bench5	49,2	1151,5	2786,1	

(a) Frequenzauswahl

Schaltung	Standardentwurf		FAST-Entwurf
	Hybrid	Optimal	Hybrid
b14	185,1	177,4	162,1
b15	25,7	22,2	24,6
b17	168,4	153,1	153,6
b18	1493,5	1436,7	1630,5
b19	10 897,9	10 554,2	11 805,2
b20	512,5	503,5	516,0
b21	608,3	597,1	650,4
b22	1173,0	1152,9	1215,6
bench1	1768,3	1705,5	
bench2	9687,9	8496,7	
bench3	1037,3	1027,8	
bench4	2124,8	2005,2	
bench5	8078,3	7915,3	

(b) Testmusterauswahl

Tabelle A.16: Laufzeiten der Frequenz- und Testmusterauswahl (alle Werte sind in s angegeben)

Schaltung	Standardentwurf			FAST-Entwurf		
	MISR	Maske	Komb.	MISR	Maske	Komb.
b14	22,9	12,0	23,0	22,2	12,9	23,1
b15	2,9	1,7	3,0	3,1	1,8	3,0
b17	17,7	7,4	18,8	17,1	7,2	19,0
b18	403,6	80,0	509,1	418,1	80,4	537,6
b19	1548,5	177,6	2173,9	1607,4	169,4	2343,7
b20	48,3	25,9	49,0	48,2	24,3	55,0
b21	47,9	26,7	54,9	51,5	28,0	56,1
b22	88,7	48,3	96,7	85,7	45,4	100,3
bench1	675,5	10,3	753,0	52,1	11,0	59,6
bench2	451,4	55,7	332,7	254,2	61,1	291,9
bench3	44,0	11,3	50,5	81,1	10,7	75,4
bench4	463,0	44,6	501,6	256,7	38,6	382,4
bench5	1405,6	126,0	1509,1	1003,7	126,5	1447,1

(a) MISR und inkrementelle Maskierung

Schaltung	Standardentwurf		FAST-Entwurf	
	Maske	MISR + Maske	Masken	MISR + Maske
b14	12,0	23,8	11,6	25,7
b15	1,5	3,2	1,4	3,2
b17	6,6	20,8	6,8	23,3
b18	88,2	497,9	77,6	472,2
b19	187,5	2091,5	216,4	1507,7
b20	24,9	49,5	25,5	55,2
b21	25,8	51,5	26,7	56,3
b22	47,0	92,8	46,3	87,6
bench1	13,9	717,0	14,7	65,2
bench2	62,8	428,6	89,6	333,4
bench3	10,1	45,2	12,2	75,8
bench4	51,2	473,9	55,5	453,1
bench5	132,2	1407,6	148,1	1540,2

(b) Grenzwertmaskierung ($q = 1$)

Tabelle A.17: Laufzeiten Maskierung und MISR (Alle Werte sind in s angegeben)

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

Abkürzungsverzeichnis

ATE Externes Testequipment (engl. Automatic Test Equipment)

ATPG Automatisierte Testmustererzeugung (engl. Automatic Test Pattern Generation)

BIST Eingebauter Selbsttest (engl. Built-In Self-Test)

CMOS Complementary Metal Oxide Semiconductor

DFT Prüfgerechter Entwurf (engl. Design for Test)

EDT Eingebauter deterministischer Test (engl. Embedded Deterministic Test)

ELF Frühzeitiger Systemausfall (engl. Early Life Failure)

ES Erweiterte Prüfpfadarchitektur (engl. Enhanced Scan)

FAST Hochgeschwindigkeitstest (engl. Faster-than-at-Speed Test)

GPU Grafikprozessor (engl. Graphics Processing Unit)

HDF Versteckter Verzögerungsfehler (engl. Hidden Delay Fault)

LFSR Linear rückgekoppeltes Schieberegister (engl. Linear Feedback Shift Register)

LOC Launch-on Capture

LOS Launch-on Shift

MISR Signaturregister mit mehreren Eingängen (engl. Multiple-Input Signature Register)

PLL Phasenregelschleife (engl. Phase-locked Loop)

PPA Pseudo-primärer Ausgang

PPE Pseudo-primärer Eingang

SDD Kleiner Verzögerungsdefekt (engl. Small Delay Defect)

SDF Kleiner Verzögerungsfehler (engl. Small Delay Fault)

SDQL Statistical Delay Quality Level

SDQM Statistical Delay Quality Model

STUMPS Self-Test Using MISR and Parallel SRSG

Literaturverzeichnis

[1] V. D. Agrawal, C. R. Kime und K. K. Saluja. „A Tutorial on Built-In Self-Test. Part 1: Principles“. In: *IEEE Design & Test of Computers* 10.1 (März 1993), S. 73–82.

[2] V. D. Agrawal, C. R. Kime und K. K. Saluja. „A Tutorial on Built-In Self-Test. Part 2: Applications“. In: *IEEE Design & Test of Computers* 10.2 (Juni 1993), S. 69–77.

[3] N. Ahmed und M. Tehranipoor. „A Novel Faster-Than-at-Speed Transition-Delay Test Method Considering IR-Drop Effects“. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28.10 (Okt. 2009), S. 1573–1582.

[4] N. Ahmed, M. Tehranipoor und V. Jayaram. „A Novel Framework for Faster-than-at-Speed Delay Test Considering IR-drop Effects“. In: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. San Jose, USA, Nov. 2006.

[5] N. Ahmed, M. Tehranipoor und V. Jayaram. „Timing-Based Delay Test for Screening Small Delay Defects“. In: *43rd ACM/IEEE Design Automation Conference (DAC)*. San Francisco, USA, Juli 2006, S. 320–325.

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

[7] M. Amodeo und B. Cory. „Defining Faster-than-at-speed Delay Tests“. In: *Cadence Nanometer Test Quarterly eNewsletter* 2.2 (Mai 2005).

[8] P. H. Bardell. „Analysis of Cellular Automata Used as Pseudorandom Pattern Generators“. In: *1990 IEEE International Test Conference (ITC)*. Washington, USA, Sep. 1990, S. 762–768.

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

[10] N. Benowitz, D. F. Calhoun, G. E. Alderson, J. E. Bauer und C. T. Joeckel. „An Advanced Fault Isolation System for Digital Logic“. In: *IEEE Transactions on Computers* C-24.5 (Mai 1975), S. 489–497.

[11] M. L. Bushnell und V. D. Agrawal. *Essentials of Electronic Testing for Digital, Memory & Mixed-Signal VLSI Circuits*. New York [u. a.]: Kluwer Academic Publishers, 2002. ISBN: 0-792-37991-8.

[12] K. Chakrabarty und J. P. Hayes. „Zero-Aliasing Space Compaction of Test Responses Using Multiple Parity Signatures“. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 6.2 (Juni 1998), S. 309–313.

[13] S. Chakravarty, N. Devta-Prasanna, A. Gunda, J. Ma, F. Yang, H. Guo, R. Lai und D. Li. „Silicon Evaluation of Faster than at-speed Transition Delay Tests“. In: *2012 IEEE 30th VLSI Test Symposium (VTS)*. Hyatt Maui, USA, Apr. 2012, S. 80–85.

[14] A. Chandra, Y. Kanzawa und R. Kapur. „Proactive Management of X's in Scan Chains for Compression“. In: *10th International Symposium on Quality Electronic Design*. San Jose, USA, März 2009, S. 260–265.

[15] V. Chickermane, B. Foutz und B. Keller. „Channel Masking Synthesis for Efficient On-Chip Test Compression“. In: *2004 IEEE International Test Conference (ITC)*. Charlotte, USA, Okt. 2004, S. 452–461.

[16] F. Corno, M. Sonza Reorda und G. Squillero. „RT-level ITC'99 Benchmarks and First ATPG Results“. In: *IEEE Design & Test of Computers* 17.3 (Juli 2000), S. 44–53. URL: `https://github.com/squillero/itc99-poli` (besucht am 09. 03. 2020).

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

[18] W. R. Daasch und R. Madge. „Variance Reduction and Outliers: Statistical Analysis of Semiconductor Test Data“. In: *2005 IEEE International Test Conference (ITC)*. Austin, USA, Nov. 2005.

[19] J. Edmonds. „Paths, Trees and Flowers“. In: *Canadian Journal of Mathematics* 17 (Feb. 1965), S. 449–467.

[20] S. Eggersglüß und R. Drechsler. „As-Robust-As-Possible Test Generation in the Presence of Small Delay Defects using Pseudo-Boolean Optimization“. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Grenoble, Frankreich, März 2011.

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

[22] J. Feil. *Der Blossom Algorithmus von Edmonds*. Lehrstuhl M9, Technische Universität München. URL: `https://www-m9.ma.tum.de/graph-algorithms/matchings-blossom-algorithm/index_de.html` (besucht am 09. 03. 2020).

[23] M. R. Garey und D. S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. New York: Freeman, 1979. ISBN: 0-7167-1044-7.

[24] R. Garg, R. Putman und N. A. Touba. „Increasing Output Compaction in Presence of Unknowns Using an X-Canceling MISR with Deterministic Observation“. In: *26th IEEE VLSI Test Symposium (VTS)*. San Diego, USA, Apr. 2008, S. 35–42.

[25] S. K. Goel, N. Devta-Prasanna und R. P. Turakhia. „Effective and Efficient Test Pattern Generation for Small Delay Defect“. In: *27th IEEE VLSI Test Symposium (VTS)*. Santa Cruz, USA, Mai 2009, S. 111–116.

[26] A.-W. Hakmi, H.-J. Wunderlich, C. G. Zoellin, A. Glowatz, F. Hapke, J. Schloeffel und L. Souef. „Programmable Deterministic Built-In Self-Test“. In: *2007 IEEE International Test Conference (ITC)*. Santa Clara, USA, Okt. 2007.

[27] S. Hellebrand, H.-G. Liang und H.-J. Wunderlich. „A Mixed Mode BIST Scheme Based on Reseeding of Folding Counters“. In: *2000 IEEE International Test Conference (ITC)*. Atlantic City, USA, Okt. 2000, S. 778–784.

[28] S. Hellebrand, J. Rajski, S. Tarnick, S. Venkataraman und B. Courtois. „Built-In Test for Circuits with Scan Based on Reseeding of Multiple-Polynomial Linear Feedback Shift Registers“. In: *IEEE Transactions on Computers* 44.2 (Feb. 1995), S. 223–233.

[29] S. Hellebrand, B. Reeb, S. Tarnick und H.-J. Wunderlich. „Pattern Generation for a Deterministic BIST Scheme“. In: *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. San Jose, USA, Nov. 1995, S. 88–94.

[30] G. Hetherington, T. Fryars, N. Tamarapalli, M. Kassab, A. Hassan und J. Rajski. „Logic BIST for Large Industrial Designs: Real Issues and Case Studies“. In: *1994 IEEE International Test Conference (ITC)*. Atlantic City, USA, Sep. 1999, S. 358–367.

[31] V. Iyengar, T. Yokota, K. Yamada, T. Anemikos, B. Bassett, M. Degregorio, R. Farmer, G. Grise, M. Johnson, D. Milton, M. Taylor und F. Woytowich. „At-Speed Structural Test For High-Performance ASICs“. In: *2006 IEEE International Test Conference (ITC)*. Santa Clara, USA, Okt. 2006.

[32] J. Jiang, M. Sauer, A. Czutro, B. Becker und I. Polian. „On the Optimality of K Longest Path Generation Algorithm Under Memory Constraints“. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Dresden, Deutschland, März 2012.

[33] S. Jin, Y. Han, H. Li und X. Li. „Unified Capture Scheme for Small Delay Defect Detection and Aging Prediction“. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 21.5 (Mai 2013), S. 821–833.

[34] W. B. Jone, Y. P. Ho und S. R. Das. „Delay Fault Coverage Enhancement Using Variable Observation Times“. In: *Journal of Electronic Testing: Theory and Applications* 11.2 (Okt. 1997), S. 131–146.

[35] S. Kaeriyama, M. Kajita und M. Mizuno. „A 1-to-2 GHz 4-Phase On-Chip Clock Generator with Timing-Margin Test Capability“. In: *Digest of Technical Papers of the 2007 IEEE International Solid-State Circuits Conference (ISSCC)*. San Francisco, USA, Feb. 2007, S. 174–175, 594.

[36] S. Kajihara, S. Morishima, M. Yamamoto, W. Xiaoqing, M. Fukunaga, K. Hatayama und T. Aikyo. „Estimation of Delay Test Quality and Its Application to Test Generation“. In: *IEEE/ACM International Conference*

on Computer-Aided Design (ICCAD). San Jose, USA, Nov. 2007, S. 413–417.

[37] R. M. Karp. „Reducibility Among Combinatorial Problems“. In: *Complexity of Computer Computations*. Hrsg. von R. E. Miller und J. W. Thatcher. New York: Plenum Press, 1972, S. 85–103. ISBN: 978-1-4684-2003-6.

[38] K. S. Kim, S. Mitra und P. G. Ryan. „Delay Defect Characteristics and Testing Strategies“. In: *IEEE Design & Test of Computers* 20.5 (Sep. 2003), S. 8–16.

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

[40] B. Koenemann, C. Barnhart, B. Keller, T. Snethen, O. Farnsworth und D. Wheather. „A SmartBIST Variant with Guaranteed Encoding“. In: *10^{th} Asian Test Symposium (ATS)*. Kyoto, Japan, Nov. 2001, S. 325–330.

[41] J. Konc und Janežič. „An improved branch and bound algorithm for the maximum clique problem“. In: *MATCH Communications in Mathematical and in Computer Chemistry* 58.3 (Jan. 2007), S. 569–590. ISSN: 340-6253.

[42] I. Koren und C. M. Krishna. „Hardware Fault Tolerance“. In: *Fault-Tolerant Systems*. Amsterdam [u. a.]: Morgan Kaufmann, 2007, S. 11–54. ISBN: 978-0-12-088525-1.

[43] B. Kruseman, A. K. Majhi, G. Gronthoud und S. Eichenberger. „On Hazard-free Patterns for Fine-delay Fault Testing“. In: *2004 IEEE International Test Conference (ITC)*. Charlotte, USA, Okt. 2004, S. 213–222.

[44] K.-J. Lee, W.-C. Lien und T.-Y. Hsieh. „Test Response Compaction via Output Bit Selection“. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30.10 (Okt. 2011), S. 1534–1544.

[45] Y. Li, S. Makar und S. Mitra. „CASP: Concurrent Autonomous Chip Self-Test Using Stored Test Patterns“. In: *Design, Automation & Test in Europe (DATE)*. München, Deutschland, März 2008.

[46] W.-C. Lien, K.-J. Lee, K. Chakrabarty und T.-Y. Hsieh. „Output-Bit Selection with X-Avoidance using Multiple Counters for Test-Response Compaction“. In: *19th IEEE European Test Symposium (ETS)*. Paderborn, Deutschland, Mai 2014.

[47] C. J. Lin und S. M. Reddy. „On Delay Fault Testing in Logic Circuits“. In: *IEEE Transactions on Computer-Aided Design* 6.5 (Sep. 1987), S. 694–703.

[48] X. Lin, R. Press, J. Rajski, P. Reuter, T. Rinderknecht, B. Swanson und N. Tamarapalli. „High-Frequency, At-Speed Scan Testing“. In: *IEEE Design & Test of Computers* 20.5 (Sep. 2003), S. 17–25.

[49] X. Lin, K.-H. Tsai, C. Wang, M. Kassab, J. Rajski, T. Kobayashi, R. Klingenberg, Y. Sato, S. Hamada und T. Aikyo. „Timing-Aware ATPG for High Quality At-speed Testing of Small Delay Defects“. In: *15th IEEE Asian Test Symposium (ATS)*. Fukuoka, Japan, Nov. 2006.

[50] Y. Liu, N. Mukherjee, J. Rajski, S. M. Reddy und J. Tyszer. „Deterministic Stellar BIST for In-System Automotive Test“. In: *2018 IEEE International Test Conference (ITC)*. Phoenix, USA, Nov. 2018.

[51] Y. Liu und Q. Xu. „On Modeling Faults in FinFET Logic Circuits“. In: *2012 IEEE International Test Conference (ITC)*. Anaheim, USA, Nov. 2012.

[52] R. Madge, B. R. Benware und W. R. Daasch. „Obtaining High Defect Coverage for Frequency-Dependent Defects in Complex ASICs“. In: *IEEE Design & Test of Computers* 20.5 (Sep. 2003), S. 46–53.

[53] Y. Maeda, J. Matsushima und R. Press. „Automotive IC on-line test techniques and the application of deterministic ATPG-based runtime test“. In: *26th IEEE Asian Test Symposium (ATS)*. Taipei, Taiwan, Nov. 2017, S. 232–236.

[54] M. Mantel und D. Ney. *CPU-Roadmap 2019-2020: Künftige AMD- und Intel-Prozessoren*. Sep. 2019. URL: `http://www.pcgameshardware.de/CPU-CPU-154106/Specials/Roadmap-CPUs-Prozessoren-Liste-AMD-Intel-1130335/` (besucht am 02. 11. 2019).

[55] W.-W. Mao und M. D. Ciletti. „A Variable Observation Time Method for Testing Delay Faults“. In: *27th ACM/IEEE Design Automation Conference (DAC)*. Orlando, USA, Juni 1990, S. 728–731.

[56] E. J. Marinissen, A. Singh, D. Glotter, M. Esposito, J. M. Carulli, A. Nahar, K. M. Butler, D. Apello und C. Portelli. „Adapting to Adaptive Testing“. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Dresden, März 2010.

[57] R. Mattiuzzo, D. Appello und C. Allsup. „Small-delay-defect Testing“. In: *Test & Measurement World* (Juni 2009), S. 37–41.

[58] E. J. McCluskey. „Techniques for Test Output Response Analysis“. In: *IEEE International Symposium on Circuits and Systems*. Singapur, Singapur, Juni 1991, S. 1869–1872.

[59] T. L. McLaurin und F. Frederick. „The Testability Features of the MCF5407 Containing the 4th Generation ColdFire® Microprocessor Core“. In: *2000 IEEE International Test Conference (ITC)*. Atlantic City, USA, Okt. 2000, S. 151–159.

[60] S. Mitra und K. S. Kim. „X-Compact: An Efficient Response Compaction Technique for Test Cost Reduction“. In: *2002 IEEE International Test Conference (ITC)*. Baltimore, USA, Okt. 2002, S. 311–320.

[61] S. Mitra, S. S. Lumetta und M. Mitzenmacher. „X-Tolerant Signature Analysis“. In: *2004 IEEE International Test Conference (ITC)*. Charlotte, USA, Okt. 2004, S. 432–441.

[62] A. S. Mutschler. „BiST Grows Up In Automotive“. In: *Semiconductor Engineering* (Juni 2019). URL: `https://semiengineering.com/bist-grows-up-in-automotive/` (besucht am 09. 03. 2020).

[63] M. Naruse, I. Pomeranz, S. M. Reddy und S. Kundu. „On-chip Compression of Output Responses with Unknown Values Using LFSR Reseeding“. In: *2003 IEEE International Test Conference (ITC)*. Charlotte, USA, Okt. 2003, S. 1060–1068.

[64] D. J. Neebel und C. R. Kime. „Cellular Automata for Weighted Random Pattern Generation“. In: *IEEE Transactions on Computers* 46.11 (Nov. 1997), S. 1219–1229.

[65] P. Nigh und A. Gattiker. „Test Method Evaluation Experiments & Data“. In: *2000 IEEE International Test Conference (ITC)*. Atlantic City, USA, Okt. 2000, S. 454–463.

[66] K. Noguchi, K. Nose, T. Ono und M. Mizuno. „A small-delay defect detection technique for dependable LSIs“. In: *2008 IEEE Symposium on VLSI Circuits*. Honolulu, USA, Juni 2008, S. 64–65.

[67] O. Novak und J. Nosek. „On Using Deterministic Test Sets in BIST“. In: *6th IEEE International On-Line Testing Workshop*. Palma de Mallorca, Spanien, Juli 2000, S. 127–132.

[68] R. S. Oliveira, J. Semião, I. C. Teixeira, M. B. Santos und J. P. Teixeira. „On-line BIST for Performance Failure Prediction under Aging Effects in Automotive Safety-Critical Applications“. In: *12th Latin American Test Workshop (LATW)*. Porto de Galinhas, Brasilien, März 2011.

[69] C. Papameletis und V. Chickermane. *Unified Compression and LBIST in a Physically Aware Environment*. Technischer Bericht. Cadence, 2019. URL: `https://www.cadence.com` (besucht am 09. 03. 2020).

[70] S. Pei, Y. Geng, H. Li, J. Liu und S. Jin. „Enhanced LCCG: A Novel Test Clock Generation Scheme for Faster-than-at-Speed Delay Testing“. In: *20th Asia and South Pacific Design Automation Conference (ASP-DAC)*. Chiba, Japan, Jan. 2015, S. 514–519.

[71] S. Pei, H. Li, S. Jin, J. Liu und X. Li. „An on-chip frequency programmable test clock generation and application method for small delay defect detection“. In: *Integration, the VLSI Journal* 49 (März 2015), S. 87–97.

[72] S. Pei, H. Li und X. Li. „An On-Chip Clock Generation Scheme for Faster-than-at-Speed Delay Testing“. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Dresden, Deutschland, März 2010.

[73] I. Pomeranz und S. M. Reddy. „3-Weight Pseudo-Random Test Generation Based on a Deterministic Test Set for Combinational and Sequential Circuits“. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 12.7 (Juli 1993), S. 1050–1058.

[74] R. Press und J. Boyer. „Easily Implement PLL Clock Switching for At-Speed Test“. In: *Chip Design* (Feb. 2006).

[75] PwC. *Spotlight on Automotive*. PwC Semiconductor Report. Sep. 2013. URL: `https://www.pwc.de` (besucht am 09. 03. 2020).

[76] X. Qian, C. Han und A. D. Singh. „Detection of Gate-Oxide Defects with Timing Tests at Reduced Power Supply“. In: *2012 IEEE 30th VLSI Test Symposium (VTS)*. Hyatt Maui, USA, Apr. 2012, S. 120–126.

[77] W. Qiu und D. M. H. Walker. „An Efficient Algorithm for Finding the K Longest Testable Paths Through Each Gate in a Combinational Circuit“. In: *2003 IEEE International Test Conference (ITC)*. Charlotte, USA, Sep. 2003, S. 592–601.

[78] W. Qiu, D. M. H. Walker, N. Simpson, D. Reddy und A. Moore. „Comparison of Delay Tests on Silicon“. In: *2006 IEEE International Test Conference (ITC)*. Santa Clara, USA, Okt. 2006.

[79] W. Qiu, J. Wang, D. M. H. Walker, D. Reddy, X. Lu, Z. Li, W. Shi und H. Balachandran. „K Longest Paths Per Gate (KLPG) Test Generation for Scan-Based Sequential Circuits“. In: *2004 IEEE International Test Conference (ITC)*. Charlotte, USA, Okt. 2004, S. 223–231.

[80] T. Rabenalt, M. Richter, F. Poehl und M. Goessel. „Highly Efficient Test Response Compaction Using a Hierarchical X-Masking Technique“. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31.6 (Juni 2012), S. 950–957.

[81] J. Rajski, J. Tyszer, M. Kassab, N. Mukherjee, R. Thompson, K.-H. Tsai, A. Hertwig, N. Tamarapalli, G. Mrugalski, G. Eide und J. Qian. „Embedded Deterministic Test for Low Cost Manufacturing Test“. In: *2002 IEEE International Test Conference (ITC)*. Baltimore, USA, 2002, S. 301–310.

[82] J. Rajski, J. Tyszer, G. Mrugalski, W.-T. Cheng, N. Mukherjee und M. Kassab. „X-Press Compactor for 1000x Reduction of Test Data“. In: *2006 IEEE International Test Conference (ITC)*. Santa Clara, USA, Okt. 2006.

[83] J. Rajski, J. Tyszer, C. Wang und S. M. Reddy. „Convolutional Compaction of Test Responses“. In: *Proceedings of 2003 IEEE International Test Conference (ITC)*. Charlotte, USA, Okt. 2003, S. 745–754.

[84] B. Reeb und H.-J. Wunderlich. „Deterministic Pattern Generation for Weighted Random Pattern Testing“. In: *Proceedings of the ED&TC European Design and Test Conference*. Paris, France, März 1996.

[85] *SAED_EDK32/28_CORE Digital Standard Cell Library Databook*. Synopsys Armenia Educational Department. 2012.

[86] Y. Sato, S. Hamada, T. Maeda, A. Takatori und S. Kajihara. „Evaluation of the Statistical Delay Quality Model“. In: *10th Asia and South Pacific Design Automation Conference (ASP-DAC)*. Shanghai, China, Jan. 2005, S. 305–310.

[87] Y. Sato, S. Hamada, T. Maeda, A. Takatori, Y. Nozuyama und S. Kajihara. „Invisible Delay Quality - SDQM Model Lights Up What Could Not Be Seen“. In: *2005 IEEE International Test Conference (ITC)*. Austin, USA, Nov. 2005.

[88] Y. Sato, S. Kajihara, T. Yoneda, K. Hatayama, M. Inoue, Y. Miura, S. Ohtake, T. Hasegawa, M. Sato und K. Shimamura. „DART: Dependable VLSI Test Architecture and Its Implementation“. In: *2012 IEEE International Test Conference (ITC)*. Anaheim, USA, Nov. 2012.

[89] M. Sauer, B. Becker und I. Polian. „PHAETON: A SAT-based Framework for Timing-Aware Path Sensitization“. In: *IEEE Transactions on Computers* 65.6 (Juni 2016), S. 1869–1881.

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

[91] M. Sauer, A. Czutro, T. Schubert, S. Hillebrecht, I. Polian und B. Becker. „SAT-based Analysis of Sensitisable Paths“. In: *14th IEEE International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*. Cottbus, Deutschland, Apr. 2011.

[92] M. Sauer, Y. M. Kim, J. Seomun, H.-O. Kim, K.-T. Do, J.-Y. Choi, K. S. Kim, S. Mitra und B. Becker. „Early-Life-Failure Detection using SAT-based ATPG“. In: *2013 IEEE International Test Conference (ITC)*. Anaheim, USA, Sep. 2013.

[93] M. Sauter. *AMDs 7-nm-CPUs lassen Intel hinter sich*. Juli 2019. URL: `https://www.golem.de/news/ryzen-3900x-3700x-im-test-amds-7-nm-cpus-lassen-intel-hinter-sich-1907-141399.html` (besucht am 09. 03. 2020).

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

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

[96] E. Schneider, S. Holst, M. A. Kochte, X. Wen und H.-J. Wunderlich. „GPU-Accelerated Small Delay Fault Simulation“. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Grenoble, Frankreich, März 2015, S. 1174–1179.

[97] E. Schneider, S. Holst, X. Wen und H.-J. Wunderlich. „Data-Parallel Simulation for Fast and Accurate Timing Validation of CMOS Circuits“. In: *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. San Jose, USA, Nov. 2014, S. 17–23.

[98] E. Schneider, M. A. Kochte, S. Holst, X. Wen und H.-J. Wunderlich. „GPU-Accelerated Simulation of Small Delay Faults“. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36.5 (Mai 2017), S. 829–841.

[99] The International Technology Roadmap for Semiconductors (ITRS 2.0). *International Technology Roadmap for Semiconductors 2015 Edition*. Executive Report. 2015. URL: `http://www.itrs2.net` (besucht am 09. 03. 2020).

[100] M. Sharma und W.-T. Cheng. „X-Filter: Filtering Unknowns from Compacted Test Responses“. In: *2005 IEEE International Test Conference (ITC)*. Austin, USA, Nov. 2005, S. 1–9.

[101] M. Sharma und J. H. Patel. „Finding a Small Set of Longest Testable Paths that Cover Every Gate“. In: *2002 IEEE International Test Conference (ITC)*. Baltimore, USA, Okt. 2002, S. 974–982.

[102] L. Shi und X. Cai. „An Exact Fast Algorithm for Minimum Hitting Set“. In: *3rd International Joint Conference on Computational Science and Optimization*. Huangshan, China, Mai 2010, S. 64–67.

[103] M. O. Simsir, A. Bhoj und N. K. Jha. „Fault Modeling for FinFET Circuits“. In: *2010 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. Anaheim, USA, Juni 2010, S. 41–46.

[104] A. Singh, C. Han und X. Qian. „An Output Compression Scheme for Handling X-States from Over-Clocked Delay Test“. In: *2010 IEEE 28th VLSI Test Symposium (VTS)*. Santa Cruz, USA, Apr. 2010, S. 57–62.

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

[106] Z. Song, W. Weng und B. Engel. „Fast Root Cause Identification using Combination of Failure Analysis and In-Line Inspection“. In: *43rd International Symposium for Testing and Failure Analysis (ISTFA)*. Pasadena, USA, Nov. 2017, S. 135–139.

[107] A. Sprenger und S. Hellebrand. „Tuning Stochastic Space Compation to Faster-than-at-Speed Test“. In: *21st IEEE Symposium on Design and Diagnostics of Electronic Systems & Circuits (DDECS)*. Budapest, Ungarn, Apr. 2018, S. 73–78.

[108] A. Srivastava, A. D. Singh, V. Singh und K. K. Saluja. „Exploiting Path Delay Test Generation to Develop Better TDF Tests for Small Delay Defects“. In: *2017 IEEE International Test Conference*. Fort Worth, USA, Okt. 2017.

[109] A. Srivastava, D. Sylvester und D. Blaauw. *Statistical Analysis and Optimization for VLSI: Timing and Power*. New York [u. a.]: Springer, 2005. ISBN: 0-387-25738-1.

[110] G. Stelzer. *FinFET-Monopol gebrochen – FinFETs für alle*. Dez. 2013. URL: `https://www.elektroniknet.de` (besucht am 09. 03. 2020).

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

[112] R. Tayade und J. A. Abraham. „On-chip Programmable Capture for Accurate Path Delay Test and Characterization“. In: *2008 IEEE International Test Conference (ITC)*. Santa Clara, USA, Okt. 2008.

[113] M. Tehranipoor, K. Peng und K. Chakrabarty. *Test and Diagnosis for Small-Delay Defects*. New York [u. a.]: Springer, 2011. ISBN: 978-1-4419-8296-4.

[114] *Tessent LogicBIST*. Mentor, A Siemens Business. URL: `https://www.mentor.com/products/silicon-yield/products/logic-bist` (besucht am 09. 03. 2020).

[115] N. A. Touba. „Survey of Test Vector Compression Techniques“. In: *IEEE Design & Test of Computers* 23.4 (Apr. 2006), S. 294–303.

[116] N. A. Touba. „X-Canceling MISR – An X-Tolerant Methodology for Compacting Output Responses with Unknowns Using a MISR“. In: *2007 IEEE International Test Conference (ITC)*. Santa Clara, USA, Okt. 2007.

[117] K. H. Tsai, S. Hellebrand, J. Rajski und M. Marek-Sadowska. „STARBIST Scan Autocorrelated Random Pattern Generation“. In: *34th ACM/IEEE Design Automation Conference (DAC)*. Anaheim, USA, Juni 1997, S. 472–477.

[118] S. Tsukiyama, M. Ide, H. Ariyoshi und I. Shirakawa. „A New Algorithm for Generating All the Maximal Independent Sets“. In: *SIAM Journal on Computing* 6.3 (1977), S. 505–517.

[119] R. Turakhia, W. R. Daasch, M. Ward und J. Van Slyke. „Silicon Evaluation of Longest Path Avoidance Testing for Small Delay Defects“. In: *2007 IEEE International Test Conference (ITC)*. Santa Clara, USA, Okt. 2007.

[120] A. Uzzaman, M. Tegethoff, L. Bibo, K. McCauley, S. Hamada und Y. Sato. „Not all Delay Tests Are the Same – SDQL Model Shows True-Time“. In: *15th IEEE Asian Test Symposium (ATS)*. Fukuoka, Japan, Nov. 2006.

[121] E. H. Volkerink und S. Mitra. „Response Compaction with any Number of Unknowns using a new LFSR Architecture“. In: *42nd ACM/IEEE Design Automation Conference (DAC)*. Anaheim, USA, Juni 2005, S. 117–122.

[122] H. Vranken, S. Kumar Goel, A. Glowatz, J. Schloeffel und F. Hapke. „Fault Detection and Diagnosis with Parity Trees for Space Compaction of Test Responses“. In: *43rd ACM/IEEE Design Automation Conference (DAC)*. San Francisco, USA, Juli 2006, S. 1095–1098.

[123] M. Wagner und H.-J. Wunderlich. „Efficient Variation-Aware Statistical Dynamic Timing Analysis for Delay Test Applications“. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. Grenoble, Frankreich, März 2013.

[124] M. Wagner und H.-J. Wunderlich. „Incremental Computation of Delay Fault Detection Probability for Variation-Aware Test Generation“. In: *19th IEEE European Test Symposium (ETS)*. Paderborn, Deutschland, Mai 2014.

[125] J. A. Waicukauski, E. Lindbloom, B. K. Rosen und V. S. Iyengar. „Transition Fault Simulation“. In: *IEEE Design & Test of Computers* 4.2 (Apr. 1987), S. 32–38.

[126] J. Wang, H. Li, Y. Min, X. Li und H. Liang. „Impact of Hazards on Pattern Selection for Small Delay Defects“. In: *15th IEEE Pacific Rim International Symposium on Dependable Computing*. Shanghai, China, Nov. 2009, S. 49–54.

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

[128] Z. Wang und K. Chakrabarty. „Test-Quality/Cost Optimization Using Output-Deviation-Based Reordering of Test Patterns“. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27.2 (Feb. 2008), S. 352–365.

[129] Y. Wei, F. Baumann, S. Lucarini, K. Barth, Z. Song und A. Katnani. „SRAM PFET and NEFT Super FIN Characterization“. In: *43rd International Symposium for Testing and Failure Analysis (ISTFA)*. Pasadena, USA, Nov. 2017, S. 140–142.

[130] P. Wohl und L. Huisman. „Analysis and Design of Optimal Combinational Compactors“. In: *2003 IEEE 21st VLSI Test Symposium (VTS)*. Napa, USA, Mai 2003.

[131] P. Wohl, J. A. Waicukauski und F. Neuveux. „Increasing Scan Compression by Using X-chains“. In: *2008 IEEE International Test Conference (ITC)*. Santa Clara, USA, Okt. 2008.

[132] P. Wohl, J. A. Waicukauski und S. Patel. „Scalable Selector Architecture for X-Tolerant Deterministic BIST“. In: *41st ACM/IEEE Design Automation Conference (DAC)*. San Diego, USA, Juli 2004, S. 934–939.

[133] P. Wohl, J. A. Waicukauski, S. Patel und M. B. Amin. „X-Tolerant Compression and Application of Scan-ATPG Patterns in a BIST Architecture“. In: *2003 IEEE International Test Conference (ITC)*. Charlotte, USA, Sep. 2003, S. 727–736.

[134] P. Wohl, J. A. Waicukauski und T. W. Williams. „Design of Compactors for Signature-Analyzers in Built-In Self-Test“. In: *2001 IEEE International Test Conference (ITC)*. Baltimore, USA, Nov. 2001, S. 54–63.

[135] T. Wood, G. Giles, C. Kiszely, M. Schuessler, D. Toneva, J. Irby und M. Mateja. „The Test Features of the Quad-Core AMD Opteron™ Microprocessor“. In: *2008 IEEE International Test Conference (ITC)*. Santa Clara, USA, Okt. 2008.

[136] G. Xu und A. D. Singh. „Achieving High Transition Delay Fault Coverage with Partial DTSFF Scan Chains“. In: *2007 IEEE International Test Conference (ITC)*. Santa Clara, USA, Okt. 2007.

[137] G. Xu und A. D. Singh. „Low Cost Launch-On-Shift Delay Test with Slow Scan Enable“. In: *11th IEEE European Test Symposium (ETS)*. Southampton, Großbritannien, Mai 2006.

[138] H. Yan und A. D. Singh. „A New Delay Test Based on Delay Defect Detection Within Slack Intervals (DDSI)“. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 14.11 (Nov. 2006), S. 1216–1226.

[139] M. Yilmaz, K. Chakrabarty und M. Tehranipoor. „Test-Pattern Grading and Pattern Selection for Small-Delay Defects“. In: *2008 IEEE 26th VLSI Test Symposium (VTS)*. San Diego, USA, Mai 2008, S. 233–239.

[140] M. Yilmaz, K. Chakrabarty und M. Tehranipoor. „Test-Pattern Selection for Screening Small-Delay Defects in Very-Deep Submicrometer Integrated Circuits“. In: *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems* 29.5 (Mai 2010), S. 760–773.

[141] T. Yoneda, K. Hori, M. Inoue und H. Fujiwara. „Faster-Than-At-Speed Test for Increased Test Quality and In-Field Reliability“. In: *2011 IEEE International Test Conference (ITC)*. Anaheim, USA, Sep. 2011.

Betreute Arbeiten

[142] M. Aftab. „X-Aware Pattern Selection for Faster-than-at-Speed Test“. Masterarbeit. Universität Paderborn, 2018.

[143] R. Borisenko. „Direct Diagnosis for FAST“. Bachelorarbeit. Universität Paderborn, 2017.

[144] R. K. Gari. „Faster-than-at-Speed Logic BIST“. Masterarbeit. Universität Paderborn, 2017.

[145] R. Mahla. „Efficient Selection of Test Frequencies for Small Delay Defects“. Masterarbeit. Universität Paderborn, 2015.

[146] T. Mertens. „Experimentelle Studie eines eingebetteten Tests mit erhöhter Betriebsfrequenz“. Bachelorarbeit. Universität Paderborn, 2017.

[147] P. Negi. „Scan-Chain Configuration Supporting Faster-than-at-Speed Test“. Masterarbeit. Universität Paderborn, 2015.

[148] H. Ngawa. „Testmusterauswahl für den Test mit erhöhter Betriebsfrequenz“. Bachelorarbeit. Universität Paderborn, 2016.

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

[150] M. Schniedermann. „SAT-basierte Testmustererzeugung für Verzögerungsfehler“. Masterarbeit. Universität Paderborn, 2018.

[151] V. Tran. „Prüfzellengruppierung zur X-toleranten Kompaktierung beim Test mit erhöhter Betriebsfrequenz“. Bachelorarbeit. Universität Paderborn, 2016.

Eigene Publikationen

[152] S. Hellebrand, T. Indlekofer, M. Kampmann, M. A. Kochte, C. Liu und H.-J. Wunderlich. „FAST-BIST: Faster-than-At-Speed BIST Targeting Hidden Delay Defects“. In: *2014 IEEE International Test Conference (ITC)*. Seattle, USA, Okt. 2014.

[153] M. Kampmann und S. Hellebrand. „Design for Small Delay Test – A Simulation Study“. In: *Microelectronics Reliability* 80 (Jan. 2018), S. 124–133.

[154] M. Kampmann und S. Hellebrand. „Design-for-FAST: Supporting X-tolerant Compaction during Faster-than-at-Speed Test“. In: *20th IEEE Symposium on Design and Diagnostics of Electronic Systems & Circuits (DDECS)*. Auszeichnung für beste Publikation im Bereich Test. Dresden, Deutschland, Apr. 2017, S. 39–45.

[155] M. Kampmann und S. Hellebrand. „X Marks the Spot: Scan-Flip-Flop Clustering for Faster-than-at-Speed Test“. In: *25th IEEE Asian Test Symposium (ATS)*. Hiroshima, Japan, Nov. 2016.

[156] M. Kampmann, M. A. Kochte, C. Liu, E. Schneider, S. Hellebrand und H.-J. Wunderlich. „Built-in Test for Hidden Delay Faults“. In: *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems* 38.10 (Okt. 2019), S. 1956–1968.

[157] M. Kampmann, M. A. Kochte, E. Schneider, T. Indlekofer, S. Hellebrand und H.-J. Wunderlich. „Optimized Selection of Frequencies for Faster-than-at-Speed Test“. In: *24th IEEE Asian Test Symposium (ATS)*. Mumbai, Indien, Nov. 2015, S. 109–114.

[158] C. Liu, E. Schneider, M. Kampmann, S. Hellebrand und H.-J. Wunderlich. „Extending Aging Monitors for Early Life and Wear-out Failure Preventi-

on“. In: *27th IEEE Asian Test Symposium (ATS)*. Hefei, China, Okt. 2018, S. 92–97.