Institut für Technische Informatik
Universität Stuttgart
Pfaffenwaldring 47
D–70569 Stuttgart

Diplomarbeit Nr. 3380

Test Rekonfigurierbarer Scan-Netzwerke

Marcel Schaal

Studiengang:	Informatik
Prüfer:	Prof. Dr. rer. nat. Hans-Joachim Wunderlich
Betreuer:	M. Sc. Rafał Baranowski Dipl. Inf. Michael Kochte
begonnen am:	8. August 2012
beendet am:	7. Februar 2013
CR-Klassifikation:	B.7.1, B.8.1, D.2.5, J.6, K.1

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Algorithmen

1. Einleitung und Motivation

Moderne Mikrochips enthalten zahlreiche Instrumente, die zur Auswertung der Betriebsparameter, zum Test oder zur Validierung der Funktionalität genutzt werden. Rekonfigurierbare Scan-Netzwerke (RSN) bieten die Möglichkeit eines effizienteren, flexibleren und skalierbareren Zugriffs auf eingebettete Instrumente gegenüber üblichen statischen Scan-Ketten.

Durch den Einsatz von Rekonfigurierbaren Scan-Netzwerken nimmt jedoch die Komplexität der Zugriffsinfrastruktur zu. Während Scan-Ketten im Wesentlichen aus Schieberegistern bestehen, wodurch ein Defekt im Scan-Pfad relativ einfach festgestellt werden kann, finden sich in Rekonfigurierbaren Scan-Netzwerken, neben einfachen Logikelementen, auch Multiplexer und möglicherweise komplexere Schaltungen. Somit können unterschiedliche Scan-Pfade und -Hierarchien gebildet werden. Allerdings können bestehende Tests für Scan-Ketten die komplexere Steuerlogik bei Rekonfigurierbaren Scan-Netzwerken nicht ausreichend testen. Deshalb ist es notwendig, neuartige Teststrategien zu entwickeln, welche speziell an die Merkmale von Rekonfigurierbaren Scan-Netzwerken angepasst sind.

In dieser Arbeit werden Strategien für den Test Rekonfigurierbarer Scan-Netzwerke analysiert und ausgewertet. Es werden mehrere neue Verfahren zur Erzeugung von Testmustern vorgestellt, welche effizient bezüglich Laufzeit als auch des Speicherplatzbedarfs arbeiten.

Einen Überblick über die notwendigen Grundlagen des Hardware-Tests für den Verlauf dieser Arbeit wird in Kapitel 2 gegeben werden. Dort wird auf die Notwendigkeit des Tests bei der Herstellung von Mikrochips eingegangen. Zum besseren Verständnis folgt darauf ein Überblick über Fehlermodelle und Fehlersimulation als auch über die automatische Testmustererzeugung.

Aufbau und Arbeitsweise Rekonfigurierbarer Scan-Netzwerke werden im dritten Kapitel erklärt. Ebenfalls findet sich darin eine Klassifizierung der Fehlerwirkung des Haftfehlermodells in Rekonfigurierbaren Scan-Netzwerken. Darüber hinaus wird auf die effiziente Modellierung auf Transaktionsebene eingegangen, da sich eine taktgenaue Modellierung als zu komplex gestaltet.

Die verwendeten Testalgorithmen für Rekonfigurierbare Scan-Netzwerke werden in Kapitel 4 vorgestellt. Es wird deren Methodik erklärt und eine erste Analyse der Testbarkeit beschrieben.

Im fünften Kapitel wird die erarbeitete Implementierung näher erläutert. Dabei wird auf die Modellierung zur Erzeugung von Testmustern auf Transaktionsebene eingegangen. Hierzu ist es notwendig, eine entsprechende Verhaltensbeschreibung des Rekonfigurierbaren Scan-Netzwerks mit den notwendigen Erweiterungen zur Testmustererzeugung in ein Modell auf

Transaktionsebene zu überführen, welches ebenfalls Teil des Implementierungskapitels ist. Abschließend wird die Modellierung von Fehlern und deren Wirkung, als auch Detektion, auf Transaktionsebene beschrieben, da diese eine Voraussetzung für die Testmustererzeugung ist.

Die Ergebnisse und Bewertung der Testalgorithmen finden sich in Kapitel 6. Hierbei wird zuerst der Aufbau der Experimente beschrieben, welche zur Analyse der betrachteten Algorithmen zur Testmustererzeugung herangezogen wurden. Im restlichen Verlauf des Kapitels werden die Ergebnisse der Experimente erläutert.

Das abschließende Kapitel 7 fasst die Arbeit und deren Ergebnisse kurz zusammen und gibt einen Ausblick, welche weiteren Verbesserungen aus Sicht des Autors sinnvoll sind.

2. Grundlagen des Hardware-Tests

In diesem Kapitel werden die notwendigen Grundlagen zum Verständnis der weiteren Arbeit vermittelt. Dabei wird Grundwissen aus der Technischen Informatik, wie sie im Grundstudium des Studiengangs Diplom Informatik gelehrt werden, vorausgesetzt. Nachgelesen werden können diese Grundlagen zum Beispiel in [WWW06] oder [PH04]. Da die Implementierung der Arbeit auf der Gatterebene realisiert ist, werden keine elektrotechnischen Kenntnisse benötigt. Ein Grundwissen in Theoretischer Informatik, wie es zum Beispiel in [Sch08] vermittelt wird, ist vorteilhaft, denn die Modellierung des Problems findet in Form der Aussagenlogik statt.

Der erste Abschnitt erläutert das Problem des Hardware-Tests und der Diagnose, welches sich bei der Fertigung von integrierten Schaltungen ergibt. Daraufhin werden Verfahren vorgestellt, um dieses Problem einfacher zu gestalten. Anschließend wird eine Übersicht über verschiedene Abstraktionsebenen integrierter Schaltungen gegeben. Defekte werden im Hardware-Test als abstrakte Fehler in Form von Fehlermodellen beschrieben, welche näher in 2.4 erläutert werden. Zur Überprüfung der Auswirkung eines Fehlers in einer integrierten Schaltung kann Fehlersimulation genutzt werden, die in Kapitel 2.5 beschrieben wird. Folgend wird die Erzeugung von Testmustern für den Hardware-Test beschrieben und auf dessen Komplexität eingegangen. Abschließend wird die Umwandlung einer auf Gatterebene modellierten Schaltung in eine Instanz des Erfüllbarkeitsproblem der Aussagenlogik dargelegt.

2.1. Test und Diagnose

Eine fehlerfreie Herstellung integrierter Schaltkreise ist mit moderner Fertigungstechnik nicht möglich. Die immer höhere Integrationsdichte, und damit verbundene geringere Strukturgrößen, führen zu einem stärkeren Einfluss der Prozessschwankung auf die Schaltungseigenschaften und damit zu geringerer Ausbeute (engl. yield) [Sta86]. Die Ausbeute gibt das Verhältnis von funktionsfähigen Chips $n_{funktional}$ zur Gesamtzahl produzierter Chips $n_{produziert}$ an, daher:

$$yield = \frac{n_{funktional}}{n_{produziert}} \tag{2.1}$$

Die Ausbeute hängt von vielen Faktoren ab, wie etwa der Ausgereiftheit des Herstellungsverfahrens, der geforderten Produktqualität, der Größe der Chips und der Qualität des Tests. Je früher ein fehlerhafter Chip entdeckt wird, desto geringer sind die Kosten, da unnötige

Schritte, wie Verpacken in ein Gehäuse oder im schlimmsten Fall Rückholung des Systems von einem Kunden, vermieden werden. Deshalb werden Chips in jeder Phase der Fertigung getestet. Zur Überprüfung, ob ein Mikrochip bestimmte Anforderungen in Form seiner Spezifikation erfüllt, das heißt funktional ist, existierten unterschiedliche Teststrategien.

Werden Testmuster außerhalb des Chips erzeugt und durch Testsysteme zugeführt, so wird von externem Test gesprochen. Testverfahren, welche dedizierte Testlogik auf einem Chip zur Erzeugung von Testmustern voraussetzen, werden als eingebauter Selbsttest (engl. Built-in self-test, BIST) bezeichnet.

Während es beim Hardware-Test im Wesentlichen lediglich um eine binäre Entscheidung geht, nämlich ob ein Mikrochip korrekt arbeitet oder nicht, steht bei der Diagnose die Frage im Vordergrund, warum ein Mikrochip oder Modul nicht richtig funktioniert. Dies kann unter anderem mit der Fehlerstelle und der Art des Fehlers beantwortet werden.

Die Diagnoseergebnisse können genutzt werden, um den Fertigungsprozess zu optimieren und damit die Ausbeute erhöhen und die Produktionskosten senken. Bestimmte physikalische Strukturen können zum Beispiel sehr empfindlich auf Schwankungen des Fertigungsprozesses sein und zu einer Häufung von Defekten führen. Diese Strukturen sollten dann mit dem fortschreitendem Prozess des Herstellungsverfahrens angepasst werden, um eine höhere Schwankungstoleranz zu gewährleisten, so dass die Defekte reduziert und die Produktqualität verbessert werden kann.

2.1.1. Fehlerabdeckung

Um eine Aussage über die Güte einer gegebenen Testmustermenge T zu treffen, wird ein Gütekriterium benötigt. Dazu kann die Fehlerabdeckung C_T (engl. fault coverage) genutzt werden, welche sich aus der Zahl der von T entdeckten Fehler D, geteilt durch die Gesamtzahl der modellierten Fehler F ergibt:

$$C_T = \frac{D}{F} \tag{2.2}$$

Die Fehlerabdeckung gibt daher die relative Zahl der detektierten Fehler an und wird als Prozentzahl ausgedrückt. Da weder die Anzahl der Testmuster, noch die benötigte Testzeit in die Aussage der Fehlerabdeckung eingehen, stellt die Fehlerabdeckung nur ein Effektivtätsmaß und kein Effizienzmaß dar.

Je nach Anwendungsbereich werden unterschiedliche Fehlerabdeckungen gefordert. In der Automobilindustrie wird von den Herstellern in der Regel eine Fehlerabdeckung von mindestens 99,999% nach ISO 26262-5 vorgeschrieben [iso12].

2.1.2. Testbarkeit von Fehlern

Ein Fehler ist detektierbar genau dann, wenn eine Folge von Testmustern M existiert, welche den Fehler aktiviert und zu einer von der fehlerfreien abweichenden Ausgabe an

den Primärausgängen führt. Ein Fehler ist nicht detektierbar, falls keine solche Folge von Testmustern existiert. Abbildung 2.1 zeigt einen untestbaren Fehler. Diese können unter anderem in fehlertoleranten Strukturen auftreten, da in diesen ein Fehler keine Auswirkung auf das Ergebnis nehmen soll, sofern keine weiteren Maßnahmen des testbaren Entwurfs eingesetzt wurden.

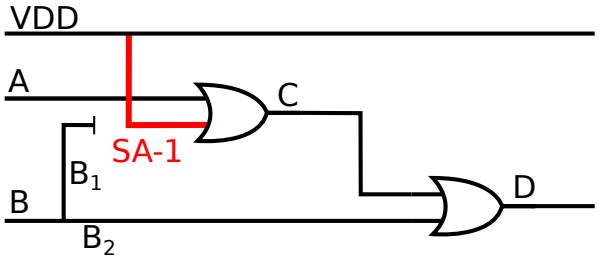

Abbildung 2.1.: Beispiel eines untestbaren Haftfehlers an Signal B_1

Wird ein Fehler $F1$ ausschließlich durch eine Teilmenge der Testmustermenge eines weiteren Fehlers $F2$ detektiert, so dominiert der Fehler $F2$ den Fehler $F1$.

Die Fehlerwirkung verschiedener Fehler kann identisch sein, so dass diese nicht voneinander unterscheidbar sind. In diesem Fall können die Fehler in Äquivalenzklassen zusammengefasst werden und müssen nicht separat behandelt werden. Dadurch kann Rechenzeit bei der Erzeugung von Testmustern eingespart werden [HKB94].

2.2. Testbarer Entwurf

Die Implementierung einer integrierten Schaltung beeinflusst ihre Testbarkeit und die entstehenden Testkosten. Testbarer Entwurf umfasst Verfahren und Strategien zum gezielten Entwurf integrierter Schaltungen mit hoher Testbarkeit.

Je schlechter eine integrierte Schaltung testbar ist, desto aufwendiger ist der benötigte Test und damit zum Beispiel die Anzahl der benötigten Testmuster. Auf der einen Seite müssen dann mehr Testmuster erzeugt werden, wodurch die benötigte Vorbereitungszeit für den Test steigt, um diese zusätzlichen Muster zu berechnen und zu validieren. Dies kann dann zu einer verlängerten Produkteinführungszeit führen und somit den Gewinn schmälern. Auf der anderen Seite müssen mehr Testmuster mittels automatischer Testsysteme (engl. automatic test equipment, ATE) in der gleichen Zeit geprüft werden, so dass entweder weniger Chips gefertigt werden können, oder weitere automatische Testsysteme benötigt werden, wodurch die Testkosten steigen.

Ein klassisches Beispiel für schlechte Testbarkeit stellt ein n-Bit-Zähler dar, welcher nur über zwei Eingänge zum Inkrementieren und Zurücksetzen verfügt und keine Optimierungen für einen effizienten Test besitzt. Um diesen Zähler vollständig funktional zu testen, müssen dazu alle 2^n Zustände iteriert werden. Unter Annahme, dass der Zähler 64 Bit besitzt, das heißt

$n = 64$ und 25 Millionen Iterationen pro Sekunde durchgeführt und getestet werden können, so beträgt die Testdauer fast 24.000 Jahre. Eine Möglichkeit des testbaren Entwurfs wäre die Aufspaltung in kleinere Zähler mit beispielsweise 8-Bit und separaten Reset-Eingängen.

Um den Test integrierter Schaltungen wirtschaftlich zu gestalten, wurden Methoden entwickelt, um die Testbarkeit stark zu verbessern. Eine davon besteht in der Integration einer zusätzliche Testinfrastruktur in die Schaltung. Scan-Design, welches im Abschnitt 2.2.1 näher erklärt wird, stellt so eine Testinfrastruktur dar.

Weitere Verfahren des testbaren Entwurf umfassen Prüfpunkte, das heißt das Hinzufügen von zusätzlichen Dateneingängen, um die Kontrollierbarkeit oder das Hinzufügen von Datenausgängen, um die Observierbarkeit zu erhöhen, sowie BIST und Testdatenkompression.

2.2.1. Scan-Ketten

Mit Scan-Ketten (engl. scan chain) wurde eine Entwurfstechnik eingeführt, welche die Erzeugung von Testmustern wesentlich vereinfacht. Dies wird erreicht, indem sequentielle Speicherelemente von außen kontrollier- und observierbar gemacht werden. Dadurch wird eine wirtschaftliche deterministische Testmustererzeugung auch für große integrierte Schaltungen ermöglicht. Scan-Ketten werden seit der Einführung durch [WA73] nahezu überall im testbaren Entwurf eingesetzt.

Die grundlegende Idee der Scan-Ketten besteht darin, dass sequentielle Speicherelemente durch Scan-Zellen ersetzt werden. Diese Scan-Zellen bestehen, neben dem ursprünglichen Daten- und Kontrollpfad des Speicherelements, aus einem weiteren Datenpfad mit zugehöriger Kontrolllogik, wie es in der Abbildung 2.2 skizziert ist. Alle Scan-Zellen werden dann über diesen zusätzliche Datenpfad in einer langen Kette verbunden, wie es in Abbildung 2.3 dargestellt wird. Die offenen Enden der Kette werden als zusätzliche Primäreinund -ausgänge, sowie deren Kontrollsignale, zur integrierten Schaltung hinzugefügt. Die zusätzlichen Signale sollen kurz erläutert werden:

Testdatenein- und -ausgänge Die eigentlichen Ein- und Ausgänge zum Transport von Daten vom automatischen Testsystem zum Chip, und vom Chip zum automatischen Testsystem. Diese bilden den Anfang (TDI) und das Ende (TDO) der Scan-Kette.

Testtakt Ein separates Taktsignal für die Testinfrastruktur kann notwendig sein, falls das Zeitverhalten im Testbetrieb anders ist, als im funktionalen Betrieb.

Testmodus Spezielles Signal, welches den Modus des Chips zwischen funktionalem Systembetrieb und Testbetrieb umschaltet.

Test-Reset Die Testinfrastruktur kann optional ein eigenes Signal zum Zurücksetzen aller Scan-Register besitzen.

Somit können Daten unabhängig von der eigentlichen Systemlogik von außerhalb der integrierten Schaltung in die Register geschrieben oder ausgelesen werden. Dies führt zur vollen Observier- und Kontrollierbarkeit der Speicherelemente.

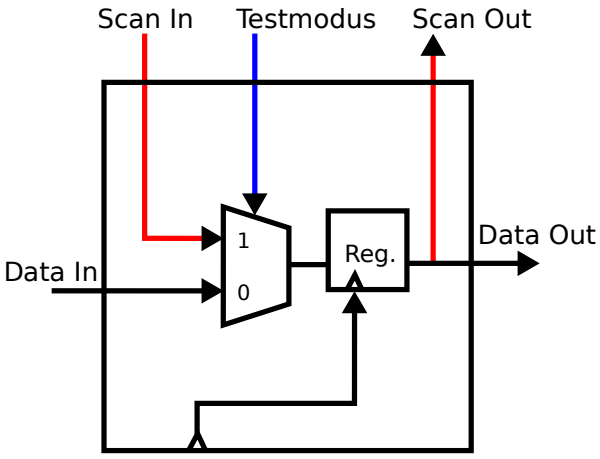

Abbildung 2.2.: Beispiel einer flankengesteuerten Scan-Zelle
Die Verbindungen des Scan-Pfads (blau) und der Aktivierung des Testmodus (rot) wurden hervorgehoben.

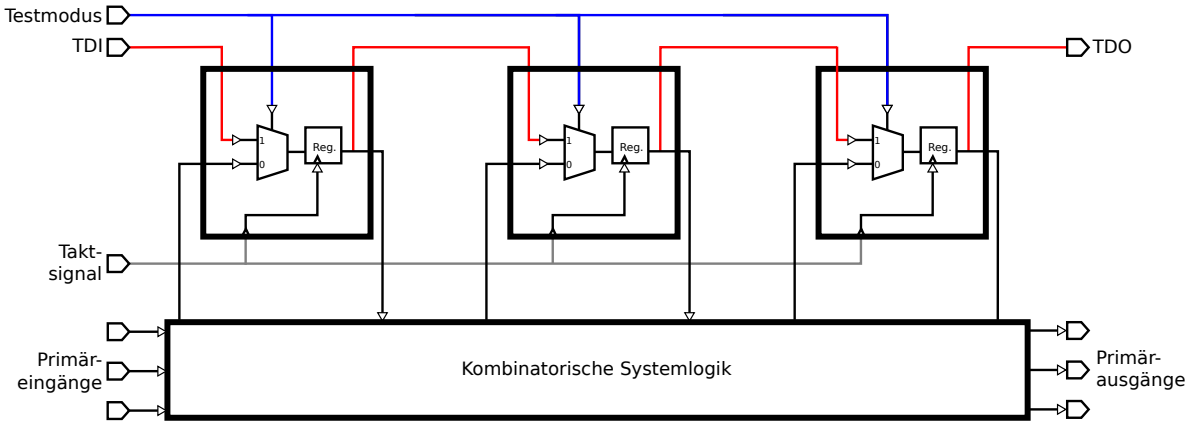

Abbildung 2.3.: Beispiel einer Scan-Kette
Alle speichernden Elemente der Schaltung wurden durch Scan-Zellen ersetzt. Die zusätzlichen Verbindungen des Scan-Pfads (rot) und der Aktivierung des Testmodus (blau) wurden hervorgehoben.

Ein Nachteil der Scan-Ketten besteht darin, dass durch das Hinzufügen der Multiplexer an den Registereingängen und der zusätzlichen Steuerlogik, der Logik- beziehungsweise Flächenbedarf für die Schaltung um bis zu 30 Prozent erhöht wird [Kun93]. Ein weiterer Nachteil betrifft das Zeitverhalten, da die Multiplexer unweigerlich auf dem kritischen Pfad liegen und die Pfadlänge erhöhen beziehungsweise die maximale Betriebsfrequenz reduzieren können.

Beim Entwurf der Testinfrastruktur gibt es hierbei verschiedene Varianten, die im Folgenden kurz erläutert werden:

Full-Scan-Design Bei einem Full-Scan-Design sind alle Register über die Scan-Ketten les- und schreibbar, so dass alle sequentiellen Elemente der Schaltung vollständig kontrollier- und observierbar sind.

Partial-Scan-Design Sofern nur eine Teilmenge der Register direkt durch die Scan-Ketten zugreifbar ist, handelt es sich um ein Partial-Scan-Design nach [Tri84]. Dies reduziert die zusätzlich benötigte Logik für die Testinfrastruktur gegenüber einem Full-Scan-Design und kann bei geeignetem Schaltungsentwurf zu ähnlich hoher Testbarkeit führen. Insbesondere bei Registern mit ungünstigem kritischem Pfad kann ein Partial-Scan-Design genutzt werden, um die geforderte Geschwindigkeit einzuhalten.

Parallele Scan-Ketten Durch die Verwendung von mehreren Scan-Ketten kann die Testzeit entsprechend reduziert werden und damit die Testkosten gesenkt werden [NGB92]. Soll die Zunahme an Primärein- und -ausgängen reduziert werden, so kann die Illinois Scan-Architektur eingesetzt werden, bei der lediglich ein einzelner Scan-Dateneingang genutzt wird [HP99].

Test-Kompression Häufig ist die Bandbreite zwischen externem Tester und zu testender Schaltung auf dem Wafer begrenzt. Deshalb werden die Testdaten vor der Übertragung komprimiert und auf dem Chip wieder dekomprimiert [VM03]. Dadurch können Kompressionsraten von 30-500 erreicht werden, was eine entsprechende Reduktion der Testzeit zur Folge hat [RTKM04].

2.2.2. Boundary Scan/JTAG

Zur Interoperabilität zwischen verschiedenen Herstellern beziehungsweise Anwendern im Verlauf der Fertigung und damit auch im Herstellungstest, wurde ein Standard zum Zugriff auf die Testinfrastruktur durch Scan-Ketten geschaffen. Diese Initiative wurde durch die Firma Philips initiiert und von verschiedenen großen Halbleiter-Herstellern in der Joint Test Action Group (JTAG) mitgetragen. Das Ziel war es eine standardisierte Schnittstelle zum Zugriff auf die Testinfrastruktur eines Mikrochips, welche ohne weitere Absprachen interoperabel ist, zu schaffen. Eine entsprechende Normierung fand im Rahmen des weltweiten Berufsverbands der Ingenieure aus Elektro- und Informationstechnik (Institute oft Electrical and Electronics Engineers, IEEE) unter der Bezeichnung IEEE 1149.1 statt [IEE01].

Verschiedene Erweiterungen wurden seit der ursprünglichen Normierung geschaffen, um den Einsatzzweck von JTAG zu erweitern. So wurde unter anderem IEEE 1149.6 zum Test von gemischten differentiellen Hochgeschwindigkeitssignalen oder 1149.4 für analoge Schaltungsteile erarbeitet.

Bei IEEE 1149/JTAG handelt es sich um eine Standardisierung für den Zugriff auf der Platinenebene. Ursprünglich entwickelt zum Test der Verbindungen auf der Platine (engl. boundary test), wird es heute ebenfalls genutzt, um die integrierte Schaltung selbst zu testen.

Dafür wurde bei der Entwicklung die Testinfrastruktur als Ganzes betrachtet und entworfen. Jedoch bestehen heutige Mikrochips nicht mehr nur aus einer einzelnen integrierten Schaltung, sondern setzen sich oft aus mehreren Modulen zu einem ganzen System auf einem Chip (engl. System-on-a-Chip, SOC) zusammen. Dies ist ohne Weiteres mit JTAG möglich, dennoch ist der Zugriff auf ein einzelnes Modul, ohne dass der gesamte Chip in den Testmodus versetzt werden muss, oft wünschenswert.

Um die Entwurfskosten niedrig zu halten, wurde ein Standard zur Wiederverwendung von Modultests gewünscht, um z.B. die Integration und Test von IP-Cores in SOCs zu vereinfachen. Ein solcher Standard fand Normierung als IEEE 1500 „Embedded Core Test“. Des weiteren ermöglicht IEEE 1500 ein Modul auf dem Scan-Pfad zu umgehen und so einen Bypass zu erzeugen, so dass ein separater Modultest möglich ist. Dabei handelt es sich allerdings nicht mehr um eine Scan-Kette mit statischer Struktur, sondern um ein Rekonfigurierbares Scan-Netzwerk, welche in Kapitel 3.1 beschrieben werden.

2.2.3. Test und Diagnose von Scan-Ketten

Eine Scan-Kette kann von einem Defekt betroffen sein und muss getestet werden. Insbesondere für die Diagnose ist es wichtig zu wissen, ob und wo eine Scan-Kette defekt ist. Für den Produkttest ist die Scan-Kette aufgrund ihrer Test- und Diagnoseeigenschaften essentiell.

Sind Scan-Ketten fehlerbehaftet, dann können diese keine Testantwort liefern. Dadurch können nur statische Werte am Ende des Scan-Ausgangs gelesen werden. In einem solchen Fehlerfall ist die Observier- und Kontrollierbarkeit extrem eingeschränkt.

Ein Testverfahren für Scan-Ketten ist der sogenannte Flush-Test. Dieses Testverfahren schreibt vordefinierte Werte in die Scan-Kette und überprüft, ob diese nach einer berechneten Zeitspanne an den Ausgängen wieder gelesen werden können.

Charakteristische Sequenzen für verschiedene Fehlermodelle auf Transistorebene wurden untersucht [MM95]. Das untersuchte Fehlermodell beschränkt sich dabei nicht nur auf einfache Haftfehler, sondern es werden zum Beispiel auch Setup-/Hold-Zeitfehler getestet, bei denen ein Datum bei bestimmten Sequenzen einen Takt zu früh beziehungsweise zu spät übernommen wird. Der so ausgelesene Testvektor weist dann den Verlust oder die Wiederholung eines Bits auf.

Ein weiteres Verfahren zum Test und zur Diagnose besteht in der Nutzung der Systemlogik. Dazu werden Testmuster in die Scan-Kette eingeschoben und ein oder mehrere Takte lang der Systemtakt aktiviert, so dass die Muster auf der Scan-Kette durch das System verarbeitet werden. Anschließend wird die Scan-Kette ausgelesen und die Testdaten untersucht [SPM92].

Die Scan-Kette kann um entsprechende Teststrukturen erweitert werden, um die Diagnoseauflösung zu erhöhen. Ein solches Verfahren basiert auf dem Hinzufügen von XOR-Gattern auf dem Scan-Pfad [EE95]. Durch einen zusätzlichen Primäreingang gesteuert können diese bei Aktivierung alle auf dem Scan-Pfad propagierten Werte invertieren. Werden diese für einen Takt aktiviert, so sind alle Werte nach der Fehlerstelle invertiert, während alle nachfolgenden

Werte weiter von der Fehlerwirkung beeinflusst sind. Aus der Länge der ausgelesenen und invertierten Werte lässt sich die Fehlerstelle präzise lokalisieren.

2.3. Abstraktionsebenen in der Modellierung

Abstraktion ermöglicht ein einfacheres Verständnis eines Systems durch eine Reduktion der Komplexität unter Verlust an Genauigkeit. In dieser Arbeit werden verschiedene Abstraktionsebenen integrierter Schaltungen genutzt. Die Zusammenhänge zwischen diesen Ebenen, sollen im Folgenden kurz erläutert werden.

Es finden zwei Arten der Abstraktion statt. Zum einen die Räumliche, anhand derer die weitere Modellierung betrachtet werden soll. Bei dieser werden verschiedene einzelne Teile zu einem größeren Block abstrahiert und dann im weiteren Verlauf gemeinsam behandelt. Die andere ist die oft mit der räumlichen einhergehende zeitliche Abstraktion. Wurde durch die räumliche Abstraktion ein System soweit vereinfacht, wie etwa beim Übergang von kontinuierlichen zu diskreten Modellen, dass eine weitere Betrachtung einer kontinuierlichen Zeit unnötig erscheint, so sollte diese auch entsprechend abstrahiert werden.

Das Verhalten von Modulen wird häufig in Verilog oder VHDL definiert. Scan-Ketten können in ihren domänen-spezifischen Sprachen (engl. domain specific language, DSL) beschrieben werden, wie zum Beispiel BSDL [PO90].

2.3.1. Gatterebene

Die erste für diese Arbeit wichtige Abstraktionsebene, die Gatterebene, betrachtet einzelne kombinatorische Logikelemente und sequentielle Speicherelemente, sowie deren Interkonnektivität. Häufig besteht die Beschreibung eines solchen Netzwerks aus einer Gatternetzliste. Diese benennt neben jedem Gatter auch die Ein- und Ausgangs- sowie die Verbindungsleitungen und enthält zusätzlich die Angabe, wie jedes Element mit den anderen verbunden ist.

Der Vorteil auf dieser Ebene ist die Observierbarkeit einzelner Signale. Der Aufwand ist jedoch für die Simulation und Modellierung deutlich höher als in einem abstrakteren Modell.

2.3.2. Register-Transfer-Ebene

Auf der höher liegenden Register-Transfer-Ebene werden Gatter zu einzelnen Teilmodulen, beispielsweise eine ALU, zusammengefasst. Die Zeit ist taktgenau modelliert, d.h. die Kommunikation findet in Form von einzelnen Datenwerten statt.

2.3.3. Transaktionsebene

Auf der Transaktionsebene werden Kommunikationsabläufe in atomare Operationen zusammengefasst. Das bedeutet, dass die Kommunikation in Form von Lese- und Schreibzugriffen, sowie eventuelle Verarbeitungsschritte, als unteilbare Transaktion stattfindet. Dies abstrahiert die taktgenaue Kommunikation durch Signale der Register-Transfer- oder Gatterebene und reduziert so die nötigen Simulationsschritte.

2.4. Strukturelle Fehlermodelle

Fehlermodelle dienen der Abstraktion von Defekten, welche bei der Fertigung von Mikrochips auftreten. Es existiert eine Vielzahl verschiedener Fehlermodelle für unterschiedliche Verwendungszwecke. Diese decken jeweils eine spezifische Untermenge von physikalischen Defekten ab und sind nicht zwangsläufig disjunkt.

Ein Fehlermodell betrachtet eine Untermenge der spezifischen Fehlerausprägungungen von Defekten, denn ohne diese Einschränkungen wären alle möglichen auftretenden Fehler denkbar. Dabei können unter Umständen nicht alle Ausprägungen von Defekten abgedeckt werden.

Wird ein strukturelles und deterministisches Fehlermodell für eine spezifische integrierte Schaltung betrachtet, so können alle möglichen Fehler in dieser aufgezählt werden. All diese Fehler in einer Menge zusammengefasst, ergeben die Fehlermenge für die entsprechende Schaltung.

Basierend auf einem Fehlermodell können entsprechende Testmuster erzeugt werden, die Fehler aktivieren und propagieren können. Mit den Testmustern kann geprüft werden, ob ein produzierter Mikrochip einen Fehler enthält oder fehlerfrei bezüglich des Fehlermodells ist.

2.4.1. Haftfehler

Ein sehr einfaches und sehr häufig eingesetztes Fehlermodell ist das Haftfehlermodell (engl. stuck-at fault model), welches auch die Grundlage für diese Arbeit bildet. Hier wird das Einfach-Haftfehlermodell betrachtet, welches ein Einfach-Fehlermodell ist, das heißt, es wird angenommen, dass immer nur genau ein einziger Fehler in der gesamten Schaltung beziehungsweise dem System auftritt.

Im Haftfehlermodell sind die Defekte als Verbindungsfehler zwischen den Gattern modelliert. Eine Verbindungsleitung ist im Fehlerfall unterbrochen und propagiert statisch entweder eine logische Null (stuck-at-0, SA-0) oder eine logische Eins (stuck-at-1, SA-1). Physikalisch lässt sich dies als direkte Verbindung der Fehlerinjektionsstelle zu Masse oder der Versorgungsspannung erklären. Die Gatter selbst sind nicht von Fehlern betroffen, das heißt die Logikfunktionen bleiben erhalten.

Ein entsprechendes Beispiel ist in Abbildung 2.4 dargestellt. Die Verbindungsleitung am Eingang A des ODER-Gatters ist unterbrochen und propagiert nun statisch eine logische Eins.

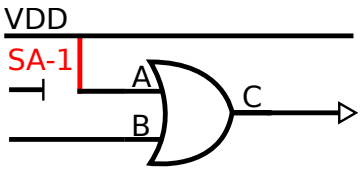

Abbildung 2.4.: Beispiel eines Haftfehlers
Eingang A weist Haftfehler an 1 auf.

Aufgrund seiner Einfachheit lässt sich dieses Fehlermodell auf Gatterebene taktgenau sehr effizient simulieren. Ein weiterer Vorteil besteht in der relativ simplen Erzeugung von Testmustern. Da dieses Fehlermodell am häufigsten eingesetzt wird, stehen entsprechend viele und ausgereifte Werkzeuge bereit.

Trotz der Einfachheit deckt es dennoch ein großes Spektrum an physikalischen Fehlerklassen aus anderen Fehlermodellen ab [BARVT82]. Zu großen Teilen trifft dies auch auf Verzögerungsfehler zu, welche ein anderes Zeitverhaltens besitzen und eine dynamischere Fehlerwirkung aufweisen.

2.4.2. Brückenfehler

Beim Brückenfehlermodell werden zwei oder mehr Verbindungsleitungen miteinander verbunden und damit kurzgeschlossen, so dass Daten zwischen unabhängigen Einheiten sich gegenseitig beeinflussen können. Dies hat zur Folge, dass der Fehler selbst zwar statisch vorhanden ist, wie beim Haftfehlermodell, aber die Fehlerwirkung abhängig von der Aktivität an den Verbindungen ist. So können beispielsweise verschiedene Taktdomänen verbunden werden. Welcher Wert bei Kurzschluss unterschiedlicher Werte resultiert, hängt von der verwendeten Technologie, Treiberstärke, den Ohmschen Widerstand, etc. ab. Ein Beispiel ist in Abbildung 2.5 gezeigt.

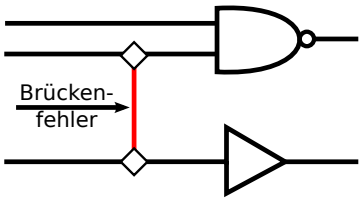

Abbildung 2.5.: Beispiel eines Brückenfehlers
Zwei unabhängige Verbindungsleitungen wurden durch einen Brückenfehler verbunden

Brückenfehler zur Versorgungsspannung V_{DD} oder zur Masse V_{SS} entsprechen den einfachen Haftfehlern. Welcher Wert als Ergebnis des Kurzschlusses entsteht, wird in weiteren Unterklassen des Brückenfehlermodells definiert. Zum Beispiel ergibt sich beim Wired-AND-Modell eine Null, sobald eines der beiden Signale eine Null führt, während es beim Wired-OR zu einer Eins kommt, sobald eines der Signale eine Eins führt. Weitere Brückenfehler-Klassifikationen für CMOS finden sich in [ESB00].

2.4.3. Weitere Fehlermodelle

Die bisher aufgezählten Fehlermodelle gehen von einer statischen Änderung der Logikfunktion aus. Jedoch müssen in einer korrekten synchronen Schaltung die Datenworte auch rechtzeitig zur Abtastung an den Registereingängen zu den Taktflanken anliegen. Ist dies nicht der Fall, so werden falsche Werte gespeichert. Solche Fehler werden unter anderem in unterschiedlichen Verzögerungsfehlermodellen betrachtet [MA98]. Jedoch können auch solche Fehler teilweise durch das einfache Haftfehlermodell beschrieben werden [MBAB99].

Das Byzantinische Fehlermodell ist ein allgemeines Fehlermodell und nimmt eine höchstmögliche Störung des Systems an. Byzantinische Fehler werden in vielen Bereichen der Informatik betrachtet und haben ihren eigentlichen Ursprung im Zwei-Generäle-Problem der Verteilten Systeme [PSL80].

2.5. Fehlersimulation

Zur Berechnung des Verhaltens einer integrierten Schaltung, kommen Simulationsmodelle zum Einsatz. Während des Entwurfs wird beispielsweise Logiksimulation zur funktionalen Validierung der Implementierung genutzt. Dahingegen wird bei der Testmustererzeugung die Fehlersimulation verwendet, welche dazu dient, einen Fehler in das Simulationsmodell einer Schaltung zu injizieren und dessen Auswirkung zu propagieren, um das Verhalten der fehlerhaften Schaltung zu untersuchen.

Die Eingabe für eine Fehlersimulation besteht aus einem Schaltungsmodell, Fehler- und einer Testmustermenge. Die Fehlersimulation untersucht dann für jeden Fehler, ob dieser von der Testmustermenge detektiert wird.

Fehlersimulation ist eine essentielle Aufgabe im Entwurf von Mikrochips und stellt aus diesem Grund einen wichtigen Forschungsbereich in der Technischen Informatik dar. Entsprechend ergiebig wurde dieser bereits untersucht und effiziente Verfahren entwickelt. Von diesen sollen nun ausgewählte Verfahren kurz vorgestellt werden.

2.5.1. Serielle Fehlersimulation

Bei der seriellen Fehlersimulation handelt es sich um ein sehr simples Verfahren: Zuerst wird die fehlerfreie Schaltung mit den Testmustern als Eingabe simuliert und die erzeugte Ausgabe gespeichert. Anschließend wird die fehlerbehaftete Schaltung mit den gleichen Eingaben simuliert und die Ausgabe mit der Gut-Simulation verglichen. Weichen die Ausgaben voneinander ab, so ist der Fehler durch die Testmuster detektierbar. Abbildung 2.6 zeigt einen möglichen Aufbau, bei dem Gut- und Fehlersimulation parallel ablaufen.

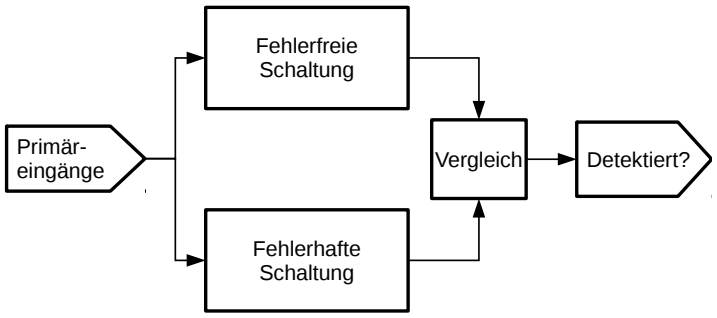

Abbildung 2.6.: Skizzierter Aufbau einer seriellen Fehlersimulation

Ein Vorteil dieser Methode ist die einfache Implementierung als doppelte Instantiierung, welche von einem Logiksimulator simuliert werden kann. Dadurch können je nach Fähigkeiten des Logiksimulators viele Fehlermodelle abgedeckt werden. Jedoch ist die Simulationszeit sehr hoch im Vergleich zu effizienten Fehlersimulationsalgorithmen, welche unter anderem ereignisgesteuert oder bitparallel rechnen.

2.5.2. Parallel Pattern Single Fault Propagation

Die Idee des PPSFP-Algorithmus besteht in der parallelen Simulation mehrere Werte der Schaltung in einem Datenwort, während gleichzeitig nur ein Fehler betrachtet wird. Darüber hinaus werden Fehler bei der Gebietsanalyse durch logisches Schließen effizient detektiert [Sch88].

Der PPSFP-Algorithmus lässt sich effizient auf Many-Core-Architekturen wie GPGPUs (General Purpose Graphical Processing Units), aufgrund seiner guten Parallelisierbarkeit, abbilden. Dadurch lässt sich die Simulationszeit bis zu einem Faktor 16 beschleunigen [KSWZ10].

2.5.3. Sequentielle Fehlersimulation

Bei der nebenläufigen Fehlersimulation (engl. concurrent fault simulation) werden die fehlerfreie, sowie alle fehlerbehafteten Schaltungen gleichzeitig ereignisgesteuert simuliert [UB73]. Dazu werden alle Gut- und Fehlerereignisse, das heißt eine Signaländerung an einem Gatter, an den Gattern selbst gespeichert. Die nebenläufige Fehlersimulation kann für verschiedene Schaltungsmodelle, unter anderem für sequentielle Schaltungen, Fehlermodelle und Zeitabstraktionen verwendet werden.

Die Differentielle Fehlersimulation bearbeitet ereignisgesteuert für ein Testmuster alle möglichen Fehler. Zunächst wird eine Gut-Simulation durchgeführt und anschließend Fehler nacheinander injiziert und simuliert. Bei abweichender Ausgabe wird ein Fehler als detektiert markiert und aus der Fehlerliste entfernt [LH92].

2.6. Automatische Testmustererzeugung

Automatische Testmustererzeugung (engl. automatic test pattern generation, ATPG) ist ein Sammelbegriff für Verfahren, welche der Erzeugung von Testmustern für Fehler in einer integrierten Schaltung dienen. Ein so erzeugtes Testmuster soll bei Anwendung den Fehler aktivieren, dessen Wirkung propagieren und so zu einem beobachtbaren, abweichenden Verhalten der fehlerbehafteten Schaltung im Gegensatz zur fehlerfreien Schaltung führen. Durch das abweichende Verhalten kann eine fehlerbehaftete Schaltung von einer fehlerfreien unterschieden werden. ATPG für kombinatorische Schaltungen und dem Haftfehlermodell gehört zur Klasse der NP-vollständigen Probleme, welche in Abschnitt 2.7.1 beschrieben werden und nach gegenwärtigem Stand der Forschung nicht effizient berechenbar sind.

Ein Beispiel für ATPG stellt der D-Algorithmus dar [Rot66]. Dieser beschreibt ein Verfahren zur Erzeugung von Testmustern für einfache Haftfehler. Der Ansatz basiert auf der Propagierung des Fehlers durch die Schaltung und Ableitung der daraus entstehenden Belegungen für die Primäreingänge durch logisches Schließen. Da unterschiedliche Belegungen möglich sind und Konflikte auftreten können, wird Backtracking genutzt, um diese durch zu probieren. Weitere Verbesserungen des D-Algorithmus stellen PODEM (path oriented decision making) als auch FAN (fanout-oriented test generation) dar, welche die Laufzeit reduzieren [GR81], [FS83].

Andere Verfahren reduzieren das Problem des ATPGs auf das Erfüllbarkeitsproblem der Aussagenlogik, welches in Kapitel 2.7 vorgestellt wird [Lar92], [TED10].

2.7. Erfüllbarkeitsproblem der Aussagenlogik

Das Erfüllbarkeitsproblem der Aussagenlogik (engl.: Boolean satisfiability-problem, SAT) stellt ein Entscheidungsproblem dar. Es überprüft, ob für die Variablen einer gegebenen

Boole'schen Funktion $f(v_1, \ldots, v_n) : \mathcal{B}^n \mapsto \mathcal{B}$ eine Belegung existiert, so dass f eine wahre Aussage ist [MMZ+01], [GN02].

Das Erfüllbarkeitsproblem, das klassische Beispiele für NP-Vollständigkeit, diente Stephen Cook zum Herleiten jener. Nachfolgend soll auf die NP-Vollständigkeit eingegangen werden, um die Komplexität der beschriebenen Probleme besser zu verstehen.

2.7.1. NP-Vollständigkeit

In der Komplexitätstheorie bezeichnet die Klasse NP alle Entscheidungsprobleme, welche von einer nichtdeterministischen Turingmaschine in polynomieller Zeit bezüglich der Eingabelänge n entschieden werden können. Eine besondere Teilmenge in NP bildet die NP-Vollständigkeit, welche von Stephen A. Cook 1971 beziehungsweise Levin postuliert wurde [Coo71], [Lev73]. NP-vollständige Entscheidungsprobleme lassen sich untereinander, mittels sogenannter Polynomialzeitreduktion $\preceq_p$, aufeinander abbilden. Eine Polynomialzeitreduktion r ist eine Abbildung, welche eine Instanz eines Entscheidungsproblems P in eine andere Instanz des Problems Q transformiert. Die Lösung für die Instanz des Problems in Q ist dann auch die Lösung für das Problem in P.

Ein Problem L ist NP-schwer (engl. NP-hard), wenn es mindestens so „schwer" wie alle anderen Probleme in NP ist. Formal muss dafür jedes Problem $K \in NP$ mittels einer Polynomialzeitreduktion durch L lösbar sein, das heißt:

$$L \in NP \Leftrightarrow \forall K \in \mathrm{NP} : L \preceq_{\mathrm{p}} K \tag{2.3}$$

Ist L zusätzlich in NP selbst, dann gehört L zur Klasse der NP-vollständigen Probleme.

2.7.2. Aussagenlogische Normalformen

Für eine gegebene Boole'sche 'Funktion kann es mehrere Repräsentationen gegeben. Im Laufe der Zeit wurden verschiedene Repräsentationformen, so genannte Normalformen, mit vorgegebenen Eigenschaften geschaffen. Eine Sonderform der Normalformen bilden kanonische Normalformen, welche strengeren Kriterien genügen, so dass es eine eindeutige Repräsentation für jede Funktion gibt.

Eine wesentliche Normalform stellt hierbei die sogenannte konjunktive Normalform (engl. conjunctive normal form, CNF) dar. Bei dieser setzen sich Formeln aus Konjunktionstermen zusammen, welche wiederum aus Disjunktionstermen bestehen. Ein Disjunktionsterm wird aus Literalen, das heißt positive oder negierte Variablen, gebildet. Als Beispiel ist in Gleichung 2.4 das Übertragsbit eines Volladdierers in CNF angegeben.

$$\begin{aligned} c_{out} &= (\overline{a} \wedge b \wedge c_{in}) \vee (a \wedge \overline{b} \wedge c_{in}) \vee (a \wedge b \wedge \overline{c_{in}}) \vee (a \wedge b \wedge c_{in}) \\ &= (b \wedge c_{in}) \vee (a \wedge c_{in}) \vee (a \wedge b) \end{aligned} \tag{2.4}$$

Als Standard für die Eingabe benutzen die Werkzeuge zum Lösen aussagenlogischer Erfüllbarkeitsprobleme die konjunktive Normalform, da für diese effiziente Lösungsverfahren existieren [GZA+02].

Die Darstellung in konjunktiver Normalform alleine führt nicht zu einer eindeutigen Repräsentation, sie ist daher nicht kanonisch. Zur Bildung der kanonischen konjunktiven Normalform werden die Produktterme aus paarweise verschiedenen Maxtermen gebildet. Ein Maxterm oder Volldisjunktion ist ein Disjunktionsterm, welcher alle Variablen der Funktion enthält und entsprechend als Literale einbettet.

Jede aussagenlogische Formel kann in eine entsprechende konjunktive Normalform umgewandelt werden. Eine händische Möglichkeit besteht in der Aufstellung der Wahrheitstabelle und das Auslesen der entsprechenden Disjunktionsterme, für welche die Wahrheitstabelle einen wahren Wert in der Ergebnisspalte enthält. Allerdings benötigt die Aufstellung der Wahrheitstabelle für n Variablen bereits 2^n Einträge und ist damit nicht effizient.

Zur Umwandlung in konjunktive Normalform können Logikäquivalenzumformungen, wie das Distributiv- oder De Morgan's-Gesetz genutzt werden. Shannons Aufzählungstheorem besagt, dass in der Regel kanonische Normalformen exponentiell länger sind als eine entsprechende freie Form der aussagenlogischen Formel [Sha49]. Es kann also vorkommen, dass zur Erzeugung der entsprechenden kanonischen Normalform exponentieller Aufwand benötigt wird und somit nicht effizient berechenbar ist.

Dies soll an folgendem Beispiel verdeutlicht werden. Gegeben sei die Formel f:

$$F \equiv \bigvee_{i=0}^{n} (X_i \wedge Y_i) \equiv (X_0 \wedge Y_0) \vee \ldots \vee (X_{n-1} \wedge Y_{n-1}) \tag{2.5}$$

Aus F lässt sich die äquivalente Gleichung G mittels Distributivgesetz in konjunktiver Normalform herleiten:

$$\begin{aligned} G &\equiv F \\ &\equiv \bigwedge_{i=0}^{n} \Big(\bigvee_{k=0}^{n-i} (X_k) \bigvee_{k=n-i}^{n} (Y_k) \Big) \\ &\equiv (X_0 \vee X_1 \vee \ldots \vee X_{n-1}) \wedge (X_0 \vee X_1 \vee \ldots \vee Y_{n-1}) \wedge \ldots \wedge (Y_0 \vee Y_1 \vee \ldots \vee Y_{n-1}) \end{aligned} \tag{2.6}$$

Während F lediglich $2n$ Literale besitzt, gibt es nach Umformung in konjunktive Normalform nach 2.6 in G 2^n Literale. Um diesen Aufwand zu vermeiden, wird die Methode der Erfüllbarkeitsäquivalenzumformung genutzt. Die Gleichung ist nach einer solchen Umformung nicht mehr exakt äquivalent, sondern lediglich erfüllbarkeitsäquivalent, das heißt bei gleichen Eingabewerten für korrespondierende Variablen ergibt die Gleichung dieselbe Lösung. Eine solche Umformung ist die Tseitin-Umformung [Tse68]. Die Größe einer Formel bei den Tseitin-Umformungen wächst lediglich linear im Verhältnis zur Ausgangsgleichung.

2.7.3. Schaltungstransformation in konjunktive Normalform

Die Erzeugung von Zugriffsmustern für Rekonfigurierbare Scan-Netzwerke kann mittels des Erfüllbarkeitsproblem für aussagenlogische Gleichungen gelöst werden. Dazu wird die Schaltungsbeschreibung in Form einer Gatternetzliste in eine aussagenlogische Formel umgeformt.

Die Modellierung kombinatorischer Logik in konjunktiver Normalform lässt sich einfach durch statische Umformung durchführen. In Tabelle 2.1 sind für die gebräuchlichen Gatter die entsprechenden Terme in konjunktiver Normalform angegeben. Mit diesen kann der Schaltungsgraph traversiert und die einzelnen Knoten in einem CNF-Modell aufgebaut werden. Jeder Primärein- und -ausgang der Schaltung sowie das Ausgangssignal eines Gatters werden separate Variablen zugeteilt.

Name	**Operation**	**CNF-Äquivalente Darstellung**
Identität	$C = A$	$(\overline{A} \vee C) \wedge (A \vee \overline{C})$
Negation	$C = \overline{A}$	$(\overline{A} \vee \overline{C}) \wedge (A \vee C)$
Konjunktion	$C = A \wedge B$	$(\overline{A} \vee \overline{B} \vee C) \wedge (A \vee \overline{C}) \wedge (B \vee \overline{C})$
Disjunktion	$C = A \vee B$	$(A \vee B \vee \overline{C}) \wedge (\overline{A} \vee C) \wedge (\overline{B} \vee C)$
Kontravalenz	$C = A \oplus B$	$(\overline{A} \vee \overline{B} \vee \overline{C}) \wedge (A \vee B \vee \overline{C}) \wedge (A \vee \overline{B} \vee C) \wedge (\overline{A} \vee B \vee C)$
Äquivalenz	$C = A \Leftrightarrow B$	$(\overline{A} \vee \overline{B} \vee C) \wedge (A \vee B \vee C) \wedge (A \vee \overline{B} \vee \overline{C}) \wedge (\overline{A} \vee B \vee \overline{C})$
Sheffer (NAND)	$C = \overline{A \wedge B}$	$(\overline{A} \vee \overline{B} \vee \overline{C}) \wedge (A \vee C) \wedge (B \vee C)$
Peirce (NOR)	$C = \overline{A \vee B}$	$(\overline{A} \vee \overline{C}) \wedge (\overline{B} \vee \overline{C})$
Implikation	$C = A \rightarrow B$	$(\overline{A} \vee B \vee \overline{C}) \wedge (A \vee C) \wedge (\overline{B} \vee C)$

Tabelle 2.1.: Logikprimitive umgeformt nach konjunktiver Normalform

Für sequentielle Schaltungen müssen die einzelnen Zeitabschnitte, wie zum Beispiel Takte, modelliert werden. Dies kann durch Ausrollen, das heißt mehrfaches Instantiieren und durch Erweitern mit zusätzlichen Transitionsklauseln, geschehen. Diese Transitionsklauseln modellieren dabei den Übergang der Daten einzelner Speicherelemente zwischen zwei Zeitabschnitten. Die Primäreingänge werden dabei ebenfalls ausgerollt, so dass in jedem Zeitabschnitt neue Daten an die Schaltung angelegt werden können. Das fiktive Anlegen im Erfüllbarkeitsproblem geschieht durch Festlegung als positives oder negatives Literal für die jeweilige Variable des Zeitabschnitts. Ebenso werden Variablen für die Primärausgänge ausgerollt und sind so in der berechneten Lösung zu finden.

3. Grundlagen Rekonfigurierbarer Scan-Netzwerke

Ohne weitere Optimierungen wächst die Zugriffszeit von Scan-Ketten linear mit der Länge des Scan-Pfads. Dies ist besonders ineffizient im Zusammenhang mit der stetigen Verbesserung der Fertigungstechnologie, insbesondere der Integrationsdichte für moderne Mikrochips. Gordon E. Moore beobachtete die Gesetzmäßigkeit, welche als Moore's Gesetz bekannt ist, dass sich seit 1970 die Zahl der Transistoren ungefähr alle zwei Jahre verdoppelt [Moo98]. Da es sich dabei um ein exponentielles Wachstum handelt und die Zahl der Register ebenfalls entsprechend wächst, ist es offensichtlich, dass sich eine reine lineare Skalierung der Scan-Ketten als nicht effizient erweist. Der Vorteil von Rekonfigurierbaren Scan-Netzwerken besteht in einer effizienteren Zugriffszeit, welche sich aus der Möglichkeit des unabhängigen Zugriffs auf Instrumente, wie zum Beispiel Temperatursensoren, ohne das andere Teilmodule in den Testmodus versetzt oder anderweitig beeinflusst werden, ergibt.

Dieses Kapitel führt die Grundlagen Rekonfigurierbarer Scan-Netzwerke ein. Zu Beginn wird deren Aufbau und Struktur erläutert und die entsprechende Terminologie erklärt. Das anschließende Kapitel befasst sich mit der Modellierung eines Rekonfigurierbaren Scan-Netzwerks auf Transaktionsebene. Abschließend wird die Fehlerwirkung des Haftfehlermodells für Rekonfigurierbare Scan-Netzwerke diskutiert und spezielle Fehlermodelle auf Transaktionsebene betrachtet.

3.1. Aufbau und Struktur

Rekonfigurierbare Scan-Netzwerke bilden einen generischen und skalierbaren Ansatz für den Zugriff auf Teilmodule und Instrumente integrierter Schaltungen. Wie bei Scan-Ketten werden sequentielle Speicherelemente durch Scan-Zellen ersetzt und von außen zugänglich gemacht, um so Kontrollier- und Observierbarkeit zu erreichen. Diese Scan-Zellen beziehungsweise Scan-Segmente befinden sich auf einer Zugriffsinfrastruktur, welche als ein gerichteter, azyklischer Graph (engl. directed, acyclic Graph, DAG) dargestellt werden kann. Durch die flexible Infrastruktur kann der Zugriff auf einzelne Teilmodule effizienter gestaltet werden als mit einer simplen linearen Verkettung.

In diesem Graphen werden die Scan-Zellen als Knoten dargestellt und Kanten bilden einfache Scan-Pfade ab. Ein Scan-Pfad stellt einen Pfad durch den Graphen vom Eingangsknoten bis zum Ausgangsknoten dar.

Zu jedem Zeitpunkt kann es nur einen einzigen Pfad durch den Graphen geben, welcher dann ähnliche Eigenschaften, wie eine Scan-Kette, aufweist. Um dies zu erreichen, besitzen Knoten beziehungsweise Scan-Zellen mit mehreren Vorgängern Multiplexer zur Auswahl eines Vorgängers. Der sich so ergebende Pfad wird als aktiver Scan-Pfad bezeichnet.

Welcher Scan-Pfad aktiv ist, ergibt sich aus der Konfiguration des Rekonfigurierbaren Scan-Netzwerkes. Abbildung 3.1 zeigt ein Beispiel eines Rekonfigurierbaren Scan-Netzwerks mit hervorgehobenen aktivem Scan-Pfad. Als Konfiguration wird der Zustand der Register bezeichnet, welche zur Steuerung der Multiplexer dienen. Diese Konfigurationsregister liegen auf der Zugriffsinfrastruktur selbst und werden, genau wie der Zugriff auf Module, durch den Schiebevorgang des Scan-Pfads kontrolliert. Dadurch können sowohl Schreib- als auch Lesevorgänge ausgeführt werden. Die Ansteuerung der Zugriffsinfrastruktur kann dabei die gleichen Anschlüsse wie Scan-Ketten nutzen. Bei JTAG wird dies als TAP (test access port) bezeichnet, womit bestehende Testwerkzeuge weiter genutzt werden können.

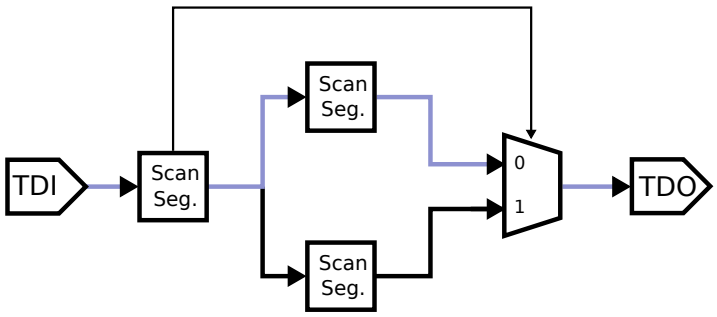

Abbildung 3.1.: Beispiel eines Rekonfigurierbaren Scan-Netzwerks
Der aktive Scan-Pfad (blau) wurde hervorgehoben. Das erste Scan-Segment entspricht einem Konfigurationsregister, da dessen Wert die Einstellung des Multiplexers steuert.

Als Scan-Pfad-Segment wird ein Pfad zwischen zwei Scan-Segmenten bezeichnet. Dieses entspricht einer Kante des DAG.

Ein Scan-Pfad ist symmetrisch zu einem anderen Scan-Pfad genau dann, wenn beide Scan-Pfade die gleiche Anzahl von Scan-Segmenten aufweisen. Die Scan-Pfade des in Abbildung 3.1 skizzierten Rekonfigurierbaren Scan-Netzwerks sind alle symmetrisch.

Die Berechnung der Stimuli für die Zugriffsinfrastruktur, um die Modulauswahl von einer Konfiguration in eine vorgegebene Konfiguration zu überführen, wird als Retargeting bezeichnet. Ein solches Muster zur Rekonfiguration heißt Zugriffsmuster .

Teilmodule können beliebig gewählt werden, allerdings scheint eine Auswahl nach einzelnen Komponenten, wie beispielsweise die ALU eines Prozessors, die BIST-Steuerung eines Speichers oder eine Gruppe von Instrumenten, als sinnvoll. IP-Cores von SOCs besitzen häufig bereits eine eigene Testschnittstelle, welche daher als Teilmodule sehr gut geeignet sind. Tatsächlich findet dies Anwendung in dem in Kapitel 2.2.2 beschriebenen Standard IEEE 1500

„Embedded Core Test“. Schaltungen nach diesem Standard können als Rekonfigurierbares Scan-Netzwerk betrachtet werden. Die Testschnittstelle der einzelnen Module ermöglicht es, diese durch den Scan-Pfad zugreifbar zu machen oder vom Scan-Pfad abzukoppeln. Dadurch ist der Scan-Pfad nicht mehr rein statisch, sondern veränderbar und bildet damit ein Rekonfigurierbares Scan-Netzwerk.

Eine Normierung zum Zugriff auf die Instrumente einer integrierten Schaltung ist derzeit in der Entstehung als IEEE P1687 „Instrument JTAG“ (IJTAG) [EB06]. In dieser Normierung wird unter anderem die Nutzung von Rekonfigurierbaren Scan-Netzwerken zum Zugriff auf die Instrumente vorgeschlagen.

3.1.1. Struktur von Scan-Segmenten

Scan-Segmente, welche als Konfigurationsregister dienen, benötigen ein zusätzliches Schattenregister, welches den gespeicherten Inhalt während des Schiebebetriebs erhält. Ansonsten würden sich die Steuersignale der Kontrolllogik während des Schiebebetriebs verändern und dadurch auch der aktive Scan-Pfad anders ausprägen. Eine solche Struktur wird bei Scan-Ketten als „Enhanced Scan“ bezeichnet. Eine Beispielinstanz ist in Abbildung 3.2 skizziert.

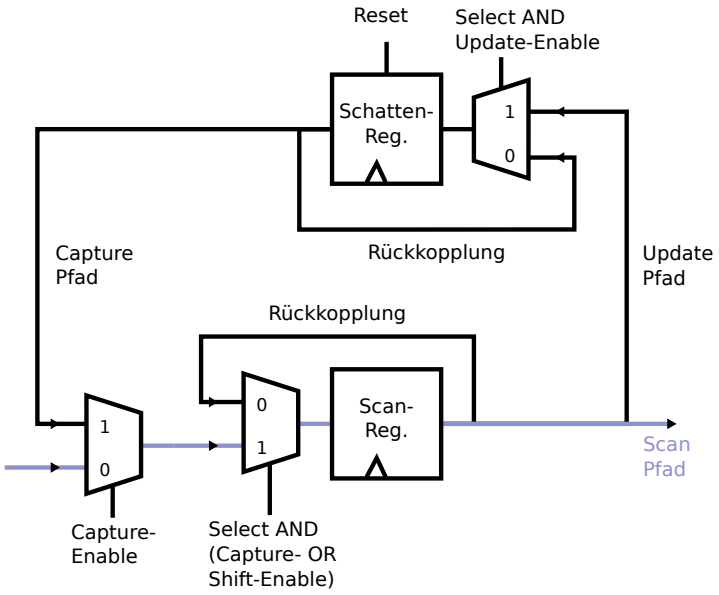

Abbildung 3.2.: Beispiel eines Scan-Segments und zugehöriger Daten- und Kontrollpfade Der Scan-Pfad (blau) wurde hervorgehoben.

Jedes Scan-Segment besitzt eine Aktivierungsbedingung in Form des Select-Signals. Dieses muss aktiv sein, sofern sich das Scan-Segment auf dem aktiven Scan-Pfad befindet und das

Rekonfigurierbare Scan-Netzwerk eine Operation ausführen soll. Das Select-Signal wird kombinatorisch aus den Konfigurationsregistern erzeugt.

Die Daten aus einem Scan-Register werden in das Schattenregister übernommen, genau dann, wenn die Signale Update-Enable und Select aktiv sind. Diese werden in Abbildung 3.3 entsprechend dargestellt. Dazu werden die Multiplexer so getrieben, dass der Update-Pfad aktiviert wird.

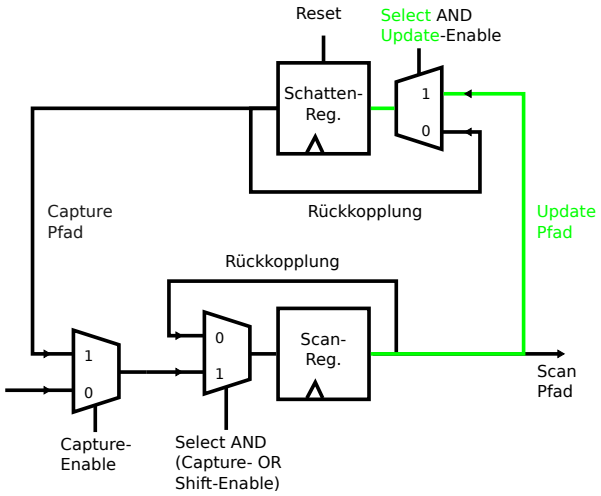

Abbildung 3.3.: Beispiel der aktiven Pfade und Signale (grün) in einem Scan-Segment während der Update-Phase. Der aktive Rückkopplungsdatenpfad des Scan-Registers wurde nicht explizit hervorgehoben.

Bei einer Capture-Operation wird das Datum des Schattenregisters in das Scan-Register übertragen und dient damit dem Auslesen des Zustands des Rekonfigurierbaren Scan-Netzwerks. Die Capture-Operation wird durch setzen des Capture-Enable- sowie Select-Signals erreicht, wie es in Abbildung 3.4 gezeigt ist.

Der Schiebebetrieb wird durch das Aktivieren des Shift-Enable- und des Select-Signals erreicht, welches in Abbildung 3.5 skizziert wird. Hierbei übernimmt das Scan-Register die Daten von seinem Vorgänger auf dem aktiven Scan-Pfad.

3.1.2. Scan-Pfadzugriff (CSU-Zyklus)

Ein Zugriff auf ein Rekonfigurierbares Scan-Netzwerk läuft, wie bei Scan-Ketten nach JTAG, in CSU-Zyklen ab. Ein CSU-Zyklus umfasst dabei das Übertragen der Werte aus dem Schattenregister zu den Scan-Registern (Capture) sowie das Herausschieben der Konfigurationsdaten und Hineinschieben von Daten in Form einer neuen Konfiguration (Shift). Wenn die neuen Daten beziehungsweise Konfiguration in die Scan-Register geschoben wurden, werden die Daten durch die Update-Operation in die Schattenregister übernommen (Update).

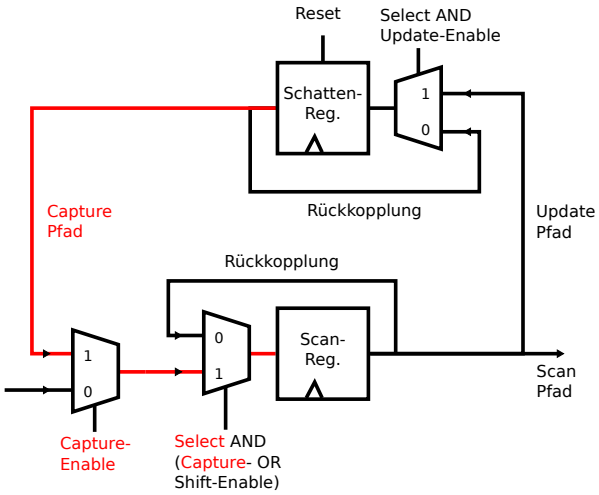

Abbildung 3.4.: Beispiel der aktiven Pfade und Signale (rot) in einem Scan-Segment während der Capture-Phase. Der aktive Rückkopplungsdatenpfad des Schattenregisters wurde nicht explizit hervorgehoben.

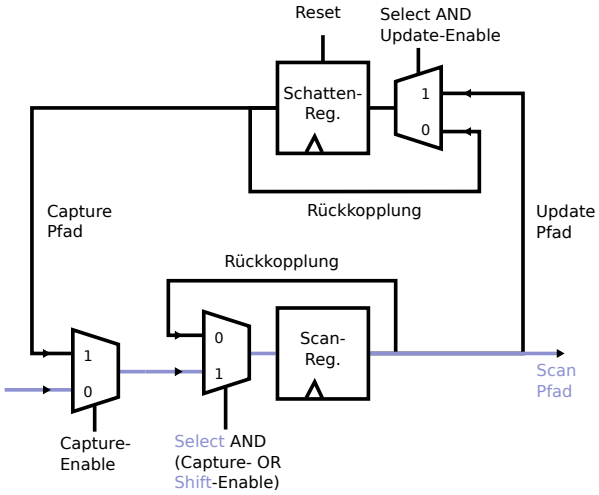

Abbildung 3.5.: Beispiel der aktiven Pfade und Signale (blau) in einem Scan-Segment während der Scan-Phase. Der aktive Rückkopplungsdatenpfad des Schattenregisters wurde nicht explizit hervorgehoben.

3.2. Modellierung auf Transaktionsebene

Die taktgenaue Modellierung eines Rekonfigurierbaren Scan-Netzwerks führt zu einem hohen Aufwand aufgrund der sequentiellen Tiefe dieser Schaltungen. Deshalb werden Rekonfigurierbare Scan-Netzwerk hier auf Transaktionsebene modelliert, so dass einzelne

getaktete Schiebeoperationen nicht notwendig sind und eine Effizienzsteigerung bei der Rechenzeit erreicht wird.

In dieser Modellierung entspricht eine Transaktion einem vollständigen CSU-Zyklus, wie er in Kapitel 3.1.2 beschrieben wurde. In einer Transaktion wird der zugreifbare Zustand eines Rekonfigurierbaren Scan-Netzwerkes, das heißt alle Scan-Elemente auf dem aktiven Scan-Pfad, neu geschrieben. Damit überführt eine Transaktion die Konfiguration eines Rekonfigurierbaren Scan-Netzwerks in eine neue Konfiguration, welche sich nur auf dem aktiven Scan-Pfad unterscheidet. Dabei ändert sich die Ausprägung des aktiven Scan-Pfads entsprechend der neuen Konfiguration.

Eine CSU-Operation entspricht dabei nur einem Simulationsschritt im Modell auf Transaktionsebene, da sowohl die Capture-, Shift- sowie die Update-Operationen nicht explizit modelliert werden. Hingegen hängt die Anzahl der Schritte im taktgenauen Modell von der Länge des aktiven Scan-Pfads ab und ist damit variabel, was die Modellierung als Instanz des Erfüllbarkeitsproblems erschwert. Die Zahl der Zyklen zur Rekonfiguration beträgt $n + 2$ Schritte, wenn der aktive Scan-Pfad n Scan-Segmente aufweist. Die mögliche Zustandsmenge kann daher auf Transaktionsebene wesentlich kleiner und damit effizienter in Rechenzeit und Speicherplatzbedarf simuliert werden.

3.2.1. Modellierung von Scan-Segmenten

Auf Transaktionsebene werden Scan- und Schattenregister als ein speicherndes Element in Form eines Scan-Segments betrachtet. Es wird angenommen, dass in einer Transaktion aktive Scan-Segmente gelesen und beschrieben werden können. Hierdurch kann der Zugriff auf den aktiven Scan-Pfad als ein einzelner Lese- und Schreibzugriff betrachtet werden.

3.2.2. Modellierung der Scan-Pfadaktivierung

Zur Erzeugung von Zugriffsmustern in Rekonfigurierbaren Scan-Netzwerken ist es notwendig zu gewährleisten, dass ein vollständig konsistenter und aktiver Scan-Pfad existiert. Dies ergibt sich aus der Konfiguration und der Kontrolllogik, welche auch auf Transaktionsebene als Boole'sche Funktion modelliert ist.

Wie auf der Register-Transfer-Ebene, besitzen die Scan-Segmente auf Transaktionsebene eine Aktivierungsbedingung, welche das Speichern neuer Werte ermöglicht. Diese ist aktiv oder wahr, sofern sich das Scan-Segment auf dem aktiven Scan-Pfad befindet. Zusätzlich gilt, dass wenn ein Scan-Segment aktiv ist, dann müssen auch genau ein eingehendes Scan-Pfad-Segment und genau ein ausgehendes Scan-Pfad-Segment aktiviert sein. Andernfalls würde kein oder mehr als ein aktiver Scan-Pfad ausgeprägt werden, was verboten ist. Abbildung 3.6 zeigt die entsprechenden Aktivierungsvariablen an Scan-Pfad-Segmenten in einem Rekonfigurierbaren Scan-Netzwerk.

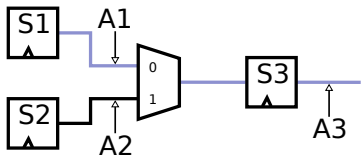

Abbildung 3.6.: Beispiel der Aktivierungsvariablen auf Transaktionsebene
Die Aktivierungsbedingungen A1 und A3 der Scan-Pfad-Segmente auf dem aktiven Scan-Pfad (blau) müssen erfüllt sein. Die Aktivierungsbedingung A2 des Scan-Segments auf einem inaktiven Scan-Pfad-Segment muss hingegen inaktiv sein.

Die in Kapitel 3.1.1 beschriebenen Aktivierungssignale werden daher als Boole'sche Formel modelliert. Gleichung 3.1 beschreibt die Aktivierung eines Scan-Segments. Ein Scan-Segment S ist aktiv, wenn die Boole'sche Aktivierungsbedingung A erfüllt ist und ein Vorgänger V als auch ein Nachfolger N aktiv sind.

$$\begin{aligned} &Aktiv(S) \Leftrightarrow A \text{ ist erfüllt} \wedge \\ &\quad \exists V \in \text{Vorgänger}(S) : Aktiv(V) \wedge \\ &\quad \exists N \in \text{Nachfolger}(S) : Aktiv(S) \end{aligned} \tag{3.1}$$

3.3. Klassifizierung der Fehlerwirkung und Testbarkeit

In diesem Abschnitt werden Fehlerstellen für das einfache Haftfehlermodell klassifiziert und diese Fehlerklassen mit deren Fehlerwirkung beschrieben.

Scan-Datenpfad Liegt ein Fehler auf dem Datenpfad eines Scan-Pfads, so ist der Fehler unmittelbar aktiviert, sobald der Scan-Pfad in einer Konfiguration aktiv ist. Mittels eines Flush-Tests ist die Fehlerwirkung observierbar, da der Scan-Pfad ab einer bestimmten Stelle nur noch konstante Werte propagiert.

Diese Fehlerklasse kann dazu führen, dass die Schaltung nicht mehr kontrolliert werden kann, nachdem ein Fehler aktiviert wurde. Dies ist aber von geringer Bedeutung für den Test, da der Fehler erfolgreich nachgewiesen wurde. Jedoch wird die Diagnose in solch einem Fall aufgrund der geringen Sichtbarkeit erschwert.

Scan-Steuerung Befindet sich ein Fehler unmittelbar im Eingangskegel eines Scan-Datenpfads, hat nur Auswirkungen auf diesen und beeinflusst dessen Aktivierung, dann führt dies zu zwei unterschiedliche Auswirkungen.

Der Fehler kann einen eigentlich inaktiven Pfad sensibilisieren und so die Werte in einem Scan-Register verändern, was zu Datenverlust führen kann. Um dies zu detektieren müsste im Scan-Register am Ausgangspunkt des fälschlich aktiven Scan-Pfads der komplementäre Wert geschrieben werden. Das Scan-Pfad-Segment müsste daraufhin deaktiviert werden und später wieder ausgelesen werden.

Andernfalls kann ein aktiver Scan-Pfad deaktiviert werden, falls der Fehler einen kontrollierenden Wert an einem Gatter propagiert, so dass der Pfad gebrochen wird. Dieser Fall ist äquivalent zum gebrochenen Scan-Datenpfad.

Scan-Adressierung Wirkt sich ein Fehler auf mehrere unterschiedliche Scan-Pfade aus und beeinflusst deren gemeinsame, wenn auch komplementäre Aktivierung, so kann dies zur Vertauschung zweier Scan-Pfade führen. Bei unterschiedlichen Scan-Pfad-Längen kann dies durch den Flush-Test erkannt werden, bereitet aber bei gleicher Scan-Pfad-Länge Probleme.

In diesem Fall kann es sein, dass beide Scan-Pfade die gleichen Datenwerte enthalten und so nicht direkt voneinander unterschieden werden. Befinden sich Konfigurationsregister auf den betroffenen Scan-Segmenten, welche asymmetrische Scan-Pfade treiben, dann können diese genutzt werden um den eigentlich Fehler in der Adressierung festzustellen. Dazu müssten entsprechende Testmuster für die kontrollierten Scan-Pfad-Segmente erzeugt werden. Sind jedoch nur Scan-Pfade betroffen, die keine Kontrollregister ansteuern, so können eventuell angeschlossene Instrumente oder Systemlogik dazu genutzt werden, um die Scan-Pfad-Segmente voneinander zu unterscheiden.

Capture-Datenpfad Liegt die Fehlerstelle auf dem Capture-Datenpfad, so können keine Datenwerte vom Schattenregister zum Scan-Register übertragen werden, sondern nur ein statischer Wert. Dies lässt sich einfach testen, indem der komplementäre Wert in einer CSU-Operation O_1 in das Scan-Register geschrieben wird und dieser vom fehlerhaften Wert in der nächsten CSU-Operation O_2 überschrieben wird. Die Scan-Phase von O_2 macht den Fehler anschließend an den Primärausgängen sichtbar.

Capture-Steuerung Ein Fehler in der Kontrolllogik des Capture-Pfads führt dazu, dass entweder der Pfad statisch desensibilisiert oder statisch aktiviert ist. Der erste Fall entspricht einem Fehler auf dem Capture-Datenpfad selbst. Beim statisch aktivierten Fall propagiert der Wert aus dem Schattenregister immer auf den Scan-Pfad zum Scan-Register. Dieser Fall ist testbar, wenn der kontrollierende Wert der Konvergenz der beiden Datenpfade im Schattenregister gespeichert ist und der nicht-kontrollierende Wert auf dem Scan-Pfad propagiert.

Update-Datenpfad Liegt der Fehler auf dem Update-Pfad selbst, so wird ein Schreiben eines abweichenden Wertes verhindert und somit das Erreichen einer bestimmten gewünschten Konfiguration verhindert. Dies lässt sich testen, indem der komplementäre Wert geschrieben und in einer folgenden CSU-Operation ausgelesen wird.

Die sequentielle Fehlerwirkung wird dadurch sichtbar, dass zum Beispiel ein Scan-Pfad nicht aktiviert wird, weil der entsprechende Wert der Konfiguration nicht gesetzt werden kann. Der Fehler kann daher durch einen abweichend von dem geforderten gewählten Scan-Pfad erkannt werden.

Update-Steuerung Ein Fehler in der Ansteuerung eines Update-Pfads kann, wie bei den bisher betrachteten Kontrolllogiken auch, den entsprechenden Datenpfad statisch aktivieren oder deaktivieren. Bei Deaktivierung lässt sich der fehlerhaft gespeicherte Wert

durch eine Update-Operation, welche den komplementären Wert schreibt, aktivieren. Eine Capture-Operation überführt den Wert in das Scan-Register, von wo aus der Schiebebetrieb den Fehler ausliest.

Die Fehlerwirkung eines statisch aktiven Update-Pfads ist sehr interessant, da diese von der Aktivität des Scan-Pfads abhängt. Dabei kann sich der aktive Scan-Pfad ständig verändern und neu ausprägen.

Rückkopplungsdatenpfad Befindet sich der Fehler auf dem Rückkopplungspfad eines Registers, so ist die Speicherfähigkeit des Registers reduziert und beschränkt sich auf einen einzelnen Wert, falls es sich um den nicht-kontrollierenden Wert der Konvergenz der Datenpfade handelt. Das Register kann, wenn es beschrieben wird, den Wert nur für einen Takt halten, sofern es nicht erneut beschrieben wird.

Scan-Register können nur getestet werden, indem das Select-Signal während des Schiebebetriebs deaktiviert wird. Andernfalls wird in der Capture-Phase der fehlerhafte Wert überschrieben und während dem Schiebebetrieb werden ständig neue Daten gespeichert, so, dass kein Wert aufgrund des Fehlers verloren gehen würde.

Fehlerhaften Daten können bei Schattenregistern nach einer Capture-Operation ausgelesen werden. Jedoch kann dies auch zu einer Veränderung des aktiven Scan-Pfads führen, wodurch eventuell ein falscher Pfad ausgelesen würde.

Besteht der Fehler aus dem kontrollierenden Wert der Konvergenz, so propagiert kein anderer Wert zum Register. Daher ist das Register nicht funktional und kann durch Lese- und Schreibzugriffe einfach getestet werden.

Rückkopplungssteuerung Beeinflusst der Fehler die Kontrolllogik des Rückkopplungspfads eines Scan- als auch Schattenregisters, so dass der Datenpfad statisch desensibilisiert ist, dann propagiert über den Rückkopplungsdatenpfad nur noch ein konstanter Wert, und es handelt sich um die entsprechende statische Fehlerklasse eines Rückkopplungsdatenpfad.

Wird hingegen ein Rückkopplungsdatenpfad statisch sensibilisiert, so tritt an der Konvergenz der Datenpfade immer der kontrollierende Wert auf, sobald er einmal propagierte. Das Register speichert den kontrollierenden Wert und propagiert über den Rückkopplungsdatenpfad wieder an die Konvergenz. Da es sich um den kontrollierenden Wert handelt, wird er wieder im Register gespeichert. Da die Fehlerwirkung statisch auftritt, sobald sie einmal aktiviert ist, ist diese Fehlerklasse einfach zu testen. Die Fehlerwirkung lässt sich durch einfachen Zugriff auf das Scan-Segment sichtbar machen.

Taktverteilung Fehler, welche die Taktverteilung beeinflussen, führen dazu, dass ein oder mehrere Register nicht funktional sind und keine Daten speichern oder weiterleiten können. Diese Fehlerklasse lässt sich sehr gut testen. Für Scan-Register ist die Fehlerwirkung äquivalent zu einem Fehler direkt auf dem Scan-Datenpfad und für Schattenregister äquivalent zum Capture-Datenpfad.

Reset-Steuerung Betrifft ein Fehler die Reset-Steuerung in der Art, dass ein oder mehrere Register nicht in ihren Initialzustand zurückgesetzt werden können, dann enthalten diese möglicherweise einen abweichenden Wert. Dieser Fehler wird nur aktiviert, falls vor dem Reset in den betroffenen Registern der komplementäre Wert gespeichert ist.

Eine Folge der fehlerhaften Konfiguration kann darin bestehen, dass ein inkorrekter Pfad nach dem Reset als aktiver Scan-Pfad ausgeprägt wird. Befindet sich ein zugehöriges Scan-Segment auf dem aktiven Scan-Pfad, so wird der Fehler unmittelbar nach der ersten CSU-Operation als abweichender Wert sichtbar.

Ebenfalls ist es möglich, dass eine ungültige Konfiguration durch den fehlerhaften Reset erreicht werden kann. Dies führt zu einem nicht funktionsfähigen oder mehreren aktiven Scan-Pfaden, welche einfach testbar sein sollten. Jedoch ist es dafür notwendig, dass das betroffene Modul aktiviert wird.

Scan-Register besitzen keine Steuerung für den Reset, denn durch die erste Capture-Operation werden gültige Daten aus den Schattenregistern in die Scan-Register geschrieben. Würden diese zurücksetzbar sein, so wären die Fehler aus diesem Grund an Scan-Registern nicht detektierbar.

Eine weitere Möglichkeit der Fehlerwirkung besteht im ständig aktiven Reset, wodurch die betroffenen Register nicht mehr funktional sind. Dies ist ein äquivalenter Fall zu einem Fehler in der Taktverteilung.

3.4. Funktionale Fehlermodelle auf Transaktionsebene

Das Einfache-Haftfehler-Modell eignet sich gut für Fehler auf Gatterebene, ist jedoch auf der Transaktionsebene, wie sie bei der CSU-Zyklen-Abstraktion betrachtet wird, schwierig anwendbar, da einige strukturelle beziehungsweise funktionale Teile der Schaltung nicht betrachtet werden. Dies umfasst im Wesentlichen die explizite Modellierung der Capture- und Update-Pfade, als auch die Rückkopplungsdatenpfade der speichernden Elemente. Daher sollen in diesem Kapitel drei mögliche funktionale Fehlermodelle beschrieben werden.

Gebrochener Scan-Pfad Besteht die Fehlerwirkung in der Deaktivierung eines Scan-Pfad-Segments des aktiven Scan-Pfads, so kann ab der Fehlerstelle nur noch der konstante Wert der Fehlerwirkung ausgelesen beziehungsweise in nachfolgende Segmente geschrieben werden. Ein solcher Fehler kann Zugriffe auf ein Rekonfigurierbares Scan-Netzwerk so beeinflussen, dass dieses bis zu einem Reset unbrauchbar ist, falls der Fehler nur noch Konfigurationen zulässt, welche aufgrund der Fehlerwirkung keinen Scan-Pfad aktivieren. Dies kann relevant für eine spätere Diagnose sein.

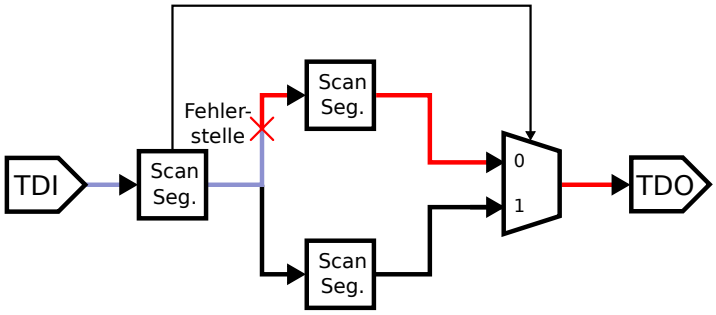

Abbildung 3.7.: Beispiel eines gebrochenen Scan-Pfads (rot)
Der Scan-Pfad ist an der Fehlerstelle unterbrochen und Werte propagieren nicht mehr weiter. Stattdessen wird nur noch ein konstanter Wert propagiert.

Falscher Scan-Pfad Beeinflusst die Fehlerwirkung die Ansteuerung eines Scan-Multiplexers, so besteht die Möglichkeit, dass statt dem eigentlich konfigurierten Scan-Pfad ein anderer Scan-Pfad aktiviert wird. Auf Transaktionsebene wirkt sich dies als Auswahl eines anderen Pfads durch den Graphen aus. Dabei sind die Gültigkeitsbedingungen immer noch erfüllt, das heißt es wird nur ein aktiver Scan-Pfad ausgeprägt.

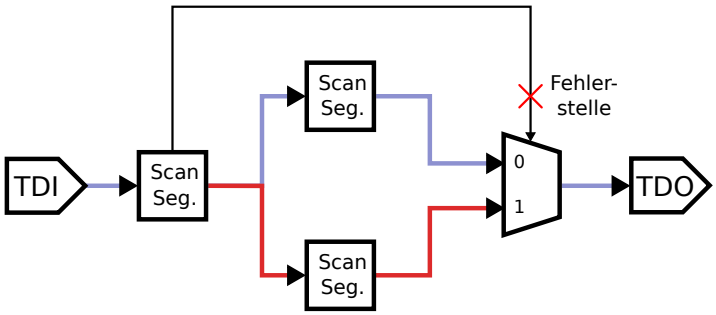

Abbildung 3.8.: Beispiel eines fälschlich aktiven Scan-Pfads (rot)
Der eigentlich ausgewählte blaue Scan-Pfad ist aufgrund des Fehlers nicht aktiv. Durch die Fehlerwirkung wurde stattdessen der rote Scan-Pfad aktiviert.

Instabiler Scan-Pfad Angenommen die Kontrolllogik eines Update-Pfads auf Gatterebene sei so betroffen, dass ein Schattenregister in jedem Takt den Wert aus dem Scan-Register übernehme. Dann basiert die Ausprägung eines aktiven Scan-Pfads auf den propagierenden Datenwerten auf dem Scan-Pfad und nicht mehr auf den durch CSU kontrollierten Datenwerten eines Konfigurationsregisters. Dadurch verändert sich der aktive Scan-Pfad entsprechend der Aktivität auf dem entsprechenden Scan-Pfad-Segment. Diese Fehlerannahme führt zu einem pseudo-dynamischen Verhalten auf Transaktionsebene, so dass ab der Fehlerstelle lediglich pseudo-zufällige Daten gelesen und geschrieben werden können.

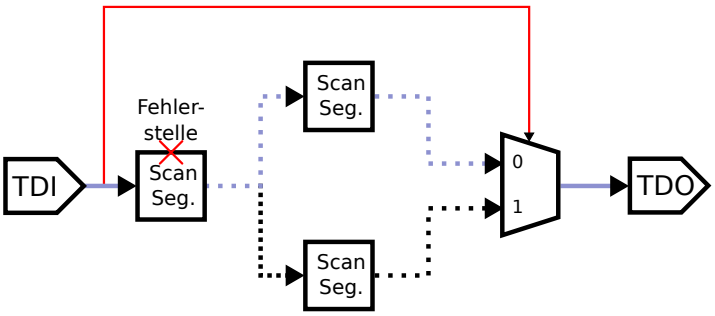

Abbildung 3.9.: Beispiel eines instabilen aktiven Scan-Pfads (schraffiert)
Die Kontrolllogik des Scan-Multiplexers wird nicht mehr aus einem stabilen Konfigurationsregister gesteuert, daher prägt sich der aktive Scan-Pfad basierend auf der Aktivität des Scan-Pfads aus.

4. Testalgorithmen für Rekonfigurierbare Scan-Netzwerke

In diesem Kapitel wird ein Überblick über unterschiedliche Testalgorithmen für Rekonfigurierbare Scan-Netzwerke geben. Zuerst wird die Möglichkeit eines pseudo-zufälligen Tests diskutiert. Darauf folgen zwei funktionale Testheuristiken, welche zum einen aus einem einmaligen Zugriff auf alle Scan-Segmente besteht und zum anderen die Funktionalität der Scan-Segmente betrachtet. Abschließend werden weitere Testalgorithmen vorgeschlagen, welche insbesondere in Hinblick auf die Bedürfnisse einer schnellen Testmustererzeugung für Rekonfigurierbare Scan-Netzwerke entwickelt worden sind.

Flush-Test

Ziel des Flush-Tests ist festzustellen, ob der Scan-Pfad Daten unverfälscht schiebt. Dieser findet in den folgenden Testalgorithmen Anwendung.

Bei einfachen Haftfehlern ist es ausreichend, dass beide Logikwerte 0 und 1 durch den Scan-Pfad propagieren. Dabei können kombinatorische Fehler aktiviert und deren Fehlerwirkung propagiert werden, welche auf dem aktiven Scan-Pfad liegen oder dessen Kontrolllogik beeinflussen.

Zur Validierung, ob ein funktionsfähiger Scan-Pfad ausgeprägt wurde, wird vor jeder Scan-Phase die Sequenz 01 in den Scan-Pfad geschrieben. Danach werden die Werte der nächsten Konfiguration geschrieben, welche genau der Länge des fehlerfreien aktiven Scan-Pfads entsprechen. Wenn die letzten beiden Werte der Konfiguration in den Scan-Pfad geschrieben werden, wird die Sequenz 01 am Ausgang des Scan-Pfads sichtbar. Wird eine andere Wertefolge gelesen, so wurde ein Fehler detektiert.

4.1. Pseudo-zufällige Testmustererzeugung

In kombinatorischen Schaltungen können pseudo-zufällig erzeugte Testmuster bereits eine hohe Fehlerabdeckung gewährleisten [BM82]. Diese Methode hat den Vorteil, dass Testmuster sehr günstig erzeugt werden können. Allerdings eignet sich diese Methode nicht für Rekonfigurierbare Scan-Netzwerke, da ungültige Zustände erreicht werden können, die keine weiteren Zugriffe auf Scan-Segmente mehr erlauben. Auf der anderen Seite ist es

unwahrscheinlich tiefe Ebenen zu erreichen. Die Wahrscheinlichkeit des Zugriffs auf ein Teilmodul einer Hierarchie soll im weiteren Verlauf erläutert werden.

Angenommen es existieren in einem Rekonfigurierbaren Scan-Netzwerk Hierarchien so, dass die Aktivierung eines Teilmoduls m von t unabhängig voneinander gesetzten Konfigurationsbits abhängt und nur in einer Konfiguration aktiviert wird. Für jedes Konfigurationsbit beträgt die Wahrscheinlichkeit, dass entsprechende Datum zufällig zu erzeugen, 50 Prozent. Dann liegt die Wahrscheinlichkeit, dass m aktiviert wird, bei:

$$P(m_{aktiviert}) \leq 2^{-t} \tag{4.1}$$

Schon für eine geringe Anzahl von Bits, zum Beispiel $t = 5$, beträgt die Wahrscheinlichkeit, dass das entsprechende Modul aktiviert wird und damit auf dem aktiven Scan-Pfad liegt, weniger als 4 Prozent.

Sind die Konfigurationsregister hierarchisch so angeordnet, dass der Zugriff auf Konfigurationsregister $i \leq t$ nur bei aktiviertem Register $i - 1$ erfolgen kann, so ergibt sich ein Fall mit noch geringer Aktivierungswahrscheinlichkeit als in Gleichung 4.1 beschrieben. Eine solche Struktur wird in Abbildung 4.1 skizziert.

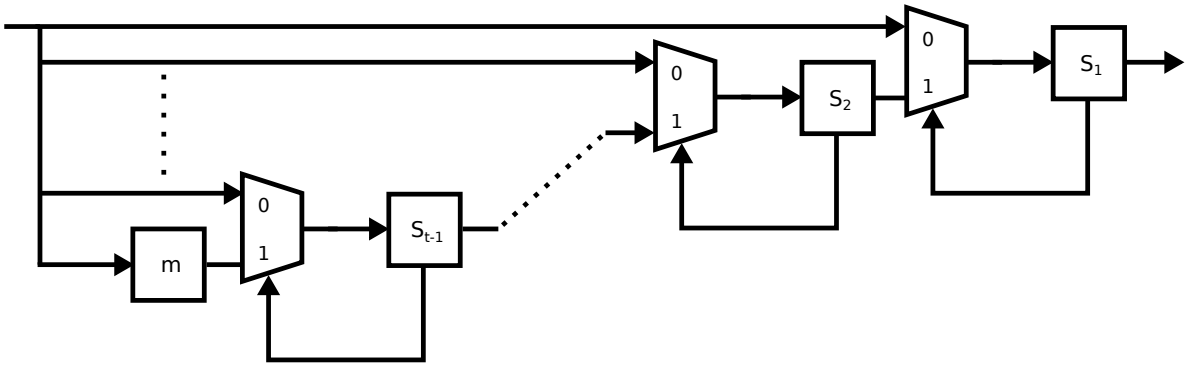

Abbildung 4.1.: Beispiel einer hierarchisch bedingten Zugriffsstruktur

Denn für jede Aktivierung eines Registers i müssen alle Register $< i$ zugreifbar sein. Damit diese zugreifbar bleiben, muss das entsprechende Datum für einen weiteren Zugriff erhalten bleiben, weil alle Daten des aktiven Scan-Pfads wieder geschrieben werden. Ansonsten würde die Hierarchieebene wieder geschlossen werden. Gleichung 4.2 beschreibt die Zugriffswahrscheinlichkeit formal:

$$\begin{aligned} P(m_{zugreifbar}) &\leq 2^{-(t-1)} .. * 2^{-1} && = \prod_{n=1}^{t-1} 2^{-n} \\ P(m_{aktiviert} \cap m_{zugreifbar}) &= P(m_{aktiviert}) * P(m_{zugreifbar}) \\ &\leq 2^{-t} * 2^{-(t-1)} * \ldots * 2^{-1} && = \prod_{n=1}^{t} 2^{-n} \end{aligned} \tag{4.2}$$

Dieses Produkt strebt sehr schnell gegen den Null so, dass die Wahrscheinlichkeit eines einmaligen zufälligen Zugriffs auf ein Teilmodul einer Tiefe von 3 schon bei ca. 1 Prozent liegt. Aus diesem Grund ist ein reiner Zufallstest für Rekonfigurierbare Scan-Netzwerke nicht effizient realisierbar. Tabelle 4.1 zeigt die Wahrscheinlichkeit für Module bis zur Tiefe 5 für die beiden Gleichungen 4.1 und 4.2.

	t	0	1	2	3	4	5
4.1	$P(m_{aktiviert})$	100%	50%	25%	12,5%	6,25%	3,13%
4.2	$P(m_{aktiviert} \cap m_{zugreifbar})$	100%	50%	12,5%	1,56%	0,10%	∼0,00%

Tabelle 4.1.: Wahrscheinlichkeit eines zufälligen Zugriffs auf ein Modul m der Tiefe t

Ein weiteres Problem besteht in der Ermittlung der aktiven Scan-Pfad-Länge. Basierend darauf, welche zufälligen Muster geschrieben wurden, verändert sich die Konfiguration und damit die Scan-Pfad-Länge. Damit ein fehlerhaft gespeichertes Datum auf dem aktiven Scan-Pfad nicht durch eine weitere Capture-Operation überschrieben wird, müssen alle Daten auf dem aktiven Scan-Pfad ausgelesen werden. Es ist also notwendig den Scan-Pfad vollständig auszulesen.

Eine Möglichkeit besteht darin, dass immer so viele Bits gelesen werden wie der längst mögliche aktive Scan-Pfad. Dies ist jedoch ineffizient, das heißt sollen keine Muster unnötig geschoben werden, muss rechtzeitig die Scan-Phase beendet werden.

Eine andere Herangehensweise besteht in der Erzeugung zufälliger Lese- und Schreibzugriffe auf die Scan-Segmente. Das später in Kapitel 5.2 beschriebene Werkzeug eda1687 ermöglicht die Erzeugung von Zugriffsmustern für Lese- und Schreibzugriffe. Da dieses einen vollständigen neuen Zustand auf den aktiven Scan-Pfad schreibt, wird auch automatisch der Zustand des aktiven Scan-Pfads ausgelesen.

4.2. Funktionale Testheuristiken

In diesem Abschnitt werden zwei funktionale Testheuristiken vorgestellt. Diese haben den Vorteil, dass sie einfach zu implementieren sind und eine kompakte Menge von Testmustern erzeugen. Beide Algorithmen können sowohl auf Gatter- als auch auf Register-Transfer-Ebene eingesetzt werden. Die Erzeugung auf Register-Transfer-Ebene ist der Gatterebene dabei vorzuziehen, da aufgrund der höheren Abstraktion weniger Elemente zu betrachten sind und somit ist diese Vorgehensweise in Rechenzeit als auch Speicherverbrauch effizienter.

4.2.1. Zugriff auf alle Scan-Segmente

Durch den simplen Zugriff auf jedes Scan-Segment können entsprechend einfach Testmuster erzeugt werden. Dadurch werden alle Scan-Register einmal auf dem aktiven Scan-Pfad kontrollier- und observierbar. Es ist zu erwarten, dass diese Heuristik ein gutes Verhältnis von Fehlerabdeckung zu Anzahl der Muster besitzt, da durch eine minimale Anzahl von Zugriffsmustern sehr viele Gatter sensibilisiert werden. Ebenfalls ist anzunehmen, dass es sich hierbei um die minimale Anzahl von Testmustern handelt, denn um einen Fehler zu testen, muss dieser observierbar sein. Für ein Rekonfigurierbares Scan-Netzwerk muss dazu der Fehler auf den aktiven Scan-Pfad einwirken.

Für jeden CSU-Zyklus wird zusätzlich ein Flush-Test durchgeführt, so dass die Wertesequenz 01 auf dem Scan-Pfad propagiert und mögliche Haftfehler detektiert. Damit werden alle Haftfehler auf dem Scan-Pfad detektiert.

4.2.2. Lese-/Schreibzugriff auf alle Scan-Segmente

Ein erweitertes, funktionales Verfahren zur Erzeugung von Testmustern stellt die Überprüfung des Lese- und Schreibzugriffs aller Scan-Segmente dar. Dieses überprüft, ob alle Scan-Register durch mindestens einen Scan-Pfad erreichbar sind und ob in der getesteten Konfiguration sowohl Capture- als auch Update-Pfad korrekt arbeiten.

Angenommen wird, dass sich auf diese Weise ein großer Teil der Fehler detektieren lässt. Insbesondere sind statische Fehler in der Taktverteilung oder Reset-Ansteuerung einfach zu entdecken, da ein entsprechend betroffenes Register nicht mehr funktional ist. Algorithmus 4.1 skizziert das Vorgehen.

Algorithmus 4.1 Schreibende funktionale Testheuristik

```
1 Für jedes Scan-Segment s:
2   Schreibe 1 in Scan-Segment s
3   Lese Scan-Segment s und Schreibe 0 in Scan-Segment s
4   Lese Scan-Segment s
```

4.3. Testmustererzeugung auf Transaktionsebene

Die deterministische Erzeugung der Testmuster setzt sich aus mehreren Schritten zusammen. Zuerst werden entsprechende Zugriffsmuster erzeugt, um eine bestimmte Konfiguration des Rekonfigurierbaren Scan-Netzwerks zu erreichen, in welcher der zu untersuchende Fehler aktiviert werden kann. Anschließend wird eine Flush-Sequenz geschrieben, so dass die Fehlerwirkung am Ausgang des Scan-Pfades sichtbar wird.

Folgenden Kapiteln führen Verfahren zur Detektion der Fehlerwirkung ein, welche genutzt werden, um festzustellen, in welchen Zuständen ein Fehler aktiviert ist.

4.3.1. Detektion durch Brechung des Scan-Pfads

Die Ausgangsidee zur Detektion von Fehlern in Rekonfigurierbaren Scan-Netzwerken basiert auf dem Brechen des Scan-Pfads. Dazu wird ein Zugriffsmuster gesucht, welches in der fehlerhaften Schaltung den aktiven Scan-Pfad durch Aktivierung des Fehlers unterbricht, da aufgrund der Fehlerwirkung lediglich konstante Werte propagiert werden. Es gibt zwei Gruppen von Fehlern, welche eine solche Wirkung verursachen können.

Die erste Gruppe besteht aus Fehlern, welche direkt auf dem Scan-Pfad liegen und diesen dadurch blockieren. Da ein solcher Fehler durch Aktivierung des entsprechenden Scan-Pfads observierbar wird, ist es ausreichend auf den Scan-Pfad zuzugreifen und mittels einer Flush-Sequenz den Fehler zu aktivieren und zu detektieren. Abbildung 4.2 zeigt einen solchen Fehler.

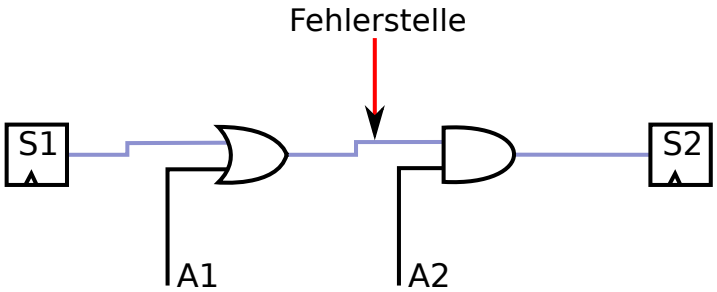

Abbildung 4.2.: Beispiel eines direkt gebrochenen Scan-Pfads
Der Scan-Pfad (blau) zwischen den Scan-Registern S1 und S2 wird durch die Fehlerwirkung eines Haftfehlers gebrochen.

Bei der anderen Gruppe befindet sich der Fehler nicht auf einem Scan-Pfad, sondern in der Kontrolllogik und steuert ein oder mehrere Scan-Pfad-Segmente, wie es in Abbildung 4.3 dargestellt wird. Zur Detektion dieses Fehlers im Erfüllbarkeitsproblem, werden die Pfadaktivierungsklauseln im fehlerfreien und fehlerbehafteten Fall verglichen. Ist ein Scan-Pfad-Segment im fehlerfreien Fall aktiv, das heißt die Pfadaktivierungklauseln evaluieren zu wahr, aber im fehlerbehafteten Fall deaktiviert, so ist dieser Pfad ein potentieller Kandidat. Zusätzlich müssen alle anderen Scan-Pfad-Segmente ihre Aktivierung beibehalten. Andernfalls könnte ein anderes Scan-Pfad-Segment aktiviert werden und einen funktionsfähigen aktiven Scan-Pfad ausbilden.

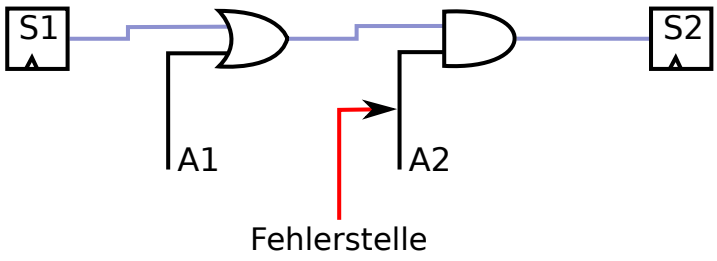

Abbildung 4.3.: Beispiel eines durch fehlerhafte Kontrolllogik gebrochenen Scan-Pfads
Der Scan-Pfad (blau) wird durch einen Haftfehler an 1 am Kontrollsignal A2 gebrochen.

Gleichung 4.3 drückt die Detektionsbedingung formal aus. Dabei bezeichnet SPS die Menge aller im Rekonfigurierbaren Scan-Netzwerk vorhandenen Scan-Pfad-Segmente. Für jedes Scan-Pfad-Segment $p \in SPS$ existieren zwei Boole'sche Funktionen. Im fehlerfreien Fall als p_{gut} und im fehlerbehafteten als p_{saf} bezeichnet. Diese sind wahr genau dann, wenn $p_{gut/saf}$ auf dem aktiven Scan-Pfad liegt.

(4.3) $\exists p \in SPS, \forall q \in \text{SPS} : (p_{gut} \wedge \neg p_{saf}) \wedge (q \neq p) \wedge (q_{gut} = q_{saf})$

4.3.2. Detektion durch Deaktivierung des Scan-Pfads

Eine Alternative zum Brechen eines Scan-Pfads besteht darin, dass lediglich ein im fehlerfreien Fall aktiver Scan-Pfad durch den Fehler deaktiviert wird, aber andere Pfade dafür auch aktiviert werden dürfen. Dies entspricht dem Brechen von Scan-Pfaden, aber ohne die Bedingung, dass kein anderer Pfad aktiviert werden darf. Somit könnte diese Bedingung auch als optimistische Detektion bezeichnet werden. Im Gegensatz zur ursprünglichen Idee der Brechung des Scan-Pfads, kann ein anderer Scan-Pfad aktiviert werden, so dass der Fehler nicht durch die Flush-Sequenz robust detektierbar ist.

Die Gleichung 4.4 basiert auf der Gleichung 4.3 und ist nur entsprechend um den zweiten Teil gekürzt. Der Vorteil dieses Verfahrens besteht in der geringen notwendigen Zahl an Bedingungen.

(4.4) $\exists p \in SPS : p_{gut} \wedge \neg p_{saf}$

4.3.3. Detektion durch Änderung des Scan-Pfad-Präfixes

Ein weiteres optimistisches Verfahren wird vorgestellt, dessen Detektionsbedingung auf der Änderung der eingehenden Datenpfade an einem Scan-Segment besteht.

Der Ansatz basiert darauf, dass ein eingehender Pfad eines aktiven Scan-Segments eine Änderung aufweist. Dies kann sowohl die Deaktivierung eines aktiven Scan-Pfads als auch

die Aktivierung eines zweiten, aber nicht aktiven Datenpfads, wie zum Beispiel Capture-, Scan- oder der Rückkopplungsdatenpfad, sein.

Im Wesentlichen wird hierbei die Scan-Pfad-Deaktivierung mit einer möglichen Scan-Pfad-Aktivierung kombiniert. Wird das aktive Scan-Pfad-Segment deaktiviert, so ist der Scan-Pfad gebrochen, sofern kein weiterer Pfad aktiviert wird. Aktiviert der Fehler einen weiteren Scan-Pfad, so kann der Fehler noch detektiert werden, falls dieser nicht gleichlang, das heißt nicht symmetrisch, ist. Es kann dabei auch der Capture- als auch der Rückkopplungsdatenpfad aktiviert werden, welche durch eine mögliche Propagierung eines kontrollierenden Werts den Scan-Pfad bricht.

Die Detektionsbedingung ist in Formel 4.5 gegeben. Darin findet sich der Deaktivierungsteil der Gleichung 4.4 wieder. Zusätzlich wird die Menge aller im Rekonfigurierbaren Scan-Netzwerk vorhandenen Scan-Segmente S eingeführt, sowie die Menge aller Eingangspfade s_{in} eines Scan-Segments s:

$$\begin{aligned} &\exists p \in SPS : (p_{gut} \wedge \neg p_{saf}) \vee \\ &\exists s \in S, \exists p \in s_{in}, \exists q \in s_{in} : p_{gut} \wedge (q \neq p) \wedge (q_{gut} \neq q_{saf}) \end{aligned} \tag{4.5}$$

4.3.4. Detektion durch Änderung der Scan-Pfad-Länge

Eine pessimistische Methode besteht in der Beobachtung der Länge des aktiven Scan-Pfads. Unterscheidet sich die Länge des aktiven Scan-Pfads im fehlerbehafteten vom fehlerfreien Fall, so kann dies entsprechend mit einer Flush-Sequenz detektiert werden.

Der Nachteil dieser Methode besteht in der Modellierung der Detektionsbedingungen, nämlich dem Zählen der aktiven Scan-Pfad-Längen. Denn dieses muss als Boole'sche Funktion dargestellt beschrieben werden. Die Variablen einer solchen Funktionen können lediglich Boole'sche Werte annehmen und somit Wahr oder Falsch darstellen. Da nur logische und keine arithmetischen Operationen direkt unterstützt werden, müssen Binärzahlen sowie einfache Additions- und Vergleichsoperationen in Klauseln explizit modelliert werden.

Alternativ lässt sich die Detektionsbedingung auch als Pseudo-Boole'sches-Problem darstellen. Dabei werden Boole'sche Variablen in arithmetischen Ausdrücken genutzt. So können die Variablen addiert oder multipliziert werden und die Länge der Scan-Pfad-Segmente im fehlerfreien $|s_{gut}|$ und im fehlerbehafteten $|s_{saf}|$ Fall dargestellt werden. Die Detektionsbedingung besteht dann aus der Differenz und dem Vergleich beider Variablen:

$$|s_{gut}| - |s_{saf}| \neq 0 \tag{4.6}$$

Pseudo-Boole'sche-Instanzen lassen sich als Instanz des aussagenlogischen Erfüllbarkeitsproblems ausdrücken [ES06].

Jedoch kann die Modellierung der Detektionsbedingung zur Erzeugung einer sehr komplexen Boole'schen Funktion führen. Falls die Fehlerwirkung auf viele Scan-Pfad-Segmente Einfluss hat, müssen entsprechend viele Variablen miteinander kombiniert werden.

4.3.5. Grenzen des Ansatzes

Folgend sollen bestimmte Bedingungen zusammengefasst werden, für die dieser Ansatz keine Testmuster deterministisch erzeugen kann.

Die fehlende Modellierung von Daten schließt Fehler aus, welche spezifische Werte an bestimmten Registern erwarten, um aktiviert werden zu können. Der Fall, wenn bestimmte Daten auf den Scan-Pfad propagiert werden müssen, damit ein Scan-Pfad von einem anderen unterschieden werden kann, wie beispielsweise bei symmetrischen Scan-Pfaden, kann dieser Ansatz ebenfalls nicht abdecken. Die beiden Fehlerklassen, der Fehler in der Taktverteilung und der Reset-Steuerung, sind aufgrund dessen nicht für dieses Modell geeignet.

In der derzeitigen Implementierung werden die Capture- und Update-Phasen nicht explizit modelliert, so dass die eigentliche Fehlerwirkung auf diesen Pfaden auch nicht modelliert werden kann. Es lässt sich das Modell um diese Phasen erweitern, indem die Klauseln entsprechend öfters ausgerollt und die Eingangssignale passend gesetzt werden.

Eine andere Kategorie sind Fehler, die gar nicht im Rahmen der CSU-Transaktion aktiviert werden können. Es könnte etwa ein Fehler nur bei gleichzeitig aktivem Capture- und Update-Enable-Signal aktiviert sein, was den zulässigen Zuständen des CSU-Zyklus widerspricht. Das Modell könnte um eine entsprechende Phase erweitert werden, indem keine Bedingungen auf die Eingangsvariablen gesetzt werden. Dies hängt jedoch von der Implementierung des Schaltnetzes ab.

4.4. Vergleich der Testalgorithmen

Tabelle 4.2 zeigt eine Übersicht der Testbarkeit der betrachteten Fehlerklassen durch die Heuristik zur Erzeugung von Testmustern, als auch durch die Maßnahmen des Scan-Pfad-Brechens sowie der Beobachtung anhand der Scan-Pfad-Länge.

Klasse	AH	WH	SPB	SPL
Scan-Datenpfad	+	+	+	-
Scan-Steuerung	o	o	+	-
Scan-Adressierung	o	o	-	+
Update-Datenpfade	o	+	-	-
Update-Steuerung	o	o	-	-
Capture-Datenpfade	o	+	-	-
Capture-Steuerung	o	o	-	-
Rückkopplungsdatenpfad	o	+	o	-
Rückkopplungsteuerung	o	o	o	-
Taktverteilung	o	+	-	-
Reset-Steuerung	o	o	o	-

Tabelle 4.2.: Zusammenfassung der Testbarkeit verschiedener Fehlerklassen
+ : gut, - : schlecht, o : unbestimmt
AH: Einfache Heuristik, **WH**: Schreibende Heuristik, **SPB**: Scan-Pfad-Brechen, **SPL**: Scan-Pfad-Länge

5. Implementierung

Dieses Kapitel stellt den Aufbau einer Boole'schen Formel f in konjunktiver Normalform (CNF) dar. Diese enthält ein Modell des Schaltungsgraphen, sowie die Fehlerinjektion und -detektion. Die Formel soll genau dann erfüllbar sein, wenn der Fehler detektierbar ist. Aus der Lösung lassen sich Zugriffsmuster ableiten, welche als Testmuster genutzt werden können, um den Fehler während des Tests zu detektieren.

Im ersten Kapitel wird eine Übersicht über das entwickelte Testverfahren gegeben. Danach folgt die Überführung aus der Verhaltensbeschreibung auf Register-Transfer-Ebenen zum Modell auf Transaktionsebene. Die nachfolgenden Kapitel beschäftigen sich mit der Fehlermodellierung und abschließend mit der Fehlersimulation.

5.1. Übersicht des Testverfahrens

Ausgang des hier betrachteten Verfahrens ist eine abstrakte Beschreibung des Rekonfigurierbaren Scan-Netzwerks auf Register-Transfer-Ebene. Diese Beschreibung kann zum Beispiel in der von IEEE P1687 definierten Sprache „Instrument Connectivty Language“ (ICL) vorliegen und wird in eine entsprechende Verilog-Verhaltensbeschreibung umgewandelt. Dadurch ist es möglich das Rekonfigurierbare Scan-Netzwerk mit gängigen Simulationsprogrammen zu simulieren und zu untersuchen.

Um die Testbarkeit zu untersuchen und Testmuster für das einfache Haftfehlermodell zu erzeugen, wird die Verhaltensbeschreibung in ein Modell auf Gatterebene übersetzt. Dies geschieht durch Synthese in eine Gatternetzliste. Diese Netzliste kann mit den gleichen Werkzeugen wie für die Register-Transfer-Ebene, simuliert und beispielsweise das Verhalten bei einem vorliegenden Fehler untersucht werden.

Im nächsten Schritt wird die synthetisierte Gatternetzliste eingelesen und ein internes Graphenmodell erstellt. Das Modell entspricht im Wesentlichen einem Graph der Gatternetzliste mit zusätzlichen Informationen, welche zur Erzeugung der Klauseln und Fehlermenge benötigt werden.

Aus dem Graphenmodell wird das Verhalten der Schaltung als Instanz eines aussagenlogischen Erfüllbarkeitsproblems modelliert, wobei jedoch der Scan-Pfad separat als Modell auf Register-Transfer-Ebene gehandhabt wird. Dadurch ist es möglich, das sequentielle Verhalten auf Transaktionsebene zu beschreiben. Die bestehende Implementierung des Werkzeugs eda1687, welches in Kapitel 5.2 näher beschrieben wird, konnte entsprechend zur Erzeugung von Testmustern erweitert werden.

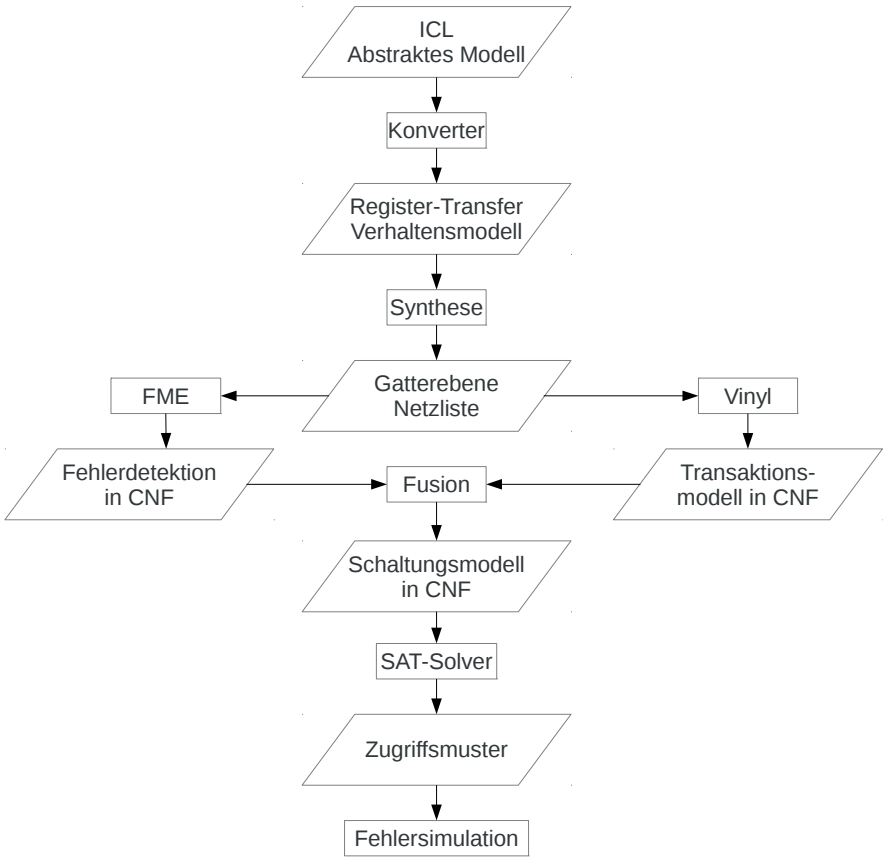

Abbildung 5.1.: Flowchart des Testverfahrens für einen Fehler
FME: Fehlermodellierung, Vinyl: Parser und CNF-Konverter

Des Weiteren wird die Fehlerliste, das heißt die Menge aller Fehler des Fehlermodells die untersucht werden soll, aus dem Graphen erzeugt. Dabei werden nach Möglichkeit äquivalente Fehler zusammengeführt und gemeinsam betrachtet, um Rechenzeit einzusparen.

5.2. Zugriffsmustererzeugung – eda1687

eda1687 ist ein Programm zur Erzeugung von Zugriffsmustern für Rekonfigurierbare Scan-Netzwerke. Als Basis dient eine abstrakte Beschreibung des Rekonfigurierbaren Scan-Netzwerks auf Register-Transfer-Ebene [BKW12], [BKW13].

Das Rekonfigurierbare Scan-Netzwerk wird von eda1687, ähnlich des im Kapitel 3.2 beschriebenen Vorgehens, in eine Instanz des aussagenlogischen Erfüllbarkeitsproblems überführt. Dies gestaltet sich in ICL einfacher als auf Gatterebene, da in der domänen-spezifische

Beschreibung Meta-Informationen, wie zum Beispiel die Aktivierungsbedingungen eines Scan-Segments, explizit ausgeführt sind und nicht extrahiert werden müssen.

5.3. Einlesen und Verarbeiten von Netzlisten

Eine Netzliste ist eine Beschreibung der Verbindung von Gatterelementen eines Schaltungsmoduls. Diese enthält im Wesentlichen folgende Komponenten:

- Name des Moduls
- Bezeichner der Ein- und Ausgänge
- Bezeichner für Verbindungsleitungen
- Zuweisungen für Ein- oder Ausgänge, Verbindungsleitungen oder Konstanten
- Instanzen von Gatterelementen und welche Verbindungsleitungen angeschlossen sind

Das implementierte ATPG-Programm (Vinyl) liest eine Verilog-Netzliste ein und erzeugt daraus ein Zwischenmodell, welches anschließend in ein Graphenmodell umgewandelt wird. Ein Knoten im Graphenmodell kann dabei ein beliebiges Element der Netzliste, das heißt ein Gatter, Register, Primärein- oder -ausgang sowie eine Verbindungsleitung sein. Da Verbindungsleitungen ebenfalls als Knoten betrachtet werden, beschreiben Kanten die Verbindung von Elementen zu anderen Elementen. Dabei besitzen die gerichteten Kanten Bezeichner, welche den benannten Anschlüssen an den Elementen entsprechen.

5.3.1. Parser – ANTLR

Zum Einlesen und Verarbeiten der Strukturbeschreibung der Schaltung in Form von Verilog-Netzlisten wurde auf den objektorientierten Parser-Generator ANTLR (ANother Tool for Language Recognition[1]) zurück gegriffen [Par07]. Dieser wurde von Professor Terence Parr als LL*-Parser in Java entwickelt und ist frei unter einer BSD-Lizenz verfügbar. Als Eingabe dient eine Beschreibung der zu verarbeitenden Grammatik. ANTLR erzeugt daraus Quellcode für eine vorgegebene Programmiersprache. Eine Grammatik besteht aus Regeln zur Ableitung einer Sprache. ANTLR kann dabei Lexer, Parser und/oder Tree-Parser erzeugen.

Ein Lexer liest eine Aneinanderreihung von Symbolen ein und zerlegt diese in einzelne, logisch zusammengehörende Teilstücke, sogenannte Tokens. Ein Token kann zum Beispiel eine natürliche Zahl oder ein Bezeichner für eine Variable sein. Dieser Schritt wird als lexikalische Analyse bezeichnet.

Diese Tokens werden dann an einen Parser weitergegeben. Ein Parser analysiert Tokens und überprüft diese auf syntaktische Korrektheit, das heißt ob die Kombination und Anordnung den Regeln der Grammatik entsprechen. Parser werden meistens als Automaten realisiert

[1] http://www.antlr.org/

und leiten ein entsprechendes Modell, beispielsweise in Form eines Abstrakten Syntaxbaums (engl. abstract syntax tree, AST), ab. In ANTLR kann zu jeder Token-Regel Programmcode angegeben werden, welcher bei Ableitung des Tokens aufgerufen wird. So kann ein ereignisbasierter Parser implementiert werden.

Ein abstrakter Syntaxbaum ist die Repräsentation der abstrakten, syntaktischen Struktur der Instanz einer Grammatik, welcher die inhaltlichen Zusammenhänge wiedergibt. Hierbei stellen die Knoten des Baums Tokens und deren Werte dar. Da durch die Baumstruktur eine explizite Reihenfolge vorgegeben wird, können Kontrolltokens, wie zum Beispiel Klammern, wegfallen.

Die erarbeitete Implementierung nutzt lediglich Lexer und Parser. Der Parser erzeugt ein objektorientiertes, tabellarisches Zwischenmodell, da dies die spätere Umwandlung in ein Graphenmodell vereinfacht. Das tabellarische Zwischenmodell ermöglicht es, eine Referenz auf die Instanz eines Gatter- beziehungsweise Signalobjekts anhand seines Bezeichners zu erhalten. Dieser Zugriff wäre im Rahmen des Abstrakten Syntaxbaum deutlich schwieriger.

5.3.2. Graphenmodell

Das Graphenmodell dient der Abbildung des azyklischen, gerichteten Graphen des Rekonfigurierbaren Scan-Netzwerks in der Implementierung. Dazu werden alle Elemente eines Schaltungsgraphen, das heißt Gatter und Verbindungsleitungen sowie Ein- und Ausgänge als Knoten betrachtet. Kanten werden implizit über Vorgänger- und Nachfolgerbeziehungen realisiert. Die entsprechende Datenstruktur eines Knotens ist in Algorithmus 5.1 dargestellt.

Algorithmus 5.1 Datenstruktur eines Knotens im Graph des RSNs

```
1 struct {
2     string name;
3     string type;
4     std::map<port_name, Node_Ptr> predecessor;
5     std::map<port_name, Node_Ptr> successor;
6 } Node;
```

Zur Konvertierung des Schaltungsgraphen in konjunktive Normalform benötigen alle Gatter eine Variable zur Darstellung ihres Funktionswerts. Die Variablenbezeichner werden als einfache Ganzzahlen dargestellt. Die Abbildung der Gatter auf Variablen erfolgt durch eine `std::map` aus der C++ Standard Template Library (STL).

Allerdings reicht diese einfache Abbildung nicht aus, da in mehreren verschiedenen Arbeitsschritten der Implementierung unterschiedlicher Variablen für dasselbe Gatter benötigt werden. So zum Beispiel bei der Unterscheidung zwischen einem fehlerfreien und einem fehlerhaften Wert. Um den Aufbau der Klauseln möglichst einfach zu gestalten, wurde die Abbildung von Gatter zu Variable in einer hierarchische Datenstruktur gekapselt, welche in Abbildung 5.2 dargestellt ist. Auf diese Weise kann eine Variablenzuordnung durch einfaches Traversieren der Hierarchie gefunden werden.

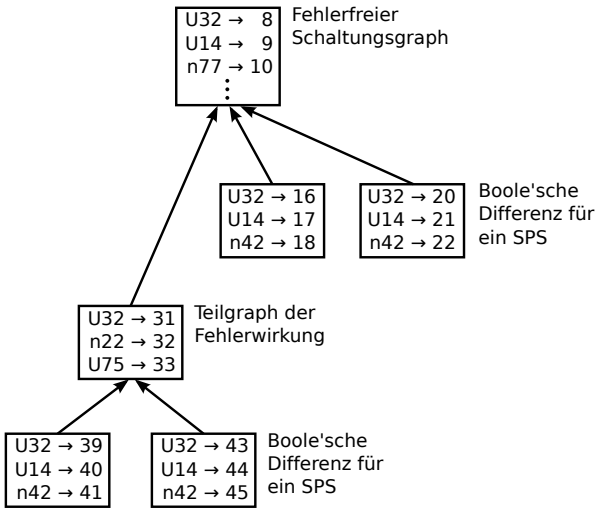

Abbildung 5.2.: Beispielinstanz der Hierarchie der Variablenabbildung
SPS : Scan-Pfad-Segment

Algorithmus 5.2 zeigt die entsprechende Datenstruktur zur Abbildung der Hierarchie der Variablen. Diese besteht lediglich aus der Abbildungsstruktur sowie einem Zeiger auf den Vorgänger in der Hierarchie.

Algorithmus 5.2 Datenstruktur der hierarchischen Abbildung von Variablen

```
1 struct {
2     map<Node_ptr, Var> node_to_var;
3     varmap_ptr parent;
4 } varmap;
```

5.4. Extraktion des Scan-Pfads aus der Netzliste

Aus der Gatternetzliste müssen die Strukturen des Rekonfigurierbaren Scan-Netzwerks wieder extrahiert werden, da die Information über deren Aufbau durch die Synthese verloren gegangen sind. Hierbei werden zuerst die Register im Graphen untersucht. Wird ein Register von einer steigenden Flanke gesteuert, so kann es sich um ein Scan-Register handeln, andernfalls um ein Schattenregister. Sollte es sich tatsächlich um ein Scan-Register handeln, so muss dieses darüber hinaus von den Signalen Capture-Enable, Shift-Enable und Select abhängen. Für ein Schattenregister sind entsprechend die Signale Update-Enable, Reset und Select notwendig.

Zur Erkennung der vorhandenen Scan-Segmente wird der Graph von den Scan-Eingängen (TDI) zu den Scan-Ausgängen (TDO) traversiert, wie es in Algorithmus 5.3 ausgeführt ist. Liegen dabei in dem kombinatorisch-transitiven Eingangskegel eines Scan-Registers ein

Testdateneingang oder ein Scan-Register, so wird das entsprechende Element als Scan-Pfad-Vorgänger vermerkt und der dazugehörige Pfad als Scan-Pfad markiert.

Da es sich bei einem Rekonfigurierbaren Scan-Netzwerk um einen azyklischen Graphen handelt, gilt für jedes Scan-Segment, dass der transitive Eingangskegel disjunkt zum transitiven Ausgangskegel ist, mit Ausnahme des zugehörigen Registers des Scan-Segments sowie sich selbst. Dies wird zur Erkennung der Scan-Segmente genutzt. Wird die Schnittmenge der Register des Eingangs- und des Ausgangskegels eines Scan-Registers gebildet, so darf diese nur aus einem einzelnen Schattenregister bestehen. Dieses ist das zugehörige Schattenregister zum entsprechenden Scan-Register.

Nachdem ein Scan-Segment erkannt wurde, können nun die obligatorischen Pfade markiert werden. Der Capture-Pfad entspricht der Schnittmenge des kombinatorischen Eingangskegel des Scan-Registers und dem kombinatorischen Ausgangskegel des Schattenregisters. Die Schnittmenge des kombinatorischen Eingangskegels des Schattenregisters und dem kombinatorischen Ausgangskegel des Scan-Registers bildet den Update-Pfad. Die jeweilige Schnittmenge der Ein- und Ausgänge eines Registers bilden die Rückkopplungspfade des selbigen.

Algorithmus 5.3 Extraktion des Scan-Pfads aus einem Schaltungsgraphen

```
1 Traversiere iterativ vom Scan-Eingang aus alle Register r:
2   Markiere r als Scan-Register, falls Capture-Enable-, Shift-Enable- und Select-Signal sowie
        ein Scan-Element-Vorgänger existieren
3   Markiere r als Update-Register, falls Update-Enable-, Reset- und Select-Signal existieren
4 Für alle Scan-Register s:
5   Schneide Ein- und Ausgangskegel und markiere Schattenregister s und Rückkopplungspfad
6   Schneide Pfade mit Schattenregister und markiere entsprechend Capture- und Updatepfad
7   Schneide Pfade mit weiteren Scan-Registern und markiere Scan-Pfade
```

Eine Rekonvergenz eines Scan-Pfad-Segments liegt vor, wenn sich ein Datenpfad an einer Signalverzweigung in separate Teilpfade aufspaltet und diese sich später an einem Gatter wieder zusammenfügen. Abbildung 5.3 zeigt eine solche Rekonvergenz.

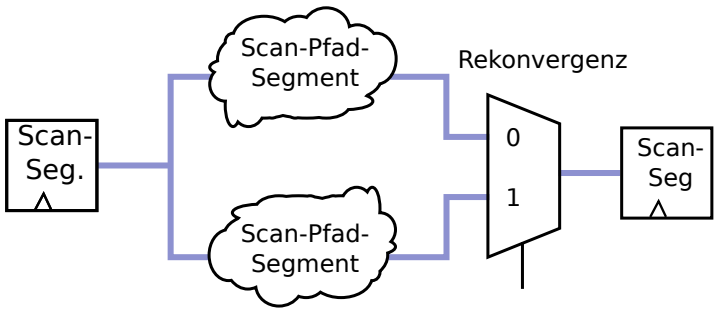

Abbildung 5.3.: Beispiel einer Scan-Pfad-Rekonvergenz
Der Scan-Pfad (blau) spaltet sich auf und rekonvergiert an einem Multiplexer.

5.5. Aktivierung von Scan-Pfad-Segmenten

Zur Modellierung der Aktivierung von Scan-Pfad-Segmenten werden zwei unterschiedliche Methoden eingesetzt. Eine einfache Variante, sofern sich auf dem Scan-Pfad-Segment keine Rekonvergenzen befinden und eine Variante basierend auf der Boole'schen Differenz, falls Rekonvergenzen vorliegen. Für jedes Scan-Pfad-Segment wird eine Variable genutzt, welche die Aktivierung des Scan-Pfad-Segments widerspiegelt.

5.5.1. Boole'sche Differenz

Die Boole'sche Differenz $\frac{\partial f}{\partial x_i}$ beschreibt die Abhängigkeit einer Boole'schen Funktion f gegenüber einer Eingangsvariable x_i. Die Boole'sche Differenz entspricht der Ableitung der Funktion f nach x_i, wodurch die erzeugte Gleichung unabhängig von x_i ist. $\frac{\partial f}{\partial x_i}$ ist genau dann wahr, wenn eine Wertänderung an x_i auch zu einer Änderung am Ergebnis von f führt. Formal wird diese in Gleichung 5.1 ausgedrückt.

$$\begin{aligned}\frac{\partial f}{\partial x_i} =& \frac{\partial f(x_1, \ldots, x_i, \ldots, x_n)}{\partial x_i} \\ =& f(x_1, \ldots, x_i = 1, \ldots, x_n) \oplus f(x_1, \ldots, x_i = 0, \ldots, x_n)\end{aligned} \tag{5.1}$$

In der Instanz des aussagenlogischen Modells ist S das zu aktivierende Scan-Pfad-Segment. Zur Modellierung der Pfadaktivierung durch die Boole'sche Differenz wird der Datenpfad von S jeweils für $x = 1$ und $x = 0$ als S_x dupliziert. Dazu wird jedem Gatterelement in S_x eine Variable zugewiesen und die Gatterelemente mit den neuen Variablen verbunden. Entsprechend wird eine Variable für x erzeugt und dem Dateneingang von S_x zugewiesen. Zum Schluss wird eine Variable A für die Aktivierung von S generiert und einer XOR-Verknüpfung der Gatter am Ausgang von S_x zugewiesen. Algorithmus 5.4 gibt den Code entsprechend wieder.

Algorithmus 5.4 Modellierung der Boole'schen Differenz

```
1  Eingabe:
2    S : Scan-Pfad-Segment als Gatterliste
3  Ausgabe:
4    A : Aktivierungsvariable
5
6  Für jeden Wert x aus {0,1}:
7    Dupliziere Pfad S als Px
8    Erzeuge Variable v aus x
9    Setze Dateneingang von Px auf v
10
11 Erzeuge Variable A
12 A = XOR-Verknüpfung der Ausgänge von P0 und P1
```

Die so erzeugte Variable kann nun genutzt werden um die Instanz des aussagenlogischen Modells zu erweitern, so dass die Variable implizit evaluiert wird.

5.5.2. Binäre Entscheidungsdiagramme

Alternativ hätten auch Binäre Entscheidungsdiagramme (engl. binary decision diagram, BDD) genutzt werden können, um die Boole'sche Differenz zu ersetzen und die Pfadaktivierung zu bestimmen [Bry86]. Ein BDD ist ein azyklischer, gerichteter Graph dessen innere Knoten die Variablen einer Boole'schen Funktion repräsentieren. Jeder Knoten hat zwei ausgehende Kanten, die einer Zuweisung von Wahr beziehungsweise Falsch an die Variable des Ausgangsknoten entspricht. Der Baum besitzt zwei Blätter, welche die Werte Wahr und Falsch widerspiegeln. BDDs können damit Boole'sche Funktionen darstellen, wobei das Traversieren des Baumes dabei der Auswertung der Funktion, unter einer bestimmten Variablenbelegung, entspricht.

BDDs haben den Nachteil, dass deren Größe abhängig von der Variablenanordnung ist und sie bei schlechter Wahl der Ordnung exponentiell groß werden können. Bei Multiplizierern ist das entstehende BDD unabhängig von der Variablenordnung immer exponentiell groß [Bry91].

5.5.3. Pfadsensibilisierung

Liegen keine Rekonvergenzen auf dem Scan-Pfad vor, so kann eine Methode mit weniger Klauseln zur Modellierung der Scan-Pfad-Segmentaktivierung genutzt werden. Ein Datenpfad p ist genau dann sensibilisiert, wenn alle Kontrollsignale des Pfads einen nicht-kontrollierenden Wert c an den Gattern besitzen. Der kontrollierende Wert c_g eines Gatters g bestimmt den Ausgangswert unabhängig von anderen Eingangswerten. Eine Übersicht der kontrollierenden Werte findet sich in Tabelle 5.1.

Name	**Operation**	**Kontrollierender Wert** c_g
Konjunktion	$C = A \wedge B$	0
Disjunktion	$C = A \vee B$	1
Sheffer (NAND)	$C = \overline{A \wedge B}$	0
Peirce (NOR)	$C = \overline{A \vee B}$	1
Kontravalenz	$C = A \oplus B$	-
Äquivalenz	$C = A \Leftrightarrow B$	-

Tabelle 5.1.: Nicht-kontrollierende Werte von Logikgattern

Gleichung 5.2 beschreibt formal die Pfadaktivierung. Ist die Gleichung erfüllt, so kann das Datum vom Anfang des Pfads durch die jeweiligen Gatter propagieren.

(5.2) Pfad p aktiv $\Leftrightarrow \forall$ Gatter $g \in p, \forall$ eingehenden Signale $s \in g \setminus p : s \neq c_g$

Der einfache Pfadaktivierungsalgorithmus in Algorithmus 5.5 extrahiert zuerst alle Variablen der eingehenden Signale des Scan-Pfad-Segments. Die entsprechenden positiven oder negativen Literale werden in einer Liste gespeichert, basierend auf dem kontrollierenden Wert des auf dem Pfad liegenden Gatters. Anschließend wird eine neue Hilfsvariable erzeugt und dieser das Ergebnis einer UND-Verknüpfung der zuvor extrahierten Variablen zugewiesen. Existieren Kontravalenz- oder Äquivalenzgatter auf dem Datenpfad, so können diese nicht statisch modelliert werden, da sie keine kontrollierenden Werte besitzen. In diesem Fall wird auf die Boole'sche Differenz zurückgegriffen.

Algorithmus 5.5 Pfadaktivierung

```
1  Eingabe:
2    P : Scan-Pfad-Segment als Gatterliste
3  Ausgabe:
4    A : Aktivierungsvariable
5  Q: Liste von Variablen
6
7  Für jedes Gatter p in P:
8    Für jeden Vorgänger v von p:
9      Falls v in P:
10       Fahre mit nächstem Vorgänger fort
11     Falls Eins kontrollierender Wert von p:
12       Füge positive Variable von v zu Q hinzu
13     Falls Null kontrollierender Wert von p:
14       Füge negierte Variable von v zu Q hinzu
15
16 Erzeuge Variable A
17 A = UND-Verknüpfung aller Variablen in Q
```

5.6. Fehlerinjektion und -detektion

Fehlerinjektion und -detektion werden ebenfalls als Klauseln des aussagenlogischen Erfüllbarkeitsproblems modelliert. Für die Fehlerinjektion und -wirkung wird dazu ein kombinatorischer Teilgraph des Schaltungsgraphen dupliziert und der Fehler als Konstante hinzugefügt. Die Detektion findet an den sequentiellen Scan-Segmenten statt.

5.6.1. Modellierung von Fehlern in der aussagenlogischen Instanz

In der fehlerbehafteten Instanz müssen drei Aspekte eines Fehlers modelliert werden, welches durch zusätzliche Klauseln in der aussagenlogischen Instanz des Rekonfigurierbaren Scan-Netzwerks geschieht:

- Fehlerinjektion an der Fehlerstelle
- Fehlerwirkung im Ausgangskegel des Fehlers
- Strukturen zum Erkennen der Fehlerwirkung

Die Fehlerinjektion entspricht der Modellierung der Fehlerwirkung am Fehlerort. Dazu wird eine entsprechende Klausel erzeugt, welche das fehlerbehaftete Gatter beschreibt. Befindet sich der Fehler am Ausgangssignal oder entspricht dem kontrollierenden Wert des Gatters, so wird das Gatter als neue Konstante modelliert. Andernfalls wird das fehlerbehaftete Eingangssignal durch eine Konstante ersetzt.

Die Fehlerwirkung beschreibt die Propagierung eines Fehlers durch die Schaltung. Hierzu werden alle Gatter im Ausgangskegel des Fehlers dupliziert, neue Variablen erzeugt und den duplizierten Gattern zugewiesen. Ist solch ein Gattereingang mit dem Ausgang eines anderen Gatters des Fehlerkegels verbunden, so wird die entsprechende fehlerbehaftete Variable genutzt. Wenn dies nicht der Fall ist, das heißt ein eingehendes Signal in den Fehlerkegel, wird die entsprechende fehlerfreie Variable verwendet. Hierbei können die in Kapitel 5.3.2 beschriebenen hierarchischen Variablenabbildungen genutzt werden.

Die Observierung des Fehlers wird durch Vergleich der Gut-Werte mit den fehlerhaften Werten an jedem Scan-Segment im kombinatorischen Fehlerkegel durchgeführt. Dazu können die in Kapitel 4.3 beschriebenen Detektionsbedingungen genutzt werden. Detektionsbedingungen werden ebenfalls als Klauseln modelliert und derart gestaltet, dass für jedes Scan-Segment eine Variable erzeugt wird, welche zu wahr evaluiert, wenn der entsprechende Fehler detektiert wurde. Alle Detektionsvariablen werden anschließend in einer Klausel zusammengeführt, was einer ODER-Verknüpfung entspricht.

Das so erweiterte aussagenlogische Erfüllbarkeitsproblem enthält sowohl die ursprüngliche Schaltung als auch den modellierten Fehler und die notwendigen Erweiterungen zur Detektion dessen. Als Einschränkung des Lösungsraums wird nun dem Lösungsprogramm die Bedingung hinzugefügt, dass mindestens eine der Detektionsklauseln erfüllt sein muss. Als Ergebnis werden auf diese Weise Zugriffsmuster berechnet, welche die modellierten Detektionsbedingungen erfüllen.

5.6.2. Modellierung der Fehlerdetektion in der aussagenlogischen Instanz

Für die Fehlerdetektion werden zwei Fälle unterschieden. Liegt ein Fehler auf dem Scan-Pfad selbst, wie dies Abbildung 5.4 skizziert, wird keine Fehlerdetektion benötigt. Ein Fehler dieser Art unterbricht bei Aktivierung den Scan-Pfad des entsprechenden Scan-Pfad-Segments. Daher ist es bereits ausreichend einen Zugriff auf das fehlerhafte Gatter zu erzwingen und die dadurch erzeugten Zugriffsmuster als Testmuster zu verwenden.

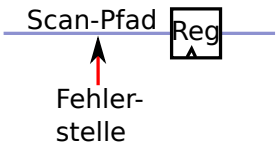

Abbildung 5.4.: Scan-Pfad (blau) mit Fehler auf dem Scan-Pfad

Der zweite Fall liegt vor, falls der zu modellierende Fehler ein Kontroll- oder Adressierungssignal eines Scan-Pfad-Segments beeinflusst, wie es in Abbildung 5.5 dargestellt wird. Dann wird der Fehler, sowie dessen Wirkung, durch zusätzliche Variablen und Klauseln wie in Kapitel 5.6.1 beschrieben, modelliert.

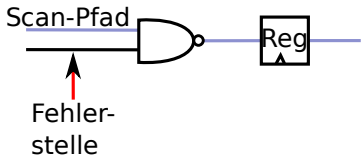

Abbildung 5.5.: Scan-Pfad (blau) mit Fehler an eingehendem Kontrollsignal

Zur Fehlerdetektion wird das in Kapitel 4.3.1 beschriebene Brechen des Scan-Pfads eingesetzt. Algorithmus 5.6 skizziert die Erzeugung der entsprechenden Klauseln. Die Detektionsbedingungen werden dazu an jedem Scan-Register, welches kombinatorisch von der Fehlerwirkung betroffen ist, hinzugefügt. Auf diese Weise kann sichergestellt werden, dass später die Fehlerwirkung ausgelesen werden kann.

Algorithmus 5.6 Pessimistische Fehlerdetektion

```
1  Eingabe:
2    F : Fehlerinjektionsstelle
3  Ausgabe:
4    D : Detektionsvariable
5  L : Variablenliste
6
7  Für jedes Scan-Register s in der Fehlerwirkung von F:
8    # Aktiver Pfad wird inaktiv
9    Erzeuge Klausel k1 : "s_gut aktiv und s_saf inaktiv"
10   Füge k1 in L ein
11
12   # Inaktiver Pfad bleibt inaktiv
13   Erzeuge Klausel k2 : "s_gut aktiv oder s_saf inaktiv"
14   Füge k2 zu CNF hinzu
15
16 Erzeuge Variable A
17 A = ODER-Verknüpfung aller Variablen in L
18 Füge A zu CNF hinzu
```

5.6.3. Fehleräquivalenzklassen

Zur Reduktion der Fehlermenge werden Fehler aus äquivalenten Fehlerklassen kollabiert und damit nur einmal in die Fehlermenge aufgenommen. Beispielhaft sind die entsprechenden Fehleräquivalenzklassen in Tabelle 5.2 für das Haftfehlermodell dargestellt. An verzweigungsfreien Verbindungsleitungen ist die Fehlerwirkung per Definition durch die Identität äquiva-

lent und dementsprechend werden die Fehler auch über die Verbindungsleitungen kollabiert.

Name	Operation	Fehlerklassen
Identität	$C = A$	$\{A_0, C_0\}; \{A_1, C_1\}$
Negation	$C = \overline{A}$	$\{A_0, C_1\}; \{A_1, C_0\}$
Konjunktion	$C = A \wedge B$	$\{A_0, B_0, C_0\}; \{A_1\}; \{B_1\}; \{C_1\}$
Disjunktion	$C = A \vee B$	$\{A_1, B_1, C_1\}; \{A_0\}; \{B_0\}; \{C_0\}$
Kontravalenz	$C = A \oplus B$	$\{A_0\}; \{B_0\}; \{C_0\}; \{A_1\}; \{B_1\}; \{C_1\}$
Äquivalenz	$C = A \Leftrightarrow B$	$\{A_0\}; \{B_0\}; \{C_0\}; \{A_1\}; \{B_1\}; \{C_1\}$
Sheffer (NAND)	$C = \overline{A \wedge B}$	$\{A_0, B_0, C_1\}; \{A_1\}; \{B_1\}; \{C_0\}$
Peirce (NOR)	$C = \overline{A \vee B}$	$\{A_1, B_1, C_0\}; \{A_0\}; \{B_0\}; \{C_1\}$
Implikation	$C = A \rightarrow B$	$\{A_0, B_1, C_1\}; \{A_1\}; \{B_0\}; \{C_0\}$

Tabelle 5.2.: Äquivalente Fehlerklassen
X_y entspricht einem Haftfehler an y am Gatter X

5.7. Fehlersimulation

Zur Validierung und Reduktion der vom ATPG erzeugten Zugriffsmuster werden diese mittels Fehlersimulation überprüft. Um die Simulation möglichst einfach zu implementieren, da deren Optimierung kein Kernbestandteil dieser Arbeit bildet, wurde eine einfache serielle Fehlersimulation realisiert. Die zugehörige Prüfschaltung ist in Kapitel 2.5.1 skizziert worden. Die Verilog-Netzliste des Rekonfigurierbaren Scan-Netzwerks wird dabei zweimal in der Prüfschaltung instanziiert, wobei der entsprechende Fehler in der fehlerbehafteten Instanz hinzugefügt wird. Beide Module werden von den gleichen Eingabemustern getrieben und die Ausgabemuster werden zu jedem Takt verglichen. Sollte es zu einer Abweichung kommen, so ist der injizierte Fehler durch diese Testmuster detektierbar.

Als Simulationsumgebung dient hierbei ModelSim von Mentor Graphics, welches über eine SSH-Verbindung[2] und eines dafür entwickelten TCL-Skripts direkt aus dem ATPG-Programm angesprochen werden kann. Dadurch ist es möglich die erzeugten Testmuster unmittelbar in der Simulation zu überprüfen. Jedoch entstehen dadurch unerwünschte Latenzen aufgrund der Kommunikation über den Netzwerk-Stack und der zusätzlichen Verschlüsselung durch SSH.

Es wurde deshalb zusätzlich eine alternative, stapel-verarbeitende Ansteuerung programmiert. Dessen Ansteuerung kann durch zwei Dateien erfolgen, wobei eine Datei die Fehlermenge enthält und die andere die Testmuster. Dies ist die bevorzugte Variante zur Ermittlung

[2] http://www.libssh.org/

der Fehlerabdeckung der funktionalen Testheuristik, da hierbei keinerlei Interaktion benötigt wird. Die stapel-verarbeitende Ansteuerung kann ebenfalls zur Ermittlung der Fehlerabdeckung der erzeugten Testmustermengen genutzt werden.

Dieser Aufbau bietet ein großes Optimierungspotential bezüglich der Simulationsgeschwindigkeit. Eine Möglichkeit besteht in der parallelen Simulation mehrerer fehlerhaften Instanzen, so dass mit einer Gut-Simulation mehrere Fehler auf einmal simuliert werden können. Damit könnte bei der Validierung eines durch das ATPG erzeugten Zugriffsmusters untersucht werden, ob weitere Fehler durch das Zugriffsmuster entdeckt werden.

Die Fehlerinjektion wird durch den ModelSim-Befehl „`force`“ an einem Signal realisiert, welcher einem Signal einen statischen Wert zuweist. Dieser Befehl lässt sich nativ als Haftfehler nutzen und kann der Schaltung dynamisch hinzufügt werden.

Nachdem die Simulation mit injiziertem Fehler durchgeführt wurde, wird die Fehlerinjektion mittels „`unforce`“ wieder aufgehoben und die Reset-Steuerung des Rekonfigurierbaren Scan-Netzwerks kann genutzt werden, um die Fehlerwirkung aus der Schaltung zu entfernen. Jedoch führt eine Fehlerinjektion an einem Register in ModelSim zu einem Programmfehler, so dass der Fehler nicht mehr aufgehoben wird. Daher muss die Simulationsumgebung neu gestartet werden, was eine höhere Simulationszeit zur Folge hat.

6. Ergebnisse und Bewertung

Im folgenden Kapitel wird ein kommerzielles ATPG-Werkzeug, die beiden funktionalen Testheuristiken aus Kapitel 4.2 und das in Kapitel 4.3.1 vorgeschlagene Testverfahren miteinander verglichen. Hierzu werden die benötigte Rechenzeit zur Erzeugung der Testmuster und die von den Testmustern erreichte Fehlerabdeckung zur Bewertung herangezogen.

Im ersten Unterkapitel wird eine Übersicht über die verwendeten Testschaltungen, sowie deren Aufbau gegeben. Anschließenden erfolgt die Auswertung der Testalgorithmen anhand der beschriebenen Testschaltungen. Abschließend werden die nicht detektierten Fehler für das vorgeschlagene Verfahren des Scan-Pfad-Brechens näher untersucht.

6.1. Übersicht der verwendeten Testschaltungen

Die Untersuchung der Algorithmen zur deterministischen Erzeugung der Testmuster findet mittels dreier Klassen von Testschaltungen statt. Deren Aufbau wird in den nächsten drei Abschnitten betrachtet. Die Testumgebung zur Ermittlung der Rechenzeit bestand aus Intel Core i7-2600 CPUs mit 20GB Arbeitsspeicher.

Die Klassen „SIB“ und „MUX“ bilden reguläre Zugangsstrukturen zum Aufbau hierarchischer Rekonfigurierbarer Scan-Netzwerke. Diese wurden genutzt, um eine Menge ausgewählter Testschaltungen für Systems-on-a-Chip als Rekonfigurierbare Scan-Netzwerke aufzubauen [MIC02]. Schaltungen aus der Klasse „Chain“ bestehen jeweils aus einer kombinatorischen Schaltung, welche als Kontrolllogik für ein Rekonfigurierbares Scan-Netzwerk dient.

6.1.1. SIB-basierte Testschaltungen

In den ersten Entwürfen zu IEEE P1687 wird das SIB (Segment Insertion Bit) als reguläre Struktur zum Aufbau hierarchischer Zugriffsinfrastrukturen beschrieben. Ein SIB kann entweder als Zugangspunkt für tiefer liegende Hierarchien oder als Bypass genutzt werden. Abbildung 6.1 zeigt ein Rekonfigurierbares Scan-Netzwerk, welches mit SIBs realisiert wurde.

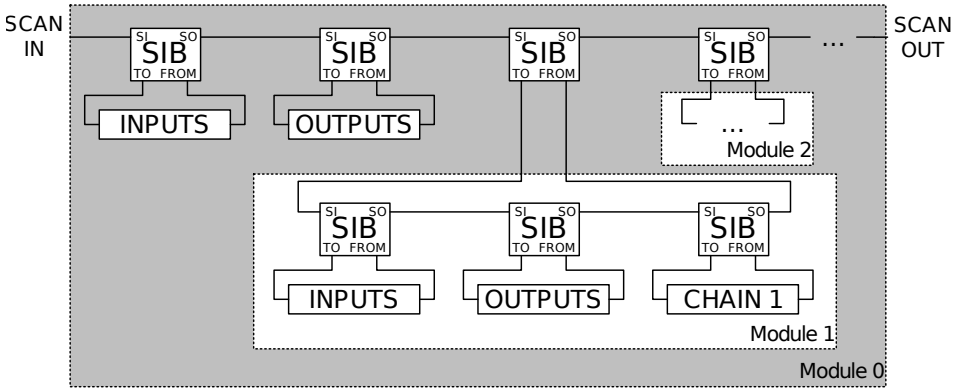

Abbildung 6.1.: Beispiel einer SIB-basierten Testschaltung [BKW12]

Ein SIB besteht aus einem 1-Bit-Konfigurationsregister und einem Scan-Multiplexer, wie es Abbildung 6.2 dargestellt wird. Die Anschlüsse `TDI` und `TDO` sind an den Scan-Pfad angeschlossen, während `TO` und `FROM` eine weitere Hierarchieebene oder weiteres Modul einbinden. Das Konfigurationsregister steuert den Multiplexer so, dass entweder ein Bypass gebildet oder die anhängende Hierarchieebene eingebunden wird.

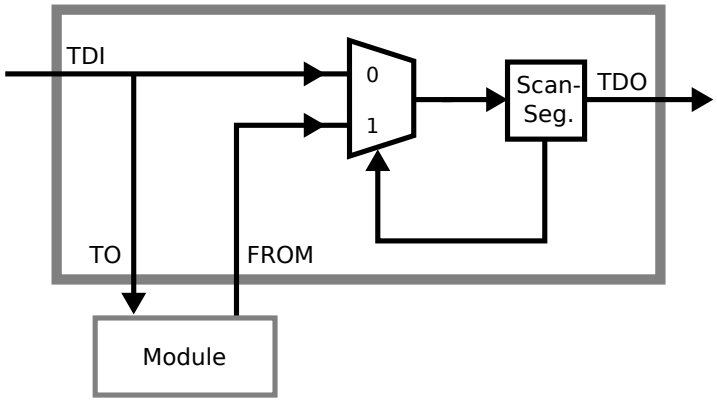

Abbildung 6.2.: Struktureller Aufbau eines SIBs

Der simple Aufbau der SIB-Struktur bietet verschiedene Vorteile. Die einfache Struktur regulärer Hierarchien erfordert keine komplexen Aktivierungsbedingungen, wodurch die Berechnung von Zugriffsmustern effizient erfolgen kann. Der nötige Hardware-Aufwand für das Scan-Segment und den Multiplexer sind sehr gering, wodurch sich ebenfalls eine geringe Zahl von möglichen Fehlern ergibt. Aufgrund des Registers auf dem Scan-Pfad, können keine langen kombinatorischen Pfade bei rekursiver Anwendung entstehen. Nachteilig wirkt sich das Scan-Register jedoch auf die minimale Scan-Pfad-Länge aus, da diese für jedes aktivierte SIB-Element erhöht wird.

6.1.2. MUX-basierte Testschaltungen

MUX-Zellen-basierte Schaltungen sind ähnlich zu SIB-basierten, denn sie binden entweder eine Hierarchieebene oder ein weiteres Modul ein, oder bilden einen Bypass für dieses. Jedoch liegen die Scan-Segmente (config, CFG) zur Steuerung der Multiplexer auf einem separaten Scan-Pfad. Abbildung 6.3 zeigt eine solche Struktur. Der separate Scan-Pfad wird von einem weiteren Scan-Segment (configuration mode, CM) kontrolliert. Dabei werden zwei Betriebsmodi unterschieden. Ein Modus zum Zugriff auf die Datenmodule (CM=0) und einen zur Konfiguration (CM=1), welche Module auf dem Scan-Pfad während des Datenzugriffs aktiv sein sollen.

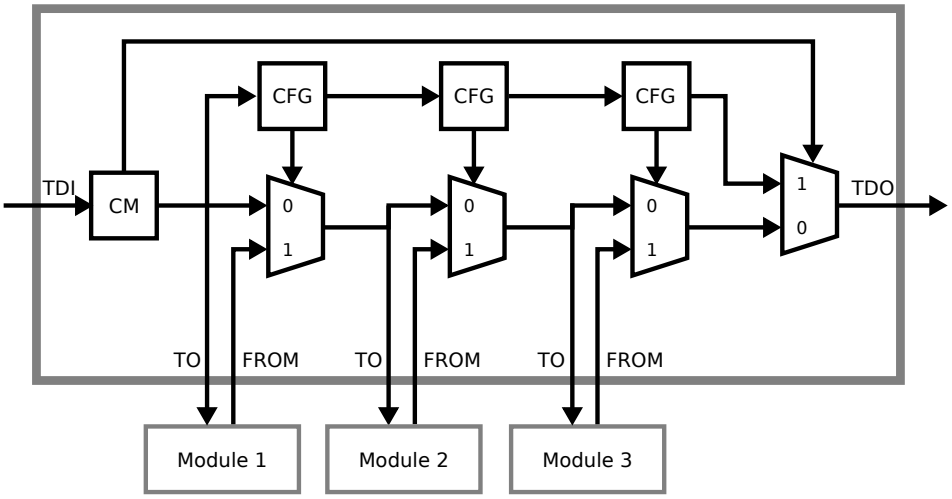

Abbildung 6.3.: Struktureller Aufbau einer MUX-Zelle

Der separate Scan-Pfad zur Konfiguration der Multiplexer reduziert die minimale Scan-Pfad-Länge des Rekonfigurierbaren Scan-Netzwerks. MUX-Strukturen können rekursiv genutzt werden, jedoch entsteht dabei ein Multiplexer-Baum. Jener enthält sehr lange kombinatorische Pfade, da jedes Modul direkt mit dem Primärausgang (TDO) der Schaltung oder einer nachfolgenden MUX-Zelle verbunden wird. Zur Reduktion der Länge des kritischen Pfads kann die Synthese verschiedene Optimierungen vornehmen, wodurch unter anderem Rekonvergenzen entstehen können. Dies führt zu einer aufwendigeren Modellierung in der aussagenlogischen Instanz des Erfüllbarkeitsproblems.

6.1.3. Chain-basierte Testschaltungen

Der Aufbau der dritten Klasse der betrachteten Testschaltungen ist in Abbildung 6.4 skizziert. Diese bestehen aus einer kombinatorischen Schaltung, welche in ein Rekonfigurierbares Scan-Netzwerk als Kontrolllogik integriert wurde. Die Eingänge der kombinatorischen Schaltung werden jeweils von einem Scan-Segment getrieben, wodurch eine vollständige

Kontrollierbarkeit der Eingänge erreicht wird. Jeder Ausgang steuert einen Multiplexer, welcher zwischen einem Bypass und einem Scan-Segment auswählt.

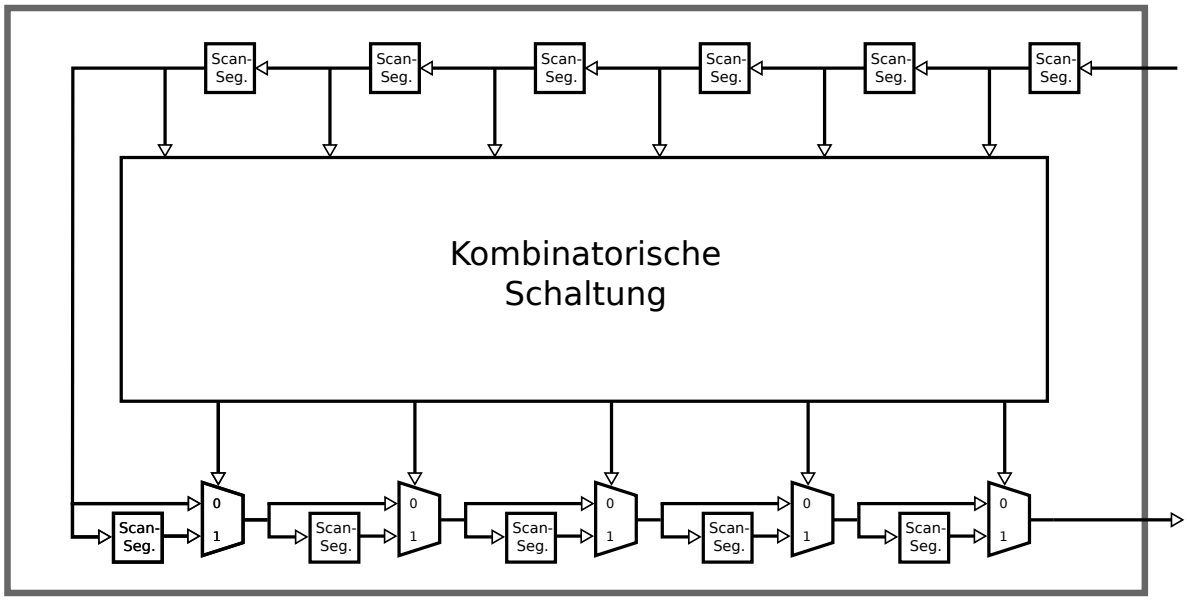

Abbildung 6.4.: Struktureller Aufbau einer Chain-Testschaltung
Eingänge der Ursprungsschaltung wurden durch Scan-Segmente ersetzt.
Jeder Ausgang steuert eine Bypass-artige Struktur mit einem Scan-Segment.

Diese Klasse der Schaltungen wurde synthetisiert, um die Effektivität der Testalgorithmen bei komplexer Steuerlogik in Rekonfigurierbaren Scan-Netzwerken zu untersuchen. Tabelle 6.1 gibt eine Übersicht über die verwendeten kombinatorischen Testschaltungen, die aus den ISCAS'85 Schaltungen ausgewählt wurden [BF85].

Name	Funktion	#PI	#PO	#Gatter	#Fehler
C432	Prioritäts-Decoder	36	7	160	524
c499	Fehlertoleranter Decoder	41	32	202	758
C880	Arithmetisch-logische Einheit	60	26	383	942
C1355	Fehlertoleranter Decoder	41	32	546	1574
C1908	Fehlertoleranter Decoder	33	25	880	1879
C3540	Arithmetisch-logische Einheit	50	22	1669	3428

Tabelle 6.1.: Übersicht über ISCAS'85 Testschaltungen
PI: Primäreingänge, **PO**: Primärausgänge

6.1.4. Schaltungsstatistiken

Tabelle 6.2 zeigt eine Übersicht der verwendeten Testschaltungen und deren charakteristischen Merkmale. Diese wurden von Synopsis Design Compiler D2010.03 für die LSI10k Bibliothek synthetisiert.

Schaltung	Gatter	Fehler	Scan			Scan-Pfad-Länge		
			Seg.	Verzw.	Konv.	Min.	Max.	∅
c432_chain	781	2334	43	13	14	36	43	39
c1908_chain	1702	4918	58	176	239	33	58	45
c880_chain	1894	5584	86	91	118	61	86	73
c499_chain	2016	5968	73	241	329	41	73	57
c1355_chain	2006	5970	73	220	310	41	73	57
c3540_chain	2507	6888	72	43	53	51	72	61
q12710_mux	956	2878	51	55	61	1	26	14
x1331_mux	1218	3644	63	67	73	1	32	16
a586710_mux	1666	4804	79	150	186	1	40	21
f2126_mux	1729	5034	81	185	230	1	41	23
DS9_mux	1860	5448	89	96	113	1	45	21
u226_mux	2061	6092	99	208	269	1	50	27
h953_mux	2081	6188	109	153	173	1	55	28
d281_mux	2363	6952	117	236	293	1	59	32
g1023_mux	3329	9814	159	321	389	1	80	40
p34392_mux	5050	14856	245	568	757	1	123	65
t512505_mux	6779	19880	319	740	953	1	160	78
d695_mux	7070	20312	335	832	1138	1	168	89
p22810_mux	10991	32302	565	1120	1376	1	283	141
q12710_sib	679	2106	46	25	25	4	46	27
x1331_sib	799	2552	56	29	31	6	56	346
a586710_sib	1010	3142	71	34	39	6	71	43
f2126_sib	1057	3316	76	38	40	6	76	47
DS9_sib	1188	3702	80	40	44	5	80	52
u226_sib	1269	3992	89	48	49	11	89	56
h953_sib	1404	4412	100	51	54	10	100	58
d281_sib	1523	4754	108	55	58	8	108	64
g1023_sib	2074	6440	144	78	79	16	144	83
p34392_sib	3157	9920	225	107	122	6	225	130
t512505_sib	4138	12914	287	155	159	33	287	164
d695_sib	4463	13950	324	164	167	10	324	180
p22810_sib	7471	23278	536	271	282	24	536	289

Tabelle 6.2.: Übersicht der charakteristischen Merkmale der betrachteten Testschaltungen
Seg.: Segmente, **Verzw**.: Verzweigungen, **Konv**.: Konvergenzen

6.2. Auswertung der Testalgorithmen

Folgend findet sich die Auswertung der betrachteten Testalgorithmen. Die Tabelle 6.3 auf Seite 68 listet die Fehlerabdeckung und die benötigte Rechenzeit auf. In Tabelle 6.4 auf Seite 69 findet sich die Anzahl und Länge der erzeugten Scan-Testmustern außer für das sequentielle ATPG, da es sich dabei nicht um Scan-Testmuster handelt.

6.2.1. Sequentielles ATPG mit kommerziellem Werkzeug

Als erstes Verfahren wird ein kommerzielles ATPG-Werkzeug für sequentielles ATPG betrachtet und die Fehlerabdeckung und Rechenzeit ermittelt. Damit soll untersucht werden, wie effizient für Rekonfigurierbare Scan-Netzwerke mit bereits bestehenden Werkzeugen Testmuster erzeugt werden können.

Für Testschaltungen mit geringer minimaler Scan-Pfad-Länge und Gatterzahl, wie zum Beispiel a586710_mux, f2126_mux, q12710_mux oder q12710_sib, lassen sich Fehlerabdeckungen bis zu 93 Prozent erreichen, jedoch beträgt die Laufzeit dabei bereits 22 bis 37 Stunden. Durch Erhöhung der maximalen Zahl an Backtracking-Schritten lässt sich die Fehlerabdeckung weiter steigern, jedoch steigt dadurch auch die benötigte Laufzeit. Des Weiteren muss beachtet werden, dass die Scan-Ketten der verwendeten Testschaltungen lediglich ein Bit lang sind. In einer realen Schaltung ist die Länge der Scan-Ketten deutlich größer und damit auch der Suchraum beim sequentiellen ATPG.

Schaltungen mit hoher minimaler Scan-Pfad-Länge stellen ein Problem für das kommerzielle Werkzeug dar. Bei größeren Schaltungen, wie beispielsweise c3540_chain, c880c, DS9_sib oder p22810_sib, liegt die Fehlerabdeckung bei unter 20 Prozent, bei einer Laufzeit von bis zu zwei Tagen. Die Erhöhung der möglichen Schritte des Backtrackings würde hierbei zu einer impraktikabel langen Laufzeit führen.

6.2.2. Funktionale Testmustererzeugung

Die funktionalen Heuristiken unterscheiden sich von den anderen betrachteten Algorithmen in der Vorgehensweise bei der Erzeugung von Testmustern. Die Heuristiken betrachten keine individuellen Fehler und Erzeugen dafür Testmustern, sondern es wird findet nur eine Erzeugung von Zugangsmustern statt, welche als Testmuster verwendet werden. Daher beträgt die Laufzeit zur Erzeugung der Muster nur wenige Sekunden. Die erzeugte Testmustermenge ist dadurch ebenfalls sehr kompakt.

Wie zu erwarten, weist die um Lese- und Schreibzugriffe erweiterte Heuristik eine höhere Fehlerabdeckung gegenüber der reinen Zugriffsheuristik auf. Die Abweichung der Unterschiede liegt etwa zwischen 3 (d295_mux) und 17 Prozent (c432_chain). Im Durchschnitt ergibt sich eine um 9-10 Prozent gesteigerte Fehlerabdeckung, bei ungefähr vierfacher Menge von Testmustern.

Die empirischen Daten zeigen eine gute Fehlerabdeckung bei SIB-basierten Strukturen. Aufgrund des Aufbaus der SIB-Strukturen und der Bedingung, dass alle Scan-Segmente einmal zugreifbar und aktiv sein müssen, werden alle Scan-Pfad-Segmente durch das iterative Aktivieren getestet.

Im Gegensatz hierzu gibt es bei MUX-basierten Schaltungen Scan-Pfad-Segmente, die durch die Zugriffsmuster nicht aktiviert werden. In MUX-basierten Schaltungen kann jedes Modul mit allen nachfolgenden Verbunden sein, so dass sehr viele mögliche Scan-Pfad-Segmente entstehen. Die Heuristiken testen jedoch nur einen Bruchteil dieser, wodurch Fehler auf den Scan-Pfaden und der Steuerlogik nicht detektiert werden.

Die erweiterten ISCAS'85 Schaltungen weisen eine durchschnittliche Fehlerabdeckung auf. Der erforderte Zugriff durch die Testheuristiken für die Scan-Segmente der Eingänge entspricht einem Test mit dem Nullvektor (alle Bits 0) und dem Einsvektor (alle Bits 1). Darüber hinaus erwirkt die Aktivierung der Scan-Segmente, dass alle Primärausgänge der kombinatorischen Schaltung einmal von einer logischen Null und einer Eins getrieben werden.

Im Vergleich zum sequentiellen ATPG-Werkzeug zeichnen sich die Heuristiken durch eine robustere Fehlerabdeckung bei SIB- und MUX-basierten Schaltungen ab und weisen weniger starke Streuungen auf.

Die Erzeugung der Zugriffsmuster auf Register-Transfer-Ebene ist für große Schaltungen um bis zu einem Faktor 150 (p22810_mux) schneller als auf Gatterebene. Dies liegt zum Einen in der größeren Anzahl von Logikelementen, als auch in der aufwendigeren Modellierung begründet. Aufgrund der Rekonvergenzen in MUX-basierten Schaltungen muss die Aktivierung der Scan-Segmente durch die Boole'sche Differenz modelliert werden und erfordert daher die Auswertung einer größeren Instanz des aussagenlogischen Erfüllbarkeitsproblems, wie in Kapitel 5.5.1 beschrieben.

Schaltung	Seq. ATPG		Heur. TPG			BSP		WH + BSP	
			AH	WH	WH				
	FA	LZ	FA	FA	LZ	FA	LZ	FA	LZ
c432_chain	85%	0:22.07	59%	76%	<0:00.01	88%	0:00.39	92%	0:00.11
c1908_chain	83%	0:56.21	53%	63%	<0:00.01	82%	0:16.47	83%	0:05.32
c880_chain	12%	3:20.54	59%	70%	<0:00.01	81%	0:13.00	88%	0:05.00
c499_chain	30%	4:24.47	40%	51%	<0:00.01	85%	0:35.22	87%	0:17.14
c1355_chain	29%	4:25.16	39%	47%	<0:00.01	78%	0:35.27	81%	0:20.14
c3540_chain	14%	7:12.22	56%	71%	<0:00.01	82%	0:11.52	88%	0:03.26
∅	42%	3:26.58	51%	63%	<0:00.01	83%	0:18.51	86%	0:08.36
q12710_mux	94%	0:22.17	70%	79%	<0:00.01	81%	0:01.26	86%	0:00.20
x1331_mux	52%	1:15.27	73%	82%	<0:00.01	85%	0:01.42	88%	0:00.20
a586710_mux	96%	0:26.32	67%	75%	<0:00.01	83%	0:05.55	89%	0:01.13
f2126_mux	94%	37:24.16	62%	71%	<0:00.01	82%	0:07.59	89%	0:01.58
DS9_mux	23%	1:10.15	74%	81%	<0:00.01	84%	0:05.55	89%	0:01.00
u226_mux	45%	2:13.54	65%	72%	<0:00.01	84%	0:12.42	90%	0:04.08
h953_mux	69%	1:20.00	70%	78%	<0:00.01	82%	0:10.29	87%	0:02.44
d281_mux	90%	1:08.03	66%	75%	<0:00.01	84%	0:15.17	90%	0:03.14
g1023_mux	88%	2:11.46	67%	75%	0:00.02	82%	1:02.43	88%	0:16.28
p34392_mux	74%	5:40.43	64%	72%	0:00.07		2:46.51		0:48.58
t512505_mux	55%	10:28.15	67%	74%	0:00.08		7:08.45		1:34.47
d695_mux	62%	5:09.14	67%	70%	0:00.06		5:12.24		1:41.18
p22810_mux	22%	17:38.49	66%	74%	0:00.50		25:48.23		6:18.28
∅	64%	7:10.36	67%	75%	0:00.06	83%	3:34.55	89%	0:54.33
q12710_sib	95%	0:23.51	76%	86%	<0:00.01	76%	0:00.32	87%	0:00.01
x1331_sib	48%	1:38.12	83%	89%	<0:00.01	83%	0:00.49	89%	0:00.02
a586710_sib	58%	3:17.58	78%	87%	<0:00.01	77%	0:00.59	88%	<0:00.01
f2126_sib	75%	2:23.47	77%	87%	<0:00.01	77%	0:01.15	87%	<0:00.01
DS9_sib	16%	6:41.03	75%	84%	<0:00.01	73%	0:01.58	85%	0:00.18
u226_sib	38%	5:59.01	79%	88%	<0:00.01	79%	0:01.36	88%	0:00.05
h953_sib	37%	5:52.55	78%	88%	<0:00.01	78%	0:02.09	88%	0:00.03
d281_sib	33%	6:24.02	77%	87%	<0:00.01	78%	0:02.20	88%	0:00.02
g1023_sib	17%	11:16.24	78%	88%	<0:00.01	76%	0:04.34	88%	0:00.20
p34392_sib	46%	14:03.20	78%	87%	<0:00.01	77%	0:12.13	88%	0:00.12
t512505_sib	17%	11:47.22	88%	88%	<0:00.01	77%	0:19.45	88%	0:01.33
d695_sib	31%	22:40.55	76%	87%	<0:00.01	75%	0:26.32	88%	0:00.09
p22810_sib	12%	47:50.54	77%	87%	<0:00.01		1:22.46	88%	0:04.47
∅	40%	7:35.33	78%	87%	<0:00.01	77%	0:12.07	88%	0:00.35
Gesamt ∅	51%	7:44.06	68%	77%	<0:00.01	81%	1:30.44	88%	0:22.52

Tabelle 6.3.: Ergebnisse für Fehlerabdeckung und Laufzeit verwendeter Testalgorithmen **FA**: Fehlerabdeckung, **LZ**: Laufzeit [h:m.s], **TPG**: Testmustererzeugung, **AH**: Einfache Heuristik, **WH**: Schreibende Heuristik, **BSP**: Brechung des Scan-Pfads

Schaltung	AH		WH		BSP		BSP abzgl. WH	
	Anzahl	Länge	Anzahl	Länge	Anzahl	Länge	Anzahl	Länge
c432_chain	2	79	8	316	269	10060	327	12257
c1908_chain	2	101	8	374	145	7250	103	5131
c880_chain	2	148	8	588	209	14361	221	15019
c499_chain	1	73	8	456	141	8942	163	10443
c1355_chain	1	73	8	456	95	6138	153	9918
c3540_chain	3	181	12	727	93	5498	123	7246
∅	2	109	9	486	159	8708	182	10002
q12710_mux	5	88	18	321	269	2186	67	454
x1331_mux	9	198	32	680	574	5957	116	1152
a586710_mux	7	191	25	679	748	9854	473	5907
f2126_mux	5	137	18	500	731	9815	578	8081
DS9_mux	1	511	67	1811	2687	31771	684	8046
u226_mux	5	174	18	634	915	16562	619	11376
h953_mux	5	187	18	682	630	9347	269	3605
d281_mux	5	195	18	712	888	14346	504	8093
g1023_mux	5	274	18	999	1122	21238	614	11450
p34392_mux	7	569	25	2053	3122	93151	1798	51311
t512505_mux	5	548	18	1998	3016	119623	1311	49249
d695_mux	5	526	18	1925	4121	163969	3057	117619
p22810_mux	7	1330	25	5137	5272	321995	2714	151365
∅	7	403	25	1484	1986	68136	1061	35605
q12710_sib	3	75	12	300	42	864	1	4
x1331_sib	5	167	20	572	86	1958	1	6
a586710_sib	4	177	16	544	24	624	3	28
f2126_sib	3	124	12	488	24	529	2	22
DS9_sib	10	348	40	1216	358	8028	67	1059
u226_sib	3	149	12	596	88	1560	0	0
h953_sib	3	166	12	656	20	707	2	30
d281_sib	3	174	12	696	44	1174	0	0
g1023_sib	3	241	12	956	101	4154	0	0
p34392_sib	4	514	16	1780	163	11554	0	0
t512505_sib	3	481	12	1916	267	24235	2	76
d695_sib	3	501	12	2004	50	2993	0	0
p22810_sib	4	1322	16	3660	101	16857	5	154
∅	4	341	16	1183	105	5787	6	106
Gesamt ∅	5	310	18	1130	840	30100	447	15726

Tabelle 6.4.: Ergebnisse für die Anzahl und Länge der erzeugten Testmuster
AH: Einfache Heuristik, **WH**: Schreibende Heuristik,
BSP: Brechung des Scan-Pfads, **BSP abzgl. WH**: Brechung des Scan-Pfads für die verbleibende Fehlermenge nach WH

6.2.3. Brechung des Scan-Pfads

Der betrachtete Ansatz des „Brechens des Scan-Pfads“ untersucht unter anderem alle Fehler, welche sich auf den Scan-Pfad-Segmenten selbst befinden und erzeugt für diese Zugriffsmuster. Durch die große Anzahl der möglichen Fehler entstehen jedoch sehr viele Testmuster, wodurch die Testmustermenge ebenfalls groß wird. Die Testmuster können durch Fehleraufgabe reduziert werden.

Sind Testmuster unabhängig, das heißt liegen die Fehler auf unterschiedlichen Scan-Pfad-Segmenten, welche sich nicht gegenseitig beeinflussen und gleichzeitig aktiv sein können, dann lassen sich die Testmuster zusammenfassen und so kann die Testmustermenge weiter reduziert werden. Dies könnte als parallele Modellierung der Fehler in einer aussagenlogischen Instanz des Erfüllbarkeitsproblems implementiert werden.

Eine höhere Fehlerabdeckung, als bei den heuristischen Verfahren, wird bei den Chain-Schaltungen erreicht, da für diese die Fehler in der kombinatorischen Logik der Schaltung explizit betrachtet werden. Hierbei ist zu beachten, dass bei diesen Schaltungen die Scan-Multiplexer an den Ausgängen der Schaltung ein Brechen des Scan-Pfads verhindern. Ist die Fehlerwirkung an einem Primärausgang der kombinatorischen Schaltung beobachtbar, dann wird ein zwar falscher Scan-Pfad gewählt, aber ein aktiver Scan-Pfad bleibt erhalten.

Die einfache Heuristik zur Testmustererzeugung kann bereits alle Fehler des Scan-Pfads in SIB-basierten Schaltungen detektieren, weshalb sich keine Verbesserung durch das Brechen des Scan-Pfads ergibt. Die Fehlerabdeckungen sind sehr ähnlich, jedoch unterscheiden sich die erzeugten Testmustermengen stark voneinander. Denn die Heuristiken erzeugen lediglich Zugriffsmuster, die einmalig alle Scan-Segmente aktivieren, während bei der Brechung des Scan-Pfads Testmuster für jeden Fehler separat erzeugt werden.

Bei MUX-basierten Schaltungen ist eine Verbesserung der Fehlerabdeckung gegenüber den Heuristiken beobachtbar, da Fehler auf den nicht aktivierten Scan-Pfad-Segmenten detektiert werden. Jedoch sind Testmustermenge als auch die Testschaltungen, wie etwa bei d695_mux oder p22810_sib, so groß, dass für die Stapel-verarbeitende Fehlersimulation keine praktikable Laufzeit erreicht und daher keine Ergebnisse ermittelt werden konnten. Mögliche Verbesserungen werden in Kapitel 7 aufgezeigt.

6.2.4. Kombination aus Heuristik und Brechung des Scan-Pfads

Abschließend wird eine Kombination aus der lesenden und schreibenden Heuristik und der Brechung des Scan-Pfads betrachtet. Hierbei kombinieren sich die Vorteile beider Verfahren, indem zuerst alle Fehler von der Heuristik untersucht werden. Anschließend wird für die übrig gebliebene Fehlermenge das Brechen des Scan-Pfads eingesetzt, so dass die aufwendige Modellierung jedes einzelnen Fehlers reduziert wird. Dadurch soll die benötigte Laufzeit reduziert und insgesamt die Fehlerabdeckung verbessert werden. Die Zahl und Länge der erzeugten Scan-Testmuster halbiert sich ungefähr im Vergleich zum exzessiven Scan-Pfad-Brechen.

Für SIB-basierte Schaltungen zeigt sich keine Verbesserung gegenüber der Heuristik. Dies liegt daran, dass nahezu alle Fehler auf dem Scan-Pfad bereits durch die Zugriffsheuristik erkannt wurden und die Brechung des Scan-Pfads nur Fehler dieser Klasse betrachtet. Es ist daher unnötig, das Brechen des Scan-Pfads mit Heuristik für SIB-basierte Schaltungen zu kombinieren.

Für MUX-basierte und Chain-Schaltungen ergibt sich eine Steigerung der Fehlerabdeckung durch Detektion von Fehlern auf den nicht durch die Heuristik aktivierten Scan-Pfad-Segmenten. Durch die Reduktion der zu betrachtenden Fehlermenge nach Nutzung der Heuristik sinkt die Laufzeit zur Erzeugung der Testmuster.

6.3. Klassifizierung nicht detektierter Fehler

In diesem Kapitel sollen die nicht detektierten Fehler anhand der Ergebnisse von drei Beispielschaltungen detaillierter untersucht werden. Da die Fehlersimulation in ModelSim auch „Unbekannte“-Werte modelliert, führen Fehler an diesen Stellen von Registern nur zu potentiell detektierbaren Fehlern. Ein potentiell detektierbarer Fehler liegt vor, wenn ein Wert in der fehlerbehafteten Schaltung „Unbekannt“ ist und es somit nicht feststeht, ob der Wert tatsächlich vom Gutwert abweicht.

Folgende Gründe erklären, warum diese dennoch deterministisch detektiert werden:

- Es ist davon auszugehen, dass Haftfehler an der Taktsteuerung von einer Flush-Sequenz erkannt werden, da aufgrund der fehlenden Taktflanke keine Datenwerte in den Registern gespeichert werden. Damit wird nur ein statischer Wert propagiert und die Flush-Sequenz gestört.
- Haftfehler am aktiven Logikwert der Reset-Steuerung führen ebenfalls zu einem stationären Verhalten der Schattenregister. Da die Reset-Steuerung an Registern immer Vorrang vor dem Dateneingang hat, ergibt sich dadurch ebenfalls die Propagierung eines statischen Werts.
- Kann ein Schattenregister nicht zurückgesetzt werden, weil die Fehlerwirkung das Reset-Signal blockiert, so lässt sich dies einfach testen. Dazu wird der komplementäre Wert des Initialzustands in das Schattenregister geschrieben und anschließend ein Reset ausgeführt. Bei einem ersten Zugriff durch eine CSU-Operation wird das fehlerhafte Datenwort ausgelesen.

Anhand einer aus jeder Schaltungsklasse ausgewählten Schaltung wird folgend näher auf die nicht detektierten und potentiell detektierbaren Fehler eingegangen. Bei dem untersuchten Testmustererzeugungsverfahren handelt es sich um die Kombination aus lesender und schreibender Heuristik, sowie dem Brechen des Scan-Pfads.

6.3.1. Untersuchung der verbliebenen Fehler in c499_chain

Verbleibende Fehler finden sich vor allem in der Scan-Adressierung, wie es in Abbildung 6.5 gezeigt ist. Im Fall der Chain-Schaltungen sind dies alle Fehler der kombinatorischen Schaltung. Da diese durch die Brechung des Scan-Pfads nicht detektiert werden können und die Heuristik diese Fehler nicht explizit betrachtet, ergibt sich eine geringe Fehlerabdeckung. Weitere undetektierte Fehler auf dem Scan-Datenpfad ergeben sich aus der Erzeugung von Rekonvergenzen durch die Synthese, welche durch den Aufbau der Scan-Multiplexer nicht gebrochen werden können.

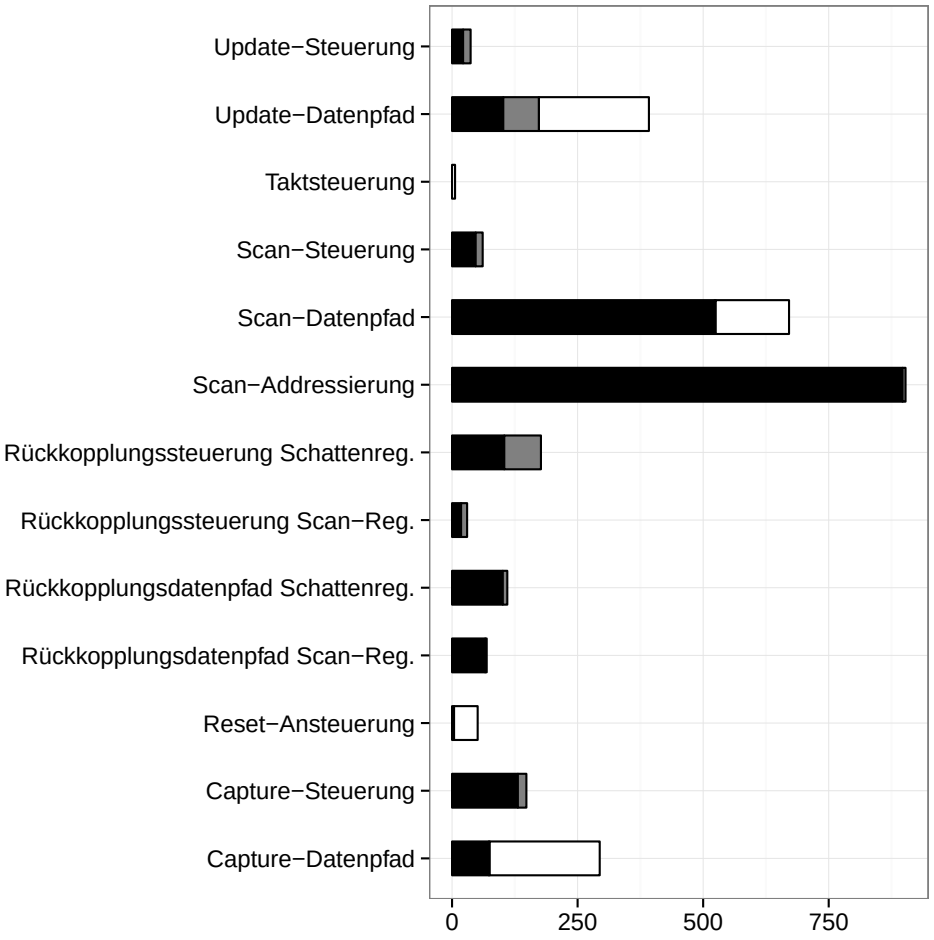

Abbildung 6.5.: Anzahl verbleibender Fehler in c499_chain
Schwarz: Undetektierte Fehler
Grau: Potentiell detektierbare Fehler
Weiß: Pot. detektierbare Fehler die nach 6.3 detektiert werden

6.3.2. Untersuchung der verbliebenen Fehler in f2126_mux

Ähnlich zu den Chain-Schaltungen finden sich in MUX-Schaltungen hauptsächlich verbleibende Fehler auf dem Scan-Datenpfad und der Scan-Adressierung. Abbildung 6.6 zeigt dies anhand von f2126_mux. Aufgrund von Scan-Pfad-Segmenten, die nicht durch die Fehlerwirkung an der Adressierung des Scan-Multiplexers gebrochen werden können, werden keine Testmuster erzeugt.

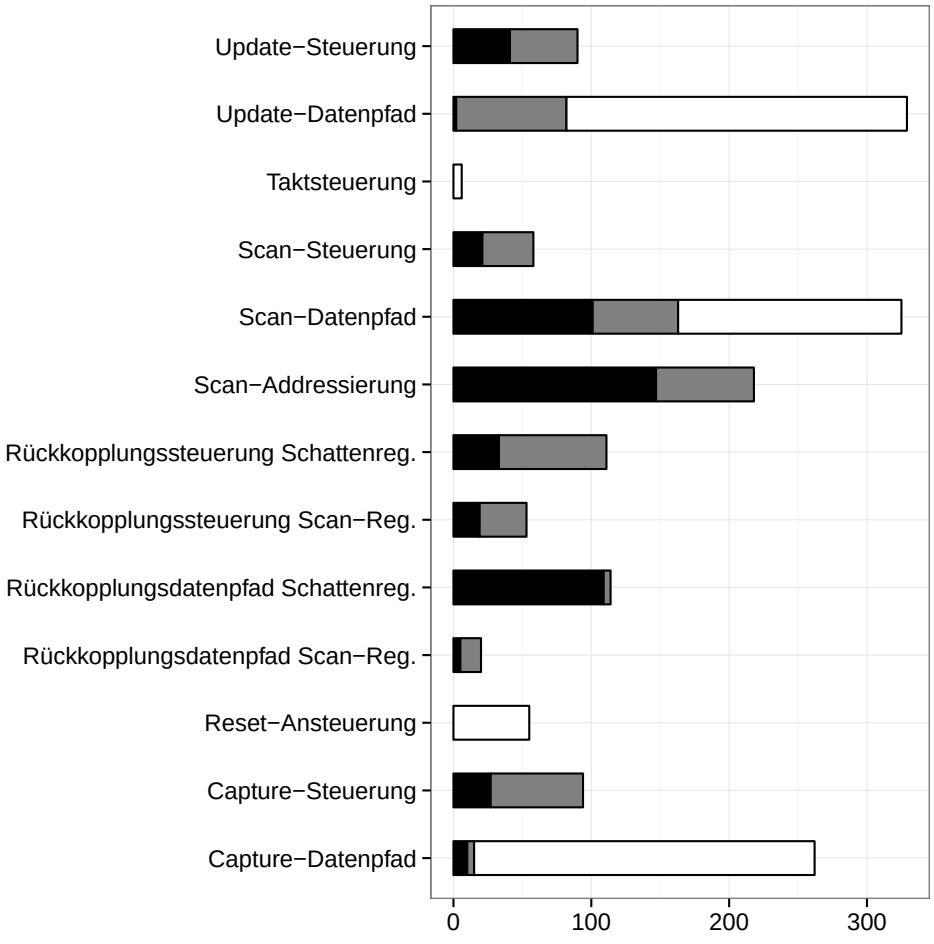

Abbildung 6.6.: Anzahl verbleibender Fehler in f2126_mux
Schwarz: Undetektierte Fehler
Grau: Potentiell detektierbare Fehler
Weiß: Pot. detektierbare Fehler die nach 6.3 detektiert werden

6.3.3. Untersuchung der verbliebenen Fehler in f2126_sib

Beispielhaft für SIB-basierte Schaltungen, lassen sich in f2126_sib alle Fehlerklassen, bis auf den Rückkopplungsdatenpfad der Schattenregister, durch die lesend und schreibend zugreifende Heuristik bereits detektieren. Abbildung 6.7 gibt die Fehlerstellen für f2126_sib entsprechend wieder. Bei den verbleibenden Fehlern auf dem Rückkopplungsdatenpfad handelt es sich um Aktivierungssignale, welche ständig aktiv sein müssen, um einen aktiven Scan-Pfad auszuprägen. Die entsprechenden undetektierten Haftfehler propagieren den gleichen Wert und lassen sich so nicht detektieren.

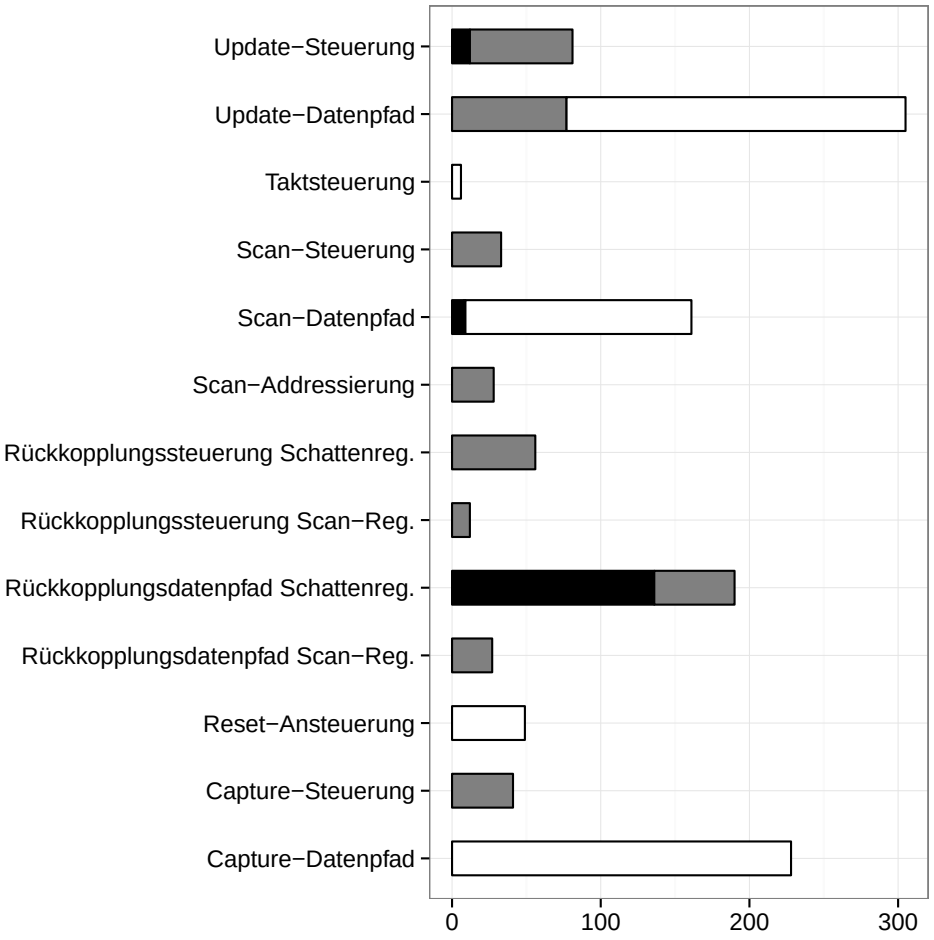

Abbildung 6.7.: Anzahl verbleibender Fehler in f2126_sib
Schwarz: Undetektierte Fehler
Grau: Potentiell detektierbare Fehler
Weiß: Pot. detektierbare Fehler die nach 6.3 detektiert werden

6.3.4. Zusammenfassung

Drei Fehlerklassen konnten bei der Untersuchung der verbleibenden Fehler als schlecht testbar durch das vorgeschlagenen Verfahren zur Testmustererzeugung ausgemacht werden:

Scan-Adressierung Das Brechen des Scan-Pfads funktioniert nicht bei Fehlern an der Ansteuerung der Scan-Multiplexer. Betrifft die Fehlerwirkung diese, so kann sich dennoch ein aktiver Scan-Pfad ausprägen, weshalb kein Testmuster erzeugt wird.

Scan-Datenpfad Ein Teil der der Scan-Pfad-Segmente werden nicht getestet, da die Heuristik nur eine Teilmenge dieser aktiviert. Das Brechen dieser Scan-Pfade ist nicht immer möglich, wodurch die Fehlerabdeckung sinkt.

Rückkopplungsdatenpfad Schattenregister Müssen an bestimmten Kontrollsignalen Datenwerte anliegen, welche notwendig sind um einen aktiven Scan-Pfad auszuprägen oder eine CSU-Operation zu ermöglichen, so können Haftfehler mit gleichem Logikwert nur schlecht getestet werden.

Weitere Fehlerklassen, wie die Update- und Capture-Steuerung, werden nicht explizit durch die Modellierung des Verfahrens betrachtet, weshalb keine entsprechenden Testmuster für diese erzeugt werden.

7. Zusammenfassung und Ausblick

Rekonfigurierbare Scan-Netzwerke sind wichtig für die Inbetriebnahme einer integrierten Schaltung und müssen fehlerfrei arbeiten. Hierzu ist ein Test mit hoher Fehlerabdeckung essentiell. Klassische Testalgorithmen für statische Scan-Ketten sind jedoch nicht ausreichend, da die kombinatorische und sequentielle Komplexität von Rekonfigurierbaren Scan-Netzwerken bedeutend höher ist. Deshalb wurde ein Verfahren zur Testmustererzeugung in Rekonfigurierbaren Scan-Netzwerken entwickelt, welches speziell an deren Erfordernisse angepasst ist. Durch die Kombination zweier Verfahren, nämlich einer Heuristik und dem Brechen des Scan-Pfads, konnte eine durchschnittliche Steigerung der Fehlerabdeckung gegenüber kommerziellen Werkzeugen von 172 Prozent erreicht werden, bei Reduktion der Laufzeit um bis zu einem Faktor 20. Weitere Optimierungen werden in den folgenden Kapiteln aufgezeigt, wodurch sich die Fehlerabdeckung weiter steigern, beziehungsweise die Laufzeit reduzieren lässt.

Erweiterung der funktionalen Testheuristiken

Die funktionalen Testheuristiken erzeugen lediglich Zugriffsmuster zur Aktivierung aller Scan-Segmente. Je nach Klasse der Zugriffsinfrastruktur werden dadurch sehr viele Scan-Pfade bereits aktiviert, wie dies beispielsweise bei SIB-basierten Scan-Netzwerken der Fall ist. Zur weiteren Optimierung der Heuristiken wäre es wünschenswert, dass sich jedes Scan-Pfad-Segment mindestens einmal auf dem aktiven Scan-Pfad befindet, da die bisherigen Heuristiken lediglich den Zugriff auf die Scan-Segmente erfordern. Dadurch können Fehler auf diesen Scan-Pfad-Segmenten sehr leicht detektiert werden. Darüber hinaus sind die entsprechenden Bedingungen der aussagenlogischen Instanz zum Zugriff leicht modellier- und lösbar.

Die Aktivierung eines Scan-Pfad-Segments im Erfüllbarkeitsproblem der Aussagenlogik wurde als Pfadaktivierung modelliert. Durch die Betrachtung aller möglichen Scan-Pfad-Segmente zwischen den Scan-Segmenten, kann es bei ungünstigen Zugriffsstrukturen, wie den MUX-basierten Schaltungen, zu einem Aufblähen der möglichen Pfade kommen. In MUX-basierten Schaltungen kann jedes Modul mit nahezu jedem anderen Modul verbunden werden, wodurch es entsprechend viele möglichen Scan-Pfad-Segmente und Kombinationen gibt. In diesen Fällen ist das Berechnen der Zugriffsmuster aufwändig und wurde deshalb nicht weiter betrachtet. Eine Lösung dieses Problems könnte in der Aufspaltung der Scan-Pfad-Segmente an Multiplexern und Verzweigungen bestehen.

Fehlerdetektierung durch Gebietsanalyse

Ein großes Problem der Laufzeit bei der erarbeiteten Implementierung rührt von der expliziten Modellierung jedes Fehlers her. Ähnlich wie bei der Gebietsanalyse des PPSFP-Algorithmus könnte die Detektierbarkeit weiterer Fehler durch logisches Schließen ermittelt werden. Durch die Testheuristiken sind bereits alle Datenwerte der Register bekannt.

Dabei treten jedoch zwei Probleme hervor:

- Durch die Fehlerwirkung kann es zu abweichenden Datenwerten in den Registern kommen, welche nicht durch die Testheuristiken detektiert wurden. Dann bliebe ein eventuell als detektiert berechneter Fehler undetektiert. Eine genauere Untersuchung des Fehlerverhaltens muss daher in Betracht gezogen werden.
- Im PPSFP-Algorithmus sind während der Fehlersimulation sowohl Gut- als auch fehlerbehaftete Werte berechnet worden. Diese fehlen jedoch bei der Modellierung im aussagenlogischen Erfüllbarkeitsproblem.

Partitionierung des Rekonfigurierbaren Scan-Netzwerks

Die Module und Hierarchien in Rekonfigurierbaren Scan-Netzwerken sind oft unabhängig voneinander. Beispielsweise gilt dies für aneinandergereihte MUX- und SIB-Strukturen, welche nur über den Scan-Pfad verbunden sind. Lässt sich ein relativ abgeschlossener Teilgraph finden, so können kleinere Instanzen der modellierten Schaltung betrachtet werden, was zu einer verbesserten Laufzeit führt. Insbesondere ermöglicht dies einen erschöpfenden funktionalen Test.

Diagnose

Bei Scan-Ketten schränkt ein Fehler die Observierbarkeit stark ein, falls dieser den Scan-Pfad bricht. Verschiedene Verfahren zur Diagnose von Scan-Ketten wurden in Kapitel 2.2.3 beschrieben, welche in der Regel zu zusätzlichem Hardware-Aufwand führen.

Im Gegensatz dazu verlieren Rekonfigurierbare Scan-Netzwerke ihre Observierbarkeit nur bei bestimmten Fehlern, wie etwa auf dem Taktbaum oder in der ersten Hierarchie. Hierdurch lassen sich allerdings schon erste Schlüsse auf die Fehlerursache ziehen.

Ein Verfahren zur Lokalisierung eines Fehlers kann in der iterativen Freischaltung einzelner Module oder Hierarchien bestehen. Wird ein Fehler detektiert, so kann beispielsweise ein Reset durchgeführt oder wieder versucht werden, die Hierarchie zu schließen. In der nächsten Phase kann die Hierarchie dann vorübergehend ausgesetzt und weitere Hierarchien getestet werden.

Full-Scan-Design für Schattenregister

Scan-Register sind in Rekonfigurierbaren Scan-Netzwerken durch ihre Lage auf dem aktiven Scan-Pfad observier- und kontrollierbar. Dahingegen kann auf Schattenregister nicht direkt zugegriffen werden, sondern nur über die zugehörigen Scan-Register. Eine Möglichkeit des direkten Zugriffs kann durch Hinzufügen der Schattenregister zu einem Scan-Pfad erreicht werden, um diese von außen observier- und kontrollierbar zu machen.

Hierdurch lässt sich auch das Problem der Diagnose einfacher lösen, da auch bei gebrochenem Scan-Pfad Zugriff auf die Konfigurationsregister besteht. So können eventuelle Übertragungsfehler in die Konfigurationsregister gefunden werden. Durch CSU Operationen mit nur einer einzelnen Scan-Operation pro Scan-Phase kann der Schiebebetrieb exakt verfolgt werden.

Eventuell besitzt die integrierte Schaltung bereits einen Scan-Pfad für den Test, wodurch keine weiteren Primärein- und -ausgänge benötigt werden. Der Verzicht auf die Reset-Logik könnte den zusätzlich Hardware-Aufwand reduzieren, da die Initialisierung durch Scan durchgeführt werden kann.

A. Anhang

Algorithmus A.1 ICL-Beispiel: Zwei Konfigurationsregister zur Steuerung eines Multiplexers mit einem Register oder Beipass

```
1  Module SelReg1 {
2    ResetPort reset;
3    ScanInPort scanIn;
4    ScanOutPort scanOut { Source reg; Enable select; }
5    SelectPort select;
6    ShiftEnPort shiftEn;
7    DataOutPort dataOut { Source reg; }
8    ScanRegister reg {
9      ScanInSource scanIn;
10 } }
11 Module ICL_Example {
12   ResetPort reset;
13   ScanInPort scanIn;
14   ScanOutPort scanOut { Source mux; }
15   SelectPort select;
16   ShiftEnPort shiftEn;
17   Instance cfg1 Of SelReg1 {
18     InputPort reset = reset;
19     InputPort select = select;
20     InputPort shiftEn = shiftEn;
21     InputPort scanIn = scanIn;
22   }
23   Instance cfg2 Of SelReg1 {
24     InputPort reset = reset;
25     InputPort select = select;
26     InputPort shiftEn = shiftEn;
27     InputPort scanIn = cfg1.scanOut;
28   }
29   LogicSignal reg_en {
30     cfg1.dataOut, cfg2.dataOut == 'b11;
31   }
32   Instance reg Of SelReg1 {
33     InputPort scanIn = cfg2.scanOut;
34     InputPort reset = reset;
35     InputPort select = reg_en;
36     InputPort shiftEn = shiftEn;
37   }
38   ScanMux mux reg_en {
39     0 : cfg2.scanOut;
40     1 : reg.scanOut;
41 } }
```

Literaturverzeichnis

[BARVT82] C. C. Beh, K. H. Arya, C. E. Radke, E. K. Vida-Torku. Do Stuck Fault Models Reflect Manufacturing Defects? In *Proc. of IEEE International Test Conference*, S. 35–42. IEEE, 1982. (Zitiert auf Seite 18)

[BF85] F. Brglez, H. Fujiwara. A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran. In *Proc. of International Symposium Circuits and Systems (ISCAS 85)*, S. 677–692. IEEE, 1985. (Zitiert auf Seite 64)

[BKW12] R. Baranowski, M. A. Kochte, H.-J. Wunderlich. Modeling, Verification and Pattern Generation for Reconfigurable Scan Networks. In *Proc. of IEEE International Test Conference*. IEEE, 2012. (Zitiert auf den Seiten 48 und 62)

[BKW13] R. Baranowski, M. A. Kochte, H.-J. Wunderlich. Optimal Scan Pattern Generation for Reconfigurable Scan Networks. Submitted for Review ETS 2013. (Zitiert auf Seite 48)

[BM82] P. H. Bardell, W. H. McAnney. Self-testing of multiple logic modules. In *Proc. of IEEE International Test Conference*, S. 200–204. IEEE, 1982. (Zitiert auf Seite 37)

[Bry86] R. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677 –691, 1986. (Zitiert auf Seite 54)

[Bry91] R. Bryant. On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication. *IEEE Transactions on Computers*, 40(2):205 –213, 1991. (Zitiert auf Seite 54)

[Coo71] S. A. Cook. The complexity of theorem-proving procedures. In *Proc. of the third annual ACM symposium on Theory of computing*, STOC '71, S. 151–158. ACM, 1971. (Zitiert auf Seite 22)

[EB06] B. Eklow, B. Bennetts. New Techniques for Accessing Embedded Instrumentation: IEEE P1687 (IJTAG). In *Proc. of Eleventh IEEE European Test Symposium*, S. 253 –254. IEEE, 2006. (Zitiert auf Seite 27)

[EE95] S. Edirisooriya, G. Edirisooriya. Diagnosis of scan path failures. In *Proc. of VLSI Test Symposium*, S. 250–255. IEEE, 1995. (Zitiert auf Seite 15)

[ES06] N. Eén, N. Sörensson. Translating Pseudo-Boolean Constraints into SAT. *JSAT*, 2(1-4):1–26, 2006. (Zitiert auf Seite 43)

[ESB00] J. Emmert, C. Stroud, J. Bailey. A new bridging fault model for more accurate fault behavior. In *Proc. of AUTOTESTCON*, S. 481 –485. IEEE, 2000. (Zitiert auf Seite 19)

[FS83] H. Fujiwara, T. Shimono. On the Acceleration of Test Generation Algorithms. *IEEE Transactions on Computers*, C-32(12):1137 –1144, 1983. (Zitiert auf Seite 21)

[GN02] E. Goldberg, Y. Novikov. BerkMin: A fast and robust SAT-solver. In *Proc. of Design, Automation and Test in Europe Conference (DATE)*, S. 142 –149. 2002. (Zitiert auf Seite 22)

[GR81] P. Goel, B. Rosales. PODEM-X: An Automatic Test Generation System for VLSI Logic Structures. In *Proc. of 18th Conference on Design Automation*, S. 260 – 268. IEEE, 1981. (Zitiert auf Seite 21)

[GZA+02] M. Ganai, L. Zhang, P. Ashar, A. Gupta, S. Malik. Combining strengths of circuit-based and CNF-based algorithms for a high-performance SAT solver. In *Proc. of IEEE/ACM Design Automation Conference*, S. 747 – 750. IEEE/ACM, 2002. (Zitiert auf Seite 23)

[HKB94] R. Hahn, R. Krieger, B. Becker. A hierarchical approach to fault collapsing. In *Proc. of European Design and Test Conference (EDAC)*, S. 171 –176. 1994. (Zitiert auf Seite 11)

[HP99] I. Hamzaoglu, J. Patel. Reducing test application time for full scan embedded cores. In *Proc. of International Symposium on Fault-Tolerant Computing*, S. 260 –267. 1999. (Zitiert auf Seite 14)

[IEE01] IEEE. Standard Test Access Port and Boundary - Scan Architecture. *IEEE Std 1149.1-2001*, 2001. (Zitiert auf Seite 14)

[iso12] ISO 26262-5:2012 Road vehicles - Functional safety - Part 5: Product development: hardware level, 2012. (Zitiert auf Seite 10)

[KSWZ10] M. Kochte, M. Schaal, H.-J. Wunderlich, C. Zoellin. Efficient fault simulation on many-core processors. In *Proc. of IEEE/ACM Design Automation Conference (DAC)*, S. 380 –385. IEEE/ACM, 2010. (Zitiert auf Seite 20)

[Kun93] S. Kundu. On diagnosis of faults in a scan-chain. In *Proc. of VLSI Test Symposium*, S. 303 –308. 1993. (Zitiert auf Seite 13)

[Lar92] T. Larrabee. Test pattern generation using Boolean satisfiability. *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(1):4 –15, 1992. (Zitiert auf Seite 21)

[Lev73] L. A. Levin. Universal sorting problems. *Problems of Information Transmission*, 9:265–266, 1973. (Zitiert auf Seite 22)

[LH92] H. Lee, D. Ha. HOPE: an efficient parallel fault simulator. In *Proc. of IEEE/ACM Design Automation Conference (DAC)*, S. 336 –340. IEEE/ACM, 1992. (Zitiert auf Seite 21)

[MA98] A. Majhi, V. Agrawal. Delay fault models and coverage. In *Proc. of IEEE International Test Conference*, S. 364 –369. IEEE, 1998. (Zitiert auf Seite 19)

[MBAB99] S. Majumder, B. Bhattacharya, V. Agrawal, M. Bushnell. In *Proc. of International Conference On VLSI Design*, S. 492 –497. 1999. (Zitiert auf Seite 19)

[MIC02] E. Marinissen, V. Iyengar, K. Chakrabarty. A set of benchmarks for modular testing of SOCs. In *Proc. of IEEE International Test Conference*, S. 519 – 528. 2002. (Zitiert auf Seite 61)

[MM95] S. Makar, E. McCluskey. Functional tests for scan chain latches. In *Proc. IEEE International Test Conference*, S. 606–615. IEEE, 1995. (Zitiert auf Seite 15)

[MMZ+01] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, S. Malik. Chaff: engineering an efficient SAT solver. In *Proc. of Design Automation Conference (DAC)*, S. 530 – 535. 2001. (Zitiert auf Seite 22)

[Moo98] G. Moore. Cramming More Components Onto Integrated Circuits. *In Proc. of the IEEE*, 86(1):82 –85, 1998. (Zitiert auf Seite 25)

[NGB92] S. Narayanan, R. Gupta, M. Breuer. Configuring multiple scan chains for minimum test time. In *Proc. of IEEE/ACM International Conference on Computer-Aided Design*, S. 4 –8. IEEE/ACM, 1992. (Zitiert auf Seite 14)

[Par07] T. Parr. *The Definitive ANTLR Reference Guide*. Oreilly and Associate Series. O'Reilly Vlg. GmbH & Company, 2007. (Zitiert auf Seite 49)

[PH04] D. A. Patterson, J. Hennessy. *Computer Organization and Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004. (Zitiert auf Seite 9)

[PO90] K. Parker, S. Oresjo. A language for describing boundary-scan devices. In *Proc. of IEEE International Test Conference*, S. 222 –234. 1990. (Zitiert auf Seite 16)

[PSL80] M. Pease, R. Shostak, L. Lamport. Reaching Agreement in the Presence of Faults. *J. ACM*, 27(2):228–234, 1980. (Zitiert auf Seite 19)

[Rot66] J. P. Roth. Diagnosis of Automata Failures: A Calculus and a Method. *IBM Journal of Research and Development*, 10(4):278 –291, 1966. (Zitiert auf Seite 21)

[RTKM04] J. Rajski, J. Tyszer, M. Kassab, N. Mukherjee. Embedded deterministic test. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(5):776 – 792, 2004. (Zitiert auf Seite 14)

[Sch88] M. H. Schulz. *Testmustergenerierung und Fehlersimulation in digitalen Schaltungen mit hoher Komplexität*, Band 173 von *Informatik-Fachberichte*. Springer, 1988. (Zitiert auf Seite 20)

[Sch08] U. Schöning. *Theoretische Informatik - kurz gefasst*. Spektrum Hochschultaschenbücher. Spektrum Akademischer Verlag, 2008. (Zitiert auf Seite 9)

[Sha49] C. E. Shannon. The synthesis of two-terminal switching circuits. *Bell Systems Technical Journal*, 28:59–98, 1949. (Zitiert auf Seite 23)

[SPM92] J. Schafer, F. Policastri, R. McNulty. Partner SRLs for improved shift register diagnostics. In *Digest of Papers VLSI Test Symposium, 1992*, S. 198 –201. IEEE, 1992. (Zitiert auf Seite 15)

[Sta86] C. H. Stapper. On yield, fault distributions, and clustering of particles. *IBM Journal of Research and Development*, 30(3):326 –338, 1986. (Zitiert auf Seite 9)

[TED10] D. Tille, S. Eggersgluss, R. Drechsler. Incremental Solving Techniques for SAT-based ATPG. *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(7):1125 –1130, 2010. (Zitiert auf Seite 21)

[Tri84] E. Trischler. Design for Testability Using Incomplete Scan Path and Testability Analysis. In *Siemens Forsch.- u. Entwickl.- Ber. Bd. 13, Nr2*. 1984. (Zitiert auf Seite 14)

[Tse68] G. S. Tseitin. On the complexity of derivation in the propositional calculus. *Zapiski nauchnykh seminarov LOMI*, 8:234–259, 1968. (Zitiert auf Seite 23)

[UB73] E. G. Ulrich, T. Baker. The concurrent simulation of nearly identical digital networks. In *Proc. of IEEE/ACM Design Automation Conference (DAC)*, S. 145–150. IEEE/ACM, 1973. (Zitiert auf Seite 21)

[VM03] E. Volkerink, S. Mitra. Efficient seed utilization for reseeding based compression. In *Proc. of VLSI Test Symposium*, S. 232 – 237. IEEE, 2003. (Zitiert auf Seite 14)

[WA73] M. Williams, J. Angell. Enhancing Testability of Large-Scale Integrated Circuits via Test Points and Additional Logic. *Transactions on Computers*, 22(1):46–60, 1973. (Zitiert auf Seite 12)

[WWW06] L. Wang, C. Wu, X. Wen. *VLSI Test Principles and Architectures: Design for Testability*. Systems on Silicon. Elsevier Science, 2006. (Zitiert auf Seite 9)

Alle URLs wurden zuletzt am 02. 02. 2013 geprüft.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Marcel Schaal)