ALBERT-LUDWIGS-UNIVERSITÄT
FREIBURG
INSTITUT FÜR INFORMATIK

Lehrstuhl für Rechnerarchitektur
Prof. Dr. Bernd Becker

Bit-parallele Simulation von resistiven Brückenfehlern

Studienarbeit

Bettina Claudia Braitling

Juni 2007

Inhaltsverzeichnis

1 Einleitung

Im heutigen Leben begegnet man auf Schritt und Tritt elektronischen Komponenten. Häufig geschieht dies unbewusst, denn von der früher einfachen Kaffeemaschine bis hin zur Luxus-Limousine finden sich in vielen Alltags-Gegenständen eingebettete Systeme. Diese Tendenz wird in Zukunft eher noch zunehmen, denn Rechner werden immer kleiner, leistungsfähiger und komplexer.

Doch je komplexer das System ist, desto fehleranfälliger kann es werden, und je kleiner es ist, desto größer können die Auswirkungen eines einzelnen Fehlers sein.

Die Häufigkeit einer bestimmten Sorte von Defekten hat in den letzten Jahren immer mehr zugenommen: Kurzschlüsse in Schaltkreisen. Schon Ende der achtziger Jahre konnten 50% aller in Schaltkreisen auftretenden Defekte auf Kurzschlüsse zurückgeführt werden [11]. Kurzschlüsse können durch Produktionsfehler oder Verunreinigungen ausgelöst werden, die mehrere Leitungen miteinander verbinden. Da die Leitungsdichte auf den Chips aufgrund der immer geringeren Größe zunimmt, steigt auch die Wahrscheinlichkeit für derartige Defekte [10] [12].

Solche Defekte können nicht nur bei der Produktion, sondern auch während des Betriebs entstehen. Im ersten Fall besteht die Chance, dass der betroffene Chip aussortiert und zur Fehleranalyse herangezogen werden kann. Der zweite Fall kann katastrophale Folgen haben.

Es ist also wichtig, sich bereits vor der Herstellung folgende Fragen zu stellen:

Wie wirkt sich der Defekt aus und können wir diese Auswirkungen feststellen? Oder arbeitet das System womöglich weiter, als ob alles in Ordnung wäre?

Man erhofft sich, mit Hilfe der Simulation Antworten auf diese Fragen zu erhalten. Um einen Defekt korrekt zu simulieren, benötigt man ein passendes Fehlermodell und ein Verfahren, das die Eigenschaften dieses Modells ausnutzen kann. In der vorliegenden Arbeit soll ein solches Verfahren für ein bestimmtes Fehlermodell präsentiert werden.

Zu Beginn wird das Fehlermodell des resistiven Brückenfehlers mit seinen besonderen Eigenschaften vorgestellt. Im Anschluss daran werden zwei Ansätze zur Simulation dieses Modells geschildert. Der eine Ansatz wird vertieft, wobei erneut mehrere Verfahren zur Simulation vorgestellt werden. Diese Verfahren werden miteinander verglichen, was dazu führt, dass nur eines davon in die Realität umgesetzt und in einen bereits existierenden Simulator eingebaut wird.

In Kapitel 5 werden die Ergebnisse verschiedener Simulationsläufe im Vergleich mit anderen Simulationsverfahren präsentiert. Schließlich werden die Ergebnisse dieser Arbeit zusammengefasst und ein Ausblick auf mögliche Erweiterungen und Verbesserungen des hier realisierten Ansatzes gegeben.

2 Resistive Brückenfehler

Ein Kurzschluss auf einem Chip ist ein Defekt, bei dem durch ein Stück Strom leitendes Material mehrere Leitungen des Chips miteinander verbunden werden. Im Bereich der Simulation stellt man dies durch einen *Brückenfehler* dar. Im Englischen bezeichnet man dies als „*Bridging Fault*" oder kurz „*Bridge*" [15].

Brückenfehler sind eine ungewollte Verbindung zwischen verschiedenen Leitungen eines Schaltkreises. Sie können sowohl Leitungen innerhalb eines Gatters (engl. *Intra-Gate*) als auch zwischen verschiedenen Gattern (engl. *Inter-Gate*) betreffen. Dabei können zwei oder mehr Leitungen miteinander verbunden werden. Brückenfehler führen unter Umständen nicht nur zu fehlerhaften logischen Werten, sondern auch zu Zeitverzögerungen. Den letzteren Fall bezeichnet man auch als *Delay Fault*.

Die vorliegende Arbeit beschränkt sich auf statische Brückenfehler, die außerhalb der Gatter genau zwei Leitungen miteinander verbinden.

Zur Darstellung und Simulation dieses Fehlertyps existieren verschiedene Modelle, wie zum Beispiel Wired Logic [15], Voting [2], Biased Voting [3] und das resistive Modell. Auf letzteres soll hier genauer eingegangen werden, eine ausführliche Beschreibung findet sich in [9].

Die Besonderheit des resistiven Bridging Fault-Modells liegt darin, dass von einem zum Zeitpunkt der Simulation unbekannten Widerstand der Bridge ausgegangen wird. Damit steht es im Gegensatz zu anderen Bridging Fault-Modellen, die von einem im Voraus bekannten Widerstand ausgehen[1].

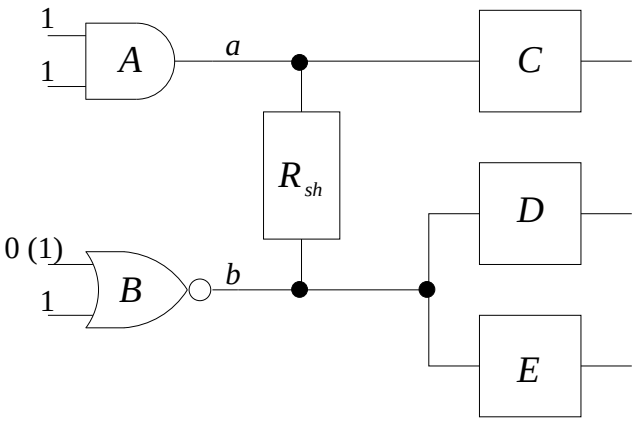

Abbildung 2.1: Schaltkreis mit resistivem Brückenfehler

[1] Die meisten anderen Modelle vernachlässigen den Widerstand sogar oder gehen davon aus, dass er so gering ist, dass sie ihn mit 0 Ω gleichsetzen können.

Abbildung 2.1 stellt einen solchen resistiven Brückenfehler dar, R_{sh} ist der Widerstand der Bridge, der einen beliebigen Wert annehmen kann. Die Gatter, deren Ausgänge durch die Bridge verbunden sind, bezeichnet man als *treibende Gatter*, die Gatter, die auf die Bridge folgen, als *nachfolgende Gatter*.

Der wesentliche Vorteil des resistiven Modells liegt in seiner Realitätsnähe. Ein Kurzschluss kann auf verschiedene Weise entstehen, zum Beispiel durch Produktionsfehler oder Verunreinigung. Der Widerstand dieser Verbindung hängt von mehreren Faktoren ab, wie dem Material und der Form oder der genauen Position auf dem Chip[2]. Dadurch ist auch in der Realität kein im Voraus bekannter Widerstand gegeben.

Je nach Widerstand kann sich eine Bridge unterschiedlich auswirken. Beträgt der Widerstand $0\,\Omega$, so entspricht die Bridge einer zusätzlichen Leitung zwischen den beiden betroffenen Signalen. In diesem Fall können die nicht-resistiven Modelle für eine realistische Simulation eingesetzt werden. Ist der Widerstand praktisch unendlich groß, so arbeitet der Schaltkreis korrekt, als ob gar kein Fehler vorläge.

Der Großteil der tatsächlich auftretenden Widerstände bei Kurzschluss-Defekten auf Chips liegt jedoch zwischen diesen beiden Extremfällen. Die meisten fallen in den Bereich zwischen $0\,\Omega$ und $500\,\Omega$ [10].

Entscheidend für die Auswirkungen der Bridge sind die Eigenschaften der davon betroffenen Gatter und die am Schaltkreis anliegende Belegung. Von diesen Auswirkungen ist es abhängig, ob der Fehler überhaupt entdeckt werden kann oder nicht.

2.1 Grenzwiderstand

Der *Grenzwiderstand* oder *kritische Widerstand* entscheidet, welchen Wert ein auf die Bridge folgendes Gatter erkennt. Liegt der Widerstand der Bridge unter dem Grenzwiderstand, so tritt der Fehlerfall ein, das heißt am nachfolgenden Gatter wird der Fehlerwert erkannt. Liegt der Widerstand über dem Grenzwiderstand, so wird der Gutwert erkannt.

Der kritische Widerstand wird durch den Schnittpunkt des logischen Schwellenwertes des nachfolgenden Gatters und der Spannung im jeweiligen Signalknoten für den Widerstand der Bridge definiert[3].

In Abbildung 2.2 sind die Spannungsverläufe U_a und U_b für die Signalknoten a und b aus Abbildung 2.1 zu sehen. Die obere Kurve zeigt den Spannungsverlauf für die Eingangsbelegung 1101, die untere für die Belegung 1111. R_{sh} ist der Widerstand der Bridge. Th_C, Th_D und Th_E sind die Schwellenwerte der Gatter C, D und E.

Für $R_{sh} = 0\,\Omega$ liegt sowohl bei a als auch bei b eine Spannung an, die für beide Signale nahezu identisch ist. Mit zunehmenden Widerstand nähert sich U_a immer mehr V_{DD} an, wohingegen U_b gegen $0\,V$ geht.

An verschiedenen Punkten schneiden die Kurven die Schwellenwerte der Gatter. R_C bezeichnet beispielsweise den Widerstand, an dem U_a für die Belegung 1101 den Schwellenwert von Gatter C, Th_C, überschreitet. R_C ist damit der kritische Widerstand des Gatters C für die Belegung 1101.

[2] Wobei die Form des Defekts nach [10] einen größeren Einfluss auf den Widerstand hat als das Material.
[3] Der Grenzwiderstand wird in [9] formal definiert.

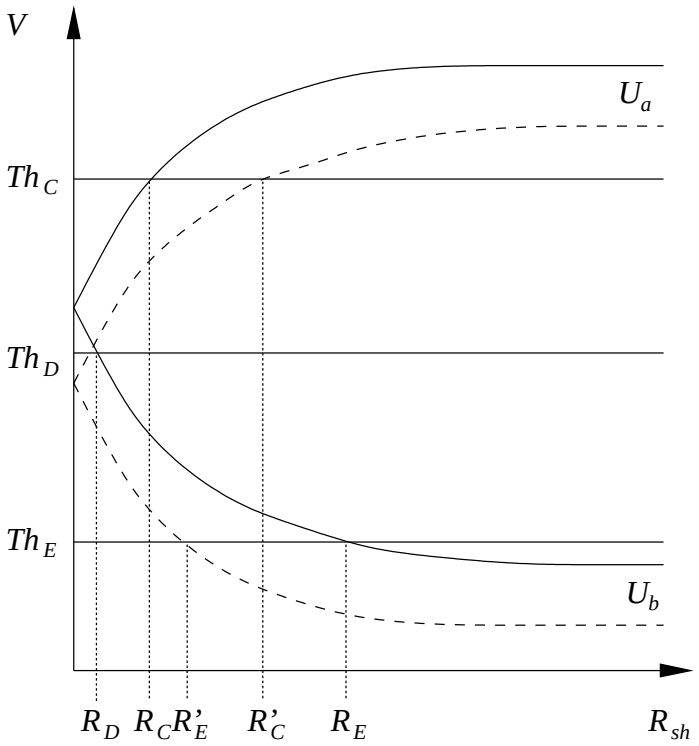

Abbildung 2.2: R_{sh}-V-Diagramm (nach [7])

Abhängig vom Widerstand der Bridge können an C für die Belegung 1101 also zwei Fälle eintreten:

$R_{sh} \leq R_C$: Die Spannung an C wird als 0 interpretiert (Fehlerwert 0/1).

$R_{sh} > R_C$: Die Spannung an C wird als 1 interpretiert.

An dieser Stelle sei darauf hingewiesen, dass in dieser Arbeit folgende Schreibweise verwendet wird, um Fehlerwerte darzustellen: Vorne steht der Wert im Fehlerfall, hinten der Gutwert.

In Abbildung 2.2 ist auch zu erkennen, dass nicht alle Gatter den gleichen Schwellenwert und damit den gleichen Grenzwiderstand haben. Selbst wenn zwei Gatter den gleichen Schwellenwert haben, kann sich ihr Grenzwiderstand unterscheiden, wenn sie nicht Nachfolger des selben treibenden Gatters sind. Die Grenzwiderstände würden in diesem Fall von verschiedenen Spannungsverläufen bestimmt werden. Dagegen haben zwei Gatter, die auf das selbe treibende Gatter folgen und den gleichen Schwellenwert haben, auch den gleichen kritischen Widerstand. Ihre kritischen Widerstände würden durch die selbe Spannungskurve definiert werden.

Es kann auch vorkommen, dass es gar keinen kritischen Widerstand gibt: Th_D, der Schwellenwert von Gatter D in Abbildung 2.2 besitzt für die Belegung 1111 keinen Schnittpunkt mit U_b. Für diese Belegung wird am betroffen Eingang von Gatter D der Gutwert 0 erkannt, unabhängig vom Widerstand der Bridge.

Andere Belegungen an den Eingängen der Gatter, die den Brückenfehler treiben, führen außerdem zu anderen Spannungsverläufen. Daraus ergeben sich auch andere kritische Widerstände für die nachfolgenden Gatter. Die untere Spannungskurve in Abbildung 2.2 steht für Belegung 1111. Wie zu erkennen, schneidet U_a Th_C an einer anderen Stelle als für die Belegung 1101. Der resultierende kritische Widerstand wird als R'_C bezeichnet.

Der kritische Widerstand ist also nicht nur von den Eigenschaften des jeweiligen betroffenen Gatters, sondern auch von der aktuellen Belegung abhängig.

Näher soll hier nicht auf die Berechnung des kritischen Widerstands eingegangen werden, die dazu benötigten Gleichungen befinden sich in [9].

2.2 Analoges Entdeckbarkeitsintervall

Nach der Berechnung des kritischen Widerstands R_C liegt am betroffenen Gattereingang ein Intervall $[0, R_C]$ vor. Befindet sich der Widerstand der Bridge innerhalb dieses Intervalls, so kann – zumindest an diesem Ort im Schaltkreis – der Fehlereffekt beobachtet werden. Außerhalb des Intervalls tritt kein Fehlereffekt auf.

Wie bereits erwähnt, haben nicht alle nachfolgenden Gatter zwingenderweise den gleichen Grenzwiderstand. Dementsprechend liegen auch an den Gattern unterschiedliche Intervalle an. Bei einer Simulation werden diese Intervalle propagiert. Dabei kann es vorkommen, dass sie rekonvergieren.

An dem in Abbildung 2.3 gezeigten Beispiel sieht man, wie die Intervalle propagiert werden können. $[0, 250] : 1/0$ bedeutet, dass im Intervall $[0, 250]$ der Fehlerwert 1 anliegt,

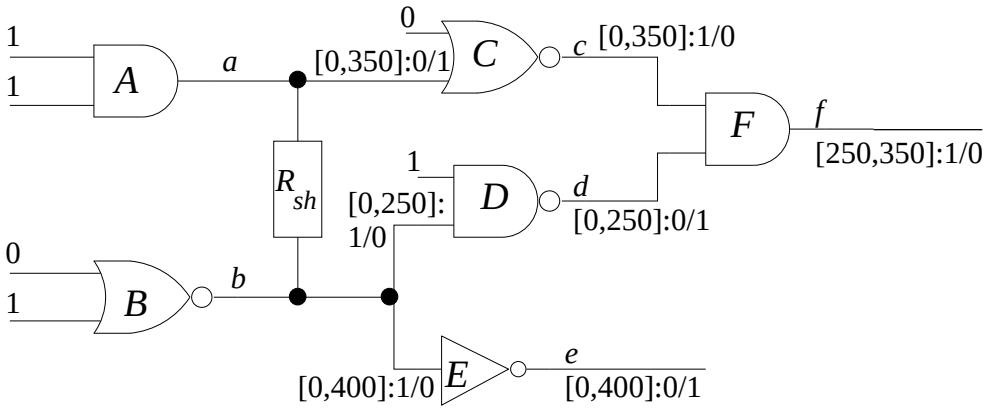

Abbildung 2.3: Simulation der Intervalle

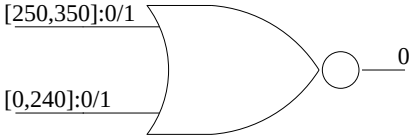

Abbildung 2.4: Auslöschung

außerhalb des Intervalls liegt der Gutwert 0 vor. Entsprechendes gilt für die anderen Intervalle.

Am Gatter F liegen zwei verschiedene Intervalle an, einmal $[0, 250] : 0/1$ und $[0, 350] : 1/0$. Entsprechend der logischen Funktion des Gatters – in diesem Fall ein AND – wird das Differenzintervall der beiden Intervalle gebildet. Man erhält $[250, 350] : 1/0$ als neues Intervall am Ausgang von F.

Gelingt es, ein solches Intervall zu einem primären Ausgang des Schaltkreises zu propagieren, so bezeichnet man es als *analoges Entdeckbarkeitsintervall* (engl. *Analog Detectability Interval*, kurz ADI). Insbesondere können an einem einzelnen Ausgang auch mehrere solcher Intervalle vorliegen. Man bezeichnet dies auch als ADI-Menge[4]. Ebenso kann es vorkommen, dass Intervalle komplett ausgelöscht werden. Abbildung 2.4 zeigt ein Beispiel dafür.

An dem NOR-Gatter in Abbildung 2.4 liegen zwei Intervalle an, das Intervall $[250, 350] : 1/0$ und das Intervall $[0, 240] : 0/1$. Entsprechend der logischen Funktion des Gatters werden die beiden Intervalle miteinander geschnitten. Am Ausgang des Gatters ist nun kein Fehlereffekt mehr zu erkennen, der Gutwert 0 liegt an.

[4] Eine formale Definition von ADI findet sich in [9].

3 Simulation

Um die ADI zu simulieren, sind verschiedene Methoden denkbar. Ein Ansatz ist die *Intervall-basierte Simulation*, ein anderer die *Segmentierung*.

3.1 Intervall-basierte Simulation

Wie in Abbildung 2.3 zu sehen, können die ADI durch Intervalle berechnet werden. Rekonvergieren die Intervalle, so werden mit Hilfe von Mengenoperationen neue Intervalle berechnet. Dieser Ansatz wird ausführlich in [14] besprochen, wo auch PROBE, ein Intervall-basierter Simulator, vorgestellt wird.

Die Berechnung der Mengenoperationen ist sehr aufwändig und benötigt viel Rechenzeit. Um die Simulation zu beschleunigen kommt in PROBE ein Pseudo-PPSFP Algorithmus zum Einsatz.

PPSFP ist die Abkürzung für *„Parallel Pattern Single Fault Propagation"* und steht für die *Muster-* oder *Pattern-parallele Simulation*. Diese Methode diente ursprünglich zur Beschleunigung von stuck-at Fault-Simulatoren, ist aber auch in anderen Bereichen einsetzbar.

Bei der PPSFP entspricht jede Position eines Bitwortes einem am Schaltkreis anliegendem *Muster* oder *Pattern*. Statt also pro Simulationsschritt nur ein Muster zu simulieren, kann man mit der PPSFP der Bitbreite entsprechend viele Muster auf einmal betrachten.

PROBE ist jedoch nicht in der Lage, die ADI Bit-parallel zu simulieren, da es sich dabei um Intervalle handelt. Stattdessen wird jede Bitposition mit dem zu dem Muster gehörenden Intervall verknüpft.

Gleiche Muster würden nun dazu führen, dass wiederholt die gleichen Berechnungen durchgeführt werden. Um dies zu vermeiden, verweist PROBE bei gleichen Muster auf die vorherige Berechnung für dieses Muster. Auf diese Weise wird eine gewisse Beschleunigung ermöglicht.

Jedoch können auch bei PROBE unnötige Berechnungen durchgeführt werden. Zu Beginn der Fehlersimulation wird meistens ein viel größerer Widerstandsbereich betrachtet, als tatsächlich propagiert werden kann. Außerdem ist es möglich, wie in Abbildung 2.4 zu sehen, dass Intervalle komplett ausgelöscht werden. Bei PROBE – und dies gilt allgemein für die Intervall-basierte Simulation – wird also ein viel größerer Widerstandsbereich simuliert, als tatsächlich benötigt wird.

3.2 Segmentierung

Segmentierung oder *Sectioning* wurde erstmals in [13] vorgeschlagen, um resistive Brückenfehler ohne aufwändige Mengenoperationen zu simulieren.

Für das Sectioning ist es nötig, alle lokalen Belegungen, die den Fehler einstellen, zu betrachten. Als *lokale Belegung* wird diejenige Belegung bezeichnet, die direkt an den treibenden Gattern der Bridge anliegt. Sorgt eine Belegung dafür, dass an einem der von der Bridge verbundenen Signalknoten der Wert 1 und an dem anderen der Wert 0 anliegt, so wird dadurch der Fehler *aktiviert* oder *eingestellt*.

Für alle lokalen einstellenden Muster wird für jedes nachfolgende Gatter der jeweilige kritische Widerstand berechnet. Die Grenzwiderstände werden der Größe nach sortiert. Man erhält auf diese Weise eine Folge von *Segmenten* oder *Sections*:

$$[0, R_{min}] = [0, R_1]\,,\ [R_1, R_2]\,,\ ...,\ [R_{n-1}, R_n]\,,\ [R_n, \infty) = [R_{max}, \infty)$$

Dabei ist n die Anzahl an Widerständen.

Die Anzahl an Widerständen ist beschränkt. Sei m die Anzahl an nachfolgenden Gattern und k die Anzahl an einstellenden Belegungen. Für jedes Gatter gibt es unter jeder Belegung höchstens einen Grenzwiderstand. Es gilt also

$$n \leq m \cdot k$$

Darüber hinaus ist n auch durch die Anzahl an verschiedenen Schwellenwerten der im Schaltkreis vorkommenden Gatter beschränkt. Wie bereits erwähnt, haben zwei Gatter mit gleichen Schwellenwert den gleichen kritischen Widerstand, wenn sie Nachfolger des selben treibenden Gatters sind.

Die Section $[R_{max}, \infty)$ wird der Vollständigkeit halber hinzugefügt. In diesem Intervall liegen nur die Gutwerte vor, da darin sämtliche kritischen Widerstände überschritten wurden. Für eine Fehlersimulation ist es deswegen nur von geringem Interesse.

Innerhalb einer Section werden von den betroffenen Gattereingängen eindeutige und feste Werte interpretiert. Am Beispiel aus Abbildung 2.1 und dem Graphen in 2.2 lässt sich dies verdeutlichen:

Es gibt insgesamt vier einstellende Belegungen: 0000, 1101, 1101 und 1111. Die Belegungen 1101 und 1110 sind in diesem Fall äquivalent zueinander, da sie zu der gleichen Anzahl an aktiven p- beziehungsweise n-Transistoren in den treibenden Gattern führen. Es reicht also aus, eine dieser Belegungen zu betrachten.

Für die Belegungen 1101 und 1111 ergeben sich die Grenzwiderstände R_D, R_C, R'_E, R'_C und R_E. Wie in Kapitel 2.1 erwähnt, gibt es für die Belegung 1111 keinen Schnittpunkt zwischen U_b und Th_D, so dass für diese Belegung am Gatter D kein Fehlereffekt beobachtet werden kann. Für die Belegung 0000 kann zur Vereinfachung angenommen werden, dass es für keines der nachfolgenden Gatter einen Grenzwiderstand gibt und damit der Fehler durch diese Belegung zwar eingestellt wird, aber keine Auswirkungen hat.

Sortiert man die kritischen Widerstände der Größe nach, ergeben sich folgende Sections:

$$[0, R_D]\,,\ [R_D, R_C]\,,\ \left[R_C, R'_E\right]\,,\ \left[R'_E, R'_C\right]\ \left[R'_C, R_E\right]\,,\ [R_E, \infty)$$

Es gilt $R_E = R_{max}$

Betrachtet man nun für die Belegung 1101 den betroffenen Eingang des Gatters C, so liegen innerhalb der Sections folgende Werte an:

$$\begin{aligned}
[0, R_D] &: 0/1 \\
[R_D, R_C] &: 0/1 \\
\left[R_C, R'_E\right] &: 1 \\
\left[R'_E, R'_C\right] &: 1 \\
\left[R'_C, R_E\right] &: 1 \\
[R_E, \infty) &: 1
\end{aligned}$$

Nur in den ersten beiden Sections lässt sich der Fehlerwert 0 an Gatter C beobachten. In den übrigen Sections wurde der Grenzwiderstand des Gatters überschritten. Dementsprechend wird nur der Gutwert 1 erkannt.

Es ist offensichtlich, dass sich innerhalb einer Section der anliegende Wert nicht ändern kann. Wäre etwas anderes der Fall, so müsste es einen weiteren kritischen Widerstand geben, der zwischen den Grenzen dieses Segments liegt. Letzteres ist jedoch nicht möglich, da zur Berechnung der einzelnen Sections alle kritischen Widerstände verwendet werden.

Jede Section kann nun einzeln simuliert werden. Werden im Anschluss an diese Simulation alle entdeckten Sections zu einem Intervall beziehungsweise zu einer Intervall-Menge vereinigt, so erhält man wieder ein ADI oder eine ADI-Menge. Diese stimmen genau mit den ADI überein, die man bei der Simulation mit Hilfe von Mengenoperationen erhält.

Gegenüber den Mengenoperationen hat die Segmentierung wesentliche Vorteile: Innerhalb einer Section reduziert sich der Brückenfehler auf eine Reihe von stuck-at Faults an den betroffenen Eingängen der nachfolgenden Gatter. Stuck-at Faults sind das klassische Fehlermodell, multiple stuck-at Faults sind eine Erweiterung davon. Die Simulation von stuck-at Faults wurde gründlich untersucht [1]. Es gibt eine Reihe von Techniken, um dieses Simulation zu beschleunigen. Eine davon ist die Bit-parallele Simulation.

Bei der Bit-parallelen Simulation entspricht jede Position eines Bitwortes einem Simulationswert. Statt also mehrere Simulationsschritte durchzuführen, kann man der Bitbreite entsprechend viele Schritte auf einmal berechnen. Dadurch reduziert sich die Rechenzeit erheblich.

Eine Anwendung der Bit-parallelen Simulation ist die bereits erwähnte PPSFP. Hierbei werden mehrere Muster parallel simuliert.

Eine andere Anwendung ist die *Fehler-parallele Simulation*. Dabei steht jede Position eines Bitwortes für einen zu simulierenden Fehler, es werden also mehrere Fehler gleich-

zeitig simuliert. Im Englischen bezeichnet man dies als *„Single Pattern Parallel Fault Propagation"* oder kurz SPPFP.

Da die Bit-parallele Simulation von stuck-at Faults möglich ist, ist sie auch bei Sections anwendbar. Shinogi et al. haben in [13] gezeigt, dass bereits durch Sectioning allein die Simulation der ADI beschleunigt werden kann. Jedoch haben sie die Bit-Parallelität nicht ausgenutzt. Es ist also noch zu vermuten, dass durch die Kombination von Sectioning und Bit-paralleler Simulation eine noch größere Beschleunigung möglich ist.

Es werden nun zwei Ansätze zur Simulation von resistiven Brückenfehlern mit Hilfe von Sectioning vorgestellt, in die sich die Bit-parallele Simulation integrieren lässt: *Bridge-orientierte Simulation* und *Section-orientiere Simulation*.

3.2.1 Bridge-orientierte Simulation

Bei diesem Verfahren entspricht ein zu simulierender Fehler einer Bridge beziehungsweise einem Brückenfehler. Jede Bridge enthält mehrere Sections.

Um die Bridge Bit-parallel zu simulieren, wird jeder Section eine Position eines Bitwortes zugewiesen. Für ein einzelnes Muster können also in einem Simulationsschritt mehrere Sections simuliert werden. Überschreitet die Anzahl der kritischen Widerstände die Bitbreite nicht, ist es sogar möglich, den gesamten Widerstandsbereich für dieses Muster auf einmal zu simulieren. Auch eine separate Gutsimulation wird überflüssig, wenn man als letzte Section das Intervall $[R_{max}, \infty)$ hinzufügt.

Ist die Anzahl an kritischen Widerständen jedoch größer als die Bitbreite, steigt der Simulationsaufwand:

Offensichtlich werden in diesem Fall für jedes am Schaltkreis anliegende Muster mehrere Simulationsschritte benötigt, um alle Sections zu simulieren. Wenn die Anzahl an Sections sich nicht ohne Rest durch die Bitbreite teilen lässt, bleiben einige Kapazitäten ungenutzt. Dies ist besonders dann der Fall, wenn die Bitbreite für das Intervall $[R_{max}, \infty)$ nicht mehr ausreicht. Somit würde eine separate Gutsimulation erforderlich, was einen wesentlichen Vorteil der Section-parallelen Simulation zunichte machen würde. Da sich die Anzahl an benötigten Simulationsschritten pro Brückenfehler nur schlecht im Voraus abschätzen lässt, würde es sich im Falle einer Implementierung sogar anbieten, die Gutsimulation von der Fehlersimulation abzukoppeln.

Auch die Verwaltung der einzelnen Brückenfehler wäre kompliziert: Es muss ohnehin vermerkt werden, ob die Bridge im gesamten Widerstandsbereich, nur in einem Teil davon oder überhaupt nicht entdeckt wurde. Doch nun käme noch hinzu, ob bereits alle Sections simuliert wurden, und wenn nicht, welche noch zu simulieren sind.

Außerdem werden bei der Bridge-orientierten Simulation, ähnlich wie beim Intervall-basierten Ansatz, unnötige Berechnungen durchgeführt werden. Wird eine Bridge für ein Muster simuliert, so werden in der Regel nicht alle Sections von diesem Muster aktiviert. Hierzu ein Beispiel:

Wie in Kapitel 3.2 bereits ausgeführt, gibt es für die Bridge aus Abbildung 2.1 sechs Sections. Betrachtet man nun den Spannungsverlauf in Abbildung 2.2 für die Belegung 1111, so werden von dieser Belegung nur die Sections $[0, R_D]$, $[R_D, R_C]$, $[R_C, R'_E]$ und $[R'_E, R'_C]$ aktiviert. In der Section $[R'_C, R_E]$ wurden bereits alle kritischen Widerstände

für dieses Muster überschritten, die Section $[R_E, \infty)$ wird ohnehin von keinem Muster aktiviert.

Man kann nun entweder die inaktiven Sections ebenfalls simulieren. Oder man versucht, die Simulation auf die aktivierten Sections zu beschränken. Beides hat Nachteile: Simuliert man die inaktiven Sections, so werden unnötig Kapazitäten verbraucht. Simuliert man sie nicht, so steigt erneut der Verwaltungsaufwand.

3.2.2 Section-orientierte Simulation

Ein anderer Ansatz zur Bit-parallelen Simulation resistiver Brückenfehler ist die Section-orientierte Simulation. In diesem Fall entsprechen die einzelnen Sections den zu simulierenden Fehlern. Bei jeder Fehlersimulation wird also eine einzelne Section betrachtet. Um Auskunft über eine zugehörige Bridge zu erhalten, müssen alle zugehörigen Sections simuliert werden. Bezogen auf das Beispiel aus Kapitel 3.2 bedeutet dies, dass man, um die Bridge zwischen den Gattern A und B aus Abbildung 2.1 zu simulieren, die Sections $[0, R_D]$, $[R_D, R_C]$, $[R_C, R'_E]$, $[R'_E, R'_C]$, und $[R'_C, R_E]$ jede für sich betrachten muss. Die Bridge selbst ist jedoch nicht Ziel der Simulation.

Eine Simulation der Section $[R_E, \infty)$ ist nicht notwendig, wenn stattdessen eine Gutsimulation stattfindet.

Allgemein ist eine Simulation von nicht aktivierten Sections nicht nötig. Werden die Sections aus dem Beispiel in Kapitel 3.2 für die Belegung 1111 simuliert, so würden nur die Sections $[0, R_D]$, $[R_D, R_C]$, $[R_C, R'_E]$ und $[R'_E, R'_C]$ betrachtet werden. Die Section $[R'_C, R_E]$ wird, wie bereits erwähnt, von diesem Muster nicht aktiviert und muss deswegen auch nicht simuliert werden.

Die Fehlerverwaltung gestaltet sich bei der Section-orientierten Simulation geradezu einfach: Es reicht aus, für die einzelnen Sections zu vermerken, ob sie entdeckt wurden oder nicht. Der Zustand des Brückenfehlers, zu dem sie gehören, lässt sich dann daraus ableiten.

3.2.2.1 Section-orientierte und Bit-parallele Simulation kombiniert

Da sich jede Section als eine Reihe von stuck-at Faults simulieren lässt, kann die Section-orientierte Simulation sehr gut mit der SPPFP oder der PPSFP kombiniert werden.

Bei der SPPFP ist es möglich, für ein Muster mehrere Sections gleichzeitig zu betrachten. Da die Sections nicht zwingend mit der entsprechenden Bridge zusammenhängen, können dabei auch Sections von verschiedenen Brückenfehlern auf einmal betrachtet werden. Außerdem wäre es, ähnlich wie bei der Bridge-orientierten Simulation, unter Umständen möglich, alle Sections einer Bridge gleichzeitig zu simulieren.

Bei der PPSFP wird nur eine Section betrachtet, dafür werden mehrere Muster gleichzeitig simuliert. Durch das gleichzeitige Betrachten verschiedener Muster erhöht sich die Wahrscheinlichkeit, dass ein Muster dabei ist, welches die Section aktiviert. Damit steigt auch die Wahrscheinlichkeit, dass die Section entdeckt wird. Wenn die Anzahl an möglichen Eingangsbelegungen des Schaltkreises unter der Bitbreite liegt, ist es sogar möglich,

in einem Simulationsschritt eine erschöpfende Simulation für die jeweilige Section durchzuführen.

Der Nachteil der PPSFP besteht darin, dass nicht alle verwendeten Muster notwendigerweise die betrachtete Section aktivieren. Für diese Bitpositionen würde eine Gutsimulation durchgeführt werden, da keine stuck-at Faults eingebaut werden. Die dazugehörenden Simulationsschritte sind jedoch notwendig, da sich an anderen Bitpositionen möglicherweise aktivierende Muster befinden. Logische Operationen auf Bitworten verbrauchen allerdings weder viel Rechenzeit noch Speicherplatz, so dass diese Redundanz der PPSFP bedenkenlos in Kauf genommen werden kann.

Bei der Simulation von stuck-at Faults hat sich gezeigt, dass die PPSFP eine größere Beschleunigung erbringt als die SPPFP[1]. Es ist zu vermuten, dass das gleiche auch bei der Section-orientierten Simulation gilt.

3.2.3 Bridge-orientierte und Section-orientierte Simulation im Vergleich

Der wesentliche Unterschied der beiden vorgestellten Ansätze liegt in ihrem Blick auf den zu simulierenden Fehler. Bei der Bridge-orientierten Simulation ist dies der Brückenfehler selbst, bei der Section-orientierten Simulation die einzelne Section. Diese beiden Betrachtungsweisen führen zu einigen Unterschieden in Hinsicht auf die Simulation und den Simulationsaufwand.

Im Idealfall benötigt man bei der Bridge-orientierten Simulation nur einen einzigen Simulationsschritt, um die Gutsimulation und die Fehlersimulation durchzuführen. Doch es kann ebenso der Fall eintreten, dass man mehrere Schritte benötigt, um einen einzigen Fehler beziehungsweise eine Bridge zu simulieren.

Bei der Section-orientierten Simulation benötigt man dagegen pro Section oder Fehler garantiert nur einen Simulationsschritt für die Fehlersimulation.

Außerdem werden, wie in Kapitel 3.2.1 ausgeführt, bei der Bridge-orientierten Simulation unnötig Sections simuliert, es sei denn, man nimmt zusätzlichen Verwaltungsaufwand in Kauf. Bei der Section-orientierten Simulation werden nicht aktivierte Sections einfach nicht simuliert, wie in Kapitel 3.2.2 beschrieben.

Darüber hinaus gestaltet sich die Fehlerverwaltung bei der Bridge-orientierten Simulation sehr komplex, was sich direkt auf eine Implementierung übertragen würde. Im Gegensatz dazu ist die Fehlerverwaltung des Section-orientierten Ansatzes einfach.

Um eine gesamte Bridge zu betrachten, müssen bei der Section-orientierten Simulation mehrere Fehler simuliert werden. Hier scheint die Bridge-orientierte Simulation einen Vorteil zu bieten, denn dabei ist es prinzipiell möglich, eine gesamte Bridge auf einmal zu simulieren, auch wenn dieser Fall vielleicht selten eintritt. Kombiniert man jedoch den Section-orientierten Ansatz mit der SPPFP, so ist es auch hier unter Umständen möglich, den kompletten Widerstandsbereich eines Brückenfehlers in einem Simulationsschritt abzudecken.

Wird die Section-orientierte Simulation mit der PPSFP kombiniert, können mehrere Muster gleichzeitig betrachtet werden, wohingegen bei der Bridge-orientierten Simulation

[1] Siehe dazu S. 155 in [16]. Wunderlich verwendet nicht den Begriff der SPPFP, sondern bezeichnet dies als parallele Fehlersimulation.

immer nur das ADI für ein einzelnes Muster berechnet wird. Führt dieses Muster nicht zur Entdeckung des Fehlers, so müssen weitere Muster simuliert werden, wodurch sich der gesamte Verwaltungsaufwand wiederholt.

Es hat sich somit gezeigt, dass der Section-orientierte Ansatz einige Vorteile gegenüber der Bridge-orientierten Simulation hat. Die Fehlerverwaltung – und damit vermutlich auch die Implementierung – gestaltet sich einfacher. Rechenkapazitäten können besser ausgenutzt werden, da keine inaktiven Sections simuliert werden müssen. Außerdem ist die Section-orientierte Simulation sehr gut mit bereits existierenden Bit-parallelen Simulationsverfahren kombinierbar, weswegen sie wahrscheinlich eine größere Beschleunigung erbringt als die Bridge-orientierte Simulation. Aufgrund dieser Vorteile wurde der Section-orientierte Ansatz für eine Realisierung gewählt.

4 Realisierung der Section-orientierten Simulation

Die Section-orientierte Simulation von resistiven Brückenfehlern lässt sich grundsätzlich in jeden Bit-parallelen Simulator einbauen, der zur Simulation multipler stuck-at Faults in der Lage ist. Außerdem sollte der Simulator es auch erlauben, stuck-at Faults gezielt an einzelnen Gattereingängen einzubauen. Der für diese Arbeit verwendete Simulator ist sowohl zur PPSFP als auch zur SPPFP fähig.

Die kritischen Widerstände werden nach den Gleichungen aus [9] berechnet. Damit unterscheidet sich der vorgestellte Simulator von anderen Ansätzen, die stattdessen im Vorfeld berechnete Tabellen verwenden, wie zum Beispiel PROBE [14] oder der Simulator aus [13].

Neben einigen Klassen zur Verwaltung der Brückenfehler und für die Berechnung der kritischen Widerstände und der lokalen aktivierenden Muster, ist vor allem die Datenstruktur zur Darstellung der einzelnen Brückenfehler interessant.

Die Ansprüche an diese Datenstruktur sind vielfältig: Einerseits sollte sie einen schnellen Zugriff auf die enthaltenen Daten liefern, aber andererseits auch nicht zu viel Speicherplatz benötigen. Um dies zu bewerkstelligen, wurde ein modularer Ansatz gewählt. Die komplette Datenstruktur besteht aus zwei Arten von Objekten, „Brückenknoten" und „Widerstandsknoten".

Ein Brückenknoten repräsentiert eine Bridge. Er enthält die Daten über den Ort der Bridge, eine Hash-Tabelle mit den lokalen einstellenden Belegungen und eine Liste der zugehörigen Widerstandsknoten.

Ein Widerstandsknoten repräsentiert eine einzelne Section. Er enthält die obere Grenze der jeweiligen Section und eine Matrix mit den Werten, die von den nachfolgenden Gattern in dieser Section erkannt werden. Dabei wird nur vermerkt, ob der Fehlerwert 1, der Fehlerwert 0 oder der Gutwert, hier dargestellt durch ein „G" vorliegt. Es wird nicht zwischen dem Gutwert 1 und dem Gutwert 0 unterschieden, da vor allem der Fehlerfall interessant ist. Wurde die Section nicht aktiviert, ist keine Fehlersimulation nötig. Außerdem können die Gutwerte auch aus den Daten einer Gutsimulation ermittelt werden.

In den Widerstandsknoten wird nur die obere Grenze der jeweiligen Section benötigt, da die untere Grenze durch den Widerstandsknoten gegeben ist, der in der Liste vorangeht. Der erste Knoten steht dementsprechend für die erste Section $[0, R_{min}]$. Für die Section $[R_{max}, \infty)$ gibt es keinen Widerstandsknoten. Für die Section-orientierte Fehlersimulation ist sie uninteressant, da sie, wie bereits erwähnt, einer Gutsimulation entspricht.

Weder im Brückenknoten noch in den Widerstandsknoten wird direkt auf die nachfolgenden Gatter verwiesen. Diese Informationen sind implizit mit dem Ort der Bridge gegeben.

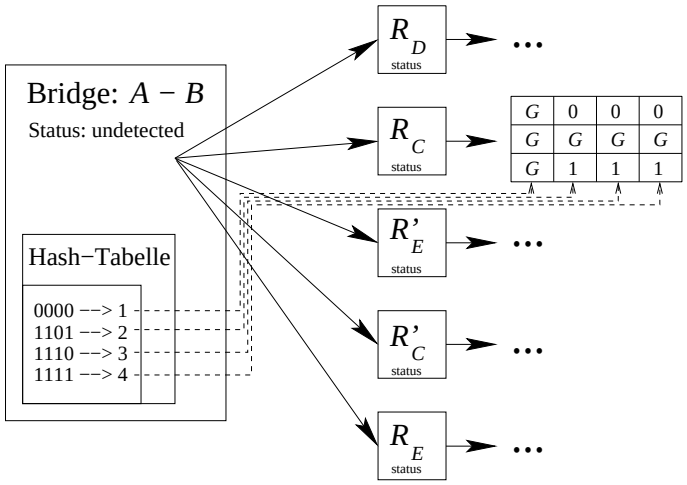

Abbildung 4.1: Datenstruktur

Um alle erkannten Werte dem passenden einstellenden Muster zuordnen zu können, muss es eine feste Abbildung geben. Diese Abbildung wird durch die Hash-Tabelle im jeweiligen Brückenknoten ermöglicht. Jedes darin gespeicherte Muster hat als Wert eine Zahl, die als Index auf eine Spalte in den Werte-Matrizen der zugehörigen Widerstandsknoten verweist. Die Zeilen dieser Matrizen stehen für die einzelnen nachfolgenden Gatter. Um für eines dieser Gatter die passende Zeile zu finden, wird die Position benötigt, die es in der Reihenfolge der nachfolgenden Gatter einnimmt, die durch die topologische Sortierung des Schaltkreises bestimmt wird. Dies lässt sich anhand von Abbildung 4.1 verdeutlichen.

Die Bridge aus Abbildung 2.1 hat – wie bereits in Kapitel 3.2 ausgeführt – vier einstellende Muster. Im Gegensatz zu den vorherigen Ausführungen müssen nun sowohl die Belegung 1101, als auch die Belegung 1110 gespeichert werden. Zwar sind diese Belegungen in Hinsicht auf das Verhalten der Bridge nach wie vor äquivalent, jedoch werden diese beiden lokalen Muster von unterschiedlichen Eingangsbelegungen des gesamten Schaltkreises hervorgerufen. Letzterer weist für diese Belegungen nicht notwendigerweise ebenfalls ein äquivalentes Verhalten auf.

Die einstellenden Muster führen zu fünf Grenzwiderständen[1].

Nun wird für Gatter C der erkannte Wert in der Section $[R_D, R_C]$ für das lokale Muster 1101 gesucht.

In der Hash-Tabelle des Brückenknotens hat das Muster 1101 den Wert „2". Gatter C steht nach der topologischen Reihenfolge des Schaltkreises bei den nachfolgenden Gattern an erster Stelle. Also findet man den gesuchten Wert in der ersten Zeile und zweiten Spalte der Matrix des Widerstandsknotens mit dem Wert von R_C. Nach Abbildung 4.1 würde an Gatter C also eine 0 anliegen, was auch den Ausführungen in Kapitel 3.2 entspricht.

In Tabelle 4.1 sind alle erkannten Fehlerwerte für die einzelnen Belegungen und Sections des Beispiels aufgeführt.

[1] Siehe Kap. 2.1 und Abb. 2.2.

Tabelle 4.1: Erkannte Fehlerwerte

Muster	Section	Fehlerwert C	D	E
0000	$[0, R_D]$	–	–	–
	$[R_D, R_C]$	–	–	–
	$[R_C, R'_E]$	–	–	–
	$[R'_E, R'_C]$	–	–	–
	$[R'_C, R_E]$	–	–	–
1101	$[0, R_D]$	0	1	1
	$[R_D, R_C]$	0	–	1
	$[R_C, R'_E]$	–	–	1
	$[R'_E, R'_C]$	–	–	1
	$[R'_C, R_E]$	–	–	1
1110	$[0, R_D]$	0	1	1
	$[R_D, R_C]$	0	–	1
	$[R_C, R'_E]$	–	–	1
	$[R'_E, R'_C]$	–	–	1
	$[R'_C, R_E]$	–	–	1
1111	$[0, R_D]$	0	–	1
	$[R_D, R_C]$	0	–	1
	$[R_C, R'_E]$	0	–	1
	$[R'_E, R'_C]$	0	–	–
	$[R'_C, R_E]$	0	–	–

```
Für jeden Brückenfehler in Fehlerliste {
   Erzeuge Brückenknoten B;
   Berechne Menge aller lokalen aktivierenden Pattern
      P = {p_1 ,...,p_n};
   Für alle p_i in P {
     Speichere p_i in Hashtabelle H von B mit Wert „i";
   }
   Lege Widerstandsknotenliste W in B an;
   Für alle p_i in H, i = 1,...,n {
      Sei G = {g_1 ,...,g_m} die Menge aller nachfolgenden Gatter;
      Für j = 1,...,m {
         Berechne kritischen Widerstand R_sh für g_j unter p_i;
         Suche Widerstandsknoten w_k mit R_sh in W {
            Wenn w_k vorhanden: {
               Speichere in [w_{k-1}, w_k] von g_j unter p_i erkannten Wert in M_{j,i};
            }
            Wenn nicht vorhanden: {
               Erzeuge neuen Widerstandsknoten w_k;
               Speichere R_sh in w_k;
               Lege Werte-Matrix M in w_k an;
               Füge w_k an entsprechender Stelle in W ein;
               Speichere in [w_{k-1}, w_k] von g_j unter p_i erkannten Wert in M_{j,i};
            }
         }
      }
   }
}
```

Abbildung 4.2: Pseudo-Code Preprocessing

Die erste Spalte enthält die aktivierenden Muster, die zweite Spalte die Sections. In der dritten, vierten und fünften Spalte von Tabelle 4.1 befinden sich die von den nachfolgenden Gattern C, D, und E erkannten Fehlerwerte für die entsprechende Belegung in der jeweiligen Section. „1" steht für den Fehlerwert 1, „0" steht für den Fehlerwert 0 und „–" bedeutet, dass am jeweiligen Gatter der Gutwert anliegt.

4.1 Preprocessing

Im Vorfeld der Simulation wird die Datenstruktur erzeugt und die entsprechenden Daten werden darin abgespeichert. Dies ist in Abbildung 4.2 als Pseudo-Code dargestellt.

Für jeden zu simulierenden Brückenfehler wird ein Brückenknoten angelegt. Anschließend werden die lokalen einstellenden Muster ermittelt und in der Hash-Tabelle gespeichert.

Für jedes Muster und für jedes nachfolgende Gatter wird nun der zugehörige Grenzwiderstand berechnet. Gibt es bereits einen passenden Widerstandsknoten, so wird dieser nun weiter bearbeitet. Gibt es noch keinen entsprechenden Widerstandsknoten in der

Widerstandsknotenliste, so wird ein neuer Knoten erzeugt und gleich an der richtigen Position in der Liste eingefügt. Schließlich werden die in den Sections von den nachfolgenden Gattern erkannten Werte in den jeweiligen Matrizen gespeichert.

4.2 Simulationsablauf

Wird eine Section simuliert, so wird zuerst überprüft, ob das lokal an der zugehörigen Bridge anliegende Muster in der jeweiligen Hash-Tabelle zu finden ist. Ist dies nicht der Fall, so wurde die Bridge nicht aktiviert, es muss kein Fehler simuliert werden. Befindet sich das Muster in der Hash-Tabelle, so wird nun für jedes einzelne Gatter überprüft, ob ein Fehlerwert vorliegt oder ob der Gutwert erkannt wird. Wurde die Section aktiviert, das heißt liegt ein Fehlerwert vor, so wird dieser Fehlerwert als stuck-at Fault am betroffenen Gattereingang eingebaut. Ansonsten wird mit dem Gutwert weiter simuliert.

Nachdem der Brückenfehler auf diese Weise in den Schaltkreis eingebaut wurde, führt der Simulator eine Fehlersimulation durch. Wird dabei ein Fehler entdeckt, so wird dies in den passenden Widerstandsknoten und Brückenknoten vermerkt.

In Abbildung 4.3 ist die Simulation für die PPSFP als Pseudo-Code zu sehen. Für die SPPFP ist es lediglich notwendig, die beiden äußersten Schleifen des Algorithmus entsprechend abzuändern.

```
Für alle zu simulierenden Widerstandsknoten w_i {
   Für jedes lokal anliegende Pattern p_j, j = 1,...,Bitbreite {
      p_j in Hash-Map H von zu w_i gehörenden Brückenknoten B? {
         Wenn 'Ja': {
            Sei G = {g_1,...,g_m} die Menge aller nachfolgenden Gatter;
            Für k = 1,...,m {
               Überprüfe Matrix-Eintrag M_{k,j} {
                  Wenn '1': Baue stuck-at-1 an g_k ein;
                  Wenn '0': Baue stuck-at-0 an g_k ein;
               }
            }
         }
      }
   }
   Führe Fehlersimulation durch;
   Wurde Section entdeckt? {
      Wenn 'Nein': Setze Status von w_i auf 'Unentdeckt';
       Wenn 'Ja': {
          Setze Status von w_i auf 'Entdeckt';
          Setze Status von B auf 'Section entdeckt';
      }
   }
}
```

Abbildung 4.3: Pseudo-Code Simulation für PPSFP

5 Ergebnisse

Im Folgenden werden nun einige Ergebnisse der Section-orientierten Simulation präsentiert. Einige dieser Ergebnisse werden auch in [8] vorgestellt.

Auf die Simulation von Feedback-Fehlern wurde bewusst verzichtet, da ihre Behandlung eine besondere Vorgehensweise erforderlich macht und den Umfang dieser Arbeit sprengen würde. Es existiert jedoch bereits ein Verfahren, um auch diesen Fehlertyp für resistive Brückenfehler zu simulieren [5]. Dieses Verfahren ließe sich grundsätzlich auch auf die Segmentierung übertragen.

Es wurden die ISCAS85 und ISCAS89 Benchmark-Schaltkreise simuliert, alle Messungen fanden auf einem Linux-Rechner, ausgestattet mit einem 2 GHz AMD Opteron und 4 GB RAM, statt. Die simulierten Brückenfehler lagen im Vorfeld bereits vor, die verwendeten ATPG-Muster stammen aus [6]. Für den Schaltkreis c6288 lagen keine ATPG-Muster vor, weswegen in diesem Fall keine Simulation stattfand.

5.1 Fehlerüberdeckung

Um die Qualität eines Testmusters zu untersuchen, ist es wichtig zu wissen, welche *Fehlerüberdeckung* (engl. *Fault Coverage*) es erzielt. Als Fehlerüberdeckung bezeichnet man den Prozentsatz an Fehlern, die bei der Simulation entdeckt wurden.

Man kann die Fehlerüberdeckung auch verwenden, um ein Simulationsverfahren zu überprüfen. Weicht die Fehlerüberdeckung eines Verfahrens von der eines bereits erprobten Verfahrens für das gleiche Testmuster ab, so ist das neue Simulationsverfahren fehlerhaft.

Wie in Kapitel 2 geschildert, liegt der Vorteil des resistiven Fehlermodells in seinem Realitätsbezug. Dieser sollte sich auch in der Berechnung der Fehlerüberdeckung widerspiegeln. Es wurde ebenfalls erwähnt, dass die meisten Bridges in der Realität einen Widerstand zwischen $0\,\Omega$ und $500\,\Omega$ haben [10]. Um nun eine möglichst realistische Fehlerüberdeckung zu berechnen, ist eine entsprechende Gewichtung nötig. Für die Ergebnisse wurde hier die Gewichtung aus [10] (so wie in [14]) verwendet.

Zur Berechnung einer möglichst exakten Fehlerüberdeckung wird normalerweise das *Globale ADI* oder G-ADI hinzugezogen [4]. Dies ist das maximale Entdeckbarkeitsintervall, außerhalb dessen der Fehler nicht entdeckt werden kann.

Als C-ADI oder *„Covered ADI"* bezeichnet man das bei der Simulation berechnete ADI [4]. Es ist in der Regel kleiner als das G-ADI.

Das G-ADI kann nur durch eine erschöpfende Simulation oder durch Simulation entsprechender ATPG-Muster ermittelt werden [6]. Jeder Fehler, der bei dieser Simulation nicht entdeckt wird, ist redundant und hat deswegen auch keinen Einfluss auf die Fehler-

überdeckung. Man bezeichnet die mit Hilfe des G-ADIs berechnete Fehlerüberdeckung auch als G-FC für englisch „*Global Fault Coverage*" [7].

$$\text{G-FC}(f) = 100\% \cdot \left(\int_{C-ADI} \rho\left(r\right) dr \right) \Big/ \left(\int_{G-ADI} \rho\left(r\right) dr \right)$$

f ist ein Brückenfehler, r bezeichnet den Widerstand, ρ ist die verwendete Verteilungsfunktion.

Der hier vorgestellte Simulator ist derzeit nur zur Berechnung der *„Excitation Fault Coverage*" oder kurz E-FC in der Lage. Es handelt sich dabei um eine Approximation der G-FC, bei der anstelle des G-ADIs das Intervall $[0, R_{max}]$ verwendet wird [7]. Es ist offensichtlich, dass das G-ADI höchstens genau so groß wie dieses Intervall ist, auf keinen Fall größer. Da das G-ADI bei der E-FC nicht bekannt ist, ist es auch nicht bekannt, ob redundante Fehler existieren oder nicht. Die E-FC fällt geringer aus als die G-FC, es handelt sich dabei also um eine pessimistische Abschätzung[1].

$$\text{E-FC}(f) = 100\% \cdot \left(\int_{C-ADI} \rho\left(r\right) dr \right) \Big/ \left(\int_{0}^{R_{max}} \rho\left(r\right) dr \right)$$

Tabelle 5.1 gibt die Fehlerüberdeckungn (E-FC) für verschiedene Simulationsläufe an. Spalte eins enthält den Schaltkreisnamen, Spalte zwei die Anzahl an simulierten Bridges pro Schaltkreis und Spalte drei die Gesamtzahl an Sections im jeweiligen Schaltkreis. In der vierten Spalte ist der Durchschnitt an Sections pro Bridge angegeben. Je nach Schaltkreis gab es pro Bridge maximal vier bis maximal 61 Sections[2]. An dieser Stelle zeigt sich, dass es von Vorteil gewesen ist, sich für den Section-orientierten Ansatz zu entscheiden. Spätestens für die Bridge mit den 61 Sections wäre es bei der Bridge-orientierten Simulation notwendig gewesen, mehrfach zu simulieren.

Die fünfte, sechste und siebte Spalte der Tabelle 5.1 enthalten die berechnete Fehlerüberdeckung in Prozent. Pro Schaltkreis wurden drei Simulationsläufe durchgeführt, einmal mit 1,000 zufälligen Testvektoren, einmal mit 10,000 zufälligen Testvektoren und einmal mit den gegebenen ATPG-Mustern.

Wie aus den Ergebnissen ersichtlich, ist die Fehlerüberdeckung sehr hoch. Man erhält dieselben präzisen Ergebnisse wie mit dem Simulator aus [7].

5.2 Laufzeiten

Bei den Simulationsläufen wurden auch Laufzeitmessungen für das Preprocessing und die Simulation durchgeführt. Neben den bisherigen Simulationsläufen wurden auch welche im SPPFP-Modus des Simulators durchgeführt. Zum Vergleich wurde der Intervall -basierte Simulator aus [7] herangezogen.

[1] Dies ist nicht zu verwechseln mit der in [7] vorgestellen „*Pessimistic Fault Coverage*" (P-FC), bei der statt des G-ADI das Intervall $[0, \infty)$ verwendet wird.

[2] Es sei nochmals darauf hingewiesen, dass die Section $[R_{max}, \infty)$ bei der Fehlersimulation nicht betrachtet wurde.

Tabelle 5.1: Fault Coverage (%)

Schaltkreis	#Bridges	#Sections total	#Sections ∅	Muster 1,000	Muster 10,000	ATPG
c0017	2	8	4.00	98.59	98.59	98.59
c0095	77	441	5.73	95.90	95.50	95.90
c0432	5253	18296	3.48	98.28	98.50	98.51
c0499	8985	16092	1.79	98.23	98.24	98.24
c0880	10000	36074	3.60	97.01	97.68	98.13
c1355	10000	32160	3.22	97.06	97.17	97.17
c1908	10000	30961	3.10	97.40	98.11	98.13
c2670	10000	28379	2.84	97.40	98.11	98.93
c3540	10000	28115	2.81	94.95	96.66	98.63
c5315	10000	28491	2.85	99.59	99.62	99.63
c6288	10000	33425	3.34	91.88	91.89	—
c7552	10000	21592	2.16	98.60	99.02	99.47
cs00027	2	10	5.00	100.00	100.00	100.00
cs00208	3986	11508	2.89	95.34	95.76	95.76
cs00298	4468	12863	2.88	97.57	97.59	97.59
cs00344	7760	21229	2.74	94.21	94.24	94.24
cs00349	7881	21914	2.78	94.12	94.15	94.15
cs00382	7809	28578	3.66	98.58	98.63	98.63
cs00386	9384	17907	1.91	96.50	96.68	96.68
cs00400	8290	31563	3.81	98.54	98.58	98.58
cs00420	10000	29142	2.91	82.10	92.22	96.33
cs00444	10000	38922	3.89	97.99	98.01	98.01
cs00510	10000	43359	4.34	95.29	95.29	95.29
cs00526	10000	33954	3.40	97.32	97.81	97.87
cs00641	10000	17494	1.75	99.14	99.41	99.52
cs00713	10000	39871	3.99	98.21	98.42	98.53
cs00820	10000	39164	3.92	91.59	94.73	95.09
cs00832	10000	40010	4.00	91.48	94.48	95.15
cs00838	10000	29463	2.95	67.17	76.86	96.36
cs00953	10000	44768	4.48	92.00	94.30	94.54
cs01196	10000	33300	3.33	93.92	97.21	97.74
cs01238	10000	34365	3.44	94.08	97.06	97.47
cs01423	10000	28598	2.86	96.40	96.95	97.17
cs01488	10000	19587	1.96	97.84	98.16	98.19
cs01494	10000	19584	1.96	97.78	98.07	98.12
cs05378	10000	28812	2.88	98.07	99.18	99.37
cs09234	10000	21819	2.18	89.70	95.93	98.63
cs13207	10000	20354	2.04	95.60	98.17	99.66
cs15850	10000	20093	2.01	96.37	98.44	99.55
cs35932	10000	27198	2.72	96.47	96.47	96.47
cs38417	10000	25893	2.59	95.58	95.98	97.82
cs38584	10000	26383	2.64	90.73	95.85	98.19

Bei der in Tabelle 5.3 angegebenen Zeiten handelt es sich um die Summen aus Preprocessing und Simulationszeit, ohne Berechnung der Fehlerüberdeckung.

Spalte eins der Tabelle 5.3 enthält wieder die Schaltkreisnamen, die Spalten zwei bis vier, fünf bis sieben beziehungsweise acht bis zehn enthalten die Laufzeiten für die Simulation im PPSFP- und im SPPFP-Modus beziehungsweise für den Intervall-basierten Simulator aus [7].

Wie aus der Tabelle 5.3 ersichtlich, ermöglicht die Kombination aus Section-orientierter Simulation und PPSFP eine erhebliche Beschleunigung gegenüber der Intervall-basierten Simulation mit Mengenoperationen. Der Beschleunigungsfaktor liegt bei bis zu 62 für die 1,000 Zufallsvektoren und bis zu 120 für die 10,000 Zufallsvektoren.

Die Simulationszeit im SPPFP-Modus fällt deutlich höher aus als im PPSFP-Modus. Damit hat sich die Vermutung aus Kapitel 3.2.2.1 bestätigt, dass die PPSFP zu einer größeren Beschleunigung führt. Jedoch liegt auch die Simulationszeit der SPPFP unter der der Intervall-basierten Simulation.

Mit $0.01\,s$ bis maximal $0.62\,s$ (für den cs00832 mit ATPG-Mustern) fällt die Zeit für das Preprocessing kaum ins Gewicht und steht im guten Verhältnis zur Simulationszeit. Mit maximal 33.6 MB (für den cs35932 und 10,000 Testvektoren) hält sich auch der Speicherplatzbedarf in Grenzen.

Tabelle 5.3: Simulationszeit (s)

	PPSFP			SPPFP			Intervall-basiert		
Schaltkreis	1,000	10,000	ATPG	1,000	10,000	ATPG	1,000	10,000	ATPG
c0017	0.02	0.02	0.02	0.03	0.07	0.02	0.00	0.08	0.00
c0095	0.08	0.53	0.02	0.42	3.84	0.03	0.78	8.13	0.01
c0432	1.10	3.53	1.25	5.78	29.80	8.12	57.46	584.48	62.16
c0499	1.01	2.16	0.72	4.14	15.63	0.89	108.23	1079.19	7.47
c0880	2.55	11.47	2.05	18.41	116.67	13.94	108.66	1072.33	75.67
c1355	3.38	20.63	1.74	22.72	187.72	4.52	217.21	2154.24	45.94
c1908	2.63	7.89	1.53	13.71	64.33	3.25	281.63	2730.71	85.44
c2670	2.56	13.02	1.21	16.26	110.01	4.37	174.50	1751.03	64.19
c3540	2.55	9.40	1.08	15.62	81.58	7.60	231.15	2342.25	117.38
c5315	1.53	3.80	1.30	5.84	26.36	4.21	209.97	2193.35	76.16
c6288	5.66	41.47	—	46.30	446.95	—	634.95	6720.93	—
c7552	2.23	8.97	1.42	11.16	70.67	3.85	247.67	2545.09	94.46
cs00027	0.01	0.01	0.01	0.01	0.02	0.01	0.02	0.14	0.00
cs00208	0.98	5.64	0.35	7.87	60.38	1.21	19.26	195.03	2.52
cs00298	0.91	4.90	0.47	5.42	44.63	1.27	28.10	279.76	3.60
cs00344	2.02	14.48	0.84	15.10	139.52	3.88	55.42	552.15	11.40
cs00349	2.12	15.30	0.87	16.40	149.83	3.88	57.86	589.74	10.94
cs00382	1.71	7.90	1.04	10.02	68.08	3.56	53.26	535.40	3.69
cs00386	2.20	10.41	0.73	17.55	100.96	1.61	40.92	412.77	4.49
cs00400	1.94	9.06	1.14	11.43	77.34	3.79	58.62	595.00	14.27
cs00420	4.09	24.31	1.20	39.31	272.00	7.23	51.48	505.67	17.43
cs00444	2.79	14.53	1.58	19.07	145.12	7.06	70.42	721.01	22.15
cs00510	3.54	23.25	1.89	27.41	244.55	10.37	70.74	696.66	22.88
cs00526	3.26	14.95	1.82	23.16	140.48	8.39	71.90	724.64	27.70
cs00641	0.88	2.29	0.63	4.54	19.98	1.84	97.33	1001.55	27.10
cs00713	1.48	7.85	0.89	9.79	72.37	3.91	105.60	1060.36	32.70
cs00820	6.00	31.97	2.87	54.93	357.59	19.61	65.54	673.51	30.47
cs00832	6.28	34.11	2.85	58.09	372.92	19.83	65.83	654.15	30.98
cs00838	6.34	47.21	1.62	57.93	476.03	10.97	57.77	574.19	30.90
cs00953	5.63	34.60	2.71	47.19	348.82	17.25	85.06	835.06	37.57
cs01196	3.65	15.61	1.87	27.98	149.77	10.27	71.55	696.53	37.31
cs01238	3.90	17.61	2.03	31.23	168.19	11.64	72.53	729.29	39.66
cs01423	2.42	12.07	1.50	17.77	114.39	8.58	86.46	830.52	57.62
cs01488	2.02	8.32	0.98	14.50	73.70	3.64	62.30	624.18	16.19
cs01494	2.13	8.89	1.04	15.17	78.04	3.78	63.42	630.46	15.59
cs05378	2.20	6.99	1.50	14.47	60.02	8.58	156.82	1552.36	144.12
cs09234	3.76	16.15	1.67	29.85	149.64	11.21	173.96	1730.58	167.83
cs13207	2.25	10.82	1.56	18.43	106.68	12.10	498.34	4929.44	564.55
cs15850	2.08	8.71	1.54	17.17	87.27	11.89	394.83	3956.68	428.38
cs35932	2.79	18.50	1.92	23.28	189.48	14.19	519.59	5088.17	406.20
cs38417	3.02	17.58	2.65	27.23	194.05	25.27	887.68	8957.95	1013.94
cs38584	3.26	15.88	3.84	28.53	163.38	41.97	654.05	6853.46	1087.09
$\sum$	110.96	582.79	58.67	851.22	5778.86	339.59	6968.87	70365.22	4948.15

6 Zusammenfassung und Ausblick

Kurzschlüsse in Schaltkreisen machten bereits vor zirka zwanzig Jahren einen großen Teil der bei Chips auftretenden Defekte aus. Aufgrund der technologischen Entwicklung ist mit einer Zunahme solcher Defekte zu rechnen. Um sie möglichst realistisch zu simulieren, wurde das Modell des resistiven Brückenfehlers entwickelt.

Derzeit gibt es gibt zwei Ansätze zur Simulation von resistiven Brückenfehlern: Die Intervall-basierte Simulation und die Segmentierung. Die Segmentierung bietet den Vorteil, dass sich die Brückenfehler auf eine Reihe von stuck-at Faults innerhalb des Schaltkreises reduzieren lassen, wohingegen der Intervall-basierte Ansatz aufwändige Mengenoperationen benötigt. Insbesondere ist mit Hilfe der Segmentierung eine Bit-parallele Simulation von resistiven Brückenfehlern möglich.

Ziel der vorliegenden Arbeit war es, ein auf der Segmentierung basierendes, Bit-paralleles Simulationsverfahren für resistive Brückenfehler zur Verfügung zu stellen. Dazu wurden zwei verschiedene Ansätze in Form der Bridge-orientierten und der Section-orientierten Simulation untersucht und miteinander verglichen. Der Aufwand der Bridge-orientierten Simulation ist im Vorfeld schwer abschätzbar, außerdem wird ein zu großer Widerstandsbereich simuliert. Bei der Section-orientierten Simulation werden dagegen nur die Sections simuliert, die tatsächlich benötigt werden. Der Section-orientierte Ansatz lässt sich außerdem sehr gut mit der PPSFP oder der SPPFP kombinieren, zwei Bit-parallele Simulationsverfahren, durch die die Simulation von stuck-at Faults beschleunigt werden kann. Diese Vorteile der Section-orientierten Simulation führten dazu, dass die Section-orientierte Simulation in die Tat umgesetzt und in einen Bit-parallelen stuck-at Fault-Simulator eingebaut wurde.

Bei Simulationsläufen hat sich gezeigt, dass Sectioning zusammen mit der Bit-parallelen Simulation eine erhebliche Beschleunigung gegenüber der Intervall-basierten Simulation ermöglicht. Dabei wurde nichts an Genauigkeit eingebüßt. Die Fehlerüberdeckung ist sehr hoch und entspricht den Ergebnissen bereits existierender Simulatoren.

Es gibt mehrere Ansätze für weiterführende Arbeiten. Die Implementierung der Section-orientierten Simulation ließe sich weiter optimieren. So ließe sich der Speicherplatzbedarf der verwendeten Datenstruktur weiter reduzieren oder die Berechnung der Fehlerüberdeckung durch verschiedene Verteilungsfunktionen erweitern.

Ein speziell für die Segmentierung ausgelegter Simulator wäre ebenfalls denkbar. Dieser könnte unter Umständen einige Eigenschaften des Sectioning besser ausnützen als ein Bit-paralleler stuck-at Fault Simulator. Im Rahmen dessen würde es sich auch anbieten, den Bridge-orientierten Ansatz nochmals genauer zu betrachten. Es sollte untersucht werden, wieviele kritische Widerstände pro Bridge im Durchschnitt existieren. Dadurch könnte der Simulationsaufwand besser abgeschätzt werden. Außerdem ist es möglich, dass die Zunahme der Bitbreite zu einer Verringerung dieses Aufwandes führt.Darüber hinaus könnte auch eine Umsetzung des Simulation-Verfahrens für Feedback-Fehler aus [5] in Betracht gezogen werden.

Literaturverzeichnis

[1] M. Abramovici, M. A. Breuer, A. D. Friedman. Digital Systems Testing And Testable Design. Chap. 4, pages 93-129. IEEE Press, 1990 (revised edition).

[2] J. M. Acken, S. D. Millman. Accurate modeling and simulation of bridging faults. *Custom Integrated Circuits Conf.*, pages 17.4.1-17.4.4, 1991.

[3] R. C. Aitken and P. C. Maxwell. Biased voting: A method for simulating CMOS bridging faults in the presence of variable gate logic thresholds. In *Int'l Test Conf.*, pages 63-72, 1993.

[4] F. Azaïs, Y. Bertrand, and M. Renovell. Detection of defects using fault model oriented test sequences. *Jour. of Electronic Testing: Theory and Applications*, 14(1-2):13-22, 1999.

[5] B. Becker, P. Egelke, I. Polian, and M. Renovell. Modeling feedback bridging faults with Non-Zero Resistance. *Jour. of Electronic Testing: Theory and Applications*, 21(1):57-69, 2005.

[6] B. Becker, P. Engelke, I. Polian, and M. Renovell. Automatic test pattern generation for resistive bridging faults. *Jour. of Electronic Testing: Theory and Applications*, 22(1):61-69, 2006.

[7] B. Becker, P. Engelke, I. Polian, and M. Renovell. Simulating resistive bridging and stuck-at faults. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 25(10):2181-2192, 2006.

[8] B. Becker, B. Braitling, P. Engelke, I. Polian, and M. Renovell. SUPERB: Simulator utilizing parallel evaluation of resitive bridges. To be presented at *Asian Test Symp.*, 2007.

[9] Y. Bertrand, P. Huc, and M. Renovell. The concept of Resistance Interval: A new parametric Model for realistic resistive Bridging Fault. In *VLSI Test Symp.*, pages 184-189, 1995.

[10] E. M. J. G. Bruls, J. Figuras, R. Rodríguez-Montañés. Bridging defects resistance measurements in a CMOS process. In *Int'l Test Conf.*, pages 892-899, 1992.

[11] F. J. Ferguson, J. P. Shen. Extraction of Realistic CMOS Fault using Inductive Faults Analysis. *IEEE Int'l. Test Conf.*, pages 475-484, 1988.

[12] R. S. Fetherston, S. Ma, I. Shaik. A comparison of bridging fault simulation methods. *Int'l Test Conf.*, pages 587-595, 1999.

[13] T. Hayashi, T. Kanbayashi, T. Shinogi, T. Tsuruoka, and T. Yoshikawa. Faulty resistance sectioning technique for resistive bridging fault ATPG systems. In *Asian Test Symp.*, pages 102-107, 1998.

[14] C. Y. Lee, and D. M. H. Walker. PROBE: A PPSFP simulator for resistive bridging faults. In *VLSI Test Symp.*, pages 105-110, 2000.

[15] K. C. Y. Mei. Bridging and stuck-at faults. *IEEE Trans. on Comp.*, C-23(7):720-727, 1974.

[16] H.-J. Wunderlich. Hochintegrierte Schaltungen: Prüfgerechter Entwurf und Test. Kap. 4, 117-183. Springer-Verlag, 1991.