Formaler Nachweis der Fehlertoleranz von Schaltkreisen

Görschwin Fey[1,2] André Sülflow[1] Stefan Frehse[1] Ulrich Kühne[1] Rolf Drechsler[1]

[1]*Fachbereich 3 – Mathematik und Informatik*
Universität Bremen
28359 Bremen

[2]*VLSI Design & Education Center*
Universität Tokio
Tokio, 113-8656, Japan

{*fey,suelflow,sfrehse,ulrichk,drechsle*}*@informatik.uni-bremen.de*

Abstract

While facing continuously shrinking feature sizes, the demand for fault tolerance in digital circuits increases. Numerous approaches to achieve such robustness on the design side have been presented. But ensuring that the fault tolerance is really achieved is a tough verification problem.

Here, we propose a formal model and an effective algorithm to formally prove the robustness of a digital circuit. The proposed model uses a fixed bound in time to cope with the complexity of the sequential equivalence check. The result is a lower and upper bound of the robustness. The underlying algorithm and techniques to improve the efficiency are presented. In the experiments the efficiency was evaluated on circuits with different fault detection mechanisms.

1 Einleitung

Moore's Gesetz treibt schon seit Jahrzehnten die Halbleiterindustrie an: Die Anzahl der Komponenten, die in einem Schaltkreis integriert werden, nimmt exponentiell zu. Dies ist nur durch eine stetige Verringerung der Strukturgrößen möglich. Im Resultat nimmt der Einfluss von Schwankungen im Fertigungsprozess relativ zu absoluten Kenngrößen stetig zu. Eine mögliche Folge ist in Zukunft, dass unzuverlässige, ggf. fehlerhafte Komponenten, die Bausteine eines Schaltkreises sind [2]. Eine weitere Konsequenz ist die zunehmende Anfälligkeit von Schaltkreisen gegenüber so genannten transienten Fehlern, die zum Beispiel durch Umgebungsstrahlung induziert werden [20].

Um mit diesen Problemen umzugehen, wurden verschiedene Maßnahmen auf Fertigungsebene [26] oder Entwurfsebene [10, 1] vorgeschlagen. Selbst erste vollautomatische Werkzeuge, um die Robustheit eines Schaltkreises zu erhöhen, stehen zur Verfügung [25]. Insbesondere beim Entwurf ist jedoch der Nachweis der erwünschten Fehlertoleranz gegenüber transienten Fehlern schwierig. Simulations- oder emulationsbasierte Methoden [5] erlauben nur einen kleinen Teil der möglichen Szenarien zu untersuchen. Zahlreiche formale Analysemethoden bewerten die Wahrscheinlichkeit mit der ein Fehler an einem Ausgang eines Schaltkreises auftritt, z.B. [16] – dies ist jedoch nur mit sehr hohem Rechenaufwand möglich.

Methoden wie sie in der formalen Verifikation üblich sind, ermöglichen eine solche Fehlertoleranz der Implementierung zu *beweisen*. In [3] wird dazu eine klassische Analyse von Fehlerbäumen durch symbolische Methoden realisiert. Die Spezifikation der zu betrachtenden Fehler muss jedoch manuell durchgeführt werden. Ebenso basieren [15] und [14] auf symbolischer Analyse, hier werden jedoch Mutationen genutzt bezüglich deren die Fehlertoleranz bewertet wird. In [14] wird lediglich eine „ja/nein"-Aussage über die Robustheit geliefert, während [15] auch Maßzahlen zur Veränderung des Zustandsraumes angibt. Diese Ansätze verwenden jeweils den vollständigen Schaltkreis als Spezifikation. In [18] wird die Fehlertoleranz bezüglich einer Menge formaler Eigenschaften evaluiert. Dabei wird aber lediglich eine Aussage über die Robustheit der Zustandsbits geliefert. Keiner der bisher genannten Ansätze liefert genaue Angaben über fehleranfällige Strukturen in den betrachteten Schaltkreisen.

Die hier vorgestellte Arbeit basiert auf dem Ansatz aus [11, 12]. Das dort vorgestellte nichtdeterministische Fehlermodell wird übernommen. Ein Schaltkreis wird als robust klassifiziert, sofern auch unter Fehlerannahmen kein falsches Ausgabeverhalten entstehen kann. Außerdem wird eine genaue Rückmeldung geliefert, welche Teile des Schaltkreises nicht fehlertolerant sind.

Die vorliegende Arbeit liefert drei wesentliche Beiträge gegenüber dem bisherigen Stand:

- Nutzung einer zeitlichen Schranke für die formale Prüfung
- Bestimmung einer oberen und unteren Schranke für die Robustheit eines Schaltkreises
- Steigerung der Effizienz des bisherigen Algorithmus

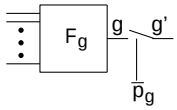

Abbildung 1. Fehlerinjektion

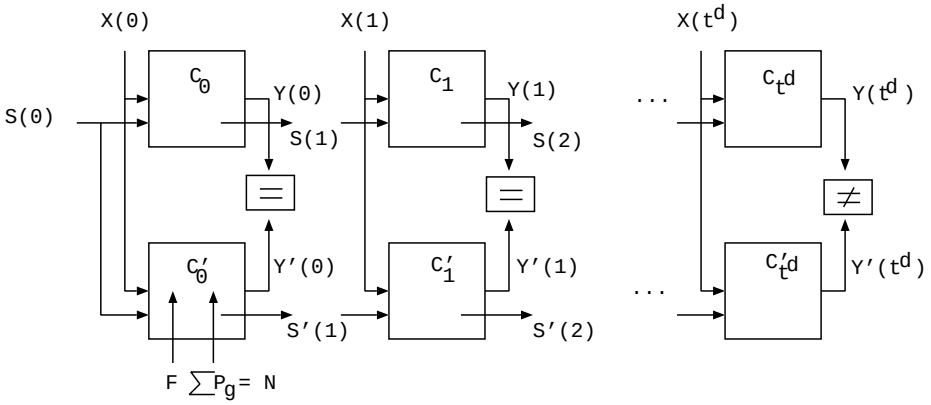

Abbildung 2. Sequentieller Äquivalenzvergleich

In Experimenten wurde der Einfluss der zeitlichen Schranke auf die Genauigkeit der Robustheitsabschätzung untersucht. Durch die Steigerung der Effizienz des Algorithmus konnte die Geschwindigkeit der Klassifizierung bis zu einem Faktor von 7 gesteigert werden.

Die Arbeit ist wie folgt gegliedert: Das allgemeine Fehlermodell und der Basisalgorithmus zur Berechnung der Robustheit werden im nächsten Abschnitt diskutiert. In Abschnitt 3 wird das Modell auf praktische Bedürfnisse angepasst und es werden Schranken für die Robustheit eines Schaltkreises eingeführt. Der resultierende Algorithmus wird in Abschnitt 4 präsentiert, in Abschnitt 5 wird die Effizienz verbessert. Nach der Darstellung der experimentellen Ergebnisse in Abschnitt 6 wird die Arbeit zusammengefasst.

2 Allgemeines Fehlermodell

Im Folgenden wird ein Schaltkreis $\mathbb{C}$ mit den primären Eingängen X, primären Ausgängen Y und Zustandsbits S betrachtet. Die Anzahl der Komponenten von $\mathbb{C}$ beträgt $|\mathbb{C}|$. Je nach gewünschter Granularität können zum Beispiel Gatter, Module oder Befehle aus der Hardware-Beschreibungssprache als Komponenten betrachtet werden.

Ein Schaltkreis ist robust, sofern die Fehlfunktion einer Komponente das Ausgabeverhalten nicht verändert. Diese Prüfung kann durch einen sequentiellen Äquivalenzvergleich des fehlerfreien und fehlerhaften Schaltkreises realisiert werden. Im fehlerhaften Schaltkreis wird dazu jeweils eine Komponente modifiziert. Bleibt der modifizierte Schaltkreis äquivalent zum fehlerfreien Schaltkreis, so gilt die betrachtete Komponente als robust.

In [11, 12] wurde ein Ansatz vorgestellt, der die Überprüfung mehrerer Komponenten gleichzeitig ermöglicht. Als Fehlermodell wird ein nichtdeterministischen Verhalten einer Komponente zugelassen. Sei g eine Komponente und F_g die Boolesche Funktion dieser Komponente. Der Ausgang von g wird ebenfalls mit der Variablen g identifiziert. Dann wird g wie in Abbildung 1 dargestellt durch $\overline{p}_g \rightarrow (g' = g)$ ersetzt. Solange das *Fehlerprädikat* p_g den Wert 0 hat, verhält sich die Komponente fehlerfrei. Nimmt p_g den Wert 1 an, kann der neue Ausgang g' der Komponente einen beliebigen Wert annehmen. Es kann also ein Fehler injiziert werden.

Für die formale Überprüfung der Robustheit einer Teilmenge M aus der Menge aller Komponenten $\mathbb{C}$ wird folgende Konstruktion verwendet. Alle Komponenten aus M werden mit Fehlerprädikaten versehen. Das so erzeugte Modell des fehlerhaften Schaltkreises wird, wie in Abbildung 2 dargestellt, einem Äquivalenzvergleich mit dem ursprünglichen Schaltkreis unterzogen. Dabei wird die Anzahl der betrachteten Takte iterativ erhöht, so dass nach und nach mehr Komponenten als nicht robust klassfiziert werden können. Die Anzahl der Fehlerprädikate, die den Wert 1 annehmen dürfen, wird auf N beschränkt. Ein Gegenbeispiel zur Äquivalenz dieser beiden Schaltkreise liefert nun einen Fehler und eine Eingabesequenz, die unter diesem Fehler eine fehlerhafte Ausgabe erzeugt. Der Fehler ist durch N Komponenten gegeben, deren Fehlerprädikate den Wert 1 haben. Als Maß für die Robustheit eines Schaltkreises wird bestimmt, welche Komponenten auch bei Fehlerinjektion kein Fehlverhalten verursachen. Diese Komponenten werden in der Menge T zusammengefasst. Die Robustheit des Schaltkreises beträgt dann $|T|/|\mathbb{C}|$.

Um den exakten Wert für die Robustheit zu berechnen, muss das Verhalten bis zur maximalen sequentiellen Tiefe des Produktautomaten eines fehlerhaften Schaltkreises und des fehlerfreien Schaltkreises[1] betrachtet werden [11]. In der Praxis ist dies jedoch nicht möglich. Deshalb werden der Robustheitsbegriff und das Modell im Folgenden auf praktische Bedürfnisse angepasst.

[1] Die sequentielle Tiefe eines Automaten ist der längste Pfad, auf dem kein Zustand zweimal vorkommt.

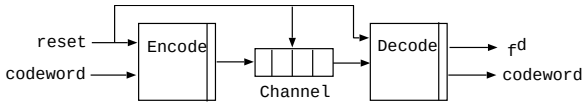

Abbildung 3. Hamming-Kode-Modell

3 Schranken für die formale Betrachtung

Im praktischen Einsatz müssen nach dem Auftreten eines internen Fehlers Maßnahmen eingeleitet werden, um damit umzugehen. Damit es nicht zur „Akkumulation" mehrerer Fehlereffekte kommt, muss diese Reaktion innerhalb einer kurzen Zeitspanne geschehen, innerhalb derer höchstens ein Fehler auftritt. Deshalb wird im Folgenden angenommen, dass die betrachteten Schaltkreise mit einer Fehlererkennungslogik ausgestattet sind. Die Erkennung eines Fehlers wird durch Setzen eines Signals f^d angezeigt. Insbesondere gilt, dass f^d im fehlerfreien Fall nie gesetzt ist. Außerdem muss jeder Fehler, der zur Veränderung der Ausgabe oder der Zustandsübergänge führt, nach spätestens t^d Takten erkannt werden. Damit reicht es für den formalen Nachweis der Robustheit aus, wenn t^d Takte betrachtet werden und eine Ein-Fehler-Annahme zu Grunde gelegt wird, wobei von einem beliebigen im fehlerfreien Fall erreichbaren Zustand gestartet wird.

Dann wird eine Fallunterscheidung genutzt, um die Robustheit einer Komponente g zu bewerten. Unter der Annahme, dass ein falscher Wert an g injiziert wurde, gilt:

1. Komponente g ist *robust*, falls
   (a) $f^d = 1$ innerhalb von t^d Takten folgt bevor eine Abweichung der Ausgabewerte auftritt oder
   (b) $f^d = 0$ ist, keine Abweichung der Ausgabewerte stattfindet und nach t^d Takten der gleiche Zustand wie im fehlerfreien Fall erreicht wird.
2. Komponente g ist *nicht robust*, falls
   eine Abweichung an mindestens einem Ausgang auftritt bevor $f^d = 1$.
3. Komponente g ist *nicht abschließend klassifiziert*, falls
   $f^d = 0$ ist, keine Abweichung der Ausgabewerte innerhalb von t^d Takten möglich ist und der Zustand nach t^d Takten vom Zustand im fehlerfreien Fall abweicht.

Beispiel 1 *Ein (7,4)-Hamming-Kode kann Einfachfehler sowohl erkennen als auch korrigieren [13]. In Abbildung 3 ist ein Modell dargestellt, das aus einem Kodierer, einem Übertragungskanal und einem Dekodierer besteht. Die Daten werden im Kodierer kodiert, über einen bit-seriellen vierstufigen Übertragungskanal zum Dekodierer übertragen und dort dekodiert. Das Fehlererkennungssignal f^d wird gesetzt, falls das empfangene Kodewort fehlerhaft war. Der zeitliche Ablauf ist wie folgt:*

- *Kodieren und zum Kanal übermitteln: 1 Takt*
- *Übertragung: 4 Takte*
- *Dekodieren und auf Ausgang schreiben, ggf. Fehler anzeigen: 1 Takt*

Bezüglich der Wahl von t^d ergeben sich folgende Fälle:

- *$t^d < 6$: Das Datenwort aus Zeittakt 0 ist noch nicht an den primären Ausgängen angekommen.*

  Die Injektion eines Fehlers in die Fehlererkennungslogik kann sich nur durch Setzen von f^d äußern. Die Fehlererkennungslogik im Dekodierer ist also robust. Das gleiche gilt für Fehler, die vor der Dekodierung im Dekodierer injiziert werden. Fehler in der Logik zur Dekodierung werden in der Regel nicht erkannt.

  Die Injektion eines Fehlers im Kodierer oder Kanal ändert den Zustand gegenüber dem fehlerfreien Modell. Die Daten wurden aber noch nicht dekodiert. Deshalb werden Fehler zum Teil noch nicht erkannt.

  Mit jeder Inkrementierung von t^d werden mehr Komponenten des Kanals klassifiziert. Teile des Kodierers bleiben unklassifiziert.

- *$t^d = 6$: Fehler, die im Kodierer injiziert werden, erreichen den primären Ausgang und werden dort ggf. angezeigt.*

- *$t^d > 6$: Die Injektion eines Fehlers im Zeittakt 0 hat keine Auswirkung auf das Modell in Zeittakten > 6, der Zustand von fehlerfreiem und fehlerhaftem Modell in Takt 7 ist gleich. Es sind also alle Komponenten klassifiziert.*

Um nun zu garantieren, dass ein Schaltkreis robust ist, dürfen weder nicht robuste noch nicht klassifizierte Komponenten vorkommen. Für eine nicht klassifizierte Komponente kann zwar keine Abweichung vom normalen Ausgabeverhalten innerhalb des betrachteten Zeitfensters nachgewiesen werden, aber es kann auch nicht ausgeschlossen werden, dass die Veränderung der Zustandsübergänge später eine falsche Antwort erzeugt.

Sei nun T die Menge der Komponenten, die als robust klassifiziert wurden, S die Menge der Komponenten, die als nicht robust klassifiziert wurden und U die Menge der Komponenten, die nicht abschließend klassifiziert wurden. Dann gilt $T \cup S \cup U = \mathbb{C}$. Eine untere Schranke R_{lb} und eine obere Schranke R_{ub} für die

Robustheit des Schaltkreises $\mathbb{C}$ können dann bestimmt werden durch:

$$R_{lb} = \frac{|T|}{|\mathbb{C}|} = 1 - \frac{|S \cup U|}{|\mathbb{C}|}$$

$$R_{ub} = \frac{|T \cup U|}{|\mathbb{C}|} = 1 - \frac{|S|}{|\mathbb{C}|}$$

Tabelle 1. Hamming-Kode

t^d	$\|T\|$	$\|S\|$	$\|U\|$	R_{lb} %	R_{ub} %
0	5	2	275	1.77	99.29
1	48	27	207	17.02	90.43
2	73	40	169	25.89	85.82
3	98	52	132	34.75	81.56
4	123	64	95	43.62	77.30
5	148	78	56	52.48	72.34
6	191	91	0	67.73	67.73

Beispiel 2 *Für das oben betrachtete Hamming-Kode-Modell ergeben sich die Werte aus Tabelle 1.*

Schaltkreise, die direkt eine Fehlerkorrektur vornehmen, statt einen Fehler zu signalisieren, können ebenfalls mit diesem Modell behandelt werden. Für einen solchen Schaltkreis wird ständig $f^d = 0$ angenommen (es kann kein Fehler signalisiert werden). Fall 1(a) aus der obigen Fallunterscheidung tritt somit nicht mehr auf.

4 Algorithmus

Der Algorithmus wurde bereits in Abschnitt 2 skizziert und ist in Abbildung 2 dargestellt. Um die Robustheit für t^d Zeitschritte zu beweisen, werden sowohl der originale Schaltkreis $\mathbb{C}$ als auch ein mit Fehlerlogik versehener Schaltkreis $\mathbb{C}'$ über t^d-Zeitschritte abgerollt. Hierbei werden die initialen Zustände beider Schaltkreise miteinander abgeglichen. In $\mathbb{C}'$ wird für den Zeitpunkt 0 für jede Komponente die Fehlerlogik eingefügt. Werden alle Fehlerprädikate p_g auf 0 gesetzt, verhält sich der Schaltkreis $\mathbb{C}'$ wie $\mathbb{C}$.

Bei der Untersuchung von Einfachfehlern ist es ausreichend, Fehlerprädikate im Zeittakt 0 einzufügen. Hierbei wird die Anzahl der gleichzeitig gesetzten Fehlerprädikate auf eins beschränkt.

Das Modell unterstützt damit sowohl Fehler in den *Flip-Flops* (FFs) als auch in der kombinatorischen Logik. Inkrementell werden nun für jeden Zeitschritt die robusten, die nicht robusten und die nicht klassifizierbaren Komponenten ermittelt. Sobald eine Komponente als robust oder nicht robust klassifiziert ist, kann in der weiteren Betrachtung das Fehlerprädikat auf 0 gesetzt bzw. die Logik zur Fehlerinjektion deaktiviert werden.

Im Folgenden wird das Problem in eine Instanz *Boolescher Erfüllbarkeit* (SAT) [6] überführt. Dazu

```
1  function robustness (C, t^d)
2   create a copy C'_0 of C
3   foreach component g ∈ C'_0
4     replace g by g'[g, p_g];
5   done
6   convert to SAT instance;
7   force init states of C'_0 and C_0 to be
        equal;
8   constrain ∑ p_g == 1;
9
10  T := ∅;
11  S := ∅;
12  U := all components g ∈ C'_0;
13  t := 0;
14  while (t ≤ t^d && U ≠ ∅)
15   if (t > 0) then
16    create copies C'_t and C_t of C;
17    connect S'(t) to C'_t and S(t) to C_t;
18   fi
19   connect PIs of C'_t and C_t;
20
21   cmpPOs:=at least one pair of POs is
         different;
22   cmpFFs:=at least one pair of FFs is
         different;
23
24   add constraint UR := (cmpPOs & !f^d) = 1;
25   S' := extractAllSolutions();
26   remove constraint UR;
27
28   add constraint UC :=
         (!f^d & !cmpPOs & cmpFFs) = 1;
29   U' := extractAllSolutions();
30   remove constraint ∀g ∈ U' : p_g = 0;
31   remove constraint UC;
32
33   T' := (U \ S') \ U';
34   add constraint ∀g ∈ T' : p_g = 0;
35
36   T := T ∪ T';
37   U := U';
38   S := S ∪ S';
39
40   t := t + 1;
41   remove cmpPOs;
42   remove cmpFFs;
43  done;
44 end function;
```

Abbildung 4. Algorithmus

```
1 function extractAllSolutions ()
2    M := ∅;
3    while (satisfiable) do
4         G = {g|p_g == 1};
5         M := M ∪ G;
6         add constraint p_g = 0;
7    done;
8    return M;
9 end function;
```

Abbildung 5. Ermittlung aller Lösungen

wird das Problem in konjunktiver Normalform dargestellt und dann durch einen SAT-Beweiser [7, 9] gelöst. Der komplette Algorithmus ist in Abbildung 4 dargestellt.

Zunächst werden der originale Schaltkreis und eine

Kopie mit Fehlerlogik in eine konjuktive Normalform überführt (Zeilen 2-6) [23]. Anschließend werden die Initialzustände der Schaltkreise $\mathbb{C}$ und $\mathbb{C}'$ miteinander abgeglichen (Zeile 7) und die Anzahl der Fehlerprädikate, die gleichzeitig eins sein dürfen, auf eins limitiert (Zeile 8). Die Menge der robusten (T), nicht robusten (S) und der nicht klassifizierten (U) Komponenten werden initialisiert (Zeile 10-12). Nun werden die primären Eingänge (PI) der beiden Schaltkreise miteinander verbunden, um gleiche Eingabesequenzen zu erzeugen (Zeile 19). Zur Ermittlung von Unterschieden an den primären Ausgängen und an den Zustandsvariablen[2] wird eine Vergleichslogik hinzugefügt, deren Ausgänge $cmpPOs$ und $cmpFFs$ (Zeilen 21-22) sind. Eine 1 an $cmpPOs$ bzw. $cmpFFs$ signalisiert Unterschiede in den Ausgängen/Zuständen von $\mathbb{C}'$ und $\mathbb{C}$, während eine 0 die Gleichheit anzeigt. Es erfolgt die Ermittlung aller nicht robusten Komponenten wie in Abschnitt 3 beschrieben (Zeilen 24-26).

Die Funktion $extractAllSolutions$ (siehe Abbildung 5) extrahiert alle Komponenten, deren Fehlverhalten zu einer Abweichung, d.h. einer erfüllenden Belegung führt. Solange die Instanz erfüllbar ist, wird die Komponente g ermittelt, deren Fehlerprädikat gesetzt ist, und g der Menge M hinzugefügt. Anschließend wird das Fehlerprädikat p_g „geblockt", d.h. konstant auf 0 gesetzt (Zeilen 3-7). Sind alle Komponenten extrahiert, erfolgt die Rückgabe der Menge M.

In den Zeilen 28-31 werden alle nicht klassifizierbaren Komponenten ermittelt. Im Anschluss an die Ermittlung werden die geblockten Fehlerprädikate wieder aktiviert, um die Komponenten auch im nächsten Zeittakt betrachten zu können (Zeile 30). Aus der Menge der nicht robusten und der nicht klassifizierbaren Komponenten wird die Menge der robusten Komponenten berechnet und dauerhaft blockiert (Zeile 33-34). Im Anschluss daran werden die Mengen U und S aktualisiert, die erzeugte Vergleichslogik für den Zeitpunkt t wieder entfernt und t um eins erhöht (Zeilen 36-42). Solange die maximale Tiefe (t^d) noch nicht erreicht worden ist und es noch nicht klassifizierte Komponenten gibt, wird die Analyse fortgeführt. Mit jedem Zeitschritt $t > 0$ wird je ein weiterer fehlerfreier Schaltkreis $\mathbb{C}'_t$ und $\mathbb{C}_t$ in der SAT Instanz erzeugt und mit $\mathbb{C}'_{t-1}$ bzw. $\mathbb{C}_{t-1}$ verbunden (Zeile 14-17).

Die Klassifizierung der Komponenten in robuste und nicht robuste Komponenten T und S ist abgeschlossen, wenn die Menge der noch nicht klassifizierten Komponenten U leer ist oder aber die dem Algorithmus übergebene maximale Tiefe t^d erreicht ist.

5 Verbesserung des Verfahrens

In diesem Abschnitt werden Methoden vorgestellt, mit denen die Genauigkeit sowie die Effizienz des Ansatzes gesteigert werden kann.

[2]Die Vergleichslogik wird an den Eingängen der FFs angeschlossen, da diese den Zustand für den nachfolgenden Zeitschritt repräsentieren.

5.1 Vermeidung von False Negatives

Das vorgestellte Verfahren macht keine Einschränkung des Zustands zum Zeitpunkt 0. Zum Beispiel lassen sich dadurch auch im fehlerfreien Schaltkreis nicht erreichbare Zustände erzeugen, die zu *False Negatives* führen können: eine Komponente wird als nicht robust klassifiziert, obwohl sie bezüglich der erreichbaren Zustände robust ist. False Negatives lassen sich durch zusätzliche Einschränkungen der initial erlaubten Zustandsmenge vermeiden.

Wird keine Einschränkung vorgenommen, so liefert der Algorithmus aus Abschnitt 4 lediglich einen garantierten Minimalwert für die Robustheit. Insbesondere im Falle von Schaltkreisen mit *Triple-Modular-Redundancy* (TMR), liegt die ermittelte Robustheit oftmals unterhalb der realen Robustheit. Die initial erlaubten Zustände lassen sich entweder durch manuelle Angabe oder durch eine automatische Erreichbarkeitsanalyse [8] einschränken. Die Erreichbarkeitsanalyse kann sowohl für alle FFs als auch für eine Teilmenge durchgeführt werden. Die erwünschte Genauigkeit der Ergebnisse lässt sich somit skalieren.

5.2 Fanout-Free Regions

Eine Komponente mit mehr als einem Nachfolger wird als *Fanout-Punkt* bezeichnet. Alle Vorgänger dieser Komponente bis zum nächsten Fanout-Punkt, primären Eingang oder FF liegen in der gleichen *Fanout-Free Region* (FFR). Diese strukturelle Information kann wie in der SAT-basierten Diagnose [22] genutzt werden, um die Effizienz des Verfahrens zu steigern. Alle Fehler, die in einer FFR auftreten, müssen über den zugehörigen Fanout-Punkt propagiert werden. Wenn also der Fanout-Punkt robust ist, dann ist auch die Logik in der FFR robust.

Im Umkehrschluss gilt, wenn Ausgänge von FFRs als nicht robust oder nicht klassifizierbar gelten, dann müssen in einem nächsten Schritt die Komponenten in den FFRs betrachtet werden, da für diese nicht automatisch die gleiche Einstufung gilt.

Bei dem Verfahren werden zunächst nur die Fanout-Punkte für die Analyse von nicht robusten Komponenten verwendet – die Suche wird im ersten Schritt auf wesentliche Komponenten beschränkt. Das Ergebnis ist eine Menge von nicht robusten Fanout-Punkten. In einem zweiten Schritt erfolgt die Einbeziehung aller Komponenten der FFRs, deren Fanout-Punkt im ersten Schritt als nicht robust klassifiziert worden sind. Somit wurden alle nicht robusten Komponenten im Schaltkreis ermittelt. Auf ähnliche Art und Weise wird anschliessend bei den nicht klassifizierten Komponenten verfahren.

5.3 Inkrementelle Beweiser

Die Geschwindigkeit des Verfahrens lässt sich durch die Verwendung eines inkrementellen Bewei-

sers [21, 24, 19] steigern. Dabei wird überprüft, ob eine gelernte Informationen nach einer Veränderung der Probleminstanz gültig bleibt und entsprechend weiterverwendet werden kann, um den Suchraum einzuschränken. Bei dem hier vorgeschlagenen Verfahren werden in jeder Iteration Klauseln für den aktuellen Zeittakt t hinzugefügt und entfernt, wohingegen die Klauseln für die vorherigen Zeittakte nicht mehr verändert werden. Ein großer Teil der Probleminstanz bleibt also erhalten.

6 Experimentelle Ergebnisse

Im Folgenden werden die Ergebnisse der Experimente präsentiert. Im ersten Abschnitt wird die Robustheit in Abhängigkeit vom gewählten t^d analysiert, während im zweiten Teil der Fokus auf der maximalen Tiefe t^d zur Klassifizierung aller Komponenten und dem Einfluss der erreichbaren Zustände auf das Robustheitsmaß im Vordergrund steht. Maßnahmen zur Beschleunigung der Klassifikation bilden den Abschluss der Untersuchungen.

Die Benchmark-Suite enthält drei Arten sequentieller Schaltkreise:

- Ohne Schutzmechanismen
- Mit *Triple-Modular-Redundancy* (TMR)
- Mit Fehlererkennung

Die Schaltkreise ohne Schutzmechanismen wurden der ITC-99 Benchmark-Suite entnommen. Sie wurden im Folgenden mit b01–b10 gekennzeichnet.

Aus den Schaltkreisen b01, b02 und b06 dieser Benchmark-Suite wurde manuell ein TMR-Schaltkreis erstellt. Dafür wurden drei Instanzen des originalen Schaltkreises erzeugt und vor den primären Ausgängen kombinatorische Logik für den Mehrheitsentscheid der Ausgänge der drei Instanzen eingefügt. Die auf diese Weise erzeugten Schaltkreise haben somit die gleiche sequentielle Tiefe wie der originale Schaltkreis.

Das Hamming-Kode-Modell ist mit Fehlererkennung ausgestattet. Zusätzlich wurden die TMR Schaltkreise von b01, b02 und b06 mit einer Fehlererkennungslogik ausgestattet. Das Fehlersignal f^d überprüft, ob der Zustand der drei Instanzen identisch ist.

Alle Experimente wurden auf einem AMD Athlon 3700+ (2.2GHz, 1GB Speicher, Linux) durchgeführt. Als SAT-Beweiser wurde Zchaff mit inkrementeller Erweiterung verwendet [17].

6.1 Einfluss von t^d

In Abbildung 6 wurde exemplarisch für zwei Schaltkreise der Verlauf der unteren und oberen Schranke für die Robustheit in Abhängigkeit von der gewählten Tiefe (t^d) abgetragen. Im fehlerfreien Fall nicht erreichbare Zustände wurden durch einen

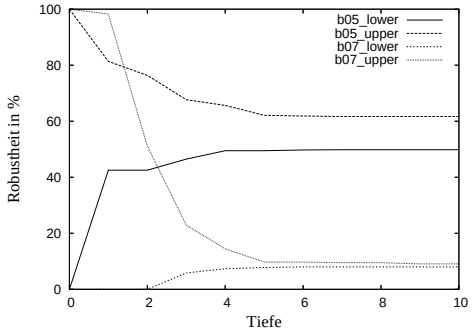

Abbildung 6. Entwicklung der Robustheit

BDD ermittelt und ausgeschlossen (siehe Abschnitt 5). Deutlich erkennbar ist die Annäherung der Schranken mit zunehmender Tiefe. Für $t^d = 5$ ist für b05 und b07 bereits eine hohe Genauigkeit erreicht, die sich in Laufe weiterer 5 Zeittakte nicht mehr stark verändert. Der Unterschied ist nach 10 Zeittakten bei b05 mit ca. 10% deutlich höher im Vergleich zu b07 mit nur 1%.

Der Verlauf der Robustheit kann so Anhaltspunkte für die zu wählende Tiefe liefern und hilft zwischen Genauigkeit und Berechnungsaufwand abzuwägen.

Tabelle 2. Maximum $t^d = 2$

circuit	$\#comp$	$\#FF$	t^d	$\|U\|$	R_{lb} %	R_{ub} %
b01	62	5	2	0	1.61	1.61
b02	33	4	2	0	3.03	3.03
b03	195	30	2	134	0.00	68.72
b04	821	66	2	88	0.00	10.72
b05	1198	34	1	0	5.59	5.59
b06	71	9	1	0	0.00	0.00
b07	512	49	2	72	0.56	14.65
b08	223	21	2	35	0.00	15.70
b09	197	28	2	45	0.00	22.84
b10	260	17	2	32	0.00	12.31
b01-tmr	210	15	2	3	1.90	3.33
b02-tmr	111	12	2	3	1.80	4.50
b06-tmr	273	27	2	0	3.66	3.66
hamming	282	7	2	169	5.67	65.60

Zusammenfassend werden in Tabelle 2 die Ergebnisse für $t^d = 2$ für die betrachteten Schaltkreise angegeben. Im Unterschied zu Abbildung 6 wurden die Ergebnisse in Tabelle 2 ohne Ausschluss der nichterreichbaren Zustände gewonnen.

Die Tabelle enthält den Namen des Schaltkreises (*circuit*), die Anzahl der Komponenten ($\#comp$) und der vorhandenen FFs ($\#FF$) im Schaltkreis. Weiterhin kennzeichnet $|U|$ die Anzahl der noch nicht klassifizierten Komponenten nach einer Abrolltiefe von t^d. Die Spalten R_{lb} und R_{ub} geben die ermittelten Schranken für die Robustheit an.

Bei vier der untersuchten Schaltkreise sind die untere und obere Schranke bereits gleich. Für diese Instanzen ist die Klassifizierung vollständig abgeschlossen. Es fallen für die Schaltkreise b03 und b09 die ho-

Tabelle 3. Einfluss der nicht erreichbaren Zustände auf die Robustheit

	—alle—			—erreichbar—		
circuit	t^d	R_{lb} %	R_{ub} %	t^d	R_{lb} %	R_{ub} %
b01	2	1.61	1.61	2	1.61	1.61
b02	2	3.03	3.03	2	3.03	3.03
b03	8	0.00	0.00	8	5.13	5.13
b05	1	5.59	5.59	26[a]	51.75	56.51
b06	1	0.00	0.00	1	0.00	0.00
b07	23	0.98	0.98	36[b]	7.23	7.42
b08	4	2.24	2.24	4	2.24	2.24
b09	8	0.00	0.00	8	0.51	0.51
b10	4	6.54	6.54	4	7.69	7.69
b01-tmr	2	3.33	3.33	9[c]	34.76	99.05
b02-tmr	2	4.51	4.51	9[d]	23.42	99.10
b06-tmr	2	3.66	3.66	9[e]	60.44	97.80
b01-tmrflt	0	4.69	4.69	0	99.06	99.06
b02-tmrflt	0	4.92	4.92	0	99.18	99.18
b06-tmrflt	0	46.13	46.13	0	97.89	97.89
hamming	6	30.50	30.50	6	67.73	67.73

[a]Memory out: $|U| = 57$, [b]Memory out: $|U| = 1$,
[c]Aborted: $|U| = 135$, [d]Aborted: $|U| = 84$, [e]Aborted: $|U| = 102$

hen Differenzen zwischen unterer und oberer Schranke auf. Es wurden also zahlreiche Komponenten noch nicht klassifiziert (siehe auch Spalte $|U|$); durch eine Erhöhung von t^d kann die Genauigkeit verbessert werden.

6.2 Erreichbarkeitsanalyse

In diesem Abschnitt folgen Untersuchung zum maximal notwendigen t^d, das zur Klassifizierung aller Komponenten notwendig ist. Darüber hinaus wird der Einfluss der erreichbaren Zustände auf die Robustheit diskutiert.

Die linke Hälfte von Tabelle 3 zeigt die Ergebnisse für alle erreichbaren Zustände und die rechte Hälfte für nur im fehlerfreien Fall erreichbare Zustände. In beiden Fällen wurde die Untersuchung bis zur Klassifizierung aller Komponenten durchgeführt. Die Spalten geben den Namen des Schaltkreises (*circuit*), t^d die Abrolltiefe, R_{lb} und R_{ub} die untere und obere Schranke an. Die im fehlerfreien Fall nicht erreichbaren Zustände werden durch eine Erreichbarkeitsanalyse basierend auf einem *Binary Decision Diagram* (BDD) [4] ermittelt, und als Initialzustand in der CNF Instanz ausgeschlossen.

Im Falle der Untersuchung aller Zustände ergibt sich eine geringe Robustheit für fast alle Schaltkreise. Aufgrund der auftretenden False Negatives werden robuste Komponenten als nicht robust klassifiziert.

Durch das Ausschließen nicht erreichbarer Zustände bleibt die Robustheit für einige Schaltkreise gleich, während sich bei anderen Schaltkreisen eine deutlich größere Robustheit ergibt. Ob eine Erreichbarkeitsanalyse notwendig ist, hängt also stark vom betrachteten Schaltkreis und Fehlererkennungsmechanismus ab. Durch die Verwendung der Erreichbarkeitsanalyse können sich, wie im Falle von b05 und b07, die notwendigen Abrolltiefen (t^d) deutlich erhöhen – bei gleichzeitiger Erhöhung der Genauigkeit.

Tabelle 4. Skalierung der ausgewählten nicht erreichbaren Zustände für $t^d = 0$

	—b06-tmr—		—b05—	
%	R_{lb} %	R_{ub} %	R_{lb} %	R_{ub} %
100	30.77	97.80	41.32	81.22
80	21.98	89.01	20.38	59.85
60	17.58	84.62	6.01	47.75
40	17.58	84.62	5.93	47.75
20	14.65	81.69	5.93	47.75
0	8.79	75.82	5.59	47.41

Besonders deutlich ist der Unterschied bei den TMR Schaltkreisen, da die redundanten Module im fehlerfreien Fall immer im gleichen Zustand sind. Dies wird bei Betrachtung aller Zustände nicht beachtet und führt somit zu False Negatives. Für TMR Schaltkreise ist die Einschränkung der initialen Zustände somit notwendig. Alternativ kann die aufwändige Erreichbarkeitsanalyse durch den manuellen Beweis einer Invariante ersetzt werden [11]. Andernfalls konvergieren die Schranken für die Robustheit auch bei hohen Abrolltiefen nicht.

Schaltkreise mit Fehlererkennung lassen sich dagegen gut handhaben. Fehlererkennung mit TMR ($tmrflt$) erlaubt es, auftretende Fehler unmittelbar festzustellen. Somit können robuste Komponenten sehr schnell klassifiziert werden. Im Fall des Hamming-Kodierers ist die Fehlererkennung verzögert, aber auch hier gelingt die Klassifizierung aller Komponenten.

Exemplarisch wurde in Tabelle 4 der Einfluss einer Überapproximation der erreichbaren Zustände untersucht. Die Menge aller nicht erreichbaren Zustände werden in einem BDD repräsentiert. Aus dem resultierenden BDD wurde dann ein Anteil (Spalte %) von Pfaden zufällig ausgewählt und blockiert. Ein Wert von 100% bedeutet, dass alle nicht erreichbaren Zustände blockiert werden.

Die Reduzierung der blockierten Pfade von 100% auf 0% führt bei beiden Instanzen zu einer Reduzierung der ermittelten Robustheit, da mehr False Negatives auftreten.

6.3 Maßnahmen zur Beschleunigung

Die in Abschnitt 5 vorgeschlagenen Verbesserungen zur Beschleunigung werden im Folgenden exemplarisch evaluiert.

Tabelle 5 gibt die Laufzeiten für $t^d = 2$ an. Es werden nur erreichbare Zustände betrachtet. In Spalte *basic* wurde kein inkrementeller Beweiser verwendet. Spalte I ist die Laufzeit bei Verwendung eines inkrementellen Beweisers (siehe Abschnitt 5.3). Für

Tabelle 5. Optimierungen

circuit	*basic*	I	F	$I+F$
b05	512.41s	91.32s	343.61s	73.83s
b06-tmr	19.06s	5.31s	16.01s	4.11s
hamming	73.88s	14.79s	70.46s	13.86s

Spalte F wurde das zweistufige Verfahren eingesetzt, das zunächst nur Fanout-Punkte betrachtet (siehe Abschnitt 5.2). Die Kombination beider Techniken liefert die Ergebnisse aus Spalte $I + F$.

Im Vergleich zum Basisalgorithmus lassen sich nach Anwendung der Optimierungstechniken Geschwindigkeitssteigerungen von bis zu einem Faktor 7 erreichen.

7 Zusammenfassung

Es wurde ein formales Modell zur Ermittlung der Robustheit eines Schaltkreises hinsichtlich Einfachfehlern vorgestellt. Zwei Schaltkreise werden einem sequentiellen Äquivalenzvergleich unterzogen, wobei in einem der Schaltkreise Fehler injiziert werden. Das Modell umgeht Komplexitätsprobleme, indem für eine gegebene Tiefe t^d die Robustheit in einer unteren und oberen Schranke angegeben wird. In detaillierten Analysen wurden der Einfluss der sequentiellen Tiefe auf die Genauigkeit und der Einfluss von False Negatives diskutiert.

Literatur

[1] T. Austin and V. Bertacco. Deployment of better than worst-case design: Solutions and needs. In *Int'l Conf. on Comp. Design*, pages 550–558, 2005.

[2] S. Borkar. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *IEEE Micro*, 25(6):10–16, 2005.

[3] M. Bozzano, A. Cimatti, and F. Tapparo. Symbolic fault tree analysis for reactive systems. In *Automated Technology for Verification and Analysis*, volume 4762 of *LNCS*, pages 162–176, 2007.

[4] R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[5] P. Civera, L. Macchiarulo, M. Rebaudengo, M. S. Reorda, and M. Violante. An FPGA-based approach for speeding-up fault injection campaigns on safety-critical circuits. *Jour. of Electronic Testing: Theory and Applications*, 18(3):261–271, 2002.

[6] S. Cook. The complexity of theorem proving procedures. In *3. ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[7] M. Davis, G. Logeman, and D. Loveland. A machine program for theorem proving. *Comm. of the ACM*, 5:394–397, 1962.

[8] R. Drechsler, W. Günther, and B. Stubert. Efficient (non-)reachability analysis of counterexamples. In *ITG/GI/GMM-Workshop "Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen"*, pages 250–259, 2004.

[9] N. Eén and N. Sörensson. An extensible SAT solver. In *SAT 2003*, volume 2919 of *LNCS*, pages 502–518, 2004.

[10] D. Ernst, N. S. Kim, S. Das, S. Pant, T. Pham, R. Rao, C. Ziesler, D. Blaauw, T. Austin, and T. Mudge. Razor: A low-power pipeline based on circuit-level timing speculation. In *Micro Conference*, 2003.

[11] G. Fey and R. Drechsler. Ein formaler Ansatz zum Robustheitsnachweis. In *GMM/GI/ITG-Fachtagung Zuverlässigkeit und Entwurf*, pages 101–108, 2007.

[12] G. Fey and R. Drechsler. A basis for formal robustness checking. In *Int'l Symp. on Quality Electronic Design*, pages 784–789, 2008.

[13] R. Hamming. Error detection and error correction codes. *The Bell System Technical Journal*, 24(2):147–160, 1950.

[14] U. Krautz, M. Pflanz, C. Jacobi, H. W. Tast, K. Weber, and H. T. Vierhaus. Evaluating coverage of error detection logic for soft errors using formal methods. In *Design, Automation and Test in Europe*, pages 176–181, 2006.

[15] R. Leveugle. A new approach for early dependability evaluation based on formal property checking and controlled mutations. In *IEEE International On-Line Testing Symposium*, pages 260–265, 2005.

[16] M. Miskov-Zivanov and D. Marculescu. Circuit reliability analysis using symbolic techniques. *IEEE Trans. on CAD*, 25(12):2638–2649, 2006.

[17] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Design Automation Conf.*, pages 530–535, 2001.

[18] S. A. Seshia, W. Li, and S. Mitra. Verification-guided soft error resilience. In *Design, Automation and Test in Europe*, pages 1442–1447, 2007.

[19] O. Shacham and K. Yorav. On-the-fly resolve trace minimization. In *Design Automation Conf.*, pages 594–599, 2007.

[20] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvis. Modeling the effect of technology trends on the soft error rate of combinational logic. In *Int'l Conf on Dependable Systems and Networks*, pages 389–398, 2002.

[21] O. Shtrichman. Pruning techniques for the SAT-based bounded model checking problem. In *CHARME*, volume 2144 of *LNCS*, pages 58–70, 2001.

[22] A. Smith, A. Veneris, M. Fahim Ali, and A.Viglas. Fault diagnosis and logic debugging using boolean satisfiability. *IEEE Trans. on CAD*, 24(10):1606–1621, 2005.

[23] G. Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic, Part 2*, pages 115–125, 1968. (Reprinted in: J. Siekmann, G. Wrightson (Ed.), Automation of Reasoning, Vol. 2, Springer, Berlin, 1983, pp. 466-483.).

[24] J. Whittemore, J. Kim, and K. Sakallah. SATIRE: A new incremental satisfiability engine. In *Design Automation Conf.*, pages 542–545, 2001.

[25] C. Zhao and S. Dey. Improving transient error tolerance of digital VLSI circuits using RObustness COmpiler (ROCO). In *Int'l Symp. on Quality Electronic Design*, pages 133–140, 2006.

[26] Q. Zhou and K. Mohanram. Gate sizing to radiation harden combinational logic. *IEEE Trans. on CAD*, 25(1):155–166, 2006.