Anwendungsbezogene Analyse der Robustheit von Digitalen Schaltungen

André Sülflow Stefan Frehse Görschwin Fey Rolf Drechsler

Fachbereich 3 – Mathematik und Informatik Universität Bremen, 28359 Bremen
{suelflow,sfrehse,fey,drechsle}@informatik.uni-bremen.de

Kurzfassung

Die kontinuierliche Reduzierung der Strukturgrößen führt zu einer erhöhten Fehleranfälligkeit von digitalen Schaltungen. Sowohl Produktionsfehler als auch transiente Fehler können die Funktionsfähigkeit beeinflussen. Aufgrund dessen wird die Untersuchung der Fehlertoleranz im Entwicklungsprozess zunehmend wichtig. Während simulative Methoden im Allgemeinen nur einen kleinen Anteil aller potentiell fehlerhaften Szenarien betrachten, können mit formalen Techniken alle Szenarien und alle potentiellen Fehlertypen untersucht werden.
In dieser Arbeit wird gezeigt, wie anwendungsspezifisches Wissen bei der formalen Untersuchung eingebracht werden kann. Hierzu werden zunächst Randbedingungen einer Anwendung ermittelt und im Anschluss zur Untersuchung der Fehlertoleranz unter Anwendungsbedingungen verwendet. Die Experimente zeigen, dass die Fehlertoleranz unter Berücksichtigung einer bestimmten Anwendung oftmals deutlich höher als die Fehlertoleranz ohne Restriktionen ist.

Abstract

Continuously shrinking feature sizes lead to an increasing vulnerability of digital circuits. Manufacturing failures and transient faults may tamper the functionality. Thus, analyzing the fault tolerance of a given implementation becomes an important step in the design process. This can be done by simulation based fault injection which does not cover all potential scenarios or by formal techniques which conservatively cover any scenario and any potential fault.
In this paper, we show how to use application specific knowledge when formally analyzing an implementation. Constraints are extracted from an application to evaluate the design under more realistic conditions. The experimental results show, that fault tolerance with respect to a certain application can be significantly higher than the fault tolerance without any restrictions.

1 Einleitung

Die stetige Verringerung der Strukturgrößen führt zu einer Zunahme des Einflusses von Umweltstrahlung auf die Funktionsfähigkeit digitaler Schaltungen. Durch Umweltstrahlung induzierte transiente Fehler können die Funktionsfähigkeit von Systemen massiv beeinflussen. Zusätzlich nimmt die Umweltstrahlung mit zunehmender Höhe zu, so dass sich bereits heute deutliche Beschränkungen für die Einsatzfähigkeit von Systemen ergeben. Dies ist von besonderer Bedeutung, wenn Standardkomponenten, wie zum Beispiel ein Mikro-Controller, in sicherheitskritischen Bereichen eingesetzt werden. Je nach Anwendung ergeben sich unterschiedliche Anforderungen für die Robustheit der Systeme. Um die Robustheit der Systeme abzusichern werden daher Verfahren auf Fertigungsebene [1] und auf Entwurfsebene [2] eingesetzt. Im Folgenden bezeichnet Robustheit die Fehlertoleranz eines Designs gegenüber transienten Fehlern. Transiente Fehler sind kurzzeitig auftretende Abweichungen des spezifizierten Verhaltens eines Schaltkreises und werden zum Beispiel durch *Single Event Upsets* (SEUs) ausgelöst.

Der Nachweis der Robustheit basiert heute oftmals auf simulations- bzw. emulationsbasierten Verfahren [3], welche Aufgrund der partiellen Untersuchungen nur eine ungenaue Aussage hinsichtlich der Robustheit treffen können. Analysemethoden, die die Verlässlichkeit analysieren, liefern typischerweise ein Maß zur Fehlerwahrscheinlichkeit [4]. Oft ist jedoch von Interesse, ob ein Fehler das Ausgabeverhalten ändern kann oder nicht.

Dies wird mit Methoden erreicht, die eine formale Analyse des Verhaltens unter Fehlerannahmen vornehmen [5, 6,

7, 8, 9, 10, 11]. Diese Verfahren sind vollständig bezüglich aller Eingabeszenarien und Zustände. Darüber hinaus sind die Verfahren aus [5, 7, 9] vollständig bezüglich betrachteter Eigenschaften, während [6, 8, 9, 10, 11] vollständig bezüglich des Verhaltens eines Referenz-Modells sind.

Die Verfahren sind konstruktiv, indem aufgezeigt wird, welche Komponente bei einem Ausfall beobachtbares Fehlverhalten hervorruft. Dort können entsprechende Maßnahmen getroffen werden. Die formalen Verfahren können jedoch zu einer „Überanpassung“ führen, wenn Eingabeszenarien betrachtet werden, die im praktischen Einsatz gar nicht vorkommen können. Der Entwerfer würde konservativ vorgehen und Schutzmechanismen konstruieren, die unnötig sind. Dies verursacht unnötigen Zeit- und vor allem Kostenaufwand.

Der Fokus der vorliegenden Arbeit ist es daher die Anwendung bei der formalen Analyse zu berücksichtigen. Hierzu werden funktionale Randbedingungen berücksichtigt, die bezüglich der Anwendung gültig sind. Die Randbedingungen können entweder symbolisch in Form von Eigenschaften formuliert werden oder aus Simulationsdaten gewonnen werden, zum Beispiel mittels einer Testbench. Hierbei wird das Verfahren aus [10, 11] erweitert. Im Resultat werden weniger Komponenten berechnet, die zusätzlicher Schutzmechanismen bedürfen – es kann also Chip-Fläche eingespart werden. Gleichzeitig bleibt die formale Analyse bezüglich aller möglichen Fehler, die auftreten können, vollständig und ist so der Simulation überlegen. Eine Beschränkung der Analyse auf eine Teilmenge des erwarteten Verhaltens wie z.B. in [7] wird ebenfalls nicht vorgenommen. Erste experimentelle Ergebnisse zeigen die deutliche Zunahme der Robustheit unter Berücksichtigung von Anwendungsdaten. Die qualitative Aussage des formalen Robustheitsnachweises wird erhöht und liefert dem Entwerfer hilfreiche Informationen.

Die Arbeit ist wie folgt gegliedert: Im folgenden Abschnitt werden Grundlagen erläutert. Abschnitt 3 untersucht die Auswirkungen der Randbedingungen auf die Ergebnisse der Robustheitsberechnung. In Abschnitt 4 wird gezeigt, wie die Randbedingungen für die Robustheitsanalyse aus Invarianten oder in einem simulations-basierten Verifikationsablauf gewonnen werden können. Experimentelle Ergebnisse werden in Abschnitt 5 angegeben und erläutert. In Abschnitt 6 wird die Arbeit zusammengefasst und offene Fragen werden aufgezeigt.

2 Grundlagen

2.1 Boolesche Erfüllbarkeit und Schaltkreise

Das *Boolesche Erfüllbarkeitsproblem* (SAT, von engl. satisfiability) ist ein Entscheidungsproblem, welches fragt ob es eine Variablenbelegung einer Booleschen Funktion $f : \mathbb{B}^n \rightarrow \mathbb{B}$ gibt, sodass die Funktion f zu logisch 1 auswertet. Wenn solch eine Belegung existiert, dann wird f erfüllbar genannt, andernfalls unerfüllbar. Das SAT-Problem liegt in der Klasse der NP-vollständigen Probleme [12]. Durch die fortgeschrittenen Techniken neuer SAT-Beweiser können praktische Problem-Instanzen meist effektiv gelöst werden [13, 14]. Als Eingabe der SAT-Beweiser dient die *Konjunktive Normalform* (KNF), welche eine Boolesche Funktion repräsentiert.

Betrachtet wird ein synchroner sequentieller Schaltkreis $\mathbb{C}$ mit *primären Eingängen* $\mathrm{PI}(\mathbb{C})$, *primären Ausgängen* $\mathrm{PO}(\mathbb{C})$ und *Zustandsbits* $\mathrm{S}(\mathbb{C})$. Die Anzahl der Komponenten eines Schaltkreises wird durch $|\mathbb{C}|$ beschrieben. Komponenten können von unterschiedlicher Granularität sein, wie z.B. ein einzelnes Gatter, ein Modul oder ein Quellcode-Ausdruck einer Hardware-Beschreibungssprache. Ein Schaltkreis kann in linearer Zeit mit linearem Speicherbedarf in $|\mathbb{C}|$ in eine KNF übersetzt werden [15].

2.2 Formale Robustheitsberechnung

Im Folgenden wird das Verfahren aus [11], eine Weiterentwicklung von [10], eingesetzt. Dieses Verfahren berechnet die Robustheit eines Schaltkreises gegenüber *einzeln* auftretenden transienten Fehlern, sogenannten SEUs. Die Analyse wird dabei auf einen Beobachtungszeitraum von t^d Taktzyklen beschränkt. Hier liegt die Annahme zu Grunde, dass ein Fehlereffekt nach kurzer Zeit behoben werden muss, ansonsten ist das System nicht robust.

Eine Komponente gilt als *nicht-robust*, falls ein transienter Fehler an dieser Komponente innerhalb von t^d Takten am Ausgang beobachtet werden kann. Alle nicht-robusten Komponenten werden in der Menge $\mathbb{S}$ zusammengefasst. Eine Komponente gilt als *nicht-klassifiziert*, falls ein transienter Fehler an dieser Komponente zwar nicht am Ausgang beobachtet werden kann, aber zu einer Änderung im Speicher führen kann, d.h. es tritt eine *Silent Data Corruption* (SDC) auf. Die Menge $\mathbb{U}$ enthält alle nicht-klassifizierten Komponenten. Daraus ergibt sich die Menge $\mathbb{T}$ der robusten Komponenten durch $\mathbb{T} = \mathbb{C} \setminus (\mathbb{U} \cup \mathbb{S})$. Mittels dieser Mengen werden Schranken für die Robustheit bestimmt:

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

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

Tatsächlich hängen die Schranken von t^d ab. Je länger der betrachtete Zeitraum ist, desto weniger Komponenten bleiben nicht-klassifiziert.

Intern betrachtet das Verfahren den Schaltkreis nur über t^d Takte, um die Komplexität zu reduzieren[1]. Der fehlerfreie

[1] Für eine vollständige Analyse ist t^d gemäß der sequentiellen Tiefe des Schaltkreises zu wählen. Für komplexe Schaltkreise kann die sequentielle Tiefe allerdings hoch sein.

Schaltkreis wird mit einem fehlerhaften Schaltkreis verglichen. Durch Verwendung eines SAT-Beweisers ist eine Probleminstanz ausreichend, um alle möglichen Fehler und alle möglichen Eingabesequenzen zu analysieren.

Damit Vollständigkeit bezüglich aller Zustände erreicht wird, beginnt die Betrachtung nicht beim Startzustand, sondern bei einer Menge S von Zuständen. Entspricht S der Menge der erreichbaren Zustände S^*, so ergeben sich exakte Schranken. Oft kann die Menge der erreichbaren Zustände aber nur approximativ bestimmt werden. Bei einer Überapproximation $S^{\uparrow} \supseteq S^*$ der erreichbaren Zustände, können robuste Komponenten als nicht-robust klassifiziert werden. Daraus ergibt sich eine untere Abschätzung der unteren Schranke der Robustheit $R_{lb}(S^{\uparrow}) \leq R_{lb}(S^*)$. Bei einer Unterapproximation $S^{\downarrow} \subseteq S^*$ werden nicht-robuste Komponenten als robust eingestuft. Es ergibt sich eine obere Abschätzung der oberen Schranke $R_{ub}(S^{\downarrow}) \geq R_{ub}(S^*)$. Im Folgenden werden nur Unterapproximationen der erreichbaren Zustände verwendet, so dass immer R_{ub} betrachtet wird. Der Einfachheit halber wird statt R_{ub} einfach R geschrieben.

So wie in [10, 11] beschrieben betrachtet das Verfahren *alle* möglichen Eingabeszenarien bzgl. $\mathrm{PI}(\mathbb{C})$ und $\mathrm{S}(\mathbb{C})$. Hier soll diese Menge entsprechend der Anwendung angepasst werden. Dazu werden die zulässigen Eingabewerte durch ihre charakteristische Funktion I angegeben. Insbesondere sind alle Eingabewerte zugelassen, falls I der konstanten Booleschen Funktion $\underline{1}$ entspricht. Insgesamt ist die Robustheit R somit in Abhängigkeit von t^d, S und der Menge der zugelassenen Eingabewerte I zu bestimmen und kann geschrieben werden als $R(t^d, S, I)$.

3 Berücksichtigung von Anwendungsdaten

Das in Abschnitt 2.2 eingeführte formale Verfahren berechnet die Robustheit eines Schaltkreises bezüglich aller erreichbaren Zustände, d.h. $S = S^*$, und unter *allen* Eingabebelegungen, d.h. $I = \underline{1}$.

Typischerweise unterliegen im praktischen Einsatz sowohl S als auch I Restriktionen. Zum Beispiel können bestimmte Eingaben einer Kodierung unterliegen. So ist bei einer *one-hot*-Kodierung nur eines der Eingabebits gesetzt oder ein fehlererkennender Kode lässt nur bestimmte Eingaben zu. Solches Wissen kann bei der Implementierung genutzt werden, um den Platzbedarf zu reduzieren. Werden diese Randbedingungen bei der Robustheitsberechnung nicht berücksichtigt, werden unter Umständen mehr Schutzmechanismen implementiert als notwendig. Durch eine Einschränkung der möglichen Eingaben, wird auch die Menge der im praktischen Einsatz erreichbaren Zustände verändert, wodurch die Robustheitsanalyse beeinflusst wird. In dieser Arbeit werden daher Anwendungsdaten eingesetzt um Randbedingungen für S und I zu ermitteln, so dass genauere Aussagen über die Robustheit bezüglich einer Anwendung getroffen werden. Die Umgebung wird hierbei als fehlerto-

```
1 vunit t1(circuit){
2  property P1 =
3    always (
4       X_1 == X_2
5       && X_3 == 0
6    );
7 }
```

Abbildung 1. Spezifikation von Invarianten in PSL

lerant ausgelegt angenommen. Ist das nicht der Fall, müssen die Randbedingungen entsprechend gelockert werden.

Im Einsatz ergeben sich Randbedingungen für die möglichen Eingaben. Hier werden diese Randbedingungen durch eine Invariante α beschrieben, die nur bestimmte Wertekombinationen an den Eingängen zulässt. Ebenso kann durch die Anwendung ein neuer Startzustand bzw. eine Menge neuer Startzustände beschrieben werden, zum Beispiel in Form eines auszuführenden Programms im Speicher eines Prozessors. Diese Startzustände werden durch eine Invariante β charakterisiert[2]. Technisch werden beide Arten von Invarianten in einer Teilmenge der *Property Specification Language* (PSL) [16] spezifiziert. Hierbei sind kombinatorische Beschränkungen für die primären Eingänge und die initialen Zustände erlaubt. Das bedeutet, alle Beschränkungen beziehen sich auf exakt einen Zeittakt.

Beispiel 1 *Ein Beispiel ist in Abbildung 1 angegeben. Die primären Eingänge* `X_1` *und* `X_2` *erhalten die Daten vom gleichen Sensor und sind somit immer identisch. Der primäre Eingang* `X_3` *ist hingegen dauerhaft auf logisch* false *gesetzt.*

Die Invariante α charkaterisiert die primären Eingänge für alle Zeittakte auf gleiche Weise[3]. Das bedeutet, Eingangsbelegungen, die nicht den spezifizierten Randbedingungen entsprechen, können nicht mehr erzeugt werden. Auf diese Weise verringert sich der Eingaberaum, so dass Komponenten, die unter allen Eingaben $\underline{1}$ als nicht-robust galten, nun robust bzw. nicht-klassifizierbar sind. In Folge dessen werden weniger Komponenten als nicht-robust klassifiziert, das heißt, unter den Randbedingungen α wird eine höhere Robustheit R des Schaltkreises beobachtet und vom Verfahren berechnet. Im Allgemeinen gilt deshalb: $R(t^d, S, \alpha) \geq R(t^d, S, \underline{1})$.

Durch eine Erreichbarkeitsanalyse wird die Menge der erreichbaren Zustände S_{reach} ausgehend von einer Menge von Zuständen S_0 bestimmt. Sei $reach : S \times I \rightarrow S$ eine Funktion zur Berechnung von S_{reach}. Sei S_0 die Menge der

[2] Die Invarianten α und β charakterisieren eine Menge von Eingabewerten bzw. von Zuständen. Zur Vereinfachung werden im Folgenden die Symbole α und β sowohl für die Invariante als auch die jeweilige Menge verwendet.

[3] Eine Erweiterung auf komplexere zeitabhängige Charakterisierungen ist möglich.

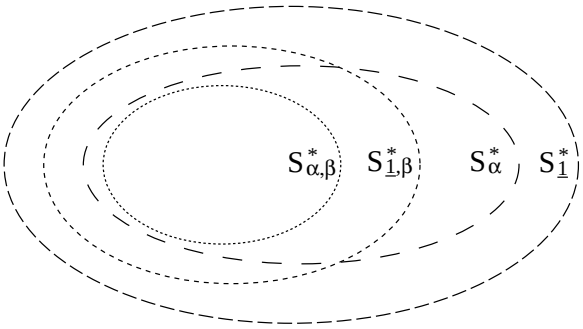


Abbildung 2. S^* **unter Randbedingungen**

Startzustände (Reset-Zustände) des Schaltkreises, dann gibt $S^*_{\underline{1}} = reach(S_0, \underline{1})$ die Menge der erreichbaren Zustände ohne Beschränkung der Eingaben und $S^*_{\alpha} = reach(S_0, \alpha)$ die erreichbaren Zustände unter Randbedingungen an. Analog bezeichnet $S^*_{\underline{1},\beta} = reach(\beta, \underline{1})$ die erreichbaren Zustände ohne Randbedingungen und $S^*_{\alpha,\beta} = reach(\beta, \alpha)$ mit Randbedingungen für die Eingänge, für $\beta \subseteq S^*_{\underline{1}}$. Abbildung 2 visualisiert diese Mengen.

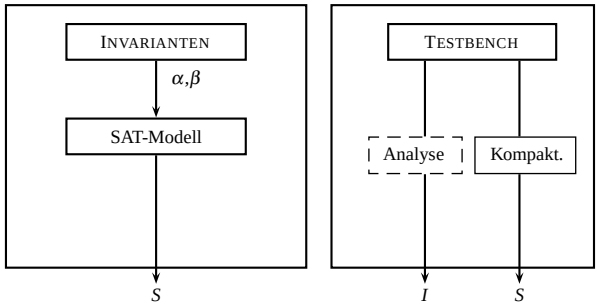
Es gilt dann: $S^*_{\underline{1}} \supseteq S^*_{\alpha}$ und $S^*_{\underline{1},\beta} \supseteq S^*_{\alpha,\beta}$. Für die Robustheit ergibt sich: $R(t^d, S^*_{\underline{1}}, I) \leq R(t^d, S^*_{\alpha}, I)$ und $R(t^d, S^*_{\underline{1},\beta}, I) \leq R(t^d, S^*_{\alpha,\beta}, I)$.

Die bisherige Betrachtung beschränkt sich auf formale Analysen. Tatsächlich stehen bezüglich einer bestimmten Anwendung aber oft auch Daten aus einer Simulation oder aus existierenden realen Systemen zur Verfügung. Diese beobachtbaren Anwendungsdaten sind typischerweise eine Unterapproximation aller möglichen Anwendungsdaten, da Testbenches und Simulationsläufe nur einen Teil aller Szenarien abdecken können. Analog zu $S^*_{\underline{1}}$ lassen sich dann alle beobachteten erreichten Zustände ohne Anwendung der Randbedingungen als $S^{sim}_{\underline{1}}$ bezeichnen. Werden nun nur die Simulationsläufe berücksichtigt, die den Randbedingungen α genügen, ergibt sich eine Menge S^{sim}_{α}. Es gilt also: $S^*_{\underline{1}} \supseteq S^{sim}_{\underline{1}} \supseteq S^{sim}_{\alpha}$, daraus folgt $R(t^d, S^*_{\underline{1}}, I) \leq R(t^d, S^{sim}_{\underline{1}}, I) \leq R(t^d, S^{sim}_{\alpha}, I)$.

Um die Randbedingungen im Verfahren aus Abschnitt 2.2 berücksichtigen zu können, muss eine kompakte Formulierung für α und β verfügbar sein, die als Zusatzbedingungen in einer SAT-Instanz berücksichtigt werden können. Der folgende Abschnitt erläutert, wie diese Beschreibungen gewonnen werden können.

4 Ermittlung von Randbedingungen

Im Nachfolgenden werden Verfahren zur Ermittlung der anwendungsspezifischen Daten für die Eingänge und von erreichbaren Zuständen vorgestellt. Der Fokus ist im ersten Teil auf der automatischen Generierung der Anwendungsdaten aus Invarianten (Abschnitt 4.1). Im zweiten Teil werden die Möglichkeiten der Extraktion von Randbedingungen aus Testbenches untersucht (Abschnitt 4.2).



(a) Invarianten (b) Testbench

Abbildung 3. Ermittlung der Randbedingungen

Einen Überblick der angewendeten Methoden liefert Abbildung 3. Die Zustandsmenge S und die Menge der betrachteten Eingaben I für die Robustheitsberechnung können entweder aus Invarianten (Abbildung 3(a)) oder aus einer Testbench (Abbildung 3(b)) gewonnen werden. Die beiden Alternativen werden in den folgenden Unterabschnitten diskutiert.

4.1 Invarianten

Über Invarianten lassen sich sowohl Angaben zu gültigen Eingaben α, z.B. konstante Eingänge oder Abhängigkeiten zwischen den Eingängen, als auch gültige (partielle) Belegungen der Zustände β spezifizieren, z.B. ein abzuarbeitendes Programm im Speicher eines Prozessors.

Die Invarianten selbst müssen zunächst manuell spezifiziert werden. Dies kann durch Ableitung aus der Spezifikation oder durch eine automatische Extraktion von Randbedingungen aus umgebenden Blöcken eines Schaltkreises geschehen.

Bei manuellen Invarianten ist es die Aufgabe des Entwerfers sicherzustellen, dass die Invarianten keine zu starke Einschränkung der erreichbaren Zustände bezüglich der Anwendung vornimmt. Sind die Invarianten zu strikt gewählt, werden nicht alle Zustände der Anwendung sichtbar und es werden zu viele Komponenten als robust klassifiziert. Die Angabe von unvollständigen Invarianten ist hingegen weniger kritisch, da hierdurch eine Überapproximation der in der Anwendung erreichbaren Zustände erreicht wird. Komponenten werden fälschlich nicht-robust klassifiziert. Als Nachteil resultiert hier, dass unter Umständen zu viele Schutzmechanismen implementiert werden, so dass zusätzlicher Logikaufwand und damit Flächenbedarf entsteht.

Sowohl α als auch β bilden die Grundlage zur Berechnung der erreichbaren Zustände $S^*_{\alpha,\beta}$ einer Anwendung. Eine exakte, vollständige Erreichbarkeitsanalyse, z.B. unter Verwendung von BDDs, ist in den meisten Fällen aufgrund von Speicher- und Zeitrestriktionen nicht durchführbar.

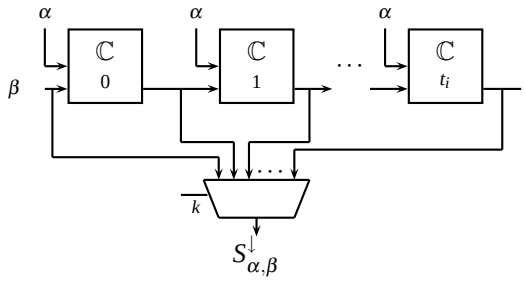


Abbildung 4. SAT-basierte Berechnung von $S^{*\downarrow}_{\alpha,\beta}$

Wie in Abbildung 3(a) dargestellt, wird daher eine Methode zur Berechnung einer Unterapproximation $S^{*\downarrow}_{\alpha,\beta} \subseteq S^{*}_{\alpha,\beta}$ mittels SAT vorgestellt.

Ein Modell für die SAT-basierte Berechnung von $S^{*\downarrow}_{\alpha,\beta}$ ist in Abbildung 4 gezeigt. Zunächst wird $\mathbb{C}$ über t_i Zeittakte abgerollt und β festgelegt. Für jeden Zeittakt werden die Belegungen der Eingänge mit α eingeschränkt. Die von β unter α innerhalb von t_i Zeittakten erreichbaren Zustände werden anschließend mit einem Multiplexer verbunden, wobei der select-Eingang (k) nicht festgelegt wird. Alle Zustände, die am Ausgang des Multiplexers sichtbar werden, bilden somit eine unterapproximierte Menge der erreichbaren Zustände $S^{*\downarrow}_{\alpha,\beta}$. Mit jeder Erhöhung von t_i werden im Allgemeinen mehr Zustände sichtbar, jedoch erhöht sich gleichzeitig auch der Berechnungsaufwand.

Der Vorteil der SAT-basierten Berechnung ist, dass zum Robustheitsnachweis kein explizites Aufzählen der erreichbaren Zustände notwendig ist. Diese Bedingungen zur Berechnung von $S^{*\downarrow}_{\alpha,\beta}$ werden der SAT-Instanz für den Robustheitsnachweis hinzugefügt. Außerdem wird die Bedingung $S == S^{*\downarrow}_{\alpha,\beta}$ eingefügt. Hierdurch ist garantiert, dass bei der Robustheitsberechnung nur Zustände aus $S^{*\downarrow}_{\alpha,\beta}$ berücksichtigt werden.

Eine weitere Möglichkeit zur Berechnung von $S^{*\downarrow}_{\alpha,\beta}$ ist der Einsatz von Simulation. Hierfür werden zunächst Stimuli erzeugt, die gemäß α gültig sind. Im Anschluss werden diese Stimuli ausgehend von dem Initialzustand β simuliert. Die durch Simulation erreichten Zustände lassen sich anschließend, z.B. durch Speicherung in einem BDD, kompaktieren und bilden $S^{*\downarrow}_{\alpha,\beta}$. Hierbei wird eine effiziente Datenstruktur zur Speicherung und zur Extraktion der nicht-erreichbaren Zustände benötigt. Allerdings werden, im Allgemeinen, durch Simulation weniger erreichbare Zustände erzeugt als durch die SAT-basierte Formulierung. Daher ist im Fokus dieser Arbeit die SAT-basierte Berechnung.

4.2 Daten aus Testbenches

Daten aus Testbenches liefern wertvolle Informationen über das Verhalten eines Schaltkreises unter Anwendungsbedingungen und es lassen sich Invarianten ableiten. Typischerweise spezifiziert eine Testbench dabei das Verhalten mehrerer Anwendungen, so dass die Menge der sichtbaren Zustände zum Einen unvollständig bzgl. einer konkreten Anwendung und zum Anderen auch Zustände anderer Anwendungen sichtbar sind. Ähnliches gilt für Randbedingungen für die Eingaben. Zur Erhöhung der Genauigkeit ist es daher erforderlich zunächst gezielt Testfälle bzgl. einer Anwendung zu selektieren.

Aus den Daten einer Testbench lassen sich automatisch partielle Abhängigkeiten der Eingänge bzw. der Zustände ermitteln. Ein Entwerfer kann auf Basis der automatisch extrahierten Informationen im Anschluss gezielt Randbedingungen formulieren. Abbildung 3(b) visualisiert dieses Vorgehen.

Methoden zur Extraktion von Eigenschaften aus Simulationsdaten können Randbedingungen α für die Eingänge liefern, z.B. [17].

Die beobachteten Zustände können zunächst kompaktiert und hinsichtlich partieller Überdeckungen untersucht werden, z.B. ein konkretes Programm im Programmspeicher eines Prozessors. Hier liegt eine reine Simulation zu Grunde, so dass für die Robustheitsberechnung $S^{sim}_{\alpha,\beta}$ geliefert wird.

Statt alle Register in den beobachteten Zuständen zu berücksichtigen, können wichtige Register, die zum Beispiel den Kontrollfluss bestimmen, ausgewählt werden. Nur bezüglich dieser Register werden dann die erlaubten Zustände eingeschränkt. Da eine Testbench zwar die meisten spezifizierten Transaktionen und damit die wichtigen Kontrollzweige, aber nicht alle möglichen Datenwerte betrachtet, kann auf diese Weise eine gute Überdeckung der relevanten Funktionalität erreicht werden.

5 Experimentelle Ergebnisse

Der folgende Abschnitt beschreibt erste experimentelle Ergebnisse des beschriebenen Verfahrens. Dabei wurden die Algorithmen in C++ implementiert und in das Verifikationswerkzeug WOLFRAM [18] integriert. Die Experimente wurden auf einem Intel Core 2 Duo Prozessor (2.33 GHz, 3 GB Speicher) unter MacOSX 10.5 durchgeführt. Als SAT-Beweiser wurde Zchaff [14] mit inkrementeller Erweiterung verwendet. Das Verfahren wurde auf einer Teilmenge von Schaltkreisen aus den ITC'99–Benchmarks evaluiert. Die Fehlertoleranz der verwendeten Schaltkreise ist gering. Die Robustheit wird bezüglich Invarianten (Tabelle 1) und einer Testbench (Tabelle 2) bestimmt. Bei der Robustheitsberechnung wird bei Erreichen einer maximalen Abrolltiefe t^d von 10 abgebrochen.

Tabelle 1. Invarianten

BENCHMARK			$S_{\underline{1}}^{*\downarrow}$ Formal ($t_i = 10$)						$S_{\alpha}^{*\downarrow}$ Formal ($t_i = 10$)					
			$\underline{1}$			α			$\underline{1}$			α		
NAME	$\lvert\mathbb{C}\rvert$	$\lvert FF\rvert$	t^d	R%	Zeit(s)	t^d	R%	Zeit(s)	t^d	R%	Zeit(s)	t^d	R%	Zeit(s)
b01	64	5	4	**0,00**	0,42	4	**0,00**	0,46	4	**0,00**	0,44	4	**0,00**	0,45
b02	34	4	3	**0,00**	0,13	10	**2,94**	0,17	4	**0,00**	0,15	10	**14,71**	0,17
b03	199	30	9	**0,50**	4,88	10	**5,03**	4,71	10	**2,01**	5,05	10	**11,56**	4,58
b04	832	66	5	**0,00**	78,81	10	**34,01**	49,74	10	**6,37**	76,63	10	**36,66**	48,50
b05	1.199	34	10	**90,33**	16,47	10	**90,83**	15,31	10	**90,33**	15,42	10	**90,83**	14,50
b06	73	9	2	**0,00**	0,49	2	**0,00**	0,52	2	**0,00**	0,52	2	**0,00**	0,56
b07	513	49	10	**87,33**	4,69	10	**87,33**	4,70	10	**87,52**	3,85	10	**87,52**	3,67
b08	232	21	10	**5,12**	7,64	10	**5,17**	8,00	10	**10,78**	7,77	10	**10,78**	7,86
b09	198	28	10	**47,47**	2,83	10	**48,48**	2,81	10	**75,76**	1,65	10	**83,84**	1,12
b10	271	17	10	**1,85**	13,30	10	**1,86**	13,76	10	**2,21**	15,77	10	**2,21**	15,79
b11	874	31	10	**8,92**	79,04	10	**8,92**	81,64	10	**20,02**	72,23	10	**20,02**	71,97
b12	1.302	121	10	**52,07**	118,59	10	**52,07**	120,14	10	**52,07**	123,56	10	**52,07**	119,14
b13	410	53	10	**52,68**	11,88	10	**54,88**	10,61	10	**52,68**	12,47	10	**54,88**	10,45

5.1 Invarianten

In Tabelle 1 sind die Ergebnisse der Robustheit bezüglich formal spezifizierter Invarianten angegeben. Die Spalte BENCHMARK gibt Name, Größe und Anzahl der Speicherelemente eines Schaltkreises an. Für die ausgewählten Schaltkreise wurden verschiedene Invarianten, in der Art wie sie in Beispiel 1 angegeben worden sind, manuell in PSL formuliert. Für die Berechnung der Robustheit wurde die Menge der betrachteten Zustände in zwei Varianten untersucht:

1. $S_{\underline{1}}^{*\downarrow}$ Formal – Beim SAT-basierten Verfahren aus Abschnitt 4.1 wurde der Initialzustand verwendet, die Eingabewerte wurden nicht eingeschränkt.

2. $S_{\alpha}^{*\downarrow}$ Formal – Wie oben, aber die Eingabewerte wurden durch die Invariante α beschränkt.

Beschränkungen für die Initialzustände (β) wurden in diesem Abschnitt nicht betrachtet, ergeben sich aber analog zu der theoretischen Analyse in Abschnitt 3.

Bei jeder dieser Varianten wurden während der Robustheitsberechnung entweder keine ($\underline{1}$) Restriktionen für Eingabewerte oder die Invariante α gewählt. Dabei werden die Spalten wie folgt interpretiert: t^d steht für die erreichte Abrolltiefe, R% für berechnete Robustheit und *Zeit* für die benötigte Laufzeit in CPU-Sekunden.

Beim Vergleich von $S_{\underline{1}}^{*\downarrow}$ und $S_{\alpha}^{*\downarrow}$ zeigt sich, dass die berechnete Robustheit teilweise zunimmt. Die Berücksichtigung anwendungsspezifischer Randbedingungen führt zu einer Einschränkung der betrachteten Zustände. So konnte beispielsweise für *b09* (Spalte $\underline{1}$) die Anzahl von nicht-robusten Komponenten um über die Hälfte von 104 (47,47%) auf 48 (75,76%) reduziert werden. Die gleiche Beobachtung gilt, wenn auch während der Robustheitsberechnung Randbedingungen auf den Eingängen berücksichtigt werden (Spalte α).

Die Laufzeit beim Vergleich von $S_{\underline{1}}^{*\downarrow}$ und $S_{\alpha}^{*\downarrow}$, sowohl mit ($\underline{1}$) als auch ohne Beschränkung der Eingaben (α), ist ähnlich. Der Aufwand zur Klassifikation von Komponenten ist trotz zusätzlicher Randbedingungen (α) nicht signifikant unterschiedlich.

5.2 Testbench

Tabelle 2 stellt die Ergebnisse der Robustheitsberechnung auf Basis einer Testbench dar. Anhand der Testbench wurden erreichbare Zustände gemäß einer Anwendung (S_{α}^{sim}) und mehrerer Anwendungen ($S_{\underline{1}}^{sim}$) generiert. Hierzu wurde zunächst versucht maximal 300 Zustände bzgl. der Anwendung zu erzeugen (S_{α}^{sim}), α wurde zuvor manuell angegeben. Im Anschluss erfolgte die Erzeugung von maximal 300 zusätzlichen Zuständen bzgl. aller Anwendungen für $S_{\underline{1}}^{sim}$. Auf dieser Basis wurde die Robustheit wieder mit (α) und ohne ($\underline{1}$) Beschränkungen der Eingänge bezüglich der Anwendung bestimmt. Angaben zur Simulation werden mit den Spalten *Traces* für die Anzahl der Simulationstraces und $|S|$ Anzahl der extrahierten Zustände dargestellt.

Es ist zu erkennen, dass die Robustheit, wie auch in Tabelle 1, mit Beschränkung der Eingänge größer ist. Der Vergleich zwischen Testdaten aus einer Anwendung (S_{α}^{sim}) und aus mehreren Anwendungen ($S_{\underline{1}}^{sim}$) zeigt, dass die Begrenzung auf eine Anwendung die Genauigkeit der Analyse deutlich steigern kann. So zeigt sich für b02 ein deutlicher Unterschied von ca. 40%. Bzgl. der Anwendung ungültige Zustände führen hier zu einer deutlichen Verschlechterung der Robustheit. Die Aufgabe eines Entwerfers ist es

Tabelle 2. Testbench

	$\lvert S_1^{sim}\rvert \leq 600$								$\lvert S_\alpha^{sim}\rvert \leq 300$							
	Simulation		$\underline{1}$			α			Simulation		$\underline{1}$			α		
NAME	$\lvert Traces\rvert$	$\lvert S\rvert$	t^d	R%	Zeit(s)	t^d	R%	Zeit(s)	$\lvert Traces\rvert$	$\lvert S\rvert$	t^d	R%	Zeit(s)	t^d	R%	Zeit(s)
b01	103.002	18	4	**0,00**	0,21	4	**0,00**	0,24	2.002	11	4	**0,00**	0,22	4	**0,00**	0,23
b02	102.001	8	3	**0,00**	0,08	10	**2,94**	0,10	1.001	4	4	**0,00**	0,08	10	**14,71**	0,10
b03	4.488	355	10	**0,05**	3,92	10	**6,03**	3,87	4.004	85	10	**2,01**	2,78	10	**11,56**	2,74
b04	696	598	10	**1,80**	355,00	10	**34,01**	342,79	390	300	10	**6,97**	89,69	10	**36,66**	74,84
b05	102.001	68	10	**56,55**	37,80	10	**56,63**	37,95	1.001	68	10	**56,55**	37,80	10	**56,63**	37,86
b06	103.002	13	2	**0,00**	0,25	2	**0,00**	0,27	2.002	5	10	**1.37**	0,28	10	**1,37**	0,31
b07	102.001	42	10	**4,87**	13,64	10	**4,87**	13,30	1.001	2	10	**87,52**	2,14	10	**87,52**	2,01
b08	56.423	526	10	**7,33**	4,49	10	**7,33**	4,82	56.085	300	10	**9,05**	4,21	10	**9,05**	4,55
b09	102.001	30	10	**10,10**	2,18	10	**17,68**	2,23	1.001	2	10	**75,76**	1,07	10	**83,84**	0,72
b10	101.214	390	10	**1,85**	8,82	10	**1,85**	10,49	100.100	103	10	**2,21**	7,47	10	**2,21**	7,85
b11	2.100	584	10	**13,39**	48,38	10	**13,39**	50,26	1.773	300	10	**20,02**	38,87	10	**20,02**	39,73
b12	771	587	10	**52,07**	78,90	10	**52,07**	80,09	307	300	10	**52,07**	70,68	10	**52,07**	70,78
b13	100.482	300	10	**21,22**	12,40	10	**23,42**	11,69	100.100	4	10	**58,29**	4,99	10	**68,54**	3,44

daher möglichst genaue anwendungsbezogene Daten zur Verfügung zu stellen.

Gleichzeitig liegt die Robustheit teilweise unter- bzw. oberhalb der in Tabelle 1 bestimmten Werte. Ursache hierfür ist, dass bei der Berechnung für Tabelle 1 der Zustandsraum nur in einer Tiefe von 10 Takten ausgehend vom Initialzustand untersucht wurde. Abhängig von der Suchraumtraversierung werden durch Simulation unterschiedliche Zustände erreicht.

Die erreichbaren Zustände werden nun aus den gespeicherten Werten extrahiert und in der SAT-Instanz kodiert. Hier werden die erreichbaren Zustände zunächst in einem BDD gespeichert und im Anschluss alle nicht-erreichbaren Zustände in der SAT-Instanz durch Klauseln ausgeschlossen. Diese Form der Kodierung ist nicht sehr kompakt, die oben verwendeten formal spezifizierten Invarianten ergeben typischerweise eine viel kleinere Struktur in der Probleminstanz.

In weiteren experimentellen Untersuchungen mit bis zu 10.000 Zuständen wurden die theoretischen Untersuchungen aus Abschnitt 4 bestätigt, dass mit Verwendung von mehr erreichbaren Zuständen mehr nicht-robuste Komponenten gefunden werden und die ermittelte Robustheit mit steigender Anzahl an Zuständen abnimmt. Gleichzeitig steigt die Laufzeit aber erwartungsgemäß stark an, da die Zustände zunächst in einem BDD kompaktiert und anschließend alle nicht-erreichbaren Zustände berechnet und ausgeschlossen werden müssen. Zum Beispiel erhöhte sich für *b12* die Laufzeit um den Faktor 24.

Insgesamt zeigt sich, dass alle vorgestellten Methoden geeignet sind, um Randbedingungen bezüglich einer Anwendung in die formale Robustheitsanalyse einzubringen. Die berechnete Robustheit der Schaltkreise hängt dabei wesentlich von (1) den verwendeten erreichbaren Zuständen und von (2) den Beschränkungen der Eingänge ab.

6 Zusammenfassung

In dieser Arbeit wurde ein Ansatz vorgestellt um Anwendungsdaten bei der formalen Robustheitsanalyse zu berücksichtigen. Dabei zeigte sich, dass die Genauigkeit der ermittelten Robustheit durch Anwendungsdaten deutlich gesteigert wird. In Folge dessen werden weniger Komponenten nicht-robust klassifiziert und müssen redundant ausgelegt werden – die Kosten verringern sich.

Eine alternative Anwendung des Verfahrens wäre die Optimierung einer existierenden Implementierung bezüglich einer Anwendung. Fehlverhalten robuster Komponenten wird nicht an den Ausgängen beobachtbar. Demzufolge können diese Komponenten entfernt werden, ohne das Verhalten bezüglich der Ausgabe zu verändern.

Aus der Arbeit ergeben sich aber auch verschiedene Fragestellungen für zukünftige Arbeiten. So sollen alternative Ansätze zur Kompaktierung der Daten aus einer Simulation untersucht werden. Bezüglich der Approximationsverfahren, sollen das SAT-basierte und das auf einer Testbench basierende Verfahren kombiniert werden. So kann der Zustandsraum in noch größerer Tiefe untersucht werden.

Literatur

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

[2] A. Pan, O. Khan, and S. Kundu. Improving yield and reliability of chip multiprocessors. In *Design, Automation and Test in Europe*, pages 490–495, 2009.

[3] A. Pellegrini, K. Constantinides, D. Zhang, S. Sudhakar, V. Bertacco, and T. Austin. CrashTest: A fast high-

fidelity FPGA-based resiliency analysis framework. In *Int'l Conf. on Comp. Design*, pages 363–370, 2008.

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

[5] 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.

[6] 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.

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

[8] J. P. Hayer, I. Polian, and B. Becker. An analysis framework for transient-error tolerance. In *VLSI Test Symp.*, pages 249–255, 2007.

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

[10] G. Fey, A. Sülflow, S. Frehse, U. Kühne, and R. Drechsler. Formaler Nachweis der Fehlertoleranz von Schaltkreisen. In *GMM/GI/ITG-Fachtagung Zuverlässigkeit und Entwurf*, pages 75–82, 2008.

[11] G. Fey, A. Sülflow, and R. Drechsler. Computing bounds for fault tolerance using formal techniques. In *Design Automation Conf.*, pages 190–195, 2009.

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

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

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

[15] 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.).

[16] Accellera. *Property Specification Language – Reference Manual*. Accellera Organization Inc., 2004. Available at http://www.accellera.org/home.

[17] F. Rogin, T. Klotz, G. Fey, R. Drechsler, and S. Rülke. Automatic generation of complex properties for hardware designs. In *Design, Automation and Test in Europe*, pages 545–548, 2008.

[18] A. Sülflow, U. Kühne, G. Fey, D. Große, and R. Drechsler. WoLFram – a word level framework for formal verification. In *IEEE International Workshop on Rapid System Prototyping*, pages 11–17, 2009.