Automatische formale Verifikation der Fehlertoleranz von Schaltkreisen

Automated Formal Verification of Fault Tolerance for Circuits

Görschwin Fey, André Sülflow, Stefan Frehse, Rolf Drechsler, Universität Bremen

Zusammenfassung Die kontinuierliche Verringerung der Strukturgrößen führt zu einer erhöhten Fehleranfälligkeit von digitalen Schaltungen. Sowohl Produktionsfehler als auch transiente Fehler können die Funktionsfähigkeit beeinflussen. Deshalb wird die Untersuchung der Fehlertoleranz im Entwicklungsprozess zunehmend wichtig. In dieser Arbeit wird ein formales Verfahren zur Berechnung der Fehlertoleranz bzgl. kurzzeitig auftretenden Fehlern vorgestellt. Darüber hinaus werden Erweiterungen zur Erhöhung der Genauigkeit, zur Beschleunigung und für die Behandlung von Mehrfachfehlern vorgeschlagen.

►►► **Summary** Continuously shrinking feature sizes lead to an increasing vulnerability of digital circuits. Manufacturing failures, transient faults, or induced bit-flips may tamper the functionality. Thus, analyzing the fault tolerance of a given implementation becomes an important step in the design process. In this work we present a formal method to compute fault tolerance with respect to single soft errors. Extensions are proposed to increase the accuracy, to improve the run time, and to cope with multiple faults.

Schlagwörter Integrierte Schaltungen, Formale Verifikation, Fehlertoleranz ►►► **Keywords** B.7 [Hardware: Integrated Circuits]; B.8.1 [Hardware: Performance and Reliability: Reliability, Testing, and Fault-Tolerance]

1 Einleitung

Integrierte Schaltkreise werden zunehmend in sicherheitskritischen Anwendungen wie zum Beispiel dem „Steer-by-Wire“ in Kraftfahrzeugen oder der Steuerung wichtiger Funktionen in Flugzeugen eingesetzt. Die Korrektheit solcher Schaltkreise wird durch den massiven Einsatz sowohl simulationsbasierter als auch formaler Verifikationsmethoden sichergestellt.

Gleichzeitig nimmt – gemäß Moore's Law – die Anzahl der Bauelemente, die in einem Gesamtsystem integriert sind, stetig zu. Die physikalische Fläche, die ein einzelnes Bauelement einnimmt, wird währenddessen ständig geringer. Als Resultat werden selbst funktional korrekt beschriebene Schaltkreise immer anfälliger gegenüber Fehlern, die erst nach der Produktion – also im praktischen Einsatz – auftreten. Hierzu zählen zum Beispiel transiente Fehler, die durch elektromagnetische Umweltstrahlung auftreten können und sich als *Single Event Upsets* (SEUs) äußern können, oder statische Fehler, die durch Elektromigration im Rahmen von Alterungsprozessen hervorgerufen werden. Ein internes Fehlverhalten muss erkannt und signalisiert werden, während das Ein-/Ausgabeverhalten weiterhin wie im fehlerfreien Fall funktioniert. Ein einfaches Mittel hierfür ist zum Beispiel die redundante Auslegung funktionaler Einheiten. Schon während des Entwurfes wird deshalb durch eine entsprechende Schaltkreisarchitektur Sorge getragen, dass die Fehlfunktion einzelner Bauteile die funktionale Korrektheit des Schaltkreises nicht gefährdet.

Um nachzuweisen, dass ein Schaltkreis auch unter Fehlerannahmen noch die Spezifikation erfüllt, werden üblicherweise simulationsbasierte oder emulationsbasierte Methoden herangezogen [12]. Diese sind jedoch unzureichend in der Überdeckung aller möglichen Systemzustände. Es kann also Systemzustände geben, in denen interne Fehlfunktionen und resultierendes fehlerhaftes Verhalten unentdeckt bleiben. Die korrekte Funktionsweise der Schaltung unter Fehlerannahme ist nicht gewährleistet.

Dagegen kann die Anwendung formaler Methoden den Nachweis liefern, dass in *jedem Systemzustand* und unter *allen Eingaben* garantiert *jede* interne Fehlfunktion

erstens detektiert wird und zweitens kein Fehlverhalten verursacht. Erste Ansätze hierfür wurden in [2; 5; 6; 8–11; 13] vorgestellt. Diese Algorithmen benötigen jedoch oft eine exakte formale Erreichbarkeitsanalyse, manuelle Interaktion oder betrachten ausschließlich kombinatorische Schaltkreise.

In der vorliegenden Arbeit wird der formale Ansatz aus [5; 6] näher vorgestellt. Ähnlich zum *Bounded Model Checking* [1] modelliert der Ansatz für sequentielle Schaltkreise ein begrenztes Zeitfenster für eine vollständige Analyse. Die Analyse liefert eine Unterteilung der Komponenten in (a) robuste Komponenten, deren Fehlfunktion keine Auswirkung auf die Ausgabe hat, (b) nicht robuste Komponenten, deren Fehlfunktion die Ausgabe verfälschen kann, und (c) nicht klassifizierte Komponenten, deren Fehlfunktion den Systemzustand verändern kann. Erweiterungen des Ansatzes erlauben es, approximative Ansätze für die Analyse der erreichbaren Zustände einzusetzen [6], anwendungsbezogene Restriktionen zu berücksichtigen [14], sowie das Versagen mehrerer Komponenten zu analysieren [7]. Die Analyse kann sowohl auf den *sequentiellen Äquivalenzvergleich* (engl.: Sequential Equivalence Checking, SEC) oder auf sequentielle *automatische Testmustergenerierung* (engl.: Automatic Test Pattern Generation, ATPG) reduziert werden. Beide Algorithmen werden experimentell verglichen.

2 Grundlagen

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 einer Hardware-Beschreibungssprache als Komponenten betrachtet werden.

In einem derart beschriebenen Schaltkreis stellt die Wertebelegung der Zustandsbits S den aktuellen Zustand dar. Typischerweise sind nicht alle Wertekombinationen in einem Schaltkreis einstellbar, diese werden als nicht erreichbare Zustände bezeichnet. Als Beispiel sei eine One-Hot-Kodierung genannt, d. h. es darf immer nur ein Bit in einem Bitvektor den Wert 1 haben, während die anderen Bits den Wert 0 haben. Mittels einer formalen Analyse lässt sich die Menge der erreichbaren Zustände für einen Schaltkreis bestimmen. Diese Menge wird im Folgenden als S^* bezeichnet. In der Praxis ist die Analyse der erreichbaren Zustände sehr aufwändig sowohl hinsichtlich der Laufzeit als auch des Speicherbedarfs.

3 Formale Analyse

In diesem Abschnitt wird zunächst das verwendete Fehlermodell erläutert. Im Anschluss werden aus praktischen Überlegungen Einschränkungen abgeleitet, die dann im Algorithmus ausgenutzt werden, um eine Beschleunigung zu erreichen.

3.1 Fehlermodell

Im Folgenden wird Robustheit eines Schaltkreises wie folgt definiert: Ein Schaltkreis ist robust, sofern die Fehlfunktion einer Komponente das Ausgabeverhalten nicht verändert.

Die Prüfung der Robustheit 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 [5] wurde ein Ansatz vorgestellt, der die Überprüfung mehrerer Komponenten gleichzeitig ermöglicht. Als Fehlermodell wird nicht-deterministisches Verhalten einer Komponente zugelassen. Seien 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 Bild 1a 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. Ein Fehler wird an g injiziert.

3.2 Zeitliche Beschränkung der Analyse

Im praktischen Einsatz müssen nach dem Auftreten eines internen Fehlers Maßnahmen eingeleitet werden. 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

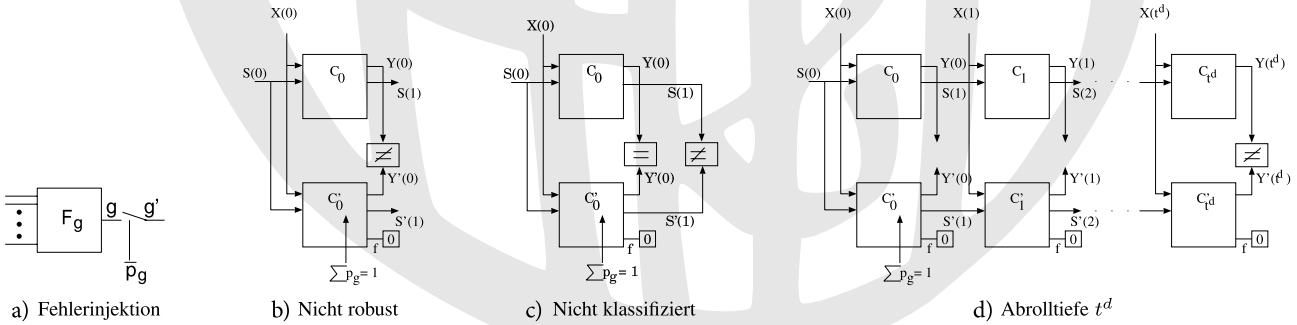


Bild 1 Modell zur Klassifizierung von Komponenten.

die betrachteten Schaltkreise mit einer Fehlererkennungslogik ausgestattet sind. Die Erkennung eines Fehlers wird durch Setzen eines Signals f angezeigt. Insbesondere gilt, dass f 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, d. h. der Menge S^* gestartet wird.

3.3 Klassifikation von Komponenten

Es 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) $f = 1$ innerhalb von t^d Takten folgt bevor eine Abweichung der Ausgabewerte auftritt oder
   (b) $f = 0$ 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 $f = 1$.
3. Komponente g ist *nicht abschließend klassifiziert*, falls $f = 0$ 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, d. h. es liegt *Silent Data Corruption* (SDC) vor.

Für die formale Überprüfung der Robustheit wird folgende Konstruktion verwendet: Alle Komponenten aus $\mathbb{C}$ werden mit Fehlerprädikaten versehen. Das so erzeugte Modell des fehlerhaften Schaltkreises wird, wie in Bild 1b dargestellt, einem Äquivalenzvergleich mit dem ursprünglichen Schaltkreis unterzogen. Dabei darf für Einfachfehler höchstens ein Fehlerprädikat den Wert 1 annehmen. Das Prädikat $S(0)$ beschreibt die Menge der erreichbaren Zustände S^* für den Schaltkreis, d. h. die Analyse betrachtet alle möglichen erreichbaren Zustände aber keine unerreichbaren Zustände. Ein Gegenbeispiel zur Äquivalenz dieser beiden Schaltkreise liefert nun einen Fehler und eine Eingabesequenz, die die fehlerhafte Ausgabe erzeugt. Der Fehler ist durch die Komponente gegeben, deren Fehlerprädikat den Wert 1 hat. Auf diese Weise werden alle Komponenten aufgezählt, die als nicht robust klassifiziert werden und in der Menge $\mathbb{S}$ zusammengefasst.

Die Menge der nicht klassifizierten Komponenten $\mathbb{U}$ wird dann mittels des Modells in Bild 1c bestimmt. Anstelle der primären Ausgänge werden hier die internen Zustände nach der Fehlerinjektion verglichen. Die restlichen Komponenten müssen robust sein und werden in der Menge $\mathbb{T}$ zusammengefasst: $\mathbb{T} = \mathbb{C} \setminus (\mathbb{S} \cup \mathbb{U})$.

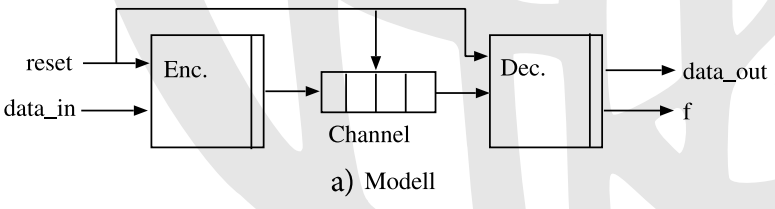
Nach der Betrachtung für einen Takt wird die Anzahl der betrachteten Takte iterativ erhöht, sodass nach und nach mehr Komponenten als nicht robust bzw. als robust klassifiziert werden können, da interne Fehler an primären Ausgängen sichtbar werden. Wie im *Bounded Model Checking* wird der Schaltkreis „abgerollt“ um das zeitliche Verhalten über mehrere Takte zu analysieren, dies ist in Bild 1d dargestellt. Im Prinzip wird dabei ein sequentieller Äquivalenzvergleich des originalen Schaltkreises mit dem Schaltkreis, in den Fehler injiziert werden, durchgeführt. Da im ersten Takt ein beliebiger erreichbarer Zustand berücksichtigt wird, muss nur im ersten Takt ein Fehler in das Modell injiziert werden. Die Mengen $\mathbb{S}$, $\mathbb{U}$ und $\mathbb{T}$ werden mit jedem Abrollschritt aktualisiert.

Beispiel 1. *Ein (7,4)-Hamming-Kode kann Einfachfehler sowohl erkennen als auch korrigieren. In Bild 2a ist ein Modell dargestellt, das aus einem Kodierer (Enc), einem Übertragungskanal (Channel) und einem Dekodierer (Dec) besteht. Die Daten (data_in) werden im Kodierer kodiert, über einen bit-seriellen vierstufigen Übertragungskanal zum Dekodierer übertragen und dort dekodiert. Das Fehlererkennungssignal f 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 äußern. Die Fehlerer-



a) Modell

t^d	$\lvert\mathbb{T}\rvert$	$\lvert\mathbb{S}\rvert$	$\lvert\mathbb{U}\rvert$	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

b) Robustheit

Bild 2 Fehlertoleranter Kanal mit Hamming-Kode.

kennungslogik 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. Da es sich hierbei um kombinatorische Logik handelt, die primäre Ausgänge treibt, werden die entsprechenden Komponenten schon bei Betrachtung eines Zeitschrittes klassifiziert.
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. Wenn zwei Zeittakte betrachtet werden, wurde das letzte Datenwort aus dem Kanal dekodiert, sodass dort injizierte Fehler erkannt bzw. an Ausgängen sichtbar werden. Entsprechend können immer mehr Komponenten klassifiziert werden. Mit weiteren Abrollschritten verringert sich die Menge der nicht klassifizierten Komponenten weiter.
Mit jeder Inkrementierung von t^d werden mehr Komponenten des Kanals klassifiziert. Teile des Kodierers verbleiben nicht abschließend klassifiziert.

- *$t^d = 6$: 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.

Aus den Mengen $\mathbb{S}$, $\mathbb{U}$ und $\mathbb{T}$ lassen sich eine untere Schranke R_{lb} und eine obere Schranke R_{ub} für die Robustheit des Schaltkreises $\mathbb{C}$ bestimmen:

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

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

Der oben skizzierte Algorithmus betrachtet immer ein Zeitfenster. Je größer dieses Zeitfenster ist, umso genauer können Komponenten klassifiziert werden. Der Algorithmus liefert also die Mengen $\mathbb{S}$, $\mathbb{U}$ und $\mathbb{T}$ in Abhängigkeit der aktuellen Abrolltiefe t^d. Ebenso ist die Menge S^* der für $S(0)$ zugelassenen Zustände für das Ergebnis wichtig. Dadurch werden auch die Schranken in Abhängigkeit von der Abrolltiefe t^d und der Menge S^* als $R_{lb}(t^d, S^*)$ bzw. $R_{ub}(t^d, S^*)$ geliefert.

Beispiel 2. *Für das oben betrachtete Hamming-Kode-Modell ergeben sich die Werte aus Bild 2b. Mit zunehmender Abrolltiefe werden mehr Komponenten klassifiziert, die Schranken nähern sich an. Mit $t^d \geq 6$ werden alle Komponenten klassifiziert.*

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

4 Erweiterungen

Das Basismodell zur Klassifikation kann auf verschiedene Arten erweitert werden, um einerseits die Effizienz der Klassifikation zu steigern oder andererseits Anforderungen, die aus praktischer Sicht sinnvoll sind, zu berücksichtigen.

4.1 Erreichbarkeit

Bisher wurde immer davon ausgegangen, dass die Menge der erreichbaren Zustände S^* für die Klassifikation bekannt ist und dass $S(0)$ ein Prädikat ist, welches diese Menge beschreibt. Allerdings ist diese Berechnung aufwändig. Oft werden *Binäre Entscheidungsdiagramme* (engl.: Binary Decision Diagrams, BDDs) [3] eingesetzt, um in einer Fixpunktiteration vom Startzustand ausgehend alle erreichbaren Zustände zu ermitteln. Diese Verfahren sind jedoch sehr zeit- und speicherintensiv.

Stattdessen wurden in [6] unterschiedliche Ansätze vorgeschlagen, um die Menge S^* zu approximieren. Hier ist sowohl eine Unterapproximation $S^{\downarrow} \subseteq S^*$ als auch eine Überapproximation $S^{\uparrow} \supseteq S^*$ möglich. Die Berechnung von Approximationen kann im Vergleich zu BDDs deutlich effizienter durchgeführt werden.

Hinsichtlich der Berechnung der Robustheit ergibt sich durch die Approximation der erreichbaren Zustände eine Abschätzung der Schranken [6]. Bei Verwendung der Überapproximation $S^{\uparrow}$ werden nicht erreichbare Zustände wie erreichbare behandelt. Dadurch können Komponenten, die bezüglich S^* robust sind, als nicht robust klassifiziert werden. Ein Beispiel dafür ist ein Hamming-kodierter Speicher. In allen zulässigen Kodeworten können Ein-Bit-Fehler korrigiert werden. Zwei-Bit-Fehler können zwar erkannt, nicht aber korrigiert werden. Werden nun durch die Verwendung der Überapproximation $S^{\uparrow}$ nicht nur Kodewörter zugelassen, kann ein weiterer Fehler evtl. weder erkannt noch korrigiert werden und eine Komponente wird als nicht robust klassifiziert. Also ergibt sich eine Abschätzung der unteren Schranke der Robustheit: $R_{lb}(t, S^*) \geq R_{lb}(t, S^{\uparrow})$.

Analog lässt sich mit einer Unterapproximation $S^{\downarrow}$ der erreichbaren Zustände die obere Schranke der Robustheit abschätzen: $R_{ub}(t, S^*) \leq R_{ub}(t, S^{\downarrow})$. Durch die Unterapproximation $S^{\downarrow}$ wird bestimmte Funktionalität von der Analyse ausgeschlossen. So werden bezüglich S^* nicht robuste Komponenten als robust klassifiziert, wenn $S^{\downarrow}$ verwendet wird.

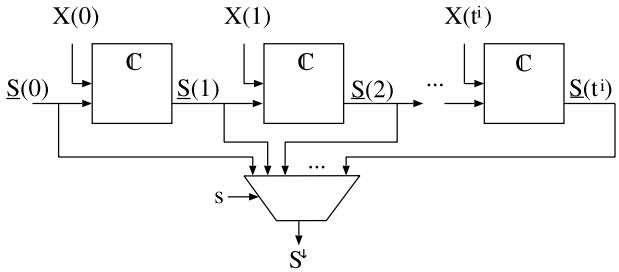


Bild 3 Unterapproximation $S^{\downarrow}$

Für die Experimente wurden einfache Approximationsmethoden eingesetzt, die sich direkt in das oben vorgestellte Modell integrieren lassen. Um eine Unterapproximation $S^{\downarrow}$ zu berechnen, wurde der Schaltkreis ausgehend vom Startzustand für t^i Takte abgerollt. Die zum Zeitpunkt t^i erreichbaren Zustände bilden dann die Einschränkung für $S(0)$. Dies ist in Bild 3 dargestellt. Die Zustandsmengen sind in dieser Abbildung mit $\underline{S}$ gekennzeichnet, um eine Verwechslung mit den Mengen aus Abschnitt 3.3 zu vermeiden.

Zur Überapproximation $S^{\uparrow}$ kann der gleiche Ansatz verwendet werden, indem $\underline{S}(0)$ nicht beschränkt wird. In den Experimenten wurden für $S^{\uparrow}$ alle Zustände erlaubt.

4.2 Anwendungsbezogene Analyse

Wird ein Schaltkreis nur innerhalb einer bestimmten Anwendung eingesetzt, ist es sinnvoll, die Analyse daraufhin anzupassen. In [14] wurden Ansätze vorgeschlagen, um die aus der Anwendung resultierenden Restriktionen zu extrahieren und bei der Analyse zu berücksichtigen. Im Prinzip liegt diesen Untersuchungen zugrunde, dass die Anwendung nur eine beschränkte Auswahl möglicher Eingabestimuli für den Schaltkreis erzeugt, diese Einschränkungen sind zunächst zu modellieren. Im Anschluss kann dann eine formale Erreichbarkeitsanalyse genutzt werden, um die resultierenden Beschränkungen für erreichbare Zustände zu ermitteln.

Alternativ kann eine Testbench eingesetzt werden, um bekannte erreichbare Zustände zu ermitteln, so wird allerdings wieder nur eine Unterapproximation der erreichbaren Zustände erreicht. Für die in [14] betrachteten Beispiele ergab sich ein um bis zu 40% höherer Wert für die Robustheit, wenn die Anwendung berücksichtigt wird.

4.3 Mehrfachfehler

Aufgrund weiter abnehmender Strukturgrößen wird die Wahrscheinlichkeit, dass Mehrfachfehler auftreten, immer größer. Hier gibt es drei Sichtweisen.

Zunächst wird durch die Verwendung von Komponenten bereits die Möglichkeit von Mehrfachfehlern berücksichtigt. Eine Komponente kann auch einen komplexeren Teilschaltkreis etwa einen Addierer repräsentieren. Ein Fehler dieser Komponente deckt dann mehrere Fehler auf Gatter- oder Transistor-Ebene ab.

Im nächsten Szenario induziert ein Alpha-Partikel transiente Fehler in mehreren benachbarten Leitungen. Dies kann im vorliegenden Modell eingebettet werden, indem bei der Vergabe von Fehlerprädikaten benachbarte Komponenten entsprechend der Layout-Information gruppiert werden.

Schließlich können mehrere zeitlich und örtlich unabhängige transiente Fehler auftreten. Der entstehende große Suchraum kann beispielsweise wie in [7] verkleinert werden.

4.4 ATPG-Ansatz

Im Prinzip werden durch den bisher vorgestellten Ansatz Testmuster für $|\mathbb{C}|$ verschiedene Fehler an Komponenten generiert. Der Vorteil ein einziges Modell zu benutzen, welches zur Klassifikation aller Komponenten verwendet wird, besteht in der Wiederverwendung gelernter Information. Um möglichst effizient eine Lösung für das gegebene Problem zu finden, lernen die verwendeten Beweiser für *Boolesche Erfüllbarkeit* (engl.: Boolean Satisfiability, SAT) sogenannte Konfliktklauseln, während der Suchraum traversiert wird. Werden aufeinanderfolgende Probleme gelöst, die gleiche Teilstrukturen haben, kann gelernte Information wiederverwendet werden [16]. Bei dem bisherigen Ansatz passiert dies implizit. Zum einen werden gelernte Informationen bei der Betrachtung verschiedener Fehler wiederverwendet. Zum anderen werden bei der Erhöhung der Abrolltiefe gelernte Informationen beibehalten. Als Nachteil muss dafür immer der gesamte Schaltkreis in der Probleminstanz für jeden Takt zweimal repliziert werden.

Eine Alternative ist der Einsatz eines sequentiellen Testmustergenerators. In diesem Fall wird für jeden Fehler eine Probleminstanz erzeugt. Dabei kann die Größe der Probleminstanz verkleinert werden, indem nur die Teile des Schaltkreises einbezogen werden, die für den betrachteten Fehler relevant sind. Dies sind die Komponenten, die entweder (a) im transitiven Fanout der Fehlerstelle liegen, d. h. von der Fehlerstelle beeinflusst werden können, oder (b) die im transitiven Fanin eines primären Ausgangs liegen, an dem der Fehler potentiell beobachtet werden kann, d. h. die für die Einstellung oder Weiterleitung des Fehlers relevant sind. Wird auch hier ein SAT-basierter Testmustergenerator verwendet, wird gelernte Information bei Erhöhung der Abrolltiefe bei entsprechender Verwendung des Beweisers automatisch wiederverwendet.

5 Experimentelle Ergebnisse

In den Experimenten wird untersucht, wie sich die Klassifizierung mit zunehmender Größe des Betrachtungsfensters t^d verhält, wie sich die Verwendung approximativer Erreichbarkeitsanalyse auswirkt und wie performant die unterschiedlichen Algorithmen (ATPG bzw. SEC) sind. Dazu wurde der vorgestellte Ansatz in der Umgebung WoLFram (Word Level Framework) [15]

umgesetzt. Als SAT-Beweiser wurde Minisat [4] mit einer Erweiterung für inkrementelles Lernen eingesetzt. Die folgenden Experimente wurden auf einem Intel Xeon (3 GHz, 32 GB RAM, Linux) ausgeführt.

Für die Untersuchung wurden die Schaltkreise aus dem ITC'99-Benchmark in folgende Varianten, die durch ein Suffix am Namen unterschieden werden, modifiziert und untersucht:

- Originalschaltkreis (Suffix *-orig*): Diese Schaltkreise sind kaum robust.
- Paritätsprüfung auf primären Ausgängen und Zustandsbits (Suffix *-parity*): Fehler, die eine gerade Anzahl von Ausgängen und Zustandsbits verändern, werden nicht erkannt.
- *Triple Modular Redundancy* (TMR) (Suffix *-tmr*): Die Schaltkreise werden dreifach redundant ausgelegt. Fehlereffekte von Einfachfehlern werden maskiert. Es kann unerkannte SDC auftreten.
- TMR mit Fehlererkennung (Suffix *-tmrflt*): Es wird ein Fehlersignal gesetzt, sobald bei einem der redundanten Module der Zustand oder der Ausgabewert abweichen.

Zusätzlich wurde ein kombinatorischer Multiplizierer untersucht, dessen Fehlersignal gesetzt wird, falls eine Modulo-3-Prüfung fehlschlägt:

$$f = [(a * b) \,\%\, 3 \neq (a \,\%\, 3) * (b \,\%\, 3)]$$

Für die Bitvektoren a und b wurden Bitbreiten von 1 bis 10 Bit betrachtet.

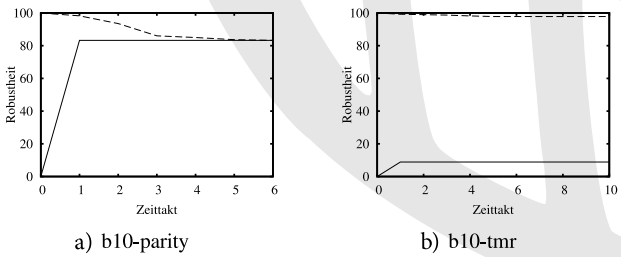
5.1 Schranken-Entwicklung

Bild 4 zeigt die Änderung der Schranken für die Robustheit mit zunehmenden Werten für t^d am Beispiel des Schaltkreises *b10*. Dargestellt sind jeweils die obere Schranke $R_{ub}(t^d, S^*)$ und die untere Schranke $R_{lb}(t^d, S^*)$.

Die Paritätsprüfung bei *b10-parity* führt entweder zu einer schnellen Entdeckung eines Fehlers oder der Fehler wird schnell an den Ausgängen sichtbar, wird aber durch die Paritätsprüfung nicht erkannt, da eine gerade Anzahl von Ausgangssignalen und Zustandsbits kippt. Wie Bild 4a zeigt, ist die Klassifikation bei $t^d = 6$ vollständig abgeschlossen.

Bei der TMR-Variante ohne Fehlersignal, zeigen sich zwar keine Fehler an den Ausgängen, intern kommt es aber zu nicht erkannter SDC. Deshalb bleibt eine große Anzahl von Komponenten nicht abschließend klassifiziert. Wie Bild 4b zeigt, nähern sich die Schranken auch bei Erhöhung von t^d nur geringfügig an.

Die Experimente zeigen, dass bereits die Zwischenergebnisse für geringere Werte von t^d als dem durch den Nutzer gewählten gute Abschätzungen für die Robustheit liefern. Der Nutzer erhält vom Algorithmus eine direkte Rückmeldung über die Konvergenz der Klassifikation.



a) b10-parity b) b10-tmr

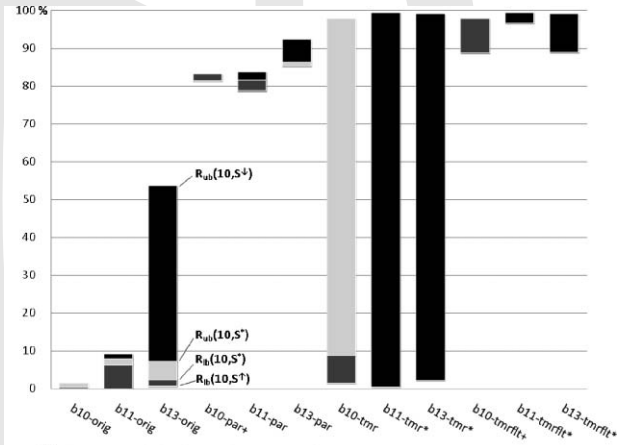
Bild 4 Schranken vs. t^d (- - $R_{ub}(t^d, S^*)$, — $R_{lb}(t^d, S^*)$).

5.2 Approximative Erreichbarkeit

Bild 5 zeigt die Ergebnisse für die exakte und die approximative Erreichbarkeitsanalyse mit $t^d = 10$. Dabei wurden für die Überapproximation $S^{\uparrow}$ alle Zustände erlaubt, für die Unterapproximation $S^{\downarrow}$ wurden der Ansatz aus Abschnitt 4.1 mit $t^i = 10$ verwendet. Für einige Schaltkreise konnte die BDD-basierte exakte Erreichbarkeitsanalyse nicht innerhalb einer Laufzeitbegrenzung von 54.000 Sekunden durchgeführt werden. In diesen mit $*$ gekennzeichneten Fällen, liegen also keine Werte für $R_{lb}(S^*)$ und $R_{ub}(S^*)$ vor. Für *b13-orig* ist im Diagramm annotiert, wie die Balken zu interpretieren sind.

Hellgraue Balken deuten an, dass auch bei bekanntem S^* in hohem Maße SDC vorkommt – die untere Schranke $R_{lb}(10, S^*)$ und die obere Schranke $R_{ub}(10, S^*)$ liegen weit auseinander. Konnte S^* berechnet werden, lässt sich ablesen, wie die Approximation der erreichbaren Zustände die Genauigkeit der Schranken beeinflusst. Der schwarze Balken zeigt an, wie die Approximation $S^{\downarrow}$ die obere Schranke erhöht. Der dunkelgraue Balken zeigt die Approximation der unteren Schranke durch Verwendung von $S^{\uparrow}$. Konnte $S*$ nicht berechnet werden, zeigt der schwarze Balken den Bereich zwischen $R_{lb}(10, S^{\uparrow})$ und $R_{ub}(10, S^{\downarrow})$.

Bezüglich der Schaltkreise gilt, dass die Originalversionen (-orig) wenig robust sind, während die Erweiterung mit Paritätsprüfung (-par) bzw. TMR (-tmrflt) eine sehr hohe Robustheit garantiert. Wird TMR eingesetzt aber keine Fehlerdetektion vorgenommen (-tmr), kommt es in hohem Maße zu SDC.



*= S^* konnte nicht berechnet werden
+= bzgl. S^* wurden alle Komponenten klassifiziert ($\mathbb{U} = \emptyset$)

Bild 5 Robustheit.

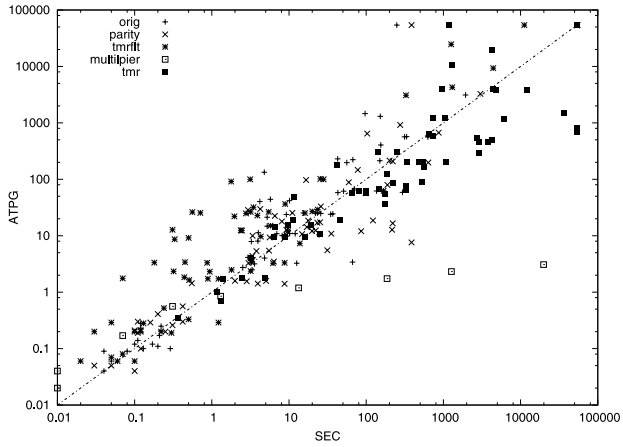


Bild 6 Laufzeiten.

Insgesamt zeigt sich, dass vor allem für größere Schaltkreise der Einsatz approximativer Erreichbarkeitsanalyse essentiell ist, um überhaupt Aussagen über die Fehlertoleranz liefern zu können.

5.3 Laufzeiten

Abschließend werden die Laufzeiten des ATPG-basierten und des SEC-basierten Algorithmus in Bild 6 auf einer logarithmischen Skala verglichen. Die Laufzeit wurde auf 54.000 Sekunden CPU-Zeit beschränkt.

Keiner der beiden Algorithmen ist immer besser. Schon bei geringen Abrolltiefen muss auch im ATPG-basierten Ansatz oft der ganze Schaltkreis in den ersten Takten instanziiert werden. Bei Schaltkreisen mit TMR aber ohne Fehlererkennung (*tmr*) ist der ATPG-basierte Ansatz schneller. Mit zusätzlicher Fehlererkennung (*tmr-flt*) ist der SEC-basierte Ansatz schneller. Der Beweiser profitiert von den über alle Aufrufe gelernten strukturellen Informationen.

Beim skalierbaren kombinatorischen Multiplizierer ist eindeutig der ATPG-Ansatz im Vorteil. Da für den kombinatorischen Schaltkreis kein Abrollen nötig ist, tritt immer eine deutliche Reduktion der Problemgröße auf, wenn alle Fehler unabhängig betrachtet werden.

Im Rahmen zukünftiger Arbeiten werden Abstraktionsansätze sowie semi-formale Methoden, die aus dem Bereich der Verifikation bekannt sind, untersucht, um eine Behandlung größerer Schaltungen zu ermöglichen.

6 Zusammenfassung

Mittels formaler Methoden lässt sich die Fehlertoleranz von Schaltkreisen prüfen. Als Ergebnis steht einerseits ein Maß für die Robustheit zur Verfügung, andererseits werden auch nicht-robuste Stellen im Schaltkreis identifiziert. Abhängig von der Art des Schaltkreises liefert das Verfahren entweder sehr genaue Aussagen über die Robustheit oder zeigt, an welchen Stellen Silent Data Corruption auftreten kann.

Danksagung

Diese Arbeit wurde zum Teil durch die DFG unter den Kennzeichen DR 287/19-1 und FE 797/5-1 gefördert.

Literatur

[1] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In: Tools and Algorithms for the Construction and Analysis of Systems, vol. 1579 of LNCS, pp. 193–207, Springer Verlag, 1999.

[2] M. Bozzano, A. Cimatti, and F. Tapparo. Symbolic fault tree analysis for reactive systems. In: Automated Technology for Verification and Analysis, vol. 4762 of LNCS, pp. 162–176, 2007.

[3] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. In: IEEE Trans. on Computers, 35(8):677–691, 1986.

[4] N. Eén and N. Sörensson. An extensible SAT solver. In: SAT 2003, vol. 2919 of LNCS, pp. 502–518, 2004.

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

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

[7] S. Frehse, G. Fey, A. Sülflow, and R. Drechsler. Robustness check for multiple faults using formal techniques. In: EUROMICRO Symp. on Digital System Design, pp. 85–90, 2009.

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

[9] M. Hunger, S. Hellebrand, A. Czutro, I. Polian, and B. Becker. ATPG-Based grading of strong fault-secureness. In: IEEE Int'l On-Line Testing Symp., pp. 269–274, 2009.

[10] 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, pp. 176–181, 2006.

[11] R. Leveugle. A new approach for early dependability evaluation based on formal property checking and controlled mutations. In: IEEE Int'l On-Line Testing Symp., pp. 260–265, 2005.

[12] 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 Computer Design, pp. 363–370, 2008.

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

[14] A. Sülflow, S. Frehse, G. Fey, and R. Drechsler. Anwendungsbezogene Analyse der Robustheit von digitalen Schaltungen. In: GMM/GI/ITG-Fachtagung Zuverlässigkeit und Entwurf, S. 45–52, 2009.

[15] A. Sülflow, U. Kühne, G. Fey, D. Große, and R. Drechsler. WoLFram – a word level framework for formal verification. In: IEEE Int'l Symp. on Rapid System Prototyping, pp. 11–17, 2009.

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

Manuskripteingang: 15. März 2010

Dr. Görschwin Fey erhielt sein Diplom in Informatik 2001 von der Martin-Luther-Universität Halle-Wittenberg. Seit 2002 ist er als wissenschaftlicher Mitarbeiter in der AG Rechnerarchitektur an der Universität Bremen tätig. Seine Promotion hat Herr Fey im Jahr 2006 unter der Betreuung von Prof. Dr. Rolf Drechsler abgeschlossen. Von November 2007 bis Juni 2008 hatte Herr Fey eine Gastprofessur an der Universität Tokyo, Japan inne. Seine Forschungsinteressen liegen in den Bereichen Verifikation und Test von digitalen Schaltkreisen.

Adresse: Universität Bremen, Bibliothekstr. 1, 28359 Bremen, Tel.: +49-421-218-63944, Fax: +49-421-218-98-63944,
E-Mail: fey@informatik.uni-bremen.de

Dr. André Sülflow studierte seit 2001 Informatik an der Universität Bremen und schloss sein Studium im Jahr 2006 erfolgreich ab. Aufbauend auf das Studium begann Herr Sülflow seine Promotion in der Arbeitsgruppe Rechnerarchitektur an der Universität Bremen unter Betreuung von Prof. Dr. Rolf Drechsler. Seit dem erfolgreichen Abschluss seiner Promotion im Jahr 2010 ist Herr Sülflow als wissenschaftlicher Mitarbeiter an der Universität Bremen tätig. Die Schwerpunkte seiner Forschung liegen in den Bereichen automatisches Debugging, formaler Nachweis der Fehlertoleranz und der formalen Verifikation von digitalen Schaltkreisen.

Adresse: Universität Bremen, Bibliothekstr. 1, 28359 Bremen, Tel.: +49-421-218-63945, Fax: +49-421-218-98-63945,
E-Mail: suelflow@informatik.uni-bremen.de

Stefan Frehse erhielt sein Diplom für Informatik im Jahr 2008 unter der Betreuung von Dr. Görschwin Fey an der Universität Bremen. Seit 2009 arbeitet er als wissenschaftlicher Mitarbeiter in der AG Rechnerarchitektur an der Universität Bremen. Seine Forschungsinteressen liegen im Bereich der formalen Verifikation fehlertoleranter Schaltkreise.

Adresse: Universität Bremen, Bibliothekstr. 1, 28359 Bremen, Tel.: +49-421-218-63953, Fax: +49-421-218-98-63953,
E-Mail: sfrehse@informatik.uni-bremen.de

Prof. Dr. Rolf Drechsler hat an der Johann-Wolfgang-Goethe Universität in Frankfurt am Main Mathematik und Informatik studiert. Im Jahr 1995 schloss er sein Studium mit der Promotion ab. Von 1995 bis 2000 war er als Wissenschaftlicher Assistent an der Albert-Ludwigs-Universität in Freiburg beschäftigt, wo er 1999 habilitierte. Im Anschluss arbeitete er als Senior Engineer in der Abteilung Corporate Technologies der Siemens AG, München, bis er im Oktober 2001 die Professur für Rechnerarchitektur am Fachbereich 3, Mathematik und Informatik, der Universität Bremen annahm. Seine wissenschaftlichen Interessen umfassen den Schaltkreis- und Systementwurf mit dem Schwerpunkt Formale Verifikation.

Adresse: Universität Bremen, Bibliothekstr. 1, 28359 Bremen, Tel.: +49-421-218-63932, Fax: +49-421-218-7385,
E-Mail: drechsle@informatik.uni-bremen.de

Vorschau auf Heft 5/2010

Unsere nächste Ausgabe ist ein Schwerpunktheft zum Thema „The Virtual Patient“ (Gastherausgeber: O. Dössel) und enthält unter anderem folgende Beiträge:

- *Seemann, G. et al.:* Electrophysiological Modeling for Cardiology: Methods and Potential Applications
- *Haag, M. und Huwendiek, S.:* The Virtual Patient for Education and Training: A Critical Review of the Literature
- *Schenkel, T. et al.:* Hemodynamics and Fluid-Structure-Interaction in a Virtual Heart
- *Weiser, M. et al.:* Craniofacial Surgery Planning based on Virtual Patient Models
- *Preusser, T. und Peitgen, H.-O.:* Patient-Specific Planning for Radio-Frequency Ablation of Tumors in the Presence of Uncertainty

Weitere Informationen über geplante Hefte, ausführliche Informationen über die in den letzten Heften der **it** erschienenen Beiträge sowie Hinweise für Autoren finden Sie im Internet unter http://www.it-information-technology.de.