Verbesserung der Robustheit und der Benutzbarkeit von Schaltkreisentwurfswerkzeugen unter Verwendung formaler Techniken

Görschwin Fey

Betreuer: Prof. Dr. Rolf Drechsler
Arbeitsgruppe Rechnerarchitektur
Fachbereich 3 – Mathematik und Informatik
Universität Bremen, 28359 Bremen
fey@informatik.uni-bremen.de

Abstract: Die Anzahl der Bauelemente, aus denen ein integrierter Schaltkreis besteht, nimmt stetig zu. Gleichzeitig resultieren aus dem Einsatz in sicherheitskritischen Anwendungen hohe Qualitätsanforderungen. Vor diesem Hintergrund müssen der etablierte Entwurfsablauf für Schaltkreise und die darin verwendeten Software-Werkzeuge ständig verbessert werden, um mit den Anforderungen Schritt zu halten.

In diesem Artikel wird der etablierte Entwurfsablauf erläutert und es werden Verbesserungsansätze vorgestellt, die die Robustheit und Benutzbarkeit der eingesetzten Werkzeuge verbessern. Die Integration dieser Ansätze führt zu einem erweiterten Entwurfsablauf. Am Beispiel der Fehlersuche in Schaltkreisen – dem Debugging – wird einer dieser Ansätze mit den resultierenden Vorteilen ausführlicher vorgestellt.

1 Einführung

Die Anwendungsgebiete *integrierter Schaltkreise* (engl. integrated circuit, IC) werden immer zahlreicher. Während ICs bis vor Kurzem lediglich in Rechnern eingesetzt wurden, sind sie heute in fast jedem Bereich des täglichen Lebens zu finden – meist verdeckt in sogenannten eingebetteten Systemen. So werden zum Beispiel Mobiltelefone, Stereoanlagen oder Mikrowellen von Schaltkreisen gesteuert. Insbesondere obliegt ihnen oft auch die Steuerung sicherheitskritischer Abläufe. Im Auto betrifft dies Systeme wie die Airbag-Steuerung, sogar Lenkung und Bremsen werden durch „Stear-by-wire" bzw. „Break-by-wire" mechanisch entkoppelt. Die korrekte Funktion der ICs ist in solchen Fällen von höchster Bedeutung – ein Versagen kann zu lebensbedrohlichen Situationen führen. An den Schaltkreisentwurf resultiert als Anforderung höchste Verlässlichkeit, um diese Korrektheit zu garantieren.

Zugleich nimmt die Komplexität eines einzelnen Schaltkreises stetig zu. Gemäß Moore's Gesetz [Moo65] verdoppelt sich die Anzahl der Elemente in einem IC etwa alle 18 Monate. Zwar ist dieses „Gesetz" eigentlich eine ökonomisch motivierte Vorhersage des Intel-Gründers Gordon Moore, trotzdem gelang es der Halbleiterindustrie in der Vergan-

genheit dieses Wachstum zu realisieren und oft sogar zu übertreffen. Mindestens für die kommenden 10 Jahre wird dieser Trend anhalten. Für den Entwurf von ICs bedeutet dies ein anhaltendes exponentielles Wachstum der Größe der Probleme, die zu betrachten sind.

Vor diesem Hintergrund – höchste Qualitätsanforderungen und exponentielles Wachstum – findet der Entwurf von ICs statt. Abbildung 1 stellt die wichtigsten Schritte des heute etablierten Entwurfsablaufs im Überblick dar. Ausgehend von einer textuellen *Spezifikation* wird ein *Systemmodell* entwickelt. Der *Syntheseschritt* zur Erzeugung des logischen Schaltkreismodells folgt. Durch die *Verifikation* werden die verschiedenen Schaltkreismodelle gegeneinander abgeglichen und auf Korrektheit gegenüber der Spezifikation geprüft. Schließlich wird die *Automatische Testmustergenerierung* (engl. Automatic Test Pattern Generation, *ATPG*[1]) durchgeführt. Mit diesen Testmustern werden die ICs nach der Fertigung auf Produktionsfehler überprüft. Zur Unterstützung der einzelnen Schritte existieren zahlreiche Software-Werkzeuge. Trotzdem hält der Produktivitätszuwachs im Entwurf nicht mit den Neuerungen in der Fertigungstechnik schritt. Es entsteht das sogenannte „Design and Verification Gap", d.h. die große Anzahl von Bauelementen die technisch für einen IC zur Verfügung stehen, sind im Entwurf und der Verifikation kaum noch handhabbar.

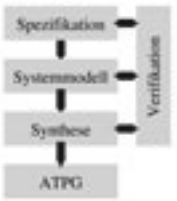

Abbildung 1: Entwurfsschritte

In der hier vorgestellten Dissertation [Fey06] werden zwei Ansatzpunkte untersucht, um die Produktivität des Entwurfsablaufs zu erhöhen:

- *Robustheit* – Dieser Begriff umfasst hier zwei Aspekte. Methodisch muss der Entwurfsablauf robust sein, um die Anfälligkeit gegen Fehler zu reduzieren. Auf algorithmischer Ebene müssen die verwendeten Werkzeuge auch auf wachsende Problemgrößen zuverlässig anwendbar sein.
- *Benutzbarkeit* – Gleichzeitig muss die Benutzbarkeit der Werkzeuge ständig verbessert werden. Komplexe Probleme verlangen komplexe Werkzeuge, deren Benutzbarkeit trotz einer Vielzahl von Parametern gewährleistet werden muss.

Es werden Ansätze entwickelt, um den etablierten Entwurfsablauf zu verbessern. Durch den Einsatz *formaler Techniken* in Form Boolescher Beweismethoden und -algorithmen werden Teilschritte automatisiert und existierende Werkzeuge verbessert. Eine wichtige Randbedingung ist, dass die neuen Ansätze in bestehende Abläufe integrierbar sind. Die einzelnen Ansätze stellen ein wichtiges Ergebnis der Dissertation dar. Das Zweite ist der *erweiterte Entwurfsablauf* der aus der Integration dieser Ansätze resultiert.

Dieser Artikel ist wie folgt gegliedert: Im folgenden Abschnitt wird der heute übliche Entwurfsablauf für ICs detaillierter eingeführt und es werden Schwachstellen identifiziert. Im Anschluss werden die neu entwickelten Ansätze zur Verbesserung und der resultierende erweiterte Entwurfsablauf in Abschnitt 3 kurz vorgestellt. Beispielhaft wird dann in Abschnitt 4 einer dieser Ansätze vertiefend erläutert. Die Fehlerlokalisierung für die Eigenschaftsprüfung wird betrachtet. Schließlich werden Zusammenfassung und Ausblick in Abschnitt 5 gegeben.

[1] Zum Teil werden die Bezeichnungen aus der englischen Fachliteratur übernommen.

2 Etablierter Entwurfsablauf

In diesem Abschnitt wird der Entwurfsablauf für integrierte Schaltkreise erläutert wie er heute in der industriellen Praxis eingesetzt wird. Dabei werden verschiedene Schwachstellen identifiziert. Abbildung 2 (auf der folgenden Seite) stellt den Gesamtablauf idealisiert dar. Die linke Seite der Abbildung zeigt die Entwurfsschritte, die rechte Seite der Abbildung gibt das Vorgehen bei der Verifikation wieder.

Ausgegangen wird von einer natürlich-sprachlichen textuellen Spezifikation des ICs (A). In zwei unabhängigen manuellen Kodierungsschritten werden aus dieser Spezifikation eine Beschreibung auf Systemebene (B) und eine synthetisierbare Beschreibung (C) erzeugt. Die Systemebene wird typischer Weise mittels Software-Programmiersprachen wie C/C++ beschrieben. Für die synthetisierbare Beschreibung werden spezielle *Hardware-Beschreibungssprachen* (engl. Hardware Description Language, HDL) verwendet. Zwei unabhängige Beschreibungen zu erstellen ist sehr zeitaufwändig. Außerdem entstehen durch das manuelle Vorgehen oft Fehler, die – wenn überhaupt – erst spät entdeckt werden.

Die synthetisierbare Beschreibung (C) wird durch ein Synthesewerkzeug automatisch in eine Beschreibung auf Gatterebene (D) überführt. Während des klassischen Syntheseschrittes stehen geringe Signallaufzeiten und eine geringe Schaltkreisgröße als Optimierungskriterien im Vordergrund.

Auf Basis der Beschreibung auf Gatterebene werden durch die *Automatische Testmustergenerierung* (ATPG) Testmuster (E) berechnet, die später auf den tatsächlichen IC angewendet werden, um Fertigungsfehler zu entdecken, die zum Beispiel durch Staubpartikel entstehen. Die Auslieferung fehlerhafter Schaltkreise soll so vermieden werden. Bei der Optimierung während des Syntheseschrittes werden Testaspekte oft aber nur teilweise berücksichtigt. Im Resultat ist der ATPG-Schritt sehr ressourcenintensiv, da ein NP-vollständiges Problem zugrunde liegt [IS75].

Neben dem Entwurf ist der Nachweis der Korrektheit des entworfenen ICs wichtig. Der Ablauf für den Korrektheitsnachweis ist in der rechten Hälfte von Abbildung 2 zu sehen. Aus der Spezifikation werden typische Szenarien abgeleitet, die die Funktionsweise des ICs beschreiben. Diese werden manuell in einer sogenannten Testbench (F) zusammengefasst, die die entsprechenden Eingangswerte und die erwarteten Ausgabewerte des Schaltkreises enthält. Diese Szenarien werden mit der Beschreibung auf Systemebene (B) und der synthetisierbaren Beschreibung (C) abgeglichen. Wird fehlerhaftes Verhalten festgestellt, resultieren Gegenbeispiele (G), d.h. Simulationsläufe, die die Fehlfunktion hervorrufen. Die vollständige Abdeckung des Verhaltens des ICs kann dabei im Allgemeinen nicht erreicht werden, da der zu betrachtende Zustandsraum exponentiell groß in der Anzahl der Speicherelemente ist. Ein IC kann dabei mehrere Tausend Speicherelemente enthalten. Insbesondere werden schwer erzeugbare Sonderfälle in der Simulation oft nicht berücksichtigt, so dass Entwurfsfehler nicht entdeckt werden. Ein bekanntes Beispiel ist der sogenannte Pentium-Bug. Ein Fehler in der Gleitkomma-Einheit der ersten Pentium-Prozessoren führte in bestimmten Fällen zu fehlerhaften Rechenergebnissen. Neben dem Imageverlust resultierte für Intel ein wirtschaftlicher Schaden von mehreren 100 Millionen Dollar [Fis95].

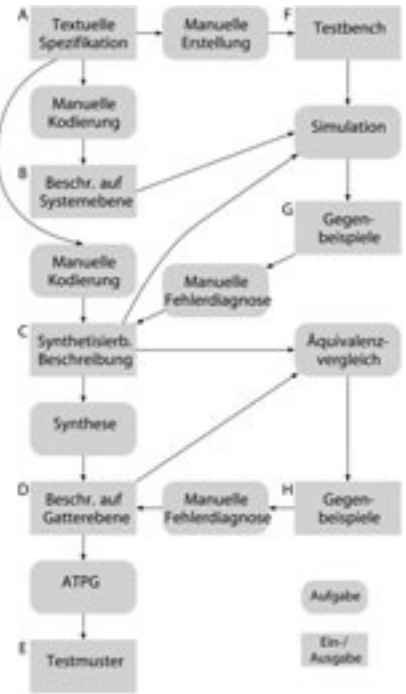

Abbildung 2: Etablierter Entwurfsablauf

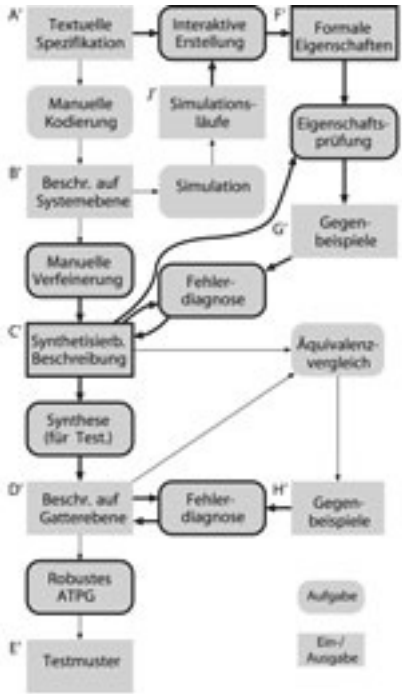

Abbildung 3: Erweiterter Entwurfsablauf

Wurde die Existenz eines Fehlers entdeckt, so muss anschließend die Fehlerstelle in der Beschreibung lokalisiert werden (G)→(C). Hierfür stehen meist nur Simulatoren zur Verfügung, im Wesentlichen wird dieser Debugging-Schritt aber von Hand durchgeführt. Werkzeuge, die das Debugging automatisieren oder hinreichend unterstützen sind bisher nicht verfügbar. Im Resultat ist die Fehlerlokalisierung meist genauso aufwändig oder sogar aufwändiger als die Entdeckung von Fehlern.

Auch der automatisierte Syntheseschritt (C)→(D) wird auf Korrektheit überprüft. Zum Einen werden durch fehlerhafte Synthesewerkzeuge Fehler in den Schaltkreis eingebracht. Zum Anderen sind oft manuelle Änderungen notwendig, um kurz vor Beginn der Fertigung letzte Änderungen vorzunehmen (sogenannte *Engineering Change Orders*). Die Korrektheitsprüfung wird – auch im industriellen Umfeld – durch den Äquivalenzvergleich (C)→(H) vollautomatisch unter Verwendung formaler Methoden durchgeführt [Dre01]. Das heißt, dass die Äquivalenz beider Beschreibungen – vor und nach der Synthese – bewiesen wird und somit in allen Systemzuständen unter allen Eingabesequenzen gilt.

Allerdings wird auch hier die Fehlerlokalisierung bzw. -diagnose (H)→(D) nicht ausreichend unterstützt. Die Existenz des Fehlers wird automatisch nachgewiesen und durch Gegenbeispiele (H) belegt, die Fehlerdiagnose bleibt häufig ein manueller Prozess.

Im Folgenden werden die neuen Ansätze aus [Fey06] vorgestellt, um die genannten Probleme zu beheben bzw. abzuschwächen. Ein besonderer Fokus liegt dabei auf der Robustheit und der Benutzbarkeit.

3 Erweiterter Entwurfsablauf

Aus Komplexitätsgründen ist es derzeit nicht möglich, wesentliche Schritte des oben vorgestellten etablierten Entwurfsablaufs gänzlich zu ersetzen und die jeweilige Transformation vollständig durch ein Werkzeug durchführen zu lassen. Außerdem ist es nicht sinnvoll den etablierten Entwurfsablauf zu stark zu ändern, da dann die Einführung in der praktischen Anwendung kaum oder nur unter extrem hohem Kostenaufwand möglich wäre. Stattdessen werden im Folgenden Ansätze vorgestellt, um die einzelnen Entwurfsschritte in Bezug auf Robustheit und Benutzbarkeit der verwendeten Werkzeuge zu verbessern und die identifizierten Probleme so zu mildern. Aus der Kombination all dieser Ansätze resultiert ein erweiterter Entwurfsablauf, der in dieser Form erstmals in [Fey06] präsentiert wurde. Die einzelnen Ansätze wurden zuvor bereits veröffentlicht, die zugehörigen Publikationen werden in der folgenden Darstellung referenziert.

Zunächst werden die zugrunde liegenden Beweisalgorithmen betrachtet. Durch eine Integration unterschiedlicher Techniken lässt sich die Effektivität zur Lösung bestimmter Problemklassen steigern [DFK06]. Weiterhin werden neue Optimierungskriterien betrachtet [FD05b], um so die Benutzbarkeit weiterer Schritte wie zum Beispiel der Fehlerdiagnose zu berücksichtigen.

Neben diesen algorithmischen Verbesserungen werden methodische Veränderungen vorgeschlagen – und ebenfalls durch Algorithmen umgesetzt. Der resultierende erweiterte Entwurfsablauf ist in Abbildung 3 dargestellt. Die Neuerungen gegenüber dem etablierten Entwurfsablauf in Abbildung 2 sind durch schwarze Umrandungen bzw. fettgedruckte Pfeile dargestellt.

Zunächst werden Verbesserungen im Entwurf diskutiert. Statt zwei unterschiedliche Beschreibungssprachen für Systemebene und synthetisierbare Beschreibung zu verwenden, wird für beide Ebenen die C++-basierte Sprache SystemC [LTG97] verwendet. Dadurch sind die beiden Beschreibungsebenen nicht mehr entkoppelt und an Stelle zweier unabhängiger Kodierungsschritte ist nur eine manuelle Verfeinerung (B')→(C') notwendig. Die Robustheit des Entwurfsablaufs wird erhöht, da nur noch einfache Transformationsschritte notwendig sind. Grundlage für dieses Vorgehen bildet ein neues Synthesewerkzeug für SystemC [FGC+04].

Während der Synthese werden im erweiterten Entwurfsablauf bereits Testaspekte berücksichtigt und so eine „Synthese für Testbarkeit" (C')→(D') durchgeführt [DSF04]. Das verwendete Verfahren schränkt die Schaltkreisstruktur ein und garantiert dadurch 100% Testbarkeit unter verschiedenen praxisrelevanten Fehlermodellen. Weiterhin kann die Testmustergenerierung in polynomieller Zeit in der Größe der Schaltkreise durchgeführt werden, obwohl eigentlich ein NP-vollständiges Problem zugrunde liegt. Die Robustheit des Ablaufs wird gesteigert.

Die genannten Restriktionen der Schaltkreisstruktur führen für bestimmte Boolesche Funktionen allerdings zu sehr großen Schaltkreisen, so dass das Syntheseverfahren in diesen Fällen nicht anwendbar ist. Stattdessen muss auf Standardalgorithmen zurückgegriffen werden, weshalb der ATPG-Schritt (D')→(E') weiter schwer bleibt. Um die Robustheit der eingesetzten Algorithmen zu erhöhen, wurden auf Boolescher Erfüllbarkeit [DP60]

beruhende Beweisalgorithmen [MMZ+01, ES04] in den ATPG-Schritt integriert. Das resultierende Werkzeug kann deutlich mehr Fehler in industriellen Schaltkreisen klassifizieren als traditionelle ATPG-Algorithmen [SFD+05].

Für die Verifikation werden die inhärent unvollständigen simulationsbasierten Verfahren weitgehend durch die formale Eigenschaftsprüfung [BCCZ99] ersetzt (F'). Diese hat bereits heute eine Reife erreicht, die den praktischen Einsatz im industriellen Umfeld ermöglicht [WTSF04].

Die aufwändige Erstellung der formalen temporallogischen Eigenschaften wird teilweise automatisiert [FD04]. Zunächst werden Eigenschaften aus Simulationsläufen automatisch abgeleitet, die dann interaktiv gewählt bzw. modifiziert werden können (A')→(F'). Dieser innovative Ansatz erleichtert die Handhabung des formalen Eigenschaftsbeweisers.

Verfahren zur teilautomatisierten Fehlerlokalisierung [FD05a, FSVD06, SFBD06] runden den erweiterten Entwurfsablauf ab. Dies gilt sowohl für die Fehlersuche nach der Eigenschaftsprüfung (G')→(C') als auch nach dem Äquivalenzvergleich (H')→(D'). Durch diese Analysemethoden wird die Benutzbarkeit der formalen Werkzeuge verbessert.

Eine schrittweise Integration der einzelnen Ansätze in den etablierten Entwurfsablauf und somit eine sanfte Migration sind möglich. Beide Aspekte – Robustheit und Benutzbarkeit – werden damit deutlich verbessert. Da die ausführliche Darstellung aller Ansätze hier nicht möglich ist, wird in diesem Abschnitt die automatisierte Fehlerdiagnose für die formale Eigenschaftsprüfung beispielhaft herausgegriffen.

4 Debugging formaler Eigenschaften

Nach einer kurzen Einführung in die Problemstellung wird der Ansatz erläutert und es werden experimentelle Ergebnisse präsentiert, die belegen, welche Vorteile der Ansatz in der Praxis hat.

Die formale Eigenschaftsprüfung dient dem Abgleich zwischen textueller Spezifikation und synthetisierbarem Modell des ICs. Der schematische Ablauf dazu wurde in Abbildung 3 bereits dargestellt. Aus der Spezifikation (A') werden temporallogische Eigenschaften (F') extrahiert, die dann vollautomatisch auf dem synthetisierbaren Schaltkreismodell (C') nachgewiesen werden. Durch die Verwendung formaler Beweismethoden ist garantiert, dass alle möglichen Zustände und Eingabesequenzen für den IC überprüft werden. Der Vorteil der formalen Eigenschaftsprüfung besteht darin, dass selbst unerwartete Eingabesequenzen, die durch zufällige Tests nur schwer erzeugbar sind, berücksichtigt werden.

Als praktisches Beispiel lässt sich eine Ampelsteuerung heranziehen. Es muss garantiert werden, dass kein Unfall passiert. Demzufolge darf für eine Fahrtrichtung niemals „grün" angezeigt werden, wenn für den kreuzende Fußgängerüberweg „grün" angezeigt wird. Diese Eigenschaft wird in eine temporallogische Formel überführt und auf dem Schaltkreismodell nachgewiesen.

Gilt die Eigenschaft nicht, so wird ein Gegenbeispiel erzeugt, d.h. eine Eingabesequenz für den Schaltkreis, die das widersprüchliche Verhalten hervorruft. Im Anschluss wird

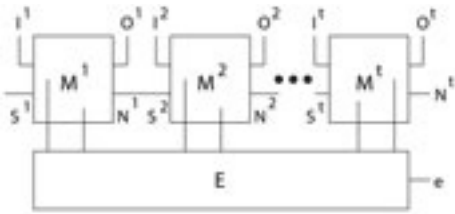

Abbildung 4: Eigenschaftsprüfung

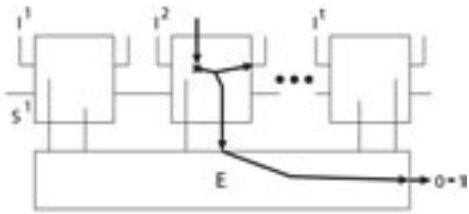

Abbildung 5: Diagnose

die Fehlerursache gesucht – wie bei der Programmierung von Software wird *Debugging* durchgeführt[2]. Die erhaltene Eingabesequenz wird dazu auf dem Schaltkreismodell simuliert, um den Fehler im Modell zu finden, der die Verletzung der Eigenschaft hervorruft.

Da dieser Debugging-Vorgang abgesehen von der Simulation manuell durchgeführt werden muss, ist er sehr zeitaufwändig. Der Entwerfer muss die Beschreibung analysieren, um zu verstehen, welche Veränderung das Fehlverhalten beseitigen würde. Stattdessen werden in [FD05a, SFBD06] erstmals Verfahren vorgestellt, die genau diesen Schritt teilweise automatisieren. Basierend auf einem oder mehreren Gegenbeispielen werden vollautomatisch Komponenten, d.h. Bereiche der Schaltkreisbeschreibung, berechnet, deren Modifikation zu einer Korrektur führen kann. Dadurch muss nur noch ein deutlich kleinerer Teil der Beschreibung durch den menschlichen Entwerfer analysiert werden, um die Korrektur abschließend durchzuführen.

Das technische Vorgehen zur Realisierung des Ansatzes wird im Folgenden beschrieben. Gegeben sind ein Schaltkreismodell M, eine Eigenschaft E und ein Gegenbeispiel G. Es liegt ein diskretes Zeitmodell zu Grunde, d.h. die Zeit ist in Abschnitte gleicher Länge unterteilt, sogenannte Takte. Die Eigenschaft E argumentiert über t Takte und beschreibt Zusammenhänge zwischen Signalen des Schaltkreises innerhalb dieser t Takte. Zur Eigenschaftsprüfung muss der Schaltkreis M nun über t Takte modelliert werden. Sind im Takt i die Eingangswerte durch I^i und die Werte aller Speicherelemente durch S^i gegeben, so liefert das Schaltkreismodell M die daraus resultierenden Ausgangswerte O^i und die neuen Werte für die Speicherelemente N^i. Dadurch kann das zeitliche Verhalten des Schaltkreises M wie in Abbildung 4 modelliert werden. Für jeden Takt i wird eine Kopie von M angelegt. Die Werte S^{i+1} der Speicherelemente in Takt $i+1$ sind dann durch die neuen Werte N^i aus dem vorherigen Takt i gegeben. Sind die Werte S^1 und eine Eingabesequenz $I^1, \ldots, I^t$ gegeben, können daraus alle weiteren Werte berechnet werden. Die Eigenschaft E wird nun ausgewertet, indem interne Werte des Schaltkreises abgegriffen und in Beziehung gesetzt werden. Die Gültigkeit der Eigenschaft unter der Eingabesequenz wird durch den Wert 1 am Ausgang e der Eigenschaft angezeigt, bei Ungültigkeit wird eine 0 erzeugt.

Für das Gegenbeispiel $G = (S^1, I^1, \ldots, I^t)$ wird also eine 0 an e erzeugt, da die Eigenschaft nicht gilt. Nun werden mittels Boolescher Beweismethoden vollautomatisch Modifikationen des Schaltkreismodells M berechnet, die den Ausgangswert e von 0 auf 1 verän-

[2]Der Begriff „Debugging" stammt aus der Anfangszeit der Rechenanlagen. Damals wurde in dem raumgroßen Rechner Mark I tatsächlich ein Fehler verursacht, als ein Käfer (engl. bug) das Schalten eines Relais verhinderte. Es musste eine „Ent-Käferung" (De-bugging) vorgenommen werden.

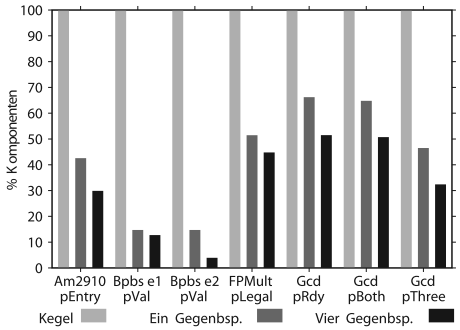

Abbildung 6: Fehlerkandidaten

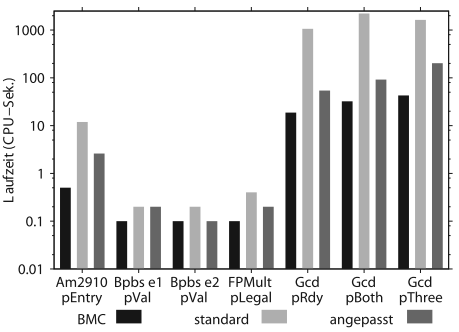

Abbildung 7: Laufzeit

dern. Dies ist in Abbildung 5 dargestellt. In das Schaltkreismodell wird an einer Stelle 'x' eine Änderung injiziert (dargestellt durch einen Pfeil). Die Auswirkung dieser Änderung wird berechnet. Falls es dadurch zur Änderung von 0 auf 1 an e kommt, so ist die Stelle 'x' eine mögliche Korrekturstelle. Die Erweiterung auf mehrere gleichzeitige Korrekturen ist möglich. Auf diese Art werden alle potenziellen Korrekturstellen berechnet, die tatsächliche Fehlerstelle muss unter diesen dann manuell ausgewählt werden. Dieser letzte manuelle Schritt bleibt notwendig, da es auf logischer Ebene meist mehrere äquivalente Korrekturmöglichkeiten gibt. Der menschliche Entwerfer muss also die beste Möglichkeit auswählen.

Der praktische Nutzen dieses Debugging-Verfahrens wurde anhand mehrerer Beispiele evaluiert. Es wurde jeweils zufällig ein Fehler in das Schaltkreismodell injiziert, indem eine Bedingung, Zuweisung oder Konstante verändert wurde. Dadurch wurden die zugehörigen Eigenschaften ungültig.

Die Vereinfachung des manuellen Debugging-Vorgangs ist der wichtigste Vorteil des Verfahrens. Um den Einfluss zu bewerten, wurde die Anzahl der Komponenten gezählt, die bei verschiedenen Vorgehensweisen zu betrachten sind. Die prozentualen Ergebnisse sind in Abbildung 6 dargestellt. An der X-Achse sind die betrachteten Schaltkreise und darunter die zugehörige Eigenschaft angegeben. Durch eine schnelle strukturelle Analyse kann berechnet werden, welche Komponenten einen Einfluss auf die betrachtete Eigenschaft haben – in der Abbildung als „Kegel" bezeichnet. Dieses Verfahren stellt mit 100% den Referenzwert dar. Ergebnisse des automatischen Debugging-Verfahrens sind dann für „ein Gegenbeispiel" und „vier Gegenbeispiele" angegeben.

Im Vergleich zur einfachen strukturelle Analyse liefert das neue Verfahren deutlich weniger Komponenten als Fehlerkandidaten. Schon durch die Verwendung nur eines Gegenbeispiels können zahlreiche Komponenten ausgeschlossen werden. Die Verwendung von vier Gegenbeispielen liefert schließlich nur noch einen kleinen Anteil als mögliche Korrekturstellen. Selbst im schlechtesten Fall werden bei „Gcd" für die Eigenschaft „pRdy" 49% aller Komponenten automatisch von der manuellen Betrachtung ausgeschlossen. Für „Bpbs e2" wird gegenüber dem strukturellen Verfahren sogar eine Reduktion um 96% erreicht. werden. Im Vergleich müssen nur noch 4% der struturell relevanten Komponenten vom Entwerfer betrachtet werden. Das Verfahren liefert das Ergebnis direkt als Annotation in der Hardware-Beschreibungssprache (nicht etwa auf Gatterebene).

Ein weiterer wichtiger Faktor ist die Laufzeit. Die gemessenen Laufzeiten bei Verwendung von vier Gegenbeispielen sind in Abbildung 7 auf der Y-Achse in einer logarithmischen Skala dargestellt. Dabei bezeichnet „standard" das automatische Debugging-Verfahren in der grundlegenden Implementierung. Durch Verbesserungen auf algorithmischer Ebene wurde das Suchverfahren der Problemstellung besser „angepasst". Schließlich gibt „BMC" zum Vergleich die Laufzeit an, die für die anfängliche Eigenschaftsprüfung aufgewendet wird. Es zeigt sich, dass das Standardverfahren oft deutlich längere Laufzeiten hat als die Eigenschaftsprüfung. Nach der Anpassung wird auch bei den Beispielen mit hoher Laufzeit (Gcd) für das Debugging-Verfahren höchstens der 5-fache Zeitaufwand der Eigenschaftsprüfung benötigt. Insgesamt wird durch die Anpassungen ein Laufzeitverhalten erzielt, dass im Bereich der Eigenschaftsprüfung liegt.

Für die Praxis bedeutet dies, dass unter Einsatz geringer Rechenressourcen der Aufwand für das manuelle Eingreifen deutlich reduziert werden kann. Neben dem positiven Einfluss auf die Entwurfskosten, da Arbeitszeit gespart werden kann, wird vor allem ein Produktivitätszuwachs erreicht, weil der Debugging-Schritt deutlich schneller durchgeführt werden kann. Dies führt zur einer Beschleunigung sowohl des Entwurfsprozesses als auch des Korrektheitsnachweises.

5 Zusammenfassung

In der vorgestellten Dissertation werden verschiedene Verfahren entwickelt, um die Robustheit und die Benutzbarkeit von Werkzeugen im Schaltkreisentwurf zu verbessern. Neben der Erarbeitung der algorithmischen und theoretischen Grundlagen für die einzelnen Verfahren, sind die experimentelle Evaluation und die Integration in den Entwurfsablauf wichtige Ergebnisse der Arbeit.

Literatur

[BCCZ99] A. Biere, A. Cimatti, E. Clarke und Y. Zhu. Symbolic model checking without BDDs. In *Tools and Algorithms for the Construction and Analysis of Systems*, *LNCS*, Band 1579, Seiten 193–207, Springer Verlag, 1999.

[DFK06] R. Drechsler, G. Fey und S. Kinder. An Integrated Approach for Combining BDD and SAT Provers. In *VLSI Design Conf.*, Seiten 237–242, 2006.

[DP60] M. Davis und H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:506–521, 1960.

[Dre01] R. Drechsler. Äquivalenzvergleich digitaler Schaltungen im industriellen Umfeld. *it+ti*, 4:200–205, 2001.

[DSF04] R. Drechsler, J. Shi und G. Fey. Synthesis of Fully Testable Circuits from BDDs. *IEEE Trans. on CAD*, 23(3):440–443, 2004.

[ES04] N. Eén und N. Sörensson. An extensible SAT solver. In *SAT 2003*, *LNCS*, Band 2919, Seiten 502–518, Springer Verlag, 2004.

[FD04] G. Fey und R. Drechsler. Improving Simulation-Based Verification by Means of Formal Methods. In *ASP Design Automation Conf.*, Seiten 640–643, 2004.

[FD05a] G. Fey und R. Drechsler. Efficient Hierarchical System Debugging for Property Checking. In *IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*, Seiten 41–46, 2005.

[FD05b] G. Fey und R. Drechsler. Minimizing the Number of Paths in BDDs - Theory and Algorithm. *IEEE Trans. on CAD*, 25(1):4–11, 2005.

[Fey06] G. Fey. *Increasing Robustness and Usability of Circuit Design Tools by Using Formal Techniques*. Dissertation, Universität Bremen, 2006.

[FGC+04] G. Fey, D. Große, T. Cassens, C. Genz, T. Warode und R. Drechsler. ParSyC: An Efficient SystemC Parser. In *Workshop on Synthesis And System Integration of Mixed Information technologies*, Seiten 148–154, 2004.

[Fis95] L.M. Fisher. COMPANY REPORTS; Intel Earnings Decline 37% On Charge for Pentium Flaw. *The New York Times*, 18. Januar 1995.

[FSVD06] G. Fey, S. Safarpour, A. Veneris und R. Drechsler. On the Relation Between Simulation-based and SAT-based Diagnosis. In *Design, Automation and Test in Europe*, Seiten 1139–1144, 2006.

[IS75] O.H. Ibarra und S.K. Sahni. Polynomially Complete Fault Detection Problems. *IEEE Trans. on Comp.*, 24:242–249, 1975.

[LTG97] S. Liao, S. Tjiang und R. Gupta. An Efficient Implementation of Reactivity for Modeling Hardware in the Scenic Design Environment. In *Design Automation Conf.*, Seiten 70–75, 1997.

[MMZ+01] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang und S. Malik. Chaff: Engineering an Efficient SAT Solver. In *Design Automation Conf.*, Seiten 530–535, 2001.

[Moo65] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.

[SFBD06] S. Staber, G. Fey, R. Bloem und R. Drechsler. Automatic Fault Localization for Property Checking. In *Haifa Verification Conference*, 2006.

[SFD+05] J. Shi, G. Fey, R. Drechsler, A. Glowatz, F. Hapke und J. Schlöffel. PASSAT: Effcient SAT-based Test Pattern Generation for Industrial Circuits. In *IEEE Annual Symposium on VLSI*, Seiten 212–217, 2005.

[WTSF04] K. Winkelmann, H.-J. Trylus, D. Stoffel und G. Fey. Cost-efficient Block Verification for a UMTS Up-link Chip-rate Coprocessor. In *Design, Automation and Test in Europe*, Seiten 162–167, 2004.

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.

Seine Forschungsinteressen liegen in den Bereichen Verifikation und Test von digitalen Schaltkreisen.