Vollständige funktionale Verifikation

Jörg Bormann
geboren in Marburg

vom Fachbereich Elektrotechnik und Informationstechnik der Universität Kaiserslautern zur Verleihung des akademischen Grades Doktor der Ingenieurwissenschaften (Dr.-Ing) genehmigte

Dissertation

Universität Kaiserslautern
Zeichen im Bibliotheksverkehr: D386

Datum der mündlichen Prüfung: 8. Juni 2009
Dekan: Prof. Dr. Gerhard Fohler
Berichterstatter: Prof. Dr. Wolfgang Kunz, Prof. Dr. Wolfram Büttner

Zusammenfassung

Diese Arbeit beschreibt einen in der Praxis bereits vielfach erprobten, besonders leistungsfähigen Ansatz zur Verifikation digitaler Schaltungsentwürfe. Der Ansatz ist im Hinblick auf die Schaltungsqualität nach der Verifikation, als auch in Bezug auf den Verifikationsaufwand der simulationsbasierten Schaltungsverifikation deutlich überlegen.

Die Arbeit überträgt zunächst das Paradigma der transaktionsbasierten Verifikation aus der Simulation in die formale Verifikation. Ein Ergebnis dieser Übertragung ist eine bestimmte Form von formalen Eigenschaften, die Operationseigenschaften genannt werden. Schaltungen werden mit Operationseigenschaften untersucht durch Interval Property Checking, einer besonders leistungsfähigen SAT-basierten funktionalen Verifikation. Dadurch können Schaltungen untersucht werden, die sonst als zu komplex für formale Verifikation gelten. Ferner beschreibt diese Arbeit ein für Mengen von Operationseigenschaften geeignetes Werkzeug, das alle Verifikationslücken aufdeckt, komplexitätsmäßig mit den Fähigkeiten der IPC-basierten Schaltungsuntersuchung Schritt hält und als Vollständigkeitprüfer bezeichnet wird. Die Methodik der Operationseigenschaften und die Technologie des IPC-basierten Eigenschaftsprüfers und des Vollständigkeitsprüfers gehen eine vorteilhafte Symbiose zum Vorteil der funktionalen Verifikation digitaler Schaltungen ein. Darauf aufbauend wird ein Verfahren zur lückenlosen Überprüfung der Verschaltung derartig verifizierter Module entwickelt, das aus den Theorien zur Modellierung digitaler Systeme abgeleitet ist.

Der in dieser Arbeit vorgestellte Ansatz hat in vielen kommerziellen Anwendungsprojekten unter Beweis gestellt, dass er den Namen "vollständige funktionale Verifikation" zu Recht trägt, weil in diesen Anwendungsprojekten nach dem Erreichen eines durch die Vollständigkeitsprüfung wohldefinierten Abschlusses keine Fehler mehr gefunden wurden. Der Ansatz wird von OneSpin Solutions GmbH unter dem Namen "Operation Based Verification" und "Gap Free Verification" vermarktet.

Lebenslauf

1970 – 74	Grundschule in Kelkheim/Ts.
1974 – 83	Taunusschule Königstein, Gymnasium, Abitur
1983 – 84	Grundwehrdienst
1984 – 90	Studium der Technomathematik an der Universität (TH) Karlsruhe Diplom-Technomathematiker
1990 – 93	Siemens AG, Zentrale Forschung Hardwareverifikation mit Theorembeweisern
1993 – 98	Siemens AG, Zentrale Forschung BDD-basiertes Model- und Equivalence Checking
1998 – 2002	Siemens AG, Zentrale Forschung, Staff Engineer SAT-basierte Hardwareverifikation: Forschung und Anwendungen
2002 – 05	Infineon Technologies, Manager Formal Verification SAT-basierte Hardwareverifikation: Methodikentwicklung, Anwendungen
2005 – 08	OneSpin Solutions GmbH, Senior Manager Verification Methodology Produktdefinition, Methodikentwicklung, Anwendungen
2008 – 09	OneSpin Solutions GmbH, Director Advanced Applications & Technology

Inhaltsverzeichnis

1 Vorwort

Diese Arbeit ist in einem industriellen Umfeld entstanden, zunächst bei der zentralen Forschung und Entwicklung der Siemens AG, dann bei Infineon, und schließlich in dem Startup OneSpin Solutions GmbH, der diese Ideen unter den Schlagworten "Operation Based Verification" und "Gap Free Verification" weiterentwickelt und vermarktet.

Dies hat dazu geführt, dass viele Ideen dieser Arbeit intensiv geprüft sind. Ohne den überzeugenden Ausgang von Pilotprojekten bis hin zum kommerziellen Einsatz hätte ich nie die Chance zur Ausarbeitung der vollständigen funktionalen Verifikation erhalten. An diesen Projekten haben unsere Kunden, sowohl die internen in unseren Siemens- und Infineonzeiten, als auch die echten Kunden großen Anteil gehabt. Ich bin ihnen dankbar für ihre Herausforderungen, für die Einsichten, die sie mir in ihre Vorgehensweisen vermittelten, und manchmal auch für die Begeisterung, mit der sie unsere Ergebnisse zur Kenntnis nahmen, obwohl diese Ergebnisse ja in den allermeisten Fällen Fehler in ihren Schaltungsentwürfen aufdeckten und zu zusätzlicher Arbeit führten.

Die hier vorgestellten Arbeiten sind in einem hochkarätigen Umfeld entstanden. So konnte ich auf einer hochklassigen Technologie aufsetzen, die viele Mütter und Väter hat. Sie einzeln zu nennen, würde den Rahmen dieses Vorworts sprengen.

Unsere Frontendleute entwickelten und betreuen die präzisen und leistungsfähigen Programmteile zum Einlesen der Schaltungen, die mit allen Finessen und allen Größen der ihnen vorgelegten RTL-Beschreibungen zurechtkommen. Die in unserer Beweisergruppe entwickelten Werkzeuge rechnen wir zu den weltweit leistungsfähigsten. Sie haben uns in einer Weise verwöhnt, dass wir heutzutage selbst bei großen Schaltungen ungeduldig werden, wenn ein Beweis mal länger braucht als 5 Minuten. Eine ganz wichtige Entwicklungsaufgabe für ein formales Verifikationswerkzeug besteht in der Aufbereitung von Benutzereingaben und in der intuitiven Rückmeldung über die Beweisergebnisse. Die damit befassten Kollegen haben ein angenehm und effizient zu bedienendes Produkt geschaffen, das mit höchsten Komplexitätsanforderungen zurecht kommt. All diesem wurde gleichsam als Krone ein professionelles graphisches Benutzerinterface aufgesetzt, das die Funktionalität auf leicht verständliche Weise darbietet. All diese in dem Produkt OneSpin 360 MV vereinigte Technologie stand mir zur Verfügung, um die hier vorgestellten Ideen zu entwickeln, die ihrerseits dann wieder das Produkt beeinflussten. Dafür bin ich den Entwicklern dankbar.

Neben den Entwicklern danke ich Sven Beyer, Martin Freibothe, Steven Obua, Jens Schönherr und Sebastian Skalberg, die mit mir in der Gruppe für Methodik, Technologie und fortgeschrittene Anwendungen der OneSpin Solutions GmbH sind oder waren, für ihre Ideen, Rückmeldungen, intensiven Diskussionen und Weiterentwicklungen. Weitere fruchtbare Diskussionen führte ich mit unseren akademischen Kooperationspartnern. Darunter danke ich besonders Professor Wolfgang Kunz für die technisch-wissenschaftlichen Diskussionen, für die Unterstützung dieser Arbeit und für die Rückmeldungen als Doktorvater. Auch Dominik Stoffel danke ich für seine vielen Hinweise.

Die Arbeiten wurden durch das Bundesministerium für Forschung und Entwicklung im Rahmen der Forschungsprojekte VALSE, Herkules und Verisoft gefördert, die auch unsere Kontakte in die akademische Welt unterstützten.

Dieses fruchtbare hochkarätige Umfeld geht auf Professor Wolfram Büttner zurück. Sein strategisches Geschick und tiefe technische Einsicht hat viele von uns bewogen, in diesem Aufgabenfeld zu arbeiten. Er hielt diese Arbeitsgruppe über 10 Jahre unter sehr wechselhaften Bedingungen zusammen und gab immer wieder die zentralen Denkanstöße. Dies gipfelte in der von ihm gegründeten OneSpin Solutions GmbH. Sein Anteil an dieser Arbeit geht aber weit über das Geschäftliche hinaus. Vielmehr hat er mir in unzähligen Gesprächen und bei vielen gemeinsamen Mittagessen geholfen, die Gedanken zu entwickeln und auf den Punkt zu bringen, sowie die Technologie in ihrer praktischen Relevanz einzuschätzen. Dafür danke ich ihm, genauso wie für seine Anregung zu dieser Arbeit, die vielen Rückmeldungen, auch zu Vorversionen, sowie insgesamt für ihre Betreuung.

Der heutigen Firmenleitung von OneSpin Solutions danke ich für die Unterstützung und Rücksichtnahme während der Anfertigung dieser Arbeit, insbesondere Michael Siegel für seine hilfreichen Kommentare.

Eine Konsequenz des industriellen Umfelds, in dem diese Arbeit entstand, ist der Patentschutz, der für einige der Konzepte und Erfindungen erwirkt oder beantragt wurde, die mit dieser Arbeit in Beziehung stehen, so z.B. für den Vollständigkeitsprüfer [Busch/Bormann 2005] und die Generierung vollständiger Eigenschaftssätze zur Verifikation von Prozessorarchitekturen gegen Prozessorpipelines [Bormann/Beyer/Skalberg 2006]. Ich muss darauf hinweisen, dass die unautorisierte Verwendung der patentierten Anteile dieser Arbeit zu Forschungszwecken oder in der industriellen Anwendung eine Patentrechtsverletzung darstellt, die rechtliche Konsequenzen hätte.

Eine Firma wie OneSpin Solutions GmbH arbeitet getrieben von echten und antizipierten Kundenwünschen. Ich möchte deswegen betonen, dass ich diese Arbeit unabhängig von der Firmenmeinung und -strategie geschrieben habe. Die hier veröffentlichten Einschätzungen, Meinungen und Schlussfolgerungen sind meine eigenen und sind unabhängig von der offiziellen Linie der OneSpin Solutions GmbH aufgestellt worden. Entsprechend macht diese Arbeit auch keine Aussagen über Entwicklungsplanungen oder Ziele der Fima OneSpin Solutions GmbH.

Meine Familie hat während der Anfertigung dieser Arbeit noch mehr auf Ehemann und Vater verzichten müssen, als vorher schon, und hin und wieder auch noch Berichte über besonders guten oder auch besonders mangelhaften Fortschritt ertragen müssen. Beate, Ines und Joachim danke ich für Geduld, Nachsicht und Unterstützung.

2 Funktionale Verifikation

2.1 Formale Verifikation und das Verification Gap

Die Klage über das “verification gap” begleitet die Halbleiterindustrie seit mindestens 15 Jahren. Konkret bedeutet diese Klage, dass heutzutage ca. 70% der R&D-Kosten für einen ASIC nicht für den eigentlichen Schaltungsentwurf, sondern für die Verifikation, d.h. die Qualitätssicherung des Entwurfs, aufgewandt werden. Trotz der enormen Kosten ist der Erfolg der Verifikation eher bescheiden: In jedem ASIC-Projekt kommt es durchschnittlich 1.7 mal zu der Situation, dass die schon eingerichtete Fertigung gestoppt werden muss, weil sich der ASIC als fehlerhaft erweist, und die Fertigungsunterlagen abgeändert werden müssen. Solche Abänderungen werden Respins genannt, und ihre Kosten liegen bei der heutigen Technologie mit Strukturgrößen von 45 bis 65 nm leicht im Bereich von 1 Mio. Euro. Diese hohe Rate an Fehlschlägen ist kein Wunder, wenn man bedenkt, dass das Hauptarbeitspferd der Verifikation immer noch Simulation ist, in der die Simulationsmodelle rund 1 Mio. mal langsamer ablaufen als die fertigen Chips. Auch wenn viele Rechner Monate lang mit Simulationen beschäftigt sind, werden während der gesamten Verifikationsphase nur einige Minuten der Lebenszeit eines Bausteins untersucht. Dass dabei nicht alle kritischen Situationen ausprobiert werden können und funktionale Fehler im Design verbleiben, ist dann nicht mehr erstaunlich.

Die Probleme kulminieren besonders in der funktionalen Verifikation. Diese soll sicherstellen, dass die einem Endkunden zu Verfügung gestellte Nutzfunktion richtig ist und betrachtet nicht physikalische Störeffekte oder Fehler bei dem Einbringen fertigungsspezifischer Funktionalität wie Testlogik, Clock-Tree-Balancing etc.

Fast genauso alt wie die Klage über das verification gap ist der Traum, dass formale Verifikation das Gap schließen könne, oder doch zumindest helfen könne, es weniger stark klaffen zu lassen. Und tatsächlich trat die formale Verifikation in Form der kombinatorischen Äquivalenzverifikation [Filkorn 1992, Brand 1993, Kunz 1993, Kuehlmann/Krohm 1997, Lohse/Warkentin 1998, Bormann/Warkentin 1999, Hoereth/Müller-Brahms/Rudlof 2002] einen Siegeszug an, der die bis dahin übliche Netzlistensimulation nahezu verdrängt hat.

Aber mit kombinatorischer Äquivalenzverifikation wird nur nachgewiesen, dass die Nutzfunktion erhalten bleibt. Es wird nicht nachgewiesen, ob die Nutzfunktion richtig im Sinne einer Spezifikation ist. Kombinatorische Äquivalenzverifikation kann deswegen nur für Prozessschritte angewandt werden, die nach der Definition der Nutzfunktion ausgeführt werden. Kombinatorische Äquivalenzverifikation ist also gerade keine funktionale Verifikation.

2.2 Assertionbasierte Formale Verifikation (ABFV)

Formale Verifikation hat wiederholt gezeigt, welchen Nutzen sie auch in der funktionalen Verifikation haben kann, wenn sie beträchtliche Beiträge zum Entwurf oder der Verifikation besonders komplizierter Bausteine wie Prozessoren [Beyer 2005, Buckow/Bormann 1993] oder Filter [Busch 1991] lieferte. Aber dann wurde sie von hochspezialisierten Experten eingesetzt, die mit Theorembeweisern, d.h. höchst komplizierten Beweiswerkzeugen, zu hantieren verstehen. Ansätze, Theorembeweiser für eine größere Anwendergruppe nutzbar zu machen sind bisher kommerziell gescheitert [Bombana et al. 1995].

Für die breite Anwendergruppe der Design- und Verifikationsingenieure hat sich nach 20 Jahren intensiver Forschung auf diesem Gebiet [Bryant 1986, Clarke/Emerson/Sistla 1986, Mayger/Harris 1991, Filkorn 1992] nur die assertionbasierte formale Verifikation (ABFV) als in der Breite anwendbares Benutzungsmodell herausgeschält. Dazu gibt es inzwischen kommer-

zielle ABFV-Werkzeuge [Solidify o.J., IFV 2005, Jasper 2007, 0-in 2005, Magellan o.J., Conquest o.J.], die auf den mit den Forschungswerkzeugen der 1990er Jahre [McMillan 1992, Bormann et al. 1995] gewonnenen Resultaten aufsetzen.

Die Erfahrungen mit den kommerziellen ABFV-Werkzeugen sind aber ernüchternd. Bis in jüngster Zeit wurden ambitionierte Designprojekte noch immer weitgehend ohne den Einsatz formaler funktionaler Verifikation durchgeführt [Shimizu et al. 2006]. Raj Mitra, der bei Texas Instruments an der Spitze der für formale Methoden Verantwortlichen steht [Mitra 2008], erkennt die Vorteile der formalen Verifikation an, benennt aber etwa als Probleme die geringe Schaltungskomplexität, die die Werkzeuge behandeln können, und die damit verbundene Notwendigkeit, ein tiefgehendes Schaltungsverständnis zu entwickeln. Ferner nennt Mitra die Unzuverlässigkeit, mit der formale Resultate in einem gegebenen Zeitraum erzielt werden können. Diese Unsicherheit entstehe, weil immer wieder benutzerdefinierte Abstraktionen [Kroening/Seshia 2007] bemüht werden müssen, um Verifikationsresultate zu erhalten. Mitra hält es für selbstverständlich, dass formale Verifikation die Schaltungsfunktionalität nicht vollständig untersuchen könne.

Ob ABFV jemals die in die formale Verifikation gesetzten Hoffnungen erfüllen wird, erscheint fraglich vor dem Hintergrund von Aussagen der Architekten von ABFV-Werkzeugen, nach denen die Werkzeuge auch heute, nach vielen Jahren intensiver Forschung immer noch Komplexitätsgrenzen bei rund 1000 Zustandsvariablen haben [Jain et al. 2007]. Denn schon in den späten 1990er Jahren wurden Komplexitäten von 300 Zustandsvariablen auf den damaligen Rechnern erreicht, sodass es fraglich ist, wie viel von der Verbesserung seither auf das Konto von Forschungsanstrengungen im Bereich formaler Methoden und wie viel auf die reine Verbesserung der Rechnerleistung und die Ausstattung mit größeren Arbeitsspeichern zurückzuführen ist. Mit Moores Law über das Wachstum der Komplexitäten von integrierten Schaltungen [Moore 1965] hat die Entwicklung der ABFV jedenfalls nicht Schritt gehalten.

Andere Autoren beklagen den Bruch zwischen ABFV und simulationsbasierter Verifikation [Bailey 2007]. Dieser Bruch äußert sich beispielsweise in der Unklarheit darüber, wie viel Simulation durch die formale Verifikation einer gegebenen Menge von Assertions ersetzt werden kann. Er äußert sich auch darin, dass die transaktionsbasierte Betrachtungsweise heutiger Simulationstestbenches keinen Widerhall in der ABFV findet: Assertions untersuchen meist lokale Aspekte des Verhaltens einzelner Signale und lassen das Gesamtbild, in das sich das Signalverhalten eigentlich einfügen sollte, außer Acht. Dies mag vor allem der mangelhaften Komplexität der ABFV-Werkzeuge geschuldet sein, aber der Bruch zieht sich bis hin dazu, dass die Standardsprachen PSL [PSL 2004] oder SVA [System Verilog 2005] der ABFV einer transaktionsbasierten Betrachtungsweise durch formale Werkzeuge entgegenstehen (siehe Abschnitt 4.3.12). Ferner werden dem Benutzer nur fallweise Ideen vermittelt, welche Assertions eigentlich aufzustellen sind [Foster et al. 2003, Foster/Krolnik 2008]. Außerdem erfordert ABFV die Erarbeitung tiefer Schaltungskenntnis, obwohl doch viele Verifikationsingenieure gar nicht in die zu verifizierenden Module hineinsehen können oder wollen. Dies ist die Folge einer lang geübten Arbeitsteilung zwischen Verifikationsingenieuren und Schaltungsentwicklern.

2.3 Heutige Arbeitsteilung zwischen Entwurf und Verifikation

Die heutige Verifikationspraxis folgt den Methoden, die schon vor 20 Jahren beim Entwurf des Alpha-Chips eingesetzt wurden. Sie wird charakterisiert durch die Verwendung von Simulation und dadurch, dass Module als Black-Boxes behandelt werden. Um einen Fehler zu erkennen, muss er also während der Simulation anhand fehlerhaften Verhaltens an den Ausgabesignalen des Moduls sichtbar werden.

Diese beobachtende Qualitätssicherung ist auch in anderen Bereichen der Technik üblich. Dort wird sie aber häufig durch eine verstehende Qualitätssicherung ergänzt, die der Implementierung des Produkts auf den Grund geht und über die Sinnhaftigkeit dieser Implementierung nachdenkt. Im Schaltungsentwurf würde diese zweite Vorgehensweise der Qualitätssicherung einem Codereview oder der in dieser Arbeit vorzustellenden vollständigen Verifikation entsprechen. Codereviews sind als Methode zu Erreichung hoher Qualitätsstandards bekannt [EN61508 2001], werden aber ganz selten eingesetzt.

Die Hauptschwäche einer beobachtenden Qualitätssicherung ist, dass Fehler nur mit gewissen Wahrscheinlichkeiten identifiziert werden. Mit dieser Schwäche hat sich der Schaltungsentwurf aber bisher arrangiert. Die Schaltungsverifikation wurde verbessert durch Verfahren, mit denen die Auftrittswahrscheinlichkeit der Fehler graduell erhöht wurde. Entsprechend ist durch Random Pattern Simulation, Hardwarebeschleunigung oder einfach nur durch schnellere Simulationsalgorithmen dafür gesorgt worden, dass in der selben Zeit mehr simuliert werden kann, oder es ist durch Coveragekriterien dafür gesorgt worden, dass die Stimuli sorgfältiger ausgesucht werden und eine größere Menge an Situationen durchsimulieren.

Dennoch arbeitet schon jeder einzelne produzierte Chip um Größenordnungen schneller als die Simulation während seiner Entwurfsphase, und dann werden unter Umständen auch noch Millionen solcher Chips in Betrieb genommen. Dabei treten selbst Fehler mit kleinsten Auftrittswahrscheinlichkeiten zu Tage. Solange man sich mit graduellen Verbesserungen der Auftrittswahrscheinlichkeiten von Fehlern zufrieden gibt, werden weiterhin Fehler im Feld auftreten, die zu unwahrscheinlich waren, um sie während der Verifikationsphase zu entdecken.

Der schlagende Vorteil der simulationsbasierten Black Box Verifikation besteht aber darin, dass er eine Arbeitsteilung zwischen Entwurf und Verifikation ermöglicht: Der Verifikationscode (d.h. die Testbench) kann unabhängig von der Implementierung entwickelt werden und ist daher idealerweise weiterverwendbar, selbst wenn die Implementierung komplett geändert wird. Der Verifikationsingenieur wird nicht durch die Kenntnis der Implementierung beeinflusst und läuft so weniger Gefahr, sich falsche Ansichten des Designers zueigen zu machen und dadurch Fehler zu akzeptieren. Der Verifikationscode kann unabhängig von der Implementierung entstehen und steht damit früh zur Verfügung. Idealerweise kann er noch während des Schaltungsentwurfs zur schnellen Qualitätssicherung neu entwickelter Schaltungsteile genutzt werden.

Diese Arbeitsteilung zwischen Entwurf und Verifikation ist inzwischen tief verwurzelt: Die Verifikationswerkzeuge und -methoden sind ganz auf die Phänomene an den Modulgrenzen eingestellt, an denen es um Interfaces und um die Transaktionen geht, die sich an diesen Interfaces beobachten lassen. Entsprechend geht es bei den Werkzeugen und Methoden um Transaktionen, ihre Erzeugung und ihre Identifikation. Es gibt zum Beispiel transaktionsbasierte Referenzmodelle und Scoreboards zum Vergleich von Transaktionsfolgen.

Die Verifikationsingenieure begreifen Schaltungen als Objekte, die durch die Folgen der Transaktionen an ihren Interfaces beschrieben werden. Die eigentliche Schaltungsbeschreibung spielt dabei eine nachgeordnete Rolle. Das geht so weit, dass Verifikationsingenieure heute gar nicht mehr in der Lage sind, Schaltungsbeschreibungen zu verstehen, weil ihnen schon die Kenntnis der Schaltungsbeschreibungssprachen (etwa VHDL oder Verilog) und erst recht die Übersicht über Schaltungsstrukturen fehlt. Eine verstehende Qualitätssicherung ist für solche Verifikationsingenieure nicht möglich.

Designer kennen natürlich die Interna der Schaltungsentwürfe. Sie sollen ihre Schaltungsentwürfe aber vor allem schnell liefern, damit die Verifikationsteams mit ihrer Arbeit beginnen können. Qualitätsgesichtspunkte werden der Entwurfsgeschwindigkeit untergeordnet. Die Designer übergeben den Code früh an die Verifikationsingenieure, und machen mit den nächsten Modulen weiter. Fehler werden dadurch später entdeckt als wenn die Designer selbst eine gewisse Qualitätssicherung durchgeführt hätten. Die späte Entdeckung erfordert, dass die Designer sich in halb vergessenen Code wieder hineindenken müssen. Die Fehlerkorrekturen werden dadurch langwieriger und unsicherer.

Diese starke Arbeitsteilung zwischen Designern und Verifikationsingenieuren und der damit verbundene mangelhafte Informationsfluss zwischen den beiden Teilaufgaben wird von Meinungsbildnern in der Industrie inzwischen als ernsthaftes strukturelles Problem betrachtet [Kranen 2008].

Alles in allem entsteht dabei schwer verständlicher RTL-Code. Wäre der Code Software, er würde den Prinzipien einer nachhaltigen, auf Wartbarkeit bedachten Entwicklung häufig nicht genügen. Ein Beleg dafür sind Module, die als firmeninterne IP-Blöcke bereit gehalten werden, um sie in verschiedenste ASICs einzubauen. Solche Module werden gewartet, d.h. sie erfahren mehrere Überarbeitungen, meist auch durch verschiedene Verantwortliche. Bei dieser Wartung rächt sich schwer verständlicher Code, denn die Überarbeitungen werden dann auf der Basis eines unvollkommenen Verständnisses des Codes durchgeführt. Damit sinkt die Qualität des Moduls mit jeder Überarbeitung.

Wenn sich dann Fehler in den IP-Blöcken häufen, wird dieser Qualitätsverlust gerne auf das Alter der Testbench und ihre veraltete Technologie zurückgeführt, obwohl die Testbench ursprünglich zur Sicherung der Qualität der ersten Version ausreichte und die Wiederverwendung ursprünglich auch als eine Maßnahme zur Steigerung der Qualität galt.

2.4 Die heutige Verifikationsmethodik und ABFV

Die Arbeitsteilung zwischen dem Schaltungsentwurf und der Verifikation und die tief verinnerlichte Praxis, Module bei der Verifikation als Black Boxes zu betrachten, führt zu Akzeptanzproblemen der ABFV. Es fängt schon damit an, dass nicht klar ist, wer denn die Assertions schreiben und beweisen soll.

Eine Klasse von Assertions wird als Designer-Assertions charakterisiert. Dabei handelt es sich um Assertions, die eigentlich nichts über die spezifizierte Funktion des Bausteins aussagen, sondern über einzelne Implementierungsdetails, häufig einfach über das lokale Zusammenspiel einiger Codezeilen. Solche Assertions werden von Designern geschrieben. Sie lassen sich leicht beweisen: Dem Designer ist klar, welcher lokale Aspekt der Funktion nachgewiesen werden soll, die Formalisierung durch eine Assertion ist einfach und der relevante Schaltungsteil ist klein, sodass es beim Einsatz der formalen Werkzeuge auch nicht zu Komplexitätsproblemen kommt. Aber solche Assertions sagen auch nicht viel aus, auch dann nicht, wenn sie zu Hunderten oder Tausenden unter Umständen sogar automatisch generiert auftreten.

Eine interessantere Klasse von Assertions sind High Level Assertions, mit denen spezifizierte Aspekte der Funktion eines Bausteins bewiesen werden sollen. In der Spezifikation schauen diese Aspekte auch immer sehr prägnant aus, aber der Teufel liegt im Detail. In der Implementierung mag es vom Entwickler intendierte Ausnahmen von der prägnanten Formulierung in der Spezifikation geben. Diese Ausnahmen müssen entweder in die Assertion eingearbeitet werden, oder sie stehen in Beziehung zu Umgebungsannahmen, die dann ihrerseits formali-

siert werden müssen. Die intendierten Ausnahmen müssen aber von den nicht intendierten Ausnahmen unterschieden werden, denn letztere weisen ja gerade auf die Fehler hin, die die Verifikation aufdecken soll. Sowohl für die Identifikation der intendierten Ausnahmen als auch für die Unterscheidung von nicht intendierten Ausnahmen ist Schaltungskenntnis erforderlich.

Wo Schaltungskenntnis nicht erworben werden soll, beschränkt sich die Verifikation von High Level Assertions auf solche, die als vorfabrizierte Verification IP eingekauft werden können. Diese Einsatzfälle beschäftigen sich häufig mit Protokollcompliance. Mit dieser Verifikationsaufgabe beschäftigt sich aber auch die simulationsbasierte Verifikation intensiv, weil anders die Transaktionen gar nicht identifiziert werden können.

Aber selbst wenn ein Verifikationsingenieur eine erste vertrauenswürdige Formalisierung einer High Level Assertion aufgestellt hat, kann es noch immer zu Komplexitätsproblemen kommen, und zwar meist um so leichter, je mehr die High Level Assertion über die Korrektheit der Schaltung aussagt. Um solche Komplexitätsprobleme zu beherrschen, wird die Aufgabe aufgespalten oder es wird eine abstraktere Version der Aufgabe bewiesen. Beides kann nicht ohne Schaltungskenntnis erfolgreich durchgeführt werden und führt wohl zu Raj Mitras Klage [Mitra 2008] über die Unvorhersehbarkeit des Aufwands für eine formale Verifikation.

Alles in allem ist es nicht möglich, High-Level-Assertions ohne Kenntnis der zu untersuchenden Schaltung sinnvoll formal zu untersuchen. Und damit sitzt die ABFV in Bezug auf High-Level-Assertions zwischen den Stühlen: Entwickler wollen keinen Aufwand investieren, um die High Level Assertions richtig zu formalisieren, und den Verifikationsingenieuren fehlen Zeit, Erfahrung und Kenntnisse, sich in die Schaltungsfunktionalität einzuarbeiten. Solange die Simulation die Haupterkenntnisquelle während der Verifikation ist, stehen darüber hinaus Aufwand und Nutzen von High-Level-Assertions in einem ungünstigen Verhältnis. Entsprechend ist das Konzept der Verifikation von High Level Assertions bisher nicht weit verbreitet.

2.5 Vollständige Verifikation

Die in dieser Dissertation vorzustellenden Arbeiten gehen einen anderen Weg als die ABFV. Zunächst wird in Abschnitt 3.2 eine allgemein anwendbare Idee davon entworfen, was zu verifizieren ist: Die Verifikation wird strukturiert anhand der Operationen einer Schaltung. Operationen einer Schaltung fassen Abläufe der Schaltung über eine eher kürzere Zeitspanne zu sinnvollen Bausteinen der Funktion zusammen. Diese Abstraktion ist so ähnlich der Abstraktion von Interfaceverhalten durch Transaktionen [Cai/Gajski 2003], dass manche Autoren statt von Operationen auch von Transaktionen eines Moduls schreiben. Beispielsweise beschreibt eine Operation eines Businterfaces die Erzeugung einer Transaktion auf dem Bus, Operationen von Prozessoren beschreiben das Holen und die Abarbeitung einer Instruktion in der Prozessorpipeline, und die Operationen eines Arbiter beschreiben die Arbitrierungszyklen.

Viele Operationen lassen sich durch verallgemeinerte Timingdiagramme (siehe Abschnitt 4.3.11) darstellen [Bormann/Spalinger 2001]. Ein Beispiel verallgemeinerter Timingdiagramme zeigt Abbildung 7. Diese Timingdiagramme lassen sich leicht in der Sprache ITL [Siegel et al. 1999] (siehe Abschnitt 4.3) oder auf der Basis der SVA-Bibliothek TiDAL [Bormann 2007] als temporale Eigenschaften ausdrücken. Diese temporalen Eigenschaften werden Operationseigenschaften genannt.

Ein Eigenschaftsprüfer beweist, dass die Schaltung die Operation so durchführt, wie in der Operationseigenschaft beschrieben ist. Für diesen Beweis stehen leistungsfähige Werkzeuge [OneSpin o.J.] zur Verfügung, die Eigenschaften über Module üblicher Größe im Minutenbe-

reich beweisen. Das größte mit diesem Ansatz untersuchte Modul umfasste deutlich mehr als 100 000 Zeilen RTL-Code.

Der Eigenschaftsprüfer verifiziert Operationen, indem er sie von jedem Zustand aus untersucht. Ein Teil seiner Leistungsfähigkeit wird allerdings dadurch erkauft, dass die Eigenschaft auch von solchen Zuständen aus verifiziert wird, die während des Betriebs der Schaltung gar nicht auftreten können. Wenn solche unerreichbaren Zustände zu Gegenbeispielen führen, müssen sie durch Erreichbarkeitsbedingungen ausgeschlossen werden. Diese Erreichbarkeitsbedingungen können teils automatisch bestimmt werden. Wo automatische Verfahren versagen, werden Vorgaben des Verifikationsingenieurs erforderlich. Diese können aus dem Schaltungsverständnis entwickelt werden, oder auch auf Hinweisen des Entwicklers der Schaltung beruhen. Solche Benutzervorgaben werden später im Rahmen der Methodik gerechtfertigt. Diese Vorgehensweise erlaubt es, das Verifikationsproblem in ein SAT-Problem zu transformieren und dann von der hohen Leistungsfähigkeit der SAT-Beweiser zu profitieren. Dieses Beweisverfahren wird Interval Property Checking (IPC) genannt.

Die verallgemeinerten Timingdiagramme legen eine Art Puzzlespiel nahe [Bormann/Spalinger 2001]: Wenn ein Eingabetrace gegeben ist, kann man versuchen, das Schaltungsverhalten nur aufgrund der Operationseigenschaften vorherzubestimmen. Wenn auf diese Weise zu jedem Eingabetrace ein Ausgabetrace eindeutig bestimmt wird, heißt die Menge der Operationseigenschaften vollständig. Sie untersucht dann jeden Aspekt der Schaltungsfunktion. Die Beschreibung eines Vollständigkeitsprüfers ist Teil dieser Arbeit. Ein vollständiger Satz von gegen die Schaltung bewiesenen Operationseigenschaften bildet eine äquivalente, abstraktere Schaltungsbeschreibung und die Basis für weitere Abstraktionen bis hin zu einer für formale Verifikation geeigneten Variante eines Transaktionsmodells. Damit nähert sich das Abstraktionsniveau der Schaltungsbeschreibung für formale Verifikation dem der Schaltungsbeschreibung für simulationsbasierte Verifikation an. Ziel der Verifikation ist eine vollständige Schaltungsbeschreibung. Es wird deswegen von vollständiger Verifikation gesprochen. Vollständige Verifikation stellt damit eine verstehende Qualitätssicherung entsprechend der Charakterisierung in Abschnitt 2.3 dar. Auftrittswahrscheinlichkeiten für Fehler, wie sie die Simulation behindern, sind hier kein Thema: Wenn ein Fehler beobachtbar ist, wird er auch gefunden.

Zur automatischen Feststellung der Vollständigkeit einer Menge von Operationseigenschaften wurde ein Vollständigkeitsprüfer entwickelt, der zum Einreichungszeitpunkt des entsprechenden Patents [Bormann/Busch 2005] der erste seiner Art war [Claessen 2006] und ein Kernproblem der formalen funktionalen Verifikation ([Katz et al. 1999, Hojati 2003, Hoskote 1999]) vollautomatisch löst: Wann hat man genügend viele Eigenschaften formuliert, um die Schaltungsfunktionalität vollständig zu erfassen? Kapitel 5 beschreibt diesen Vollständigkeitsprüfer, beweist seine Vorgehensweise und illustriert sie an einem Beispiel.

Der Vollständigkeitsprüfer untersucht Operationseigenschaften über einen Schaltungsteil, in dem die zugehörigen Operationen nacheinander durchgeführt werden. Es gibt darin also keine nebenläufigen Operationen. Ein solcher Schaltungsteil wird Cluster genannt. Größere Schaltungen enthalten im Allgemeinen nebenläufige Operationen. Solche Schaltungen werden zur Verifikation in mehrere Cluster zerlegt. Die Komposition vollständiger Verifikationen der Cluster zu einer vollständigen Verifikation der Gesamtschaltung ist nur möglich unter bestimmten Anforderungen an die Modellbildung und an sog. Integrationsannahmen, in denen Voraussetzungen und Zusicherungen über Verhalten am Interface eines Clusters aufgestellt werden. Diese Anforderungen werden in Kapitel 6 beschrieben, bewiesen und an Beispielen illustriert.

2.6 Vollständige Verifikation im Einsatz

Verifikationsprojekte mit vollständiger Verifikation und ähnlichen, weniger ausgereiften Vorgängeransätzen sind seit 1999 regelmäßig im produktiven Einsatz bei Siemens, Infineon und weiteren Halbleiterherstellern und Systemhäusern durchgeführt worden. Einen Überblick über eine Auswahl von Projekten gibt Tabelle 1. Fast alle in diesen Projekten verifizierten Schaltungsbeschreibungen funktionierten nach der Fabrikation des entsprechenden ASICs oder dem Brennen des entsprechenden FPGAs auf Anhieb fehlerfrei.

In den wenigen Projekten, in denen Schaltungsfehler übersehen wurden, ließ sich das Versehen auf menschliche Unzulänglichkeit bei der Anwendung der zu Grunde liegende Methodik zurückführen: Beispielsweise schränkten Constraints die Eingabetraces zu stark ein, oder durch fehlerhafte Beschreibungen des erwarteten Ausgabeverhaltens wurden Schaltungsfehler fälschlicherweise akzeptiert. Der Ansatz verwendet dabei ungeprüft nur Beschreibungen an den Modulgrenzen. Werden Zwischenergebnisse über Schaltungsinterna verwandt, ist es gerade eine Eigenart des Ansatzes, auf dem Beweis dieser Zwischenergebnisse zu bestehen. Dadurch wird die Wahrscheinlichkeit für Auswirkungen menschlicher Unzulänglichkeit auf die endgültige Schaltungsqualität minimiert.

Dem gegenüber wurden auch in gründlich per Simulation verifizierten Schaltungen durch vollständige Verifikation noch etliche Fehler gefunden und erlaubten so eine Bewertung der Schwächen der vorher eingesetzten Verifikationsmethodik.

Die hier beschriebene Vorgehensweise ist in der Lage, die volle Verifikationsaufgabe für ein Modul zu übernehmen. Vollständige Formale Verifikation ist also im Gegensatz zur ABFV keine Ergänzung zur simulationsbasierten Verifikation auf Modulebene, sondern eine Alternative. Im Extremfall lässt sich mit vollständiger Verifikation der Einsatz von Simulation auf die Exploration des Hardwareentwurfs reduzieren, wo sich etwa ein Designer per Simulation ein Gefühl für die Funktion der neu entworfenen Schaltung verschafft. Es steht aber einem Verifikationsverantwortlichen frei, wie der Mix zwischen Simulation und Vollständiger Formaler Verifikation gewählt wird. Wie bei der Qualitätssicherung in anderen Ingenieursdisziplinen kann aber nun eine passende Mischung aus beobachtender und verstehender Qualitätssicherung im Sinne des Absatzes 2.3 konfiguriert werden.

Damit eröffnen sich für die Vollständige formale Verifikation ganz andere Aufwandsbudgets als der ABFV. Im Rahmen dieser Aufwandsbudgets ist eine Einarbeitung in den Schaltungsentwurf möglich, mit einem günstigen Verhältnis von Aufwand und Nutzen. Diese Einarbeitung in den Schaltungsentwurf wird durch den Ansatz durch Strukturierung vereinfacht. Indem sich der Verifikationsingenieur auf eine Operation nach der anderen konzentriert, wird er/sie nicht gleichzeitig mit allen Problemen konfrontiert, wie das etwa bei einer Verifikation von High Level Assertions der Fall ist.

2.7 Simulationsbasierte und vollständige Verifikation

Wie gerade ausgeführt, ist vollständige Verifikation eine eigenständige Methode zur Schaltungsverifikation. Sie steht damit in Konkurrenz zu simulationsbasierten Verifikationsansätzen zur Modulverifikation, wie etwa Coverage Driven Random Pattern Simulation [Bergeron et al. 2005]. Dabei weist vollständige Verifikation eine Reihe attraktiver Eigenarten auf, die nachfolgend nur benannt werden sollen. Nach einer detaillierteren Darstellung der vollständigen Verifikation werden diese Vorteile in Abschnitt 3.6 genauer besprochen.

Vollständige Verifikation erreicht höchste Qualität für die damit verifizierten Module, während bei anderen Verifikationsverfahren Fehler im Design verbleiben, weil sie entweder nicht stimuliert wurden, oder weil sie von den Prüfmechanismen übersehen wurden.

Der Qualitätsgewinn betrifft nicht nur die Implementierung, sondern auch die Spezifikation, denn der Vollständigkeitsprüfer weist auch auf Spezifikationslücken hin. Die Konkurrenzverfahren haben keine Möglichkeit zur strukturierten Identifikation von Spezifikationslücken.

Zur vollständigen Verifikation wurde ein Prozess definiert, der bei der Planung eines Verifikationsprojekts beginnt und auch Anleitung dazu gibt, wie die Operationseigenschaften zu entwickeln sind. Gute Projektplanung führt dabei zu verlässlichen Bearbeitungszeiten. Die Qualität der Projektplanung beeinflusst nur die Termintreue des Projekts, aber nicht die Qualität der Schaltung nach der vollständigen Verifikation. Für simulationsbasierte Verfahren gibt es ebenfalls ausgearbeitete Prozesse. Verifikationsplanung erfordert hier die Erfassung aller Verifikationsziele. Die Qualität der Verifikationsplanung beeinflusst sowohl den Zeitplan als auch die durch die Simulationen zu erzielende Qualität.

Der Abschluss eines vollständigen Verifikationsprojekts wird erreicht, wenn ein logisches, d.h. kompromissloses Terminierungskriterium erreicht wird, das im Wesentlichen von dem Vollständigkeitsprüfer automatisch abgehakt wird. Simulationsbasierte Verifikationen werden aufgrund heuristischer Kriterien wie Coveragezahlen und Fehlerfindungskurven beendet.

Produktivität und Terminierungskriterium stehen miteinander in Beziehung. Unkundige Durchführung der vollständigen Verifikation oder die Anwendung auf schlecht durchdachten RTL-Code führt zu unter Umständen sehr langen Verifikationszeiten, während beim Abschluss einer Verifikation per Simulation immer auch eine Interpretationsaufgabe mitschwingt. Erfahrene Benutzer verifizieren ordentlich geschriebenen RTL-Code mit vollständiger Verifikation aber schneller als per Simulation, trotz des zusätzlichen Qualitätsgewinns. Eine grobe Schätzung liegt bei einer Produktivität von 2000 bis 4000 Zeilen RTL-Code, der in einem Personenmonat mit vollständiger Verifikation untersucht werden kann. In Einzelfällen wurden 8000 Zeilen pro Personenmonat erreicht.

Neben der ungewohnten Herangehensweise besteht die größte Akzeptanzhürde der vollständigen Verifikation in ihrer engeren Beziehung zur konkreten Implementierung. Die Erfahrung zeigt, dass diese Nähe nur ganz selten dazu führt, dass fehlerhafte Interpretationen der Spezifikation in den Verifikationscode übernommen werden und dadurch Fehler übersehen werden. Aber Implementierungsänderungen erfordern häufig auch Änderungen im Verifikationscode. Das ist bei inkrementellen Änderungen im Rahmen eines Entwurfs- und Verifikationsprojekts kein Problem. Bei starken Änderungen können aber erhebliche Aufwände entstehen, während bei simulationsbasierter Verifikation Aufwand gespart werden kann durch Wiederverwendung der Testbenches. Bei dieser Art von Aufwandsvergleichen muss allerdings auch beachtet werden, dass die Wiederverwendung einer Testbench auch mit der Anpassung der Coveragekriterien und der Stimuli einher gehen müsste, wenn die neue Schaltung nach der Verifikation die gleiche Qualität haben soll wie die alte. Dieser Anpassungsaufwand wird aber häufig eingespart.

2.8 Vollständige Verifikation und ABFV

Der fundamentale Unterschied zwischen ABFV und vollständiger Verifikation ergibt sich aus ihrer Rolle in einem Verifikationsprojekt: ABFV unterstützt eine Verifikation, die hauptsächlich per Simulation durchgeführt wird, während vollständige Verifikation die gesamte Aufgabe übernehmen kann.

Beiden gemeinsam ist aber das zentrale Benutzungsprinzip: Assertions bzw. Eigenschaften werden aufgestellt, sie werden mit einem automatischen Beweiswerkzeug gegen die Schaltung verifiziert, das Beweiswerkzeug konstruiert ein Gegenbeispiel, dieses wird vom Benutzer analysiert und die Analyse führt zur Erkennung eines Schaltungsfehlers oder zur Anpassung der Beweisaufgabe.

ABFV macht dabei keine Vorgaben über die Gestalt der Assertions, während vollständige Verifikation eine sehr klare Vorstellung davon vermittelt, welche Struktur die Eigenschaften haben und wie sie aufgestellt werden sollten. Die mit ABFV zu erzielende Schaltungsqualität ist damit in stärkerem Maße von Kundigkeit und Einfallsreichtum des Verifikateurs abhängig, aber dieser kann schon mit sehr wenig Ausbildung bereits Assertions schreiben und beweisen. Für jeden Schaltungstyp und meist auch für jede Implementierungsstrategie wird aber ein eigener Satz von Assertions entwickelt werden, ohne dass es eine übergeordnete Idee gibt, wie die Assertions aussehen sollten. Entsprechend gibt es viel Literatur darüber, die Schaltungsklassen fallweise besonders hilfreiche Assertions zuweisen. Bei der vollständigen Verifikation ist der methodische Unterbau breiter, und die Verifikationsaufgaben ergeben sich daraus auf natürliche Weise.

Weil Assertions sehr vielfältig sein können, werden universelle Verifikationswerkzeuge dafür angeboten. Assertions können von sehr unterschiedlicher Komplexität sein. Dabei spielt die Schaltung selbst eine große Rolle, sodass auch syntaktisch ähnliche Assertions zu unterschiedlichem Resourcenbedarf der Beweiser führen. Die Benutzer gewöhnen sich entweder daran, dass der Beweiser an manchen Assertions ohne erkennbaren Grund scheitert, oder sie entwickeln ein Gefühl für den Resourcenbedarf des Beweisers und legen ihm nur noch geeignete Assertions vor. Ungeeignete Assertions können per Simulation weiter untersucht werden.

Die Operationseigenschaften der vollständigen Verifikation stellen recht hohe Komplexitätsanforderungen. Diese können mit IPC vorteilhaft behandelt werden. Die Beweiser scheitern nur selten. Wenn doch, muss die betreffende Operation in kleinere Operationen aufgeteilt werden.

Assertions spielen auch innerhalb der vollständigen Verifikation eine wichtige Rolle, z.B. als Teil einer Spezifikation. Sie kommen aber auch ins Spiel, um Zwischenergebnisse zusammenzufassen. Einige Assertions werden einfach mit den Mitteln der ABFV verifiziert. Wo dies jedoch scheitert, können Assertions auch bewiesen werden, indem gezeigt wird, dass sie durch alle Operationen einer vollständigen Menge von Operationseigenschaften erfüllt werden.

Tabelle 1: Projekte mit vollständiger formaler Verifikation

Funktion	Projekt
Prozessoren	TriCore2 (superskalar, 32 Bit, Automobiltechnik) [Bormann et al. 2007] Multithreaded network processor [PPv2 2008] IEEE floating point processor Schwach programmierbare IP [Loitz et al. 2008]
Peripherals	Infrarot-Schnittstelle One-Wire Interface Touch Screen Measurement Interface USB Master Interface, Zähler UART Interrupt Controller A/D Converter Controller Flash Card Data Port Camera Interface Multimedia Card Interface konfigurierbarer Arbiter DMA Controller
Bus Interfaces	Bus Arbiter AHB (Master-, Slaveinterfaces, Brücken, Busmatrix [Bormann/Blank/Winkelmann 2005]) Interfaces mit proprietären Protokollen Protokollanpassung eines Legacy-Prozessors [Bormann/Spalinger 2001] CAN, LIN, Flex Ray, AXI SRC Audiobus Interface Kommunikationsinfrastruktur eines massiv parallelen Multiprozessorsystems HDLC Controller [Bormann/Blank/Winkelmann 2005]
Memory Controllers	SDRAM Controller Advanced Memory Bus SATA Caches Flash Speicher Interface
Error Correction	ECC Störungstoleranz bei Board-zu-Board-Kommunikation
Telecom	AAL2 Termination Element Addressverwaltung in ATM Switch Sonet / SDH Frame Alignment [Thomas et al. 2004] Path Overhead Behandlung eines Multi-Gigabit-Switch DSP Koprozessor-ASIC zur Korrelationsberechnung [Winkelmann et al. 2004]

3 Überblick über die vollständige Verifikation

Eine Besonderheit des in Abschnitt 2.5 skizzierten Ansatzes der vollständigen formalen Verifikation ist seine Ähnlichkeit zu transaktionsbasierter Verifikation, d.h. dem heute üblichen simulationsbasierten Verifikationsansatz [Bergeron et al. 2005]. Mit Bezug auf diese Ähnlichkeit wird nun dieser Ansatz aus Anwendungssicht vorgestellt.

Ein kurzer Abriss transaktionsbasierter Verifikation per Simulation wird in Abschnitt 3.1 gegeben. Die vollständige formale Verifikation ist eine Symbiose aus der in Abschnitt 3.2 beschriebenen operationsbasierten Schaltungsdarstellung, dem Interval Property Checking genannten Beweisverfahren aus Abschnitt 3.3 und dem Vollständigkeitsprüfer aus Abschnitt 3.4, die durch den kompositionalen Ansatz aus Abschnitt 3.5 zu einem Verfahren ergänzt wird, das prinzipiell keinen Größenbeschränkungen unterliegt. Die Vor- und Nachteile der praktischen Anwendung der vollständigen Verifikation werden im Abschnitt 3.6 diskutiert.

3.1 Transaktionsbasierte Verifikation durch Simulation

3.1.1 Transaktionen

RTL beschreibt Berechnungsschritte und die Interaktion dieser Berechnungsschritte. Jeder Operator in einer RTL-Beschreibung ist ein Berechnungsschritt, und durch Kontrollstatements wird dargestellt, wie diese Berechnungsschritte zusammenwirken.

Funktionale Verifikation stellt fest, ob diese einzelnen Berechnungsschritte im RTL so zusammenwirken, wie es eine Spezifikation vorsieht. In dieser Beziehung entspricht die funktionale Verifikation zahlreichen anderen Verifikationsaufgaben: Beim Entwurf von Zellen aus Transistoren wird etwa verifiziert, ob das Zusammenwirken der Transistoren die gewünschte Funktion der Zelle implementiert. Bei der Verifikation von Netzlisten gegen RTL wird nachgewiesen, dass das Zusammenwirken vieler solcher Zellen die einzelnen gröberen Berechnungsschritte des RTL (wie etwa eine Multiplikation) implementieren.

Wenn Simulation zur funktionalen Verifikation eines Moduls eingesetzt wird, wird vor allem das Verhalten der Interfacesignale des Moduls betrachtet, also derjenigen Signale, die die Module verbinden und z.B. Lese- oder Schreibanforderungen oder Bestätigungen signalisieren. Dieses Verhalten wirkt auf den ersten Blick unübersichtlich. Bei genauerem Hinsehen stellt sich aber heraus, dass dieses Verhalten wenigen Grundmustern genügt. Diese Grundmuster erhält man, wenn die Menge der Interfacesignale entsprechend ihrer Zughörigkeit zu Bussen gruppiert wird und die einzelnen Busse separat betrachtet werden. Die Grundmuster werden dann Transaktionen dieser Busse genannt. Die Transaktionen beschreiben die Anforderung von Aktivität des Moduls bzw. zugelieferte Leistungen anderer Module.

Welche Transaktionen es gibt, wie sie aufeinanderfolgen dürfen und wie sie durch Signalverläufe auf den Signalen eines Busses implementiert werden, wird von einer Protokollspezifikation festgelegt. Das Konzept der Transaktionen ist für die heutige simulationsbasierte funktionale Verifikation fundamental.

Transaktionen haben Parameter. Unverzichtbare Parameter sind Daten, Adressen oder die Kommunikationsrichtung. Weitere Parameter können über den Simulationszeitpunkt Auskunft geben, an dem die Transaktion gestartet oder beendet wurde und über die Zeitdauer zwischen den im Protokoll vorgesehenen Synchronisationsereignissen. Darüber hinaus mag es protokollspezifische Parameter geben wie etwa den Typ einer Basistransaktion des AHB-Protokolls [AHB 1999].

Der Begriff der Transaktion ist nicht eindeutig [Cai/Gajski 2003], vielmehr gibt es eine Hierarchie von Transaktionen. Beim AHB-Protokoll gibt es etwa die Basistransaktion, die aber ihrerseits in Adress- und Datenphase zerfällt, der – je nach Betrachtungsweise – auch noch eine Arbitrierungsphase vorangeht. Diese Phasen sind Transaktionen auf einem niedrigen Abstraktionsniveau. Auf einem hohen Abstraktionsniveau verbinden sich mehrere AHB-Basistransaktionen zu Bursts.

3.1.2 Testbenches

Die Testbench einer simulationsbasierten funktionalen Verifikation enthält all denjenigen Code, der zur Verifikation der gegebenen Schaltung entwickelt wurde. Abbildung 1 zeigt eine für transaktionsbasierte Verifikation typische Testbench, die nachfolgend beschrieben wird.

Gerichtete oder zufällige Tests erzeugen auf den Bussen der Schaltung Folgen von Signalwerten (Waveforms). Die Waveforms aller Signale des Moduls werden teilweise mit Assertions untersucht, doch die Testbench untersucht im Wesentlichen die Interfacesignale des Moduls. Deren Waveforms werden durch sog. Transaktionsextraktoren umgeformt in Folgen von Transaktionen, die jeweils durch ihren Parametersatz beschrieben werden. Während der Extraktion prüfen die Transaktionsextraktoren auch die Übereinstimmung der Folge von Signalwerten mit der Protokollspezifikation, ob z.B. Schreibdaten auch wirklich stabil gehalten werden, bis das Schreiben bestätigt wurde. Die Prüfung der Modulfunktionalität wird dann auf die Untersuchung der Transaktionen reduziert. Dabei wird z.B. festgestellt, ob die Parameter der Transaktionen richtig sind, oder ob die Reihenfolge der Transaktionen stimmt.

Es wird zwischen eingehenden und ausgehenden Transaktionen des Moduls unterschieden. Eingehende Transaktionen sind solche, bei denen ein Nachbarmodul die Initiative zu einem Datentransfer ergriffen hat. Ausgehende Transaktionen sind solche, bei denen das betrachtete Modul die Initiative zu einem Datentransfer ergriffen hat. Wenn ein Businterface eines Moduls eingehende Transaktionen erhält, heißt es Slave- oder Target-Interface, wenn es ausgehende Transaktionen produziert, heißt es Master- oder Initiator-Interface.

Die Begriffe "eingehende" und "ausgehende Transaktion" beschreiben nicht die Richtung des Datentransfers. Bei Schreibtransaktionen fließen die Daten vom Master- zum Slave-Interface, und bei Lesetransaktionen vom Slave- zum Masterinterface. Module können nur Slave-Interfaces haben, wie z.B. Speichermodule, oder nur Masterinterfaces haben, wie z.B. Prozessoren, oder sie haben beide Sorten von Interfaces, wie z.B. Busbrücken.

In einer typischen transaktionsbasierten Simulationsverifikation werden die eingehenden Transaktionen von einem transaktionsbasierten Referenzmodell weiterverarbeitet, das die Daten zu eingehenden Lesetransaktionen bereitstellt und eine Folge von ausgehenden Transaktionen erzeugt. Ein solches auf Transaktionen operierendes Modell der zu untersuchenden Schaltung heißt Transaction Level Model (TLM) der Schaltung.

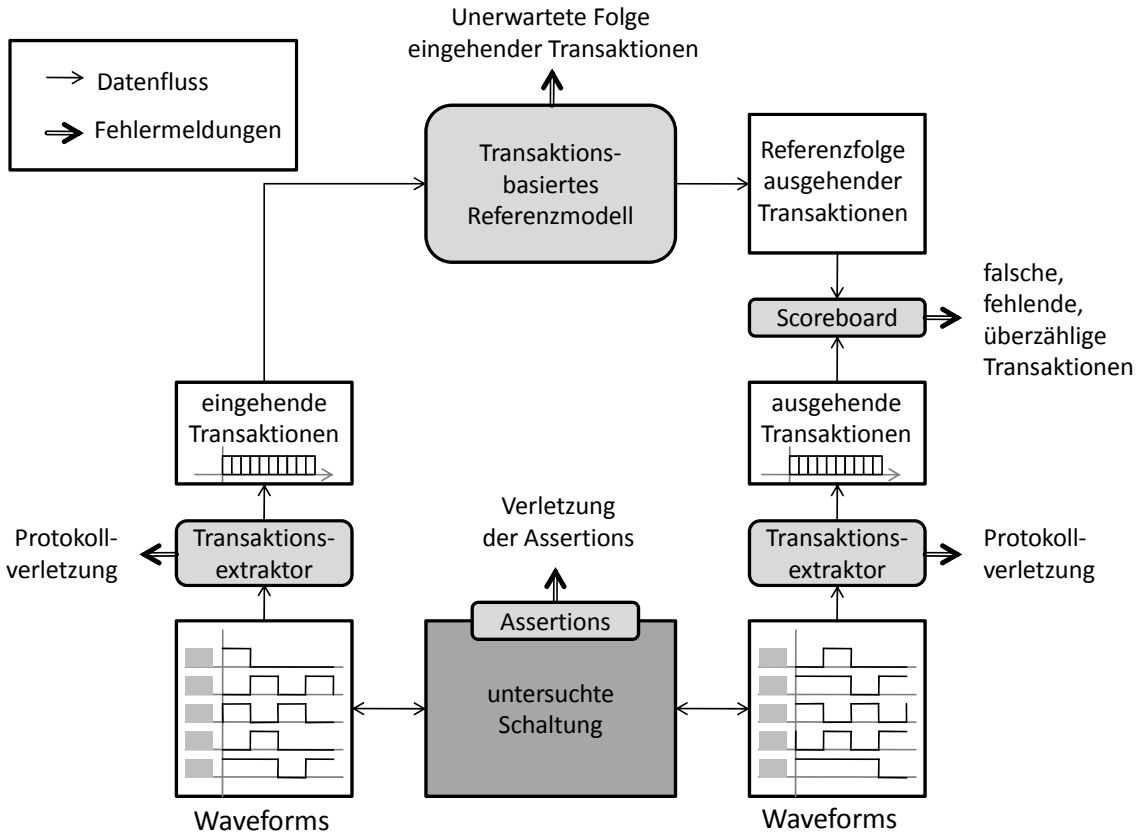

Abbildung 1: Transaktionsbasierte Testbench für Simulation

Die vom TLM erzeugte Folge von ausgehenden Transaktionen muss nicht exakt mit der Folge der ausgehenden Transaktionen übereinstimmen, die die Transaktionsextraktoren aus den Signalverläufen am Interface des untersuchten Moduls extrahieren. Aber es muss eine der Verifikationsaufgabe angepasste Übereinstimmung bestehen. Testbenchkomponenten, die diese Übereinstimmung überprüfen, heißen Scoreboards. Diese vergleichen die Folge von Transaktionen der Implementierung mit den Transaktionen des TLM und verifizieren so das Modul. Ein Scoreboard kann z.B. tolerant gegenüber lokalen Umordnungen der Folge von Transaktionen sein, meldet aber einen Fehler, wenn das Modul falsche oder überzählige Transaktionen erzeugt.

Die in Abbildung 1 skizzierte Testbench ist besonders geeignet zur Prüfung eines Moduls innerhalb einer Systemsimulation, in der das Modul von Nachbarmodulen angeregt wird. In diesem Fall extrahieren die Transaktionsextraktoren tatsächlich die Transaktionen aus dem Signalverhalten. Zur reinen Verifikation eines isolierten Moduls, z.B. eines IP-Bausteins, muss die Testbench aber auch die eingehenden Transaktionen generieren. In diesem Fall werden zumindest die Generatoren auf der Masterseite der zu untersuchenden Schaltung abstrakt beschrieben. Sie bestehen entweder aus Leseroutinen, die z.B. ein Textfile mit den Parametern der auszuführenden Transaktionen einlesen und an Transaktionsextraktor und TLM weitergeben. Die Generatoren können aber auch Zufallsgeneratoren sein, die zunächst die Parameter der auszuführenden Transaktionen zufällig bestimmen. Statt der Transaktionsextraktoren werden dann Testbenchkomponenten eingesetzt, die aus den Parametern das Signalverhalten generieren.

Mit Coveragemessungen wird festgestellt, wie intensiv die Schaltung durch die Verifikation untersucht wird, d.h. wie viele Betriebssituationen durchsimuliert wurden. Coverage wird einerseits auf dem RTL der zu untersuchenden Schaltung erhoben. Andererseits kann gerade funktionale Coverage, d.h. die Simulation bestimmter funktionaler Abläufe häufig von den Parametern der Transaktionen abgeleitet werden, sodass die Coveragemessung dasjenige Abstraktionsniveau nutzt, das für die gegebene Coveragebedingung besonders geeignet ist.

3.2 Abstraktionsebenen der vollständigen Verifikation

In diesem Abschnitt wird der Ansatz der vollständigen formalen Verifikation aus Anwendungssicht vorgestellt. Abbildung 2 zeigt die damit verbundenen Abstraktionsebenen und die durch Verfeinerungen gegebenen Beziehungen zwischen diesen Abstraktionsebenen.
Die Rolle des transaktionsbasierten Referenzmodells in der Simulation wird von einem Transaktionsautomaten übernommen, der in Abschnitt 3.2.2 eingeführt wird. Die Übergänge des Transaktionsautomaten werden durch eingehende Transaktionen ausgelöst und sie beschreiben, welche ausgehenden Transaktionen produziert werden. Der Transaktionsautomat modelliert nicht, wie die ein- oder ausgehenden Transaktionen durch Signale und Takte implementiert werden.

Eine Verfeinerung des Transaktionsautomaten repräsentiert die Transaktionen durch ihr Signalspiel. Der so verfeinerte Automat heißt Operationsautomat und entspricht dem Zusammenwirken der Transaktionsextraktoren, eines speziellen Scoreboards und des TLM-Referenzmodells. Der Operationsautomat wird in Abschnitt 3.2.3 vorgestellt.

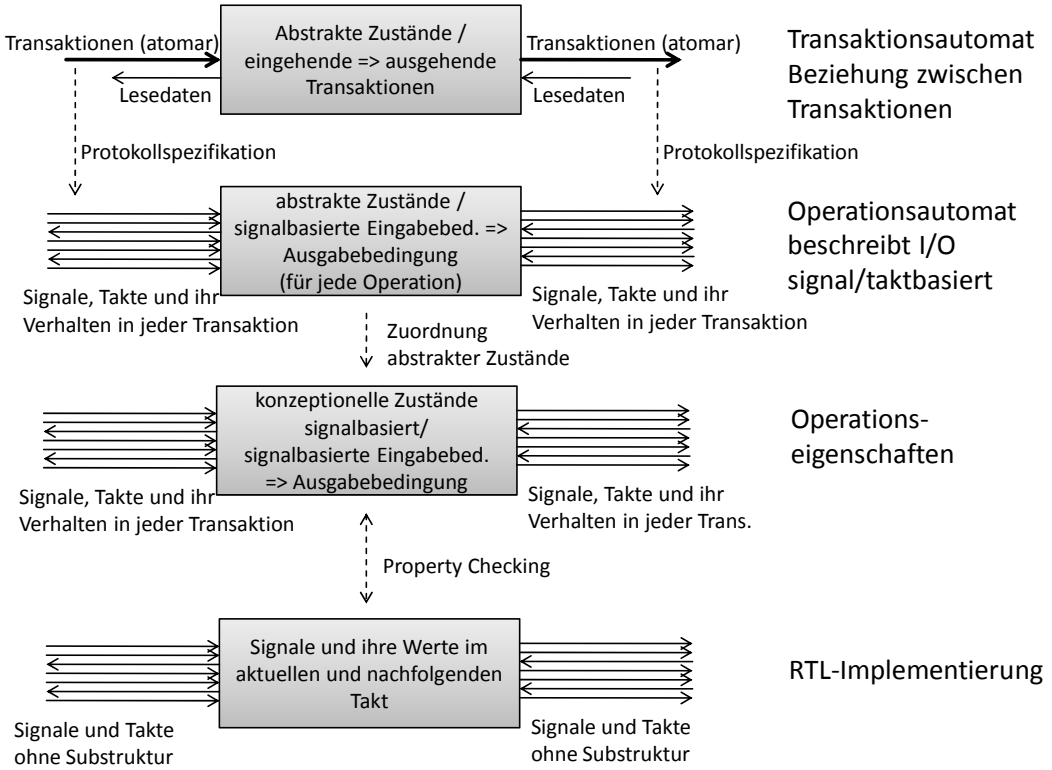

Abbildung 2: Abstraktionsebenen und ihre Beziehungen

Eine Schaltung wird verifiziert, indem nachgewiesen wird, dass ihr Operationsautomat die Spezifikation erfüllt und das gleiche Ein-/Ausgabeverhalten hat wie das zu prüfende Modul. Dazu ist eine Zuordnung der Zustände der zu untersuchenden Schaltung zu den Zuständen des Operationsautomaten erforderlich, die dem Zustandsmapping der kombinatorischen Äquivalenzverifikation ähnelt. Durch das Zustandsmapping entstehen Operationseigenschaften, über die in Abschnitt 3.2.4 gesprochen wird. Die Abstraktionen werden in Abschnitt 3.2.6 mit anderen im Schaltungsentwurf und in der formalen Verifikation üblichen Abstraktionstechniken verglichen.

Ihr Beweis mit IPC wird in Abschnitt 3.3 dargestellt und ebenso die mit dem IPC-Einsatz verbundenen Erreichbarkeitsbedingungen aus Anwendersicht.

Bei der Ausführung der vollständigen Verifikation in der Praxis konzentriert sich der Verifikateur zu einem Zeitpunkt immer nur auf die Verifikation einer Operation. Technologie und Theorie zur Beantwortung der Frage, ob die gesamte Schaltungsfunktion untersucht wurde, wird in Abschnitt 3.4 vorgestellt. Dies betrifft einerseits Schaltungsteile, die Operationen nacheinander ausführen, Cluster genannt werden, und deren Verifikation mit dem Vollständigkeitsprüfer aus Abschnitt 3.4.3 ihrerseits auf Verifikationslücken geprüft wird. Es betrifft aber auch die Zusammenschaltung von Clustern zu nebenläufigen Funktionalitäten, über die in Abschnitt 3.5 gesprochen wird.

3.2.1 Beispiel

Die im folgenden einzuführenden Konzepte sollen beispielhaft anhand einer Speicherschnittstelle illustriert werden. Wie Abbildung 3 zeigt, vermittelt die Schnittstelle den Datentransport zwischen einem Prozessor und einem SDRAM. SDRAMs sind Speicherblöcke, deren Speicherstellen man sich als in einer Matrix angeordnet vorstellen soll. Entsprechend wird von einer Zeilen- und einer Spaltenadresse gesprochen. Die Spaltenadresse wird üblicherweise durch die obere Hälfte der Leitungen des Adressbusses gegeben, die Zeilenadresse durch die untere Hälfte.

Die Steuereingänge eines SDRAMs heißen üblicherweise cs_n, we_n, ras_n und cas_n und erinnern an Namen wie Chip Select oder Write Enable. Ihre Funktionsweise ist aber besser zu verstehen, wenn die Gesamtheit der vier Signale als Kommandobus begriffen wird, über den das SDRAM Befehle erhält. Ein solche Befehl ist beispielsweise "row activate", mit dem eine Zeile aktiviert wird, was durch die Bedingung

```
cs_n = 0 and ras_n = 0 and cas_n = 1 and we_n = 1
```

über die Signale des Kommandobusses kodiert ist. Entsprechend gibt es Lese- und Schreibbefehle, die die Übertragung von Datenbursts veranlassen, einen Stop-Befehl, mit dem die Übertragung eines Bursts beendet wird, ein Precharge-Befehl mit dem die Aktivierung einer Zeile beendet wird, d.h. die Zeile geschlossen wird und ein NOP-Befehl, bei dem gar nichts passieren soll.

Auslesen oder Speichern eines Datums wird vom Prozessor durch die Aktivierung des Request-Signals req und einen passenden Wert auf dem rw-Signal verlangt. Die Adresse findet sich dann auf dem Signal address und die Schreib- bzw. Lesedaten auf dem Signal rdata bzw. wdata. Der Prozessor muss die von ihm getriebenen Signale stabil halten, bis sie von der Speicherschnittstelle durch Aktivierung des ready-Signals quittiert werden. Diese Aktivierung validiert auch die Schreibdaten.

Zum Auslesen oder Speichern sind normalerweise mehrere Befehle an das SDRAM vonnöten, die nacheinander eine Zeile der Speichermatrix aktivieren, dann mit der Spaltenadresse

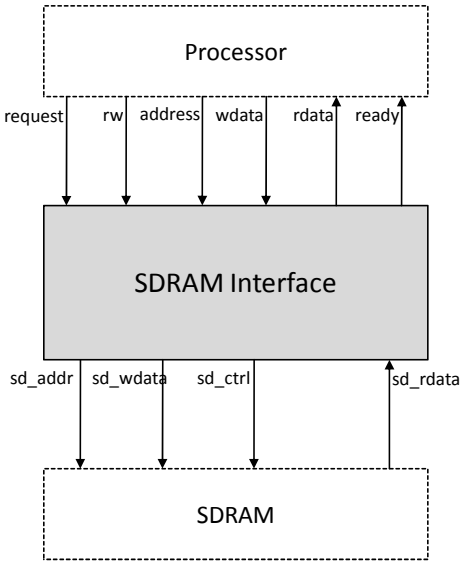

Abbildung 3: SDRAM Interface und seine Systemintegration

auf die Speicherstelle lesend oder schreibend zugreifen und zum Schluss die Speicherzeile schließen. Die entsprechenden Befehle des Kommandobusses heißen "row activate", "read" oder "write", und "precharge". Die Kommandos werden durch Werte auf der Adressleitung sdram_addr oder der Schreibdatenleitung sdram_wdata begleitet, oder sie führen dazu, dass das SDRAM eine gewisse Zeit nach dem Lesebefehl gültige Werte auf der Lesedatenleitung sdram_rdata einstellt. Die einzelnen SDRAM-Befehle müssen gewisse, speichertypabhängige Zeitabstände voneinander einhalten, die durch NOP-Befehle eingestellt werden. Nach der Aktivierung einer Zeile können mehrere Schreib- und Leseoperationen auf die Elemente dieser Zeile durchgeführt werden, ohne dass dazu diese Zeile geschlossen oder erneut aktiviert werden muss.

3.2.2 Transaktionsautomat

Der Ausgangspunkt der vollständigen Verifikation ist eine transaktionsbasierte Automatendarstellung der Spezifikation, die hier Transaktionsautomat genannt wird. Diese ist dem TLM-Referenzmodell der simulationsbasierten Verifikation recht ähnlich. Der Automat bearbeitet eingehende Transaktionen und generiert dazu Folgen von ausgehenden Transaktionen. Unterschiede in den Abstraktionsniveaus von TLM-Referenzmodellen und Transaktionsautomaten werden in Abschnitt 3.2.2 diskutiert.

Die Zustände des Transaktionsautomaten werden "abstrakte konzeptionelle Zustände" genannt. Analog der Aufteilung von Schaltungen in Kontroll- und Datenanteil bestehen die abstrakten konzeptionellen Zustände des Transaktionsautomaten aus einem Kontrollanteil und einem Datenanteil. Der Kontrollanteil heißt abstrakter wichtiger Zustand des Transaktionsautomaten. Der Datenanteil heißt sichtbarer Zustand, weil die in der Schaltung gespeicherten Daten in dieser Form in der Spezifikation sichtbar sind. Die Variablen, in denen die sichtbaren Zustände gespeichert werden, heißen sichtbare Register.

Abbildung 4 zeigt den Transaktionsautomaten des SDRAM Interfaces. Es gibt zwei wichtige Zustände, nämlich IDLE und ROW_ACT. Letzterer trägt der Tatsache Rechnung, dass bereits eine Speicherzeile aktiviert ist, und nachfolgende Zugriffsoperationen auf die gleiche Spei-

cherzeile deshalb schneller durchgeführt werden können. Im Zustand ROWACT gibt es auch einen sichtbaren Zustand actrow, der Auskunft über die gerade geöffnete Speicherzeile gibt. Im Idle-Zustand ist diese Angabe bedeutungslos.

Allgemein führt ein Transaktionsautomat abhängig von dem abstrakten konzeptionellen Zustand und – falls das Modul Slaveinterfaces besitzt – von eingehenden Transaktionen einen Zustandswechsel durch. Während dieses Zustandswechsels bestimmt der Automat die Lesedaten für eingehende Lesetransaktionen. Ferner generiert der Automat ausgehende Transaktionen, sofern das Modul über Masterinterfaces verfügt. Der Zeitbegriff eines Transaktionsautomaten ist demnach ereignisgetrieben.

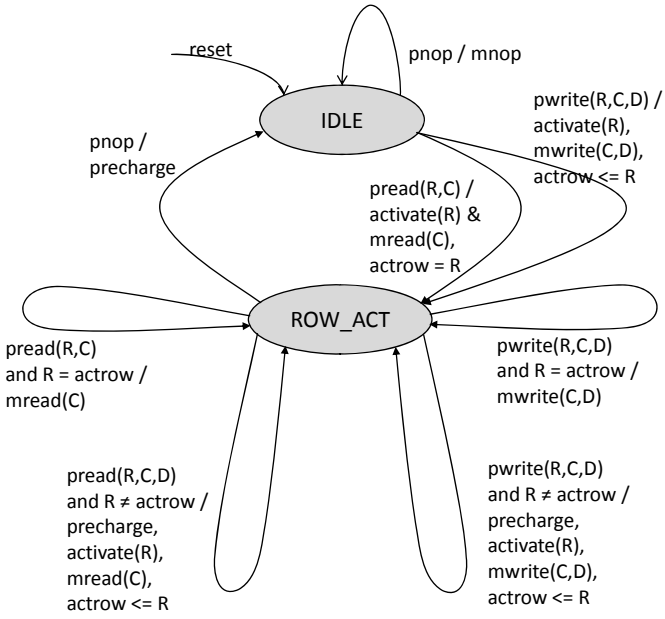

Abbildung 4: Transaktionsautomat des SDRAM Interfaces

Wenn ein Transaktionsautomat als Automatengraph dargestellt werden soll, wird die für Schaltungen mit Kontroll- und Datenanteil übliche Repräsentation genommen: Knoten des Transaktionsgraphen sind nur die wichtigen Zustände und das Verhalten des Datenpfades wird durch geeignete Ausdrücke repräsentiert, die entweder Teil der Bedingungen sind, unter denen eine Transition durchgeführt wird oder die Daten oder Adressen bestimmen, die bei der Transition von der Schaltung ausgegeben werden. Der Rest der Darstellung wird mit dieser vergröberten Sichtweise arbeiten, in der der Transaktionsautomat häufig relativ wenige wichtige Zustände und wenige Transitionen enthält.

Der Graph des Transaktionsautomaten des SDRAM Interfaces ist in Abbildung 4 dargestellt. Bei der Darstellung des Transaktionsautomaten wurde davon ausgegangen, dass der Prozessorbus die Transaktionen pread(R,C), pwrite(R,C,D) und pnop kennt, die in beliebiger Reihenfolge auftreten dürfen. R steht für die Zeilen-, C für die Spaltenadresse des Speichers und D für die Schreibdaten. Der Schreibvereinfachung wegen bleiben die Lesedaten unerwähnt. Der Speicherbus zwischen dem Interface und dem SDRAM beherrscht die Transaktionen mnop, activate(R), mwrite(C,D), mread(C), precharge, die die jeweiligen SDRAM-Befehle und die zugehörigen Zeitbedigungen enthalten sollen. Die einzelne Transaktion wird durch einen Ausdruck der Form <Bedingung> "/" <Aktion> charakterisiert. Die <Bedingung> muss dabei erfüllt sein, damit die Transition ausgeführt wird, und die Ausführung der Transaktion

bewirkt die unter <Aktion> beschriebenen Aktivitäten. Die angegebenen Transaktionen werden nacheinander versandt, und den sichtbaren Registern wird der neue sichtbare Zustand zugewiesen.

In Abbildung 4 gibt es etwa eine Transition, die durchgeführt wird, wenn das SDRAM Interface im abstrakten konzeptionellen Zustand ROW_ACT ist, wenn auf dem Prozessorbus eine pwrite(R, C, D)-Transaktion eingestellt ist, und wenn die Zeilenadresse R mit der Adresse actrow übereinstimmt, die als Teil des sichtbaren Zustands die Adresse der aktivierten Speicherstelle angibt. In diesem Fall gibt das SDRAM-Interface eine Transaktion mwrite(R, D) an das SDRAM weiter und bleibt in dem abstrakten konzeptionellen Zustand ROW_ACT, ohne den sichtbaren Zustand actrow zu verändern.

Mit jeder Transition des Transaktionsautomaten ist eine Verifikationsaufgabe verbunden: Über das zu untersuchende Modul ist nachzuweisen, dass es ausgehend von der Implementierung des wichtigen Zustand bei Eintreffen der eingehenden Transaktion unter Berücksichtigung des sichtbaren Zustands die richtige ausgehende Transaktion und die richtigen Daten erzeugt, sich in den entsprechenden neuen wichtigen Zustand begibt und den sichtbaren Zustand richtig anpasst.

3.2.3 Verfeinerung zum Operationsautomat

Um diese Beweisaufgaben untersuchen zu können, müssen die Zusammenhänge zwischen dem Transaktionsautomaten und dem Ein-/Ausgabeverhalten der zu untersuchenden Schaltung geklärt werden: Dazu müssen die vorher als atomar betrachteten Transaktionen durch das Verhalten der Ein- und Ausgabesignale des zu untersuchenden Moduls ausgedrückt werden. Die wesentliche Information hierzu kommt von den Spezifikationen der Protokolle der Busse an den Slave- und Masterinterfaces. Diese Spezifikationen legen fest, welches Verhalten seiner Eingabesignale das Modul von seiner Umgebung erwarten darf und wie sich die Ausgabesignale des Moduls im Zusammenhang damit verhalten müssen. Diesem Verfeinerungsschritt entspricht in der simulationsbasierten Verifikation die Ergänzung des TLM-Referenzmodells durch die Transaktionsextraktoren.

Nachdem die Protokollspezifikation eingebracht wurde, ist die Beschreibung zunächst noch unabhängig von einer konkreten Implementierung und legt nicht fest, wann das zu untersuchende Modul die Synchronisationsereignisse produziert, die es der Protokollspezifikation zufolge erzeugen soll, und welche zeitliche Beziehung zwischen eingehenden und ausgehenden Transaktionen herrschen sollen. Solche Freiheitsgrade werden in der simulationsbasierten Verifikation durch das Scoreboard ausgedrückt, das die möglichen zeitlichen Beziehungen der Spezifikation akzeptiert und andere zurückweist. Für formale Verifikation lassen sich diese Freiheitsgrade nur schwer ausdrücken. Der hier vorgestellte Ansatz geht daher einen etwas anderen Weg: Er verlangt, dass einzig die durch das RTL implementierte zeitliche Beziehung eingestellt wird.

Nach Einbringen dieser Informationen entsteht aus dem Transaktionsautomat ein verfeinerter Automat, der Operationsautomat genannt wird. Seine Zustände sind nach wie vor die abstrakten konzeptionellen Zustände des Transaktionsautomaten. Der Operationsautomat beschreibt aber das Ein-/Ausgabeverhalten der Schaltung präzise für Signale und Takte. Die Transitionen des verfeinerten Automaten werden aber nun ausgelöst, wenn Bedingungen an die konkreten Eingangssignale der Schaltung erfüllt sind, und sie erzeugen das entsprechende Verhalten auf den Ausgabesignalen, das wiederum durch eine Bedingung beschrieben wird. Beide Bedingungen sind im Allgemeinen sequentiell und werden sich im allgemeinen zeitlich überlappen.

Eine Transition des Operationsautomaten wird demnach durch einen abstrakten konzeptionellen Anfangs- und Endzustand und durch Bedingungen über die konkreten Eingangs- und Ausgangssignale charakterisiert. Sie soll als Operation bezeichnet werden.

Die Verfeinerung legt weiterhin die zeitliche Relation aufeinanderfolgender Operationen fest. Dazu wird in jeder Operation ein symbolischer Anfangszeitpunkt t festgelegt und ein symbolischer Referenzzeitpunkt T_{ref}, zu dem alle Nachfolgeoperationen beginnen. Der Referenzzeitpunkt einer Operation wird bestimmt durch einen zeitlichen Versatz gegenüber dem Anfangszeitpunkt t, und dieser zeitliche Versatz kann sowohl vom sichtbaren Zustand zu Beginn der Operation, als auch vom Verhalten der Eingabesignale abhängen.

Im Kontext von Abbildung 1 vereinigt der Operationsautomat die Aufgaben des Transaction Level Model, des Scoreboards und der Transaktionsextraktoren bei der Verifikation. Der Operationsautomat könnte z.B. per Simulation parallel mit dem zu verifizierenden Modul ausgeführt werden. Dann würde die Verifikation in der Prüfung bestehen, ob die Ausgaben des zu verifizierenden Moduls die Bedingungen an die Ausgabesignale erfüllen, die der Operationsautomat verlangt.

Die Verfeinerung des Transaktionsautomaten kann so exakt gewählt werden, dass der Operationsautomat jedem Eingabetrace genau einen Ausgabetrace zuordnet. In diesem Fall prüft die Verifikation die Äquivalenz zwischen dem Operationsautomaten und der zu verifizierenden Schaltung.

Abbildung 5 präsentiert die Operation, die sich aus der Transition des Transaktionsautomaten unten links in Abbildung 4 ergibt. Als Prozessorbusprotokoll dient das oben schon eingeführte einfache Request-Ready-Protokoll, bei dem nach der Aktivierung des $request$-Signals alle Angaben stabil gehalten werden müssen, bis das SDRAM mit einem $ready$-Puls den Zugriff quittiert und gleichzeitig die Werte auf dem $rdata$-Signal validiert. Auf dem SDRAM-Bus

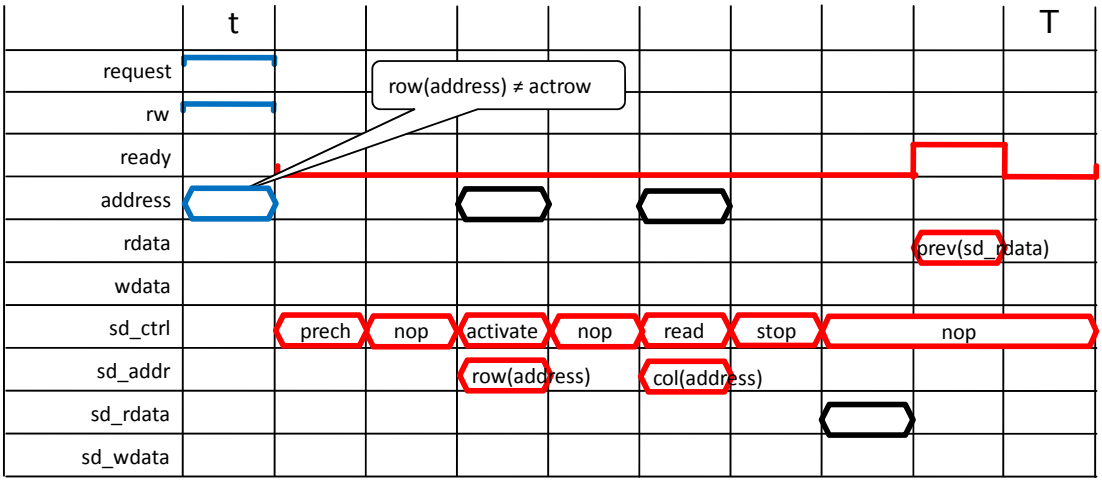

Abbildung 5: Operation Lesen mit Wechsel der Speicherzeile

wird je ein Precharge-, Activate-, Lese- und Stopkommando unter den erforderlichen Zeitbedingungen abgesetzt. Das Stopkommando ist erforderlich, weil das Read-Kommando sonst eine Burstoperation anfordern würde.

In der Operationsdarstellung von Abbildung 5 ist die Eingabebedingung der Operation blau gezeichnet, sie enthält den Vergleich zwischen der Zeilenadresse der Transaktion und dem sichtbaren Zustand. Die Beschreibung der Aktivitäten auf den Ausgabesignalen ist rot gezeichnet. Die schwarze Markierung bezeichnet Signalwerte, die an anderer Stelle in Ausdrücken referenziert werden und bezeichnet selbst keine Annahme oder Forderung.

In dieser Beschreibung wird auch über die Ausgabe der Lesedaten an den Prozessor auf dem Ausgabesignal $rdata$ Auskunft gegeben. An den schwarz markierten Werten des $address$-Signals wird deutlich, dass die Operation die Stabilität der Prozessorbussignale bis zum Ready-Puls ausnutzt. Ferner enthält die Beschreibung die Festlegung des Referenzzeitpunkts T_{ref}, der in diesem Fall gleich $t + 9$ ist. Neben den Bedingungen über die Ein- und Ausgabesignale entsprechend Abbildung 5 wird die Operation charakterisiert durch ihren wichtigen Anfangs- und Endzustand row_act, sowie durch den sichtbaren Zustand $actrow$, der im Verlauf der Operation mit der neuen Zeilenadresse belegt wird. Die Informationen über den konzeptionellen Zustand sind nicht in Abbildung 5 dargestellt.

3.2.4 Verfeinerung zu Operationseigenschaften

Wenn die gerade beschriebene Äquivalenzverifikation zwischen Operationsautomat und Schaltung mit der Qualität formaler Verifikation durchgeführt werden soll, sind zusätzliche Maßnahmen erforderlich, um praxisrelevante Schaltungsgrößen untersuchen zu können. Für den hier beschriebenen Ansatz werden die abstrakten konzeptionellen Zustände explizit mit den Zuständen der zu untersuchenden Schaltung in Beziehung gesetzt und dadurch der Äquivalenzvergleich partitioniert.

Das entspricht der Vorgehensweise bei der kombinatorischen Äquivalenzverifikation [Filkorn 1992, Kuehlmann/Krohm 1997, Lohse/Warkentin 1998, Bormann/Warkentin 1999, Hoereth/Müller-Brahms/Rudlof 2002]: Dort besteht die ursprüngliche Verifikationsaufgabe ebenfalls im Nachweis, dass die Schaltungen gleiches Ein-/Ausgabeverhalten haben. Dies ist zunächst eine sequentielle Verifikationsaufgabe, bei der beliebig lange Eingabe- und Ausgabetraces betrachtet werden müssen. Diese sequentielle Verifikationsaufgabe kann aber nur mit Algorithmen gelöst werden, die schon an ziemlich kleinen Schaltungen scheitern. Deshalb werden in einem Vorverarbeitungsschritt die Zustände der beiden zu vergleichenden Schaltungen in Beziehung gesetzt, indem deren Zustandssignale einander zugeordnet werden. Durch diese Zuordnung wird die ursprüngliche Verifikationsaufgabe zeitlich unterteilt: Die Verifikation nimmt gleiche Zustände zu einem beliebigen Anfangszeitpunkt t an und beweist die Gleichheit der Zustände zum Zeitpunkt $t + 1$ (und natürlich auch die Gleichheit der Ausgabesignale zum Zeitpunkt t). Diese zeitlich unterteilte Verifikationsaufgabe kann mit SAT-Algorithmen behandelt werden, deren Leistungsfähigkeit die Untersuchung großer Schaltungen ermöglicht. Die Reduktion auf ein kombinatorisches Problem ist zulässig, weil mit der kombinatorischen Äquivalenzverifikation Entwurfsschritte wie Synthese, Gatteroptimierungen, Einfügen von Testlogik oder Änderungen in letzter Minute verifiziert werden sollen, die die Zustandskodierung der Schaltung nicht verändern sollen.

Der Charme dieser Vorgehensweise besteht in ihrem Pessimismus. Sie liefert lieber einmal ein negatives Vergleichsergebnis, wo eigentlich bereits Äquivalenz herrscht. Wenn das Verfahren aber zwei Schaltungen als äquivalent beweist, dann haben sie verlässlich das gleiche Ein- und Ausgabeverhalten. Verkehrte Zustandszuordnungen können keine fälschlicherweise positiven Resultate bewirken, mit denen Fehler übersehen würden. Allerdings sind fälschlich negative Resultate durchaus möglich, bei denen Schaltungen mit gleichem Ein-

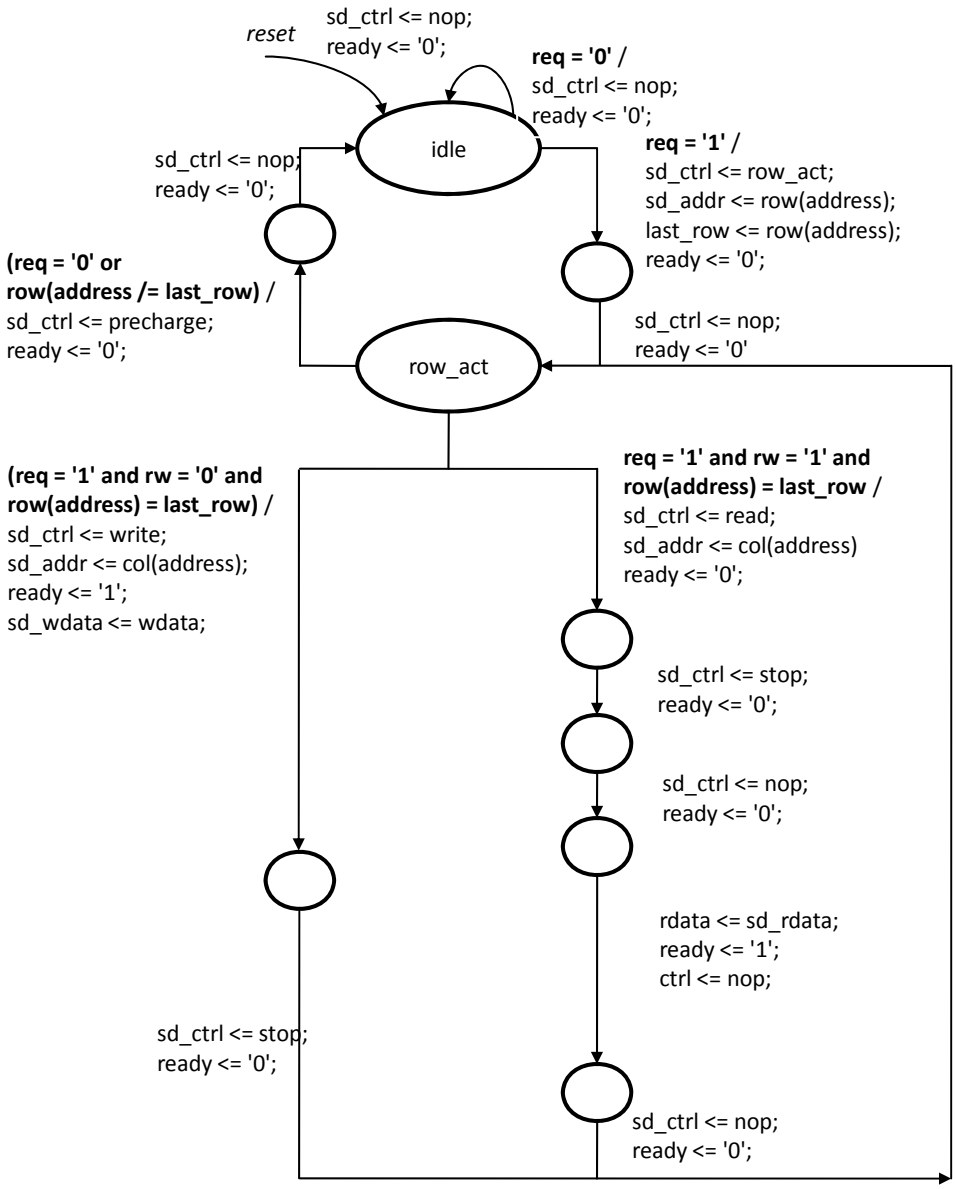

Abbildung 6: Implementierung des SDRAM Interfaces

/Ausgabeverhalten als unterschiedlich bezeichnet werden. Dann werden Gegenbeispiele erzeugt, die Hinweise auf Schaltungsfehler oder auf fehlerhafte Zustandszuordnungen enthalten.

Dieser Ansatz geht ähnlich vor: Den konzeptionellen Zuständen des Operationsautomaten werden die Zustände der zu untersuchenden Schaltung zugeordnet. Dadurch wird der Operationsautomat zu einem Automaten verfeinert, dessen Zustände durch Bedingungen über die konkreten Signale der Implementierung gegeben sind. Diese Zustandsbedingungen sollen konkrete konzeptionelle Zustände heißen. Der Vorteil dieses verfeinerten Automaten besteht darin, dass jede Transition und damit jede Operation für sich in einer separaten Verifikationsaufgabe mit der Schaltung verglichen werden kann. Diese Verifikationsaufgabe nimmt an, dass sich die Schaltung im konkreten konzeptionellen Anfangszustand einer Operation befindet und beweist, dass die Schaltung sich entsprechend der Operation verhält und sich zum Referenzzeitpunkt in einem Zustand befindet, der den konkreten konzeptionellen Endzustand erfüllt. Diese Verifikationsaufgabe lässt sich durch eine Eigenschaft ausdrücken. Diese Eigenschaft wird Operationseigenschaft genannt. Der Automat der Operationseigenschaften unterscheidet sich vom Operationsautomaten nur in Bezug auf die Zustände. Die Transitionen und Referenzzeitpunkt sind gleich. Er wird deswegen mit dieser Menge identifiziert.

Die Zustandszuordnung ist eine Funktion, die Zustände der Implementierung (zu möglicherweise mehreren Takten) auf die konzeptionellen Zustände und einen Spezialwert $\perp$ abbildet. Der Spezialwert steht für solche Zustände der Implementierung, die keinen konzeptionellen Zustand repräsentieren. In der Praxis gibt es aber für jeden wichtigen Zustand ein Prädikat, das feststellt, wann die Schaltung in einem wichtigen Zustand ist. Ferner gibt es für jeden wichtigen Zustand eine separate Funktion, die das Verhalten der Schaltung auf den zugehörigen sichtbaren Zustand abbildet.

Die Zustandszuordnung muss vorgegeben werden. Während sie beim kombinatorischen Vergleich durch geeignete Algorithmen identifiziert werden kann, ist hier der Benutzer gefragt. Der Benutzer arbeitet aber mit einem Sicherheitsnetz, das dem pessimistischen Charme der Äquivalenzverifikation entspricht: Falsche Benutzervorgaben können nur dazu führen, dass die Verifikation keine Äquivalenz zwischen Operationseigenschaften und Implementierung feststellt, obwohl eigentlich schon Äquivalenz herrscht. Es ist nicht möglich, durch falsche Benutzervorgaben zu einem fälschlich positiven Resultat zu gelangen.

Zur Bestimmung der Zustandszuordnung beim SDRAM-Interface muss die Implementierung untersucht werden. Abbildung 6 beschreibt den Zustandsübergangsgraphen der Implementierung. Eine Transition dieses Graphen entspricht einem Takt. Die Transition wird charakterisiert durch die Zuweisungen an Register, die Großteils die Ausgabesignale treiben, und ggf. durch die Bedingungen, unter der die Transition ausgeführt wird. Eine naheliegende Zustandszuordnung bildet $state = idle$ auf den wichtigen Zustand IDLE und $state = row\_act$ auf den wichtigen Zustand ROW_ACT ab, sowie den Inhalt des Registers $last\_row$ auf den sichtbaren Zustand actrow.

3.2.5 Darstellung von Operationseigenschaften

Wenn diese Zustandszuordnung zugrunde gelegt wird, kann die zu Abbildung 5 gehörende Operationseigenschaft gebildet werden. Sie wird in der Sprache ITL ausgedrückt, die so eng an Timingdiagramme angelehnt ist, dass sich die entsprechenden Eigenschaften häufig sehr anschaulich visualisieren lassen.

ITL unterscheidet zwischen einem Annahme- und einem Beweiszielteil. Der Annahmenteil entspricht der linken Seite einer Implikation und der Beweiszielteil der rechten Seite. Die Implikation wird für alle Zeitpunkte $t \geq 0$ geprüft. Annahme und Beweiszielteil bestehen aus temporalen Bedingungen, die aus einem Zustandsprädikat und zeitlichen Spezifikationen gebildet werden. Die zeitlichen Spezifikationen referenzieren Zeitpunkte explizit, relativ zu einem beliebigen aber festen Zeitpunkt t.

Durch die explizite Benennung der Zeitpunkte ist es möglich, Annahme- und Beweiszielteil zeitlich überlappen zu lassen. Dies ist für Operationseigenschaften wichtig, denn Operationen können sich auch durch die Werte von Eingabesignalen zu recht späten Zeitpunkten unterscheiden, wie z.B. Lesezugriffe eines Prozessors, die normal abgewickelt werden oder durch eine Fehlermeldung beendet werden.

In dieser Beziehung hat ITL Vorteile gegenüber der üblichen Verwendung von SVA oder PSL, bei der der Antezendent (der der ITL-Annahme entspricht) und der Sukzedent (der dem Beweiszielteil entspricht) zeitlich voneinander getrennt sein müssen. Ein weiterer Vorteil von ITL besteht darin, dass es gar nicht erst Operatoren zulässt, die bei gegebenem Anfangszeitpunkt der Eigenschaft unterschiedliche Matchings zulassen, über deren Menge dann räsoniert wird. Dies ist eine Quelle von Fehlern selbst in der Anwenderliteratur für SVA.

Die Graphik drückt die Annahme durch blaue Linien und den Beweiszielteil durch rote Linien aus. Falls sich die Eigenschaft beweisen lässt, ist es möglich, die Graphik auf eine transparente Folie zu drucken und über jeden möglichen Simulationslauf des SDRAM Interfaces zu ziehen. Wo immer die blauen Linien passen, müssen dann auch die roten Linien passen.

Die textuelle Darstellung der zu der Operation aus Abbildung 5 gehörenden Operationseigenschaft ist

```
property read_new_row is

assume:
at t:       state = row_act;
at t:       request = '1';
at t:       rw = '1';
at t:       address /= last_row;

prove:
at t+9:           state = row_act;
at t+9:           last_row = prev(row(address));

during [t+1, t+7]:      ready = '0';
at t+8:           ready = '1';
at t+9:           ready = '0';
at t+8:           rdata = prev(sd_rdata);

at t+1:           sd_ctrl = precharge;
at t+2:           sd_ctrl = nop;
at t+3:           sd_ctrl = activate;
at t+3:           sd_addr = row(address);
at t+4:           sd_ctrl = nop;
at t+5:           sd_ctrl = read;
at t+5:           sd_addr = col(address);
at t+6:           sd_ctrl = stop;
during [t+7, t+9]:      sd_ctrl = nop;

end property;
```

Die Funktionen row und col extrahieren dabei aus der Adresse die Zeilen- und Spaltenadresse. Abbildung 7 präsentiert die zugehörige Graphik. Die Eigenschaft lässt sich nur beweisen, wenn während des Beweises vorausgesetzt wird, dass sich die Schaltungsumgebung entsprechend des Protokolls verhält. Dies wird in den folgenden beiden Constraints formalisiert:

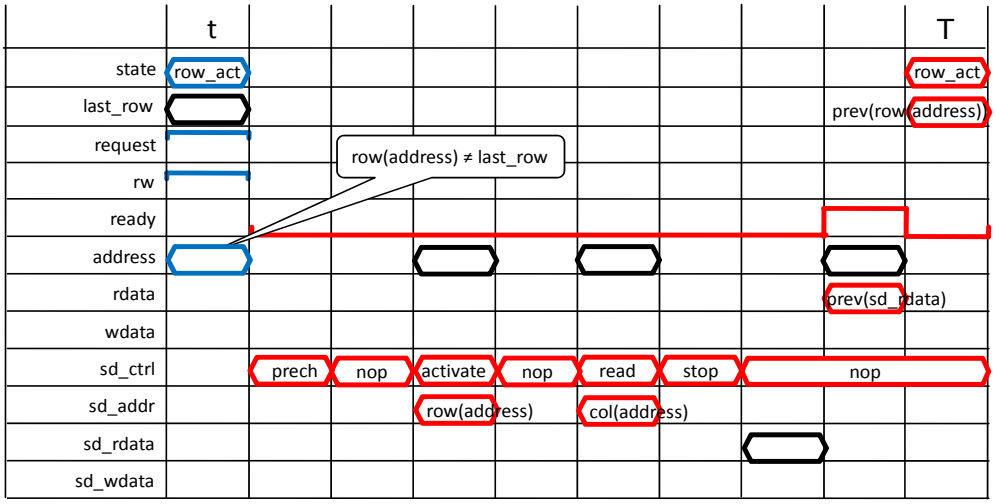

Abbildung 7: Graphische Darstellung der Eigenschaft

```
constraint no_reset :=
      reset = '0';
end constraint;

constraint processor_protocol :=
      if request = '1' and ready = '0' then
            next(request) = '1' and
            next(address) = address and
            next(rw) = rw and
            if rw = '0' then next(wdata) = wdata;
      end if;
end constraint;
```

3.2.6 Diskussion

Operationseigenschaften, Operationsautomat und Transaktionsautomat sind Schaltungsbeschreibungen mit wachsendem Abstraktionsgrad. In den Operationseigenschaften wird nur noch fallweise Auskunft über das Verhalten einiger interner Signale der Schaltung gegeben. Operationsautomaten geben nicht einmal mehr darüber Auskunft, sondern beschreiben Verhalten interner Signale nur noch in Bezug auf ihre Aufgabe, die Reihenfolge der Operationen festzulegen und für den Informationsfluß zwischen aufeinanderfolgenden Operationen zu sorgen. Operationseigenschaften und –automaten repräsentieren das Ein-/Ausgabeverhalten aber signal- und zyklenexakt. Erst der Übergang zu Transaktionsautomaten abstrahiert auch das Ein-/Ausgabeverhalten der Schaltung zu atomaren Transaktionen.

Operationseigenschaften und –automaten stellen also das Ein-/Ausgabeverhalten der Schaltung nicht weniger präzise dar als die ursprüngliche RTL-Beschreibung. Dabei sind die Operationen aber stärker strukturiert, weil in den Operationen zusammengehörende Aktivitäten auf den Ausgabesignalen in ihrem Zusammenhang untereinander und in ihrem Zusammenhang mit Eingaben dargestellt werden. Dies ist weniger eine Abstraktion als eine Strukturierung des Ein-/Ausgabeverhaltens. Diese Strukturierung macht aus einer meist völlig unübersichtlichen Vielzahl an möglichen Ausgabeaktionen des RTLs eine überschaubare Anzahl von Operationen. Dadurch läßt sich die Schaltungsfunktion leichter verstehen und es lässt sich leichter entscheiden, ob die Operation einer Schaltung richtig ist.

Der Verständnisgewinn beim Übergang von RTL zu Operationen entspricht etwa dem Verständnisgewinn, der sich dadurch einstellt, dass statt einer Netzliste die zugehörige RTL-Beschreibung untersucht wird. Auch hier wird das Schaltungsverständnis erleichtert, indem einzelne Signale zu Bitvektoren und komplizierte Logiken, die durch Gatter und verbindende interne Signale aufwändig beschrieben sind, durch mächtige oder zumindest anschaulichere Operatoren zusammengefaßt werden. Im Verhältnis zwischen Netzliste und RTL spielen Werte der Bitvektoren und die sie zusammenfassenden mächtigeren Operatoren etwa die Rolle, die Transaktionen und Operationen im Verhältnis zwischen RTL und einem Operationsautomaten spielen.

Doch führt die Strukturierung des Ein-/Ausgabeverhaltens durch Operationen bzw. Operationseigenschaften und die damit verbundene Abstraktion internen Verhaltens nicht nur zu einem besseren Verständnis der Schaltung. Es wird in den folgenden Abschnitten gezeigt werden, dass Operationen auch zu Beweisaufgaben führen, die für die Beweistechnik besonders geeignet sind, sodass gegebene Schaltungen mit geringerem technischen Aufwand verifiziert werden können, bzw. mit vorhandenen Resourcen größere Schaltungen untersucht werden können.

Dieser Effekt wird auch mit anderen in der Literatur diskutierten Abstraktionen wie etwa predicate abstraction und abstraction refinement angestrebt [Kroening/Seshia 2007, Clarke et al. 2000 , Wang et al. 2006, Chauhan et al. 2002, McMillan/Amla 2003, Gupta et al. 2003, Jain et al. 2008]. In diesen Verfahren geht es aber immer darum, das Modell einer Schaltung geeignet zu reduzieren. In dieser Arbeit wird auf derartige Reduktionen des Modells verzichtet. Alle Eigenschaften werden auf einer internen Repräsentation des gesamten RTLs durchgeführt. Die einzige Maßnahme zur Reduktion der Größe eines Modells besteht ggfs. in kompositionalen Techniken, bei denen eine Schaltung in Teilschaltungen zerlegt wird, die dann separat verifiziert werden.

3.3 Beweistechnik

In der vollständigen Verifikation sind Beweistechnik und Methodik besonders gut aufeinander abgestimmt. Die Beweistechnik der Wahl ist Interval Property Checking (IPC), ein Beweisverfahren, das die aus Operationseigenschaften resultierenden großen Verifikationsaufgaben häufig in Minutenschnelle bearbeiten kann. Denn bei vollständiger Verifikation fällt der Nachteil von IPC nicht so sehr ins Gewicht, dass Erreichbarkeitsbedingungen vorgegeben werden müssen, mit denen die im Schaltungsbetrieb erreichbaren Zustände beschrieben werden. Zur vollständigen Verifikation werden nämlich vergleichsweise wenige und einfache Erreichbarkeitsbedingungen benötigt. Darüber hinaus werden die vom Benutzer vorgegebenen Erreichbarkeitsbedingungen geprüft, sodass die Verifikation nicht durch versehentlich falsche Vorgaben korrumpiert werden kann.

So führt die Abstimmung zwischen Beweistechnik und Methodik bei der vollständigen Verifikation dazu, dass auch große Schaltungen mit Operationseigenschaften untersucht werden können, und trotzdem der Aufwand für die Erstellung der Erreichbarkeitsbedingungen akzeptabel bleibt. Der Aufwand ist jedenfalls so gering, dass die vollständige Verifikation produktiver ist als die simulationsbasierte Verifikation, selbst wenn der Qualitätsgewinn unberücksichtigt bleibt.

Beim Beweis allgemeiner Assertions mit IPC fallen schwerer zu bestimmende Erreichbarkeitsbedingungen an. In ABFV-Werkzeugen ist die Erreichbarkeitsanalyse daher weitgehend automatisiert. Dass damit nur kleinere Schaltungen untersucht werden können und/oder auf vollgültige Beweise von Assertions verzichtet werden muss, wird in Kauf genommen.

IPC wird in Abschnitt 3.3.1 behandelt. Erreichbarkeitsbedingungen werden in Abschnitt 3.3.2 exemplifiziert, und ihre Bestimmung und Rechtfertigung in den Abschnitten 3.3.3 bis 3.3.5 diskutiert.

3.3.1 Interval Property Checking (IPC) und Bounded Model Checking (BMC)

Um eine Schaltung zu verifizieren, muss sichergestellt werden, dass jede aus dem Operationsgraphen abgeleitete Operationseigenschaft für jeden Zeitpunkt $t \geq 0$ erfüllt ist.

Der entsprechende Beweis kann im Prinzip mit beliebigen formalen Beweisverfahren geführt werden. Interval Property Checking (IPC) ist allerdings besonders geeignet, weil es große Schaltungen untersuchen kann und schnell zu einem Ergebnis kommt. Interval Property Checking ist ein SAT-basiertes Verfahren [Ganai/Gupta 2007] um digitale Schaltungen zu verifizieren [Lohse/Warkentin 2001]. Die Schaltungen werden durch Mealy-Automaten repräsentiert, diese sollen bitwertige Eingabe-, Ausgabe- und Zustandsvariablen haben. Die Abbildung von RTL-Code auf derartige Automaten gehört zum Stand der Technik [Bormann 1995].

Interval Property Checking ist geeignet für Eigenschaften A, die Eingabe-, Ausgabe-, und Zustandsvariablen der Schaltung für Zeitpunkte innerhalb eines endlichen Zeitintervalls $[t, t+n]$ in Beziehung setzen. Die Eigenschaften werden für alle Zeitpunkte $t \geq 0$ und für alle Traces untersucht. Dazu werden die Zustandsübergangs- und Ausgabefunktion des Mealyautomaten $n+1$ - mal abgerollt (siehe Abbildung 8). Dadurch entstehen Instanzen der Funktionen für die Zeitpunkte $t, t+1$ usf. bis $t+n$, durch deren Variablen die Variablen der Eigenschaft substituiert werden. Das ergibt eine boolesche Funktion A', deren Argumente die Instanzen der Eingangsvariablen des Mealyautomaten für jeden der Zeitpunkte t bis $t+n$ und die Instanz der Zustandsvariablen zum Zeitpunkt t sind. Die Schaltung erfüllt die ursprüngliche Eigenschaft A, wenn A' für alle Argumente 1 ist.

Nullstellen von A' weisen auf Abläufe hin, die A verletzen. Diese sog. Gegenbeispiele werden charakterisiert durch einen Zustand zum Zeitpunkt t und durch die Folge der Eingabewerte an den Zeitpunkten t bis $t+n$. Dabei legt das Verfahren keine Rechenschaft darüber ab, ob der Zustand zum Zeitpunkt t erreichbar ist. Wenn er nicht erreichbar ist, ist das Gegenbeispiel unrealistisch, d.h. es kann im Betrieb der Schaltung gar nicht auftreten. Dann muss der Beweis unter zusätzlichen Erreichbarkeitsbedingungen wiederholt werden. Diese Eigenart beschränkt die Anwendung von IPC auf Verfahren, bei denen Erreichbarkeitsbedingungen separat bestimmt und validiert werden.

Die Nullstellen von A' werden mit einem SAT-Verfahren berechnet. Diese Verfahren können mittlerweile robust Probleme mit Millionen oder mehr Variablen lösen. Für diese Arbeit ist die exakte Funktion der SAT-Algorithmen unerheblich, sie werden etwa in [Ganai/Gupta 2007] vorgestellt. Bekannte SAT-Beweiser sind GRASP [Marques-Silva/Sakallah 1996], Chaff [Malik et al. 2001] und MiniSat [Een/Sörensson 2003]. An der Verbesserung der SAT-Beweiser wird fortlaufend gearbeitet [Brinkmann 2003, Wedler et al. 2007, Wedler et al. 2005, Novikov/Goldberg 2001, Novikov 2003, Novikov/Brinkmann 2005, Goldberg/Novikov 2002], wie auch an dem Einsatz alternativer Beweismethoden [Achterberg/Wedler/Brinkmann 2008].

IPC ist mit dem Bounded Model Checking (BMC) nach [Biere 1999] verwandt. Im Unterschied zu IPC wird bei BMC am Anfang der Instanzen der Zustandsübergangs- und Ausgabefunktionen kein beliebiger Zustand eingelesen, sondern der Resetzustand der Schaltung. Die Instanzen beschreiben damit den Takt 0, 1, 2, usf. nach dem Zurücksetzen der Schaltung. Die Eigenschaft A wird bei BMC nicht für beliebiges t untersucht. Statt dessen bestimmt ein BMC-Algorithmus eine Schranke N in Abhängigkeit von den verfügbaren Ressourcen und untersucht die Eigenschaft A nur für alle $t \leq N$. Ein Gegenbeispiel aus dieser Untersuchung ist besonders nützlich, weil es im Resetzustand beginnt und damit nicht wie bei IPC aufgrund eines unerreichbaren Anfangszustands unrealistisch sein kann. Wenn die Untersuchung aber einen Beweis liefert, so hat dieser aufgrund der Beschränkung auf N Takte nur eine beschränkte Aussagekraft, die für konstantes N auch noch von der untersuchten Schaltung abhängt.

Solange ABFV nur die simulationsbasierte Verifikation unterstützen soll, sind ihre Benutzer vornehmlich an Gegenbeispielen interessiert und weniger am Beweis dieser Eigenschaft, sodass BMC für dieses auch semiformale Verifikation genannte Einsatzszenario geeigneter scheint als IPC. Wenn allerdings komplexere Assertions wirklich bewiesen werden müssen,

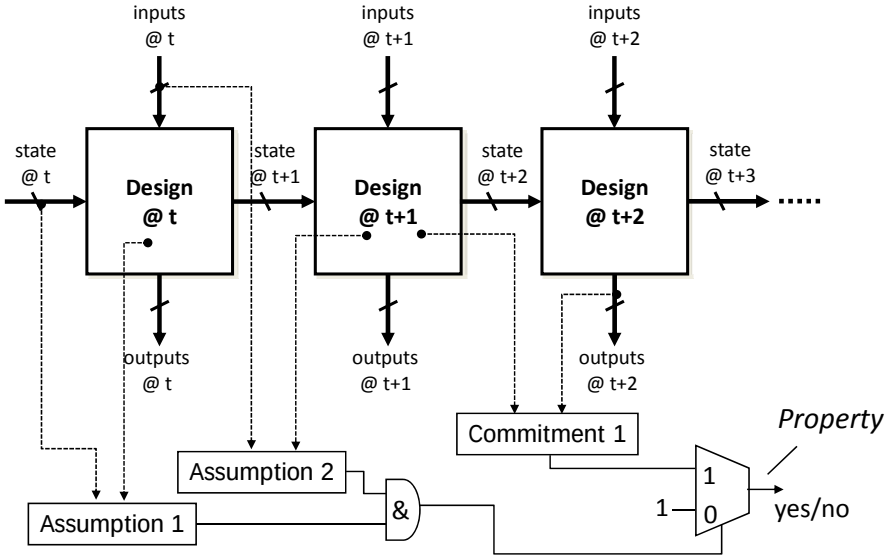

Abbildung 8: Generierung einer IPC-Beweisaufgabe

weil z.B. so die Einhaltung einer Spezifikation nachgewiesen werden soll, gerät ein Verifikateur in ein Dilemma, weil BMC ungeeignet ist, weil andere ABFV-Algorithmen am Beweis scheitern können und weil die Bestimmung von Erreichbarkeitsbedingungen zum Beweis der Assertion mit IPC aus den in Abschnitt 3.3.5 zu diskutierenden Gründen aufwändig werden kann. Ein Ausweg aus diesem Dilemma wird in Abschnitt 3.4.4 beschrieben.

Zum Beweis der Eigenschaft A mit IPC sind nur $n + 1$ Instanzen der Zustandsübergangs- und Ausgabefunktion notwendig, während BMC $N + n + 1$ Instanzen benötigt, sodass eine BMC-Verifikation trotz der eingeschränkten Beweiskraft aufwändiger ist als die entsprechende IPC-Aufgabe.

3.3.2 Beispiel einer Erreichbarkeitsbedingung

Das Beispiel des SDRAM Interfaces ist eigentlich zu klein, um die Benutzersicht auf IPC, unrealistische Gegenbeispiele und Erreichbarkeitsbedingungen zu vermitteln. Es können we-

der die geringen Laufzeiten eines IPC-basierten Beweises einer Operationseigenschaft demonstriert werden, noch der Facettenreichtum der benötigten Erreichbarkeitsbedingungen.

Um aber überhaupt etwas präsentieren zu können, soll angenommen werden, dass das SDRAM-Interface aus Abbildung 6 entsprechend Abbildung 9 modifiziert werde, d.h. durch Entfernung der Zuweisungen an das Ausgaberegister $ready$, wenn die Schaltung im Zustand $idle$ ist. Diese Änderung ist unerheblich, denn auf allen Pfaden, die in den Zustand $idle$ führen, wird diesem Register die '0' zugewiesen. Die Zuweisungen an $ready$ im $idle$-Zustand sind also redundant.

Dennoch wird auf der entsprechend Abbildung 9 modifizierten Schaltung die Eigenschaft read_new_row aus Abschnitt 3.2.5 nicht direkt per IPC zu beweisen sein. Das Gegenbeispiel wird sein, dass $ready$ zum Zeitpunkt $t + 1$ den Wert '1' annimmt. Dieses Gegenbeispiel ist irrelevant, denn die Schaltung kann nicht so betrieben werden, dass $ready$ im Zustand $idle$ den Wert 1 annimmt. Um dies beim Beweis zu berücksichtigen, muss dem Startzu-

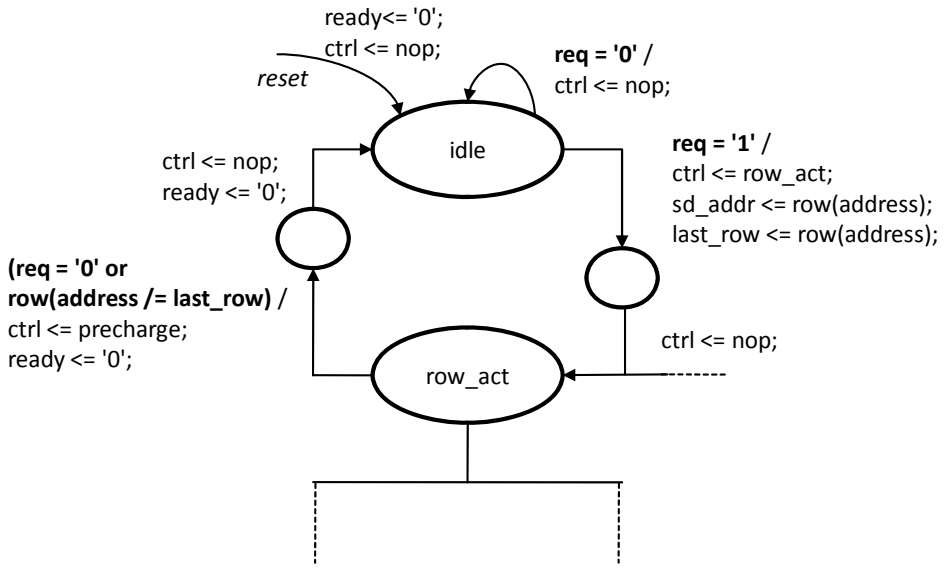

Abbildung 9: SDRAM Interface mit alternativer Implementierung

stand der Eigenschaft die Erreichbarkeitsbedingung $ready = 0$ hinzugefügt werden. Entsprechend müssen alle Eigenschaften mit $state = idle$ als Endzustandsbedingung so abgeändert werden, dass die neue Endzustandsbedingung $state = idle\ and\ ready = '0'$ heißt.

3.3.3 Erreichbarkeitsanalyse

Erreichbarkeitsbedingungen können automatisch bestimmt werden, oder der Benutzer muss sie sich selbst ausdenken.

Die automatische Bestimmung von Erreichbarkeitsbedingungen wird als Erreichbarkeitsanalyse bezeichnet und steht in enger Beziehung zu den formalen Beweisalgorithmen. Erreichbarkeitsanalyse ist ein sehr schwieriges Forschungsthema und ein Hauptgrund für die langsame Entwicklung formaler Verifikation über die letzten 15 Jahre.

Manche Erreichbarkeitsanalysen streben an, die Menge der erreichbaren Zustände exakt zu bestimmen. Diese Verfahren können nur kleine Schaltungen untersuchen und haben selbst dann unter Umständen enorme Laufzeiten.

Andere Verfahren bestimmen Ober- oder Untermengen der erreichbaren Zustände. Der Vorlauf von bis zu N Instanzen der Zustandsübergangs- und Ausgabefunktion des Mealy-Automaten bei BMC (siehe Abschnitt 3.3.1) kann als eine Erreichbarkeitsanalyse verstanden werden, mit der eine Untermenge der erreichbaren Zustände zum Zeitpunkt t der Bedingung A bestimmt wird. Derartige unterschätzende Verfahren sind für die vollständige Verifikation ungeeignet, weil sie die Aussagekraft der Eigenschaftsbeweise beschränken.

Hingegen lassen sich Verfahren zur Bestimmung von Obermengen der erreichbaren Zustände gut in die vollständige Verifikation integrieren. Besonders gut gelang dies mit dem in [Nguyen et al. 2008, Nguyen et al. 2005a, Nguyen et al 2005b] beschriebenen Verfahren. Andere Optionen ergeben sich aus den Arbeiten von [Case/Mishchenko/Brayton 2006, Stoffel et al. 2004, Wedler et al. 2003, McMillan 2003].

3.3.4 Rechtfertigung benutzerdefinierter Erreichbarkeitsbedingungen

Die automatische Erreichbarkeitsanalyse versagt spätestens in solchen Verifikationsprojekten, in denen die Erreichbarkeitsbedingungen kreative Entwurfsanstrengungen auf der Basis eines globalen Schaltungsverständnisses zusammenfassen müssen. Für eine vollständige Verifikation ist es deswegen unverzichtbar, dass Erreichbarkeitsbedingungen vom Benutzer explizit vorgegeben werden können.

Solch eine Benutzervorgabe kann Fehler enthalten. Durch den in Abschnitt 3.4 vorzustellenden Vollständigkeitsprüfer werden solche Fehler aber entdeckt. So wird verhindert dass die Verifikation durch fehlerhafte Benutzervorgaben entwertet wird. Aufgrund der geringen Laufzeit der beteiligten Beweisverfahren muss der Benutzer bei der Vorgabe von Erreichbarkeitsbedingungen nicht einmal besonders sorgfältig sein: Wenn die vollständige Verifikation bis zum Ende durchgeführt werden kann, sind alle Erreichbarkeitsbedingungen ebenfalls gerechtfertigt, und andernfalls ergeben sich Gegenbeispiele, anhand derer die Erreichbarkeitsbedingungen nachjustiert werden können.

ABFV setzt vor allem auf automatisch zu bestimmende Erreichbarkeitsbedingungen. Wenn sie gelegentlich doch von Hand vorgegeben werden, bleibt es in der Verantwortung des Benutzers, die Erreichbarkeitsbedingungen zu rechtfertigen. Teilweise wird einfach informell argumentiert. Andere Nutzer prüfen Erreichbarkeitsbedingungen per Simulation. In beiden Fällen reduziert sich die Verlässlichkeit der Verifikation beträchtlich: Es ist fraglich, welchen Wert die formale Verifikation einer Assertion noch hat, wenn sie unter Annahme einer Erreichbarkeitsbedingung verifiziert wird, die ihrerseits durch Verfahren gerechtfertigt wird, die man wegen ihrer Unzuverlässigkeit gerade vermeiden wollte.

Erreichbarkeitsbedingungen können aber auch durch separate Beweise gerechtfertigt werden. Typisch sind Induktionsbeweise. Diese erfordern häufig das Auffinden weiterer Erreichbarkeitsbedingungen, bis eine Induktionshypothese gefunden ist, die sich beweisen lässt. In der ABFV führt dieses aufwändige Verfahren zu einer Einzelanfertigung der Erreichbarkeitsanalyse für die entsprechende Assertion. Es wiederholt sich möglicherweise, wenn weitere Assertions bewiesen werden sollen.

3.3.5 Erreichbarkeitsbedingungen und Operationseigenschaften

Im Zusammenhang mit der vollständigen Verifikation werden Erreichbarkeitsbedingungen nur unter der Randbedingung gebraucht, dass die Schaltung sich in einem wichtigen Zustand befindet, denn alle Operationseigenschaften beginnen dort. Ein Operationsautomat hat meist nur recht wenige wichtige Zustände, und entsprechend sind zur vollständigen Verifikation nur wenige Erreichbarkeitsbedingungen notwendig.

Wenn ein Register in allen Operationen, die von einem wichtigen Zustand ausgehen, erst initialisiert wird, bevor sein Wert weiterverwendet wird, dann braucht dieses Register nicht in den Erreichbarkeitsbedingungen zu diesem wichtigen Zustand aufzutauchen. Falls nämlich das Beweiswerkzeug am Beginn einer Operation erst einen unerreichbaren Wert annimmt, so wird dieser Wert rechtzeitig sinnvoll überschrieben und der unerreichbare Wert hat keine Auswirkungen.

In manchen Implementierungsstilen werden allerdings die Register gleich nach der Benutzung wieder initialisiert, und dann bis zur nächsten Benutzung nicht wieder. Das Beispiel aus Abschnitt 3.3.2 ähnelt dieser Situation. Solche Implementierungsstile führen zu der Erreichbarkeitsbedingung, dass das Register in einem wichtigen Zustand den Initialisierungswert hat. In dieser Situation wird in der Tat eine Erreichbarkeitsbedingung gebraucht, die aber recht einfach und leicht zu erkennen ist.

Falls doch kompliziertere Erreichbarkeitsbedingungen im wichtigen Zustand auftreten sollten, besteht eine gute Chance, dass diese auch dem Entwickler der Schaltung bekannt sind, denn auch der Entwickler betrachtet die Operationen ja als in sich abgeschlossene funktionale Einheiten, die in den wichtigen Zuständen starten und implementiert entsprechend. Dieses Wissen des Designers kann bei der Aufstellung der Eigenschaften genutzt werden. Das Risiko einer Duplikation von Fehlern ist aus den in Abschnitt 3.3.4 genannten Gründen ausgeschlossen.

Alles in allem sind Erreichbarkeitsbedingungen für den Beweis von Operationseigenschaften also meist relativ leicht zu beschaffen. Diese Aussage soll aber nicht ausschließen, dass gerade unzulänglich implementierte Schaltungen komplizierte Erreichbarkeitsbedingungen enthalten, die nur mühsam erkannt werden können.

Bei Erreichbarkeitsbedingungen für Assertions liegen die Verhältnisse jedoch anders: Im Verlauf einer Operation werden Zwischenergebnisse berechnet, gespeichert und weiterverwendet. Dadurch können sich in der Mitte der Operation komplizierte Erreichbarkeitsbedingungen ergeben, die auch dem Entwickler nicht bewusst sind, weil er/sie den Ablauf der Operation vor Augen hat und nicht darüber nachdenkt, welche Signalwerte gleichzeitig auftreten können. Diese komplizierten Erreichbarkeitsbedingungen aus der Mitte der Operationen sind für den Beweis der Operationseigenschaften glücklicherweise nicht erforderlich. Sie können aber sehr wohl für den Beweis von Assertions gebraucht werden, die z.B. Aussagen über Signalverhalten in der Mitte von Operationen machen. Solche Erreichbarkeitsaussagen sind schon in Bezug auf einzelne Operationen nicht trivial. Besonders schwierig wird es aber, wenn eine Assertion eine Aussage über mehrere Operationen macht. Dann lässt sich der Prozess der Aufstellung der Erreichbarkeitsbedingung nicht mehr richtig strukturieren, weil die unrealistischen Gegenbeispiele immer wieder andere Abläufe aus unterschiedlichen Operationen aufzeigen. Entsprechend aufwändig kann die Aufstellung von Erreichbarkeitsinformationen für allgemeine Assertions werden. In diesem Fall kann es vorteilhaft sein, erst eine vollständige Verifikation durchzuführen, um die Assertion danach mit den Verfahren zu beweisen, die in Abschnitt 3.4.4 diskutiert werden.

3.4 Vollständigkeitsprüfung

Bei der vollständigen Verifikation wird eine Schaltung durch Operationseigenschaften aus Abschnitt 3.2 mit der Beweistechnik Interval Property Checking aus Abschnitt 3.3 untersucht. Die operationsweise Untersuchung der Schaltungsfunktion hilft nicht nur bei der Erreichbarkeitsproblematik und der Beweiskomplexität, über die im Abschnitt 3.2 diskutiert wurde. Sie hilft auch einem Benutzer, die Verifikationsaufgabe intellektuell aufzuteilen und sich zu einem Zeitpunkt konzentriert mit einer Operation und allen ihren Facetten auseinanderzusetzen.

Die Konzentration auf einzelne Operationen lässt aber die Gefahr entstehen, dass der Zusammenhalt der Operationen aus dem Blickfeld des Verifikateurs gerät. So können Verifikationslücken entstehen. Beispiele werden in Abschnitt 3.4.1 vorgestellt. Eine Maßnahme zum Ausschluss von Verifikationslücken besteht in der Untersuchung der Menge der Operationseigenschaften auf Vollständigkeit. Dazu wird untersucht, ob der Verifikationscode die Werte der Ausgabesignale zu jedem Zeitpunkt scharf genug überwacht. Die Schärfe der Überwachung wird im Begriff der Determiniertheit aus Abschnitt 3.4.2 formalisiert. Der Vollständigkeitsprüfer aus Abschnitt 3.4.3 nimmt Determinierungsannahmen über die Eingabesignale an und beweist, dass dann die Operationseigenschaften die Ausgabesignale im Rahmen vorgegebener Determinierungsforderungen überprüfen.

Mit dem Vollständigkeitsprüfer kann die vollständige Verifikation von Schaltungsteilen geprüft werden, die Operationen nacheinander abarbeiten. Für solche Schaltungsteile wird in Abschnitt 3.5.6 der Begriff Cluster eingeführt. Abschnitt 3.5.7 stellt dar, unter welchen Bedingungen eine vollständige Verifikation einer Menge von Clustern auch eine vollständige Verifikation der aus diesen Clustern zusammengesetzten Schaltung ist.

3.4.1 Verifikationslücken

Operationseigenschaften werden unter Bezug auf die Abstraktionsebenen aus Abbildung 2 entweder von oben nach unten oder von unten nach oben entwickelt.

Bei Verfügbarkeit einer guten Spezifikation wird der Verifikateur mit der höchsten Abstraktionsebene, also den Transaktionsautomaten, beginnen und sich zuerst anhand der Spezifikation einen Überblick über die Transitionen und konzeptionellen Zustände dieses Automaten verschaffen. Dieser wird weiter zu Operationsautomat und Operationseigenschaften verfeinert. Dabei fließen auch Informationen aus der Implementierung über Entwurfsentscheidungen ein, wie in den Abschnitten 3.2.3 und 3.2.4 dargestellt.

Von unten nach oben wird vorgegangen, wenn nur unzulängliche Spezifikationen vorliegen oder wenn die Implementierung die Spezifikation auf eigenwillige Weise interpretiert, sodass sich die Operationen aus der Spezifikation nicht so richtig in der Implementierung wiederfinden lassen. Zunächst werden die Operationseigenschaften entwickelt. Während dieser Entwicklung klärt sich die Gestalt des Operations- und Transaktionsautomaten. Die Übereinstimmung zwischen den Operationseigenschaften und dem, was von der Spezifikation vorhanden ist, wird kontinuierlich geprüft.

Bei beiden Vorgehensweisen konzentriert man sich auf eine Operation zu einer Zeit. Das ist nicht nur für die Beweistechnik aus Abschnitt 3.3 vorteilhaft. Vielmehr erlaubt es dem Verifikateur, sich ganz auf eine relativ kleine Anzahl von Phänomenen zu beschränken und diese besonders gründlich zu durchleuchten, bevor die nächste Operation bearbeitet wird. Dadurch erhält die einzelne Operation besonders hohe Qualität.

Während der Verifikation muss aber auch sichergestellt werden, dass sich die Operationseigenschaften zu einem Operationsautomaten zusammenfügen. Dem können eine Reihe von Versehen beim Aufstellen der Operationseigenschaften entgegenstehen, die allesamt dazu führen, dass die Gesamtheit der Operationseigenschaften auch unerwünschtes Verhalten akzeptiert. In diesem Fall hat die Verifikation Lücken, für die im Folgenden einige Beispiele präsentiert werden sollen.

Die prominenteste Verifikationslücke besteht darin, dass ganze Operationen nicht verifiziert wurden. Wenn eine Schaltung solch eine vergessene Operationen fehlerhaft ausführte, würde dies nicht entdeckt werden.

Eine andere Sorte von Verifikationslücken besteht darin, dass die Bedingungen über die Eingangssignale der Operationen zu restriktiv formuliert sind. Dann prüft die Operationseigenschaft weniger Abläufe der Schaltung als beabsichtigt und lässt andere Abläufe ungeprüft. In den ungeprüften Abläufen können sich Implementierungsfehler verbergen, die mit den vorliegenden Operationseigenschaften nicht entdeckt werden.

Weitere mögliche Verifikationslücken sind unzulängliche Bedingungen über die Ausgangssignale, die manchmal mehrere Werte zulassen, obwohl nur einer richtig ist. Wenn etwa in der Eigenschaft aus Abschnitt 3.2.5 die Bedingungen über sd_ctrl im Beweiszielteil fehlten, würde eine Schaltung nicht als fehlerhaft erkannt werden, die völlig falsche Befehle an das SDRAM übergeben würde.

Weiterhin können die Prädikate für die wichtigen Zustände falsch formuliert sein, oder es sind unterschiedliche Prädikate für den selben wichtigen Zustand eingesetzt worden.

Schließlich kann die Berechnung des aktualisierten sichtbaren Zustands unvollständig beschrieben sein oder nicht mit den Funktionen zusammenpassen, die aus den Traces der Schaltung die sichtbaren Zustände extrahieren. Wenn etwa die Bedingung "at t+9: last_row = prev(row(address))" aus der Eigenschaft aus Abschnitt 3.2.5 fehlte, wäre bei einem nachfolgenden Lesezugriff sowohl die Eigenschaft über einen Lesezugriff mit gleicher Zeilenadresse als auch eine andere Eigenschaft über einen Lesezugriff mit anderer Zeilenadresse anwendbar. Da die beiden Eigenschaften unterschiedliches Verhalten z.B. von sd_ctrl vorsehen, wäre damit das Verhalten von sd_ctrl wiederum nicht eindeutig bestimmt. Wenn die Schaltung während der Operation versehentlich das Register last_row überschriebe und der nachfolgende Zugriff zufälligerweise auch noch auf die Zeilenadresse zugriffe, die nach dem Überschreiben in last_row stünde, würde der eigentlich erforderliche Wechsel der aktiven Speicherzeile nicht durchgeführt und die falsche Speicherstelle gelesen. Ein solcher Fehler ist per Simulation kaum zu identifizieren, da er in den meisten Fällen nur dazu führt, dass die Speicherzeile häufiger gewechselt wird, aber trotzdem das richtige Datum gelesen wird. Um während der Simulation ein falsches Datum zu lesen, muss mit einer Zeilenadresse zugegriffen werden, die zufälligerweise dem falschen Wert in last_row entspricht, denn nur dann unterbleibt der eigentlich notwendige Wechsel der Speicherzeile. Nur dann wird die falsche Speicherstelle ausgelesen. Diese Situation ist in der Simulation unwahrscheinlich, aber der hier beschriebene Ansatz findet solche Fehler und stellt darüber hinaus sogar sicher, dass sich die Verifikation um dieses mögliche Fehlerszenario auch verlässlich kümmert.

3.4.2 Determiniertheit

Es bietet sich an, nach den oben genannten Verifikationslücken zu suchen, indem festgestellt wird, ob für jeden Eingabetrace der Schaltung die Konjunktion der Eigenschaften durch genau einen Ausgabetrace erfüllt wird. Dieses Vollständigkeitskriterium wird etwa bei [Claessen

2006] eingesetzt. Mit dieser Prüfung wird festgestellt, ob sich der Verifikateur zu jedem Zeitpunkt für einen festen Wert entschieden hat. Ist das nicht der Fall, muss nachgebessert werden.

Dieses Vollständigkeitskriterium verlangt ein bisschen viel, denn die Designer wollen manche Signale nicht immer eindeutig festlegen. Viele Protokollspezifikationen legen z.B. fest, wann Adress- und Datensignale "gültig" sind, wann also das sendende Modul die Information auf dem Bus verfügbar machen muss. Nur dann sollte das empfangende Modul sie vom Bus lesen. Außerhalb dieser Zeitpunkte dürfen diese Signale irgendeinen Wert annehmen, der sich nicht aus der Schaltungsfunktion, sondern aus Optimierungskriterien wie Strom- oder Platzverbrauch ergeben kann. Die Verifikation prüft diese Werte besser nicht, denn sie könnten sich während eines Entwurfsprojekts ändern, ohne dass dies funktionale Auswirkungen haben sollte.

Umgekehrt muss auch sichergestellt werden, dass die Verifikation keine Annahmen über Eingabesignale zu Zeitpunkten macht, an denen z.B. eine Protokollspezifikation festlegt, dass die Signale ungültig sind. Die Schaltung muss für alle ungültigen Signalwerte gleich funktionieren und daher dürfen solche Signalwerte nicht bei der Verifikation der Schaltung ausgewertet werden.

Für die Praxis ist es also offensichtlich erforderlich, für die Eingabe- und Ausgabetraces einer Schaltung eine gewisse Unschärfe zuzulassen. Zwei Eingabetraces dürfen unterschiedlich sein, sollen aber von der Schaltung gleich behandelt werden. Zu einem Eingabetrace darf die Schaltung verschiedene Ausgabetraces produzieren, die sich nur auf eine Weise unterscheiden dürfen, die von den Umgebungsschaltungen nicht wahrgenommen wird.

Diese Unschärfe wird durch Determinierungsbedingungen ausgedrückt. Wenn die Determinierungsbedingungen angeben, wann zwei Eingabetraces als gleich angesehen werden sollen, heißen sie Determinierungsannahmen. Wenn die Determinierungsbedingungen festlegen, wann zwei Ausgabetraces gleich sind, heißen sie Determinierungsforderungen. Wann die Inhalte sichtbarer Register als gleich angesehen werden, wird durch lokale Determinierungsbedingungen festgelegt.

Ganz allgemein könnten Determinierungsannahmen und -forderungen durch bedingte Determinierungsfunktionen gebildet werden. Die Determinierungsbedingungen blenden dabei Situationen aus, in denen es auf die konkreten Werte der Traces nicht ankommt. Die Funktionen extrahieren aus den Traces alle für die Schaltungsfunktion relevanten Informationen, ähnlich den Transaktionsextraktoren aus Abschnitt 3.1.2.

Syntaktischer Grundbaustein zur Festlegung von Determinierungsannahmen und -forderungen ist das Konstrukt

```
if g then determined(e) end if;
```

mit *g* als Bedingung und *e* als Funktion zur Informationsextraktion. Das Konstrukt

```
determined(e);
```

ist eine Kurzschreibweise für

```
if true then determined(e) end if;
```

Die Konjunktion von Determinierungsannahmen bzw. –forderungen wird durch

```
determined(e); determined(f);
```

ausgedrückt.

Neben den Determinierungsannahmen und -forderungen gibt es auch lokale Determinierungsbedingungen, die als Zwischenbeweisziele des Vollständigkeitsprüfers erforderlich werden. Sie drücken aus, dass eine Operationseigenschaft den Wert eines sichtbaren Registers am Ende der Operation eindeutig beschreibt. Anders als die Determinierungsannahmen und -forderungen werden diese lokalen Determinierungsbedingungen also nicht zu jedem Zeitpunkt geprüft, sondern nur zu ausgewählten, meistens nur zu einem Zeitpunkt relativ zum Zeitpunkt t der Operationseigenschaft. Die Syntax ist

```
at t: if g then determined(e) end if;
```

Im Folgenden werden die Begriffsbildungen zur Determinierung anhand des Beispiels über das SDRAM Interface veranschaulicht: Der Operationsautomat muss das Signal sd_ctrl zu jedem Zeitpunkt auf einen eindeutigen Wert überprüfen, damit das SDRAM keine unerwünschten Befehle erhält. Außerdem soll das Signal $ready$ in Übereinstimmung mit ähnlichen Protokollspezifikationen aus der Praxis immer eindeutig bestimmt sein, auch wenn dies vom Protokoll her nicht unbedingt erforderlich wäre. Dies wird durch die Determinierungsforderungen

```
determined(sd_ctrl);
determined(ready);
```

festgelegt. Diese Forderungen verlangen, dass die Operationseigenschaften die beiden Signale zu jedem Zeitpunkt gegen einen eindeutig definierten Wert prüfen.

Das Ausgangssignal sd_addr braucht nur gegen einen eindeutigen Wert geprüft werden, wenn das SDRAM die entsprechende Information benötigt, und das ist der Fall, wenn es die Befehle activate, read oder write erhält. Entsprechend muss das Signal sd_wdata nur gültig sein, wenn der Befehl write kommt. Das Signal rdata muss nur determiniert sein, wenn ein Read-Zugriff mit $ready = 1$ beendet wird.

Dies liefert die Determinierungsforderungen

```
if rw = 1 and ready = 1 then determined(rdata) end if;
if sd_ctrl = write then determined(sd_wdata) end if;
if sd_ctrl = read or sd_ctrl = write or sd_ctrl = activate
            then determined(sd_addr) end if;
```

Die Determinierungsannahmen des SDRAM Interfaces verlangen fortwährende Determiniertheit des $request$-Signals. Das rw-Signal und die $address$-Leitung des Prozessors wird nur als determiniert angenommen, wenn ein Request vorliegt. Schreibdaten auf dem Signal $wdata$ dürfen nur dann als determiniert angenommen werden, wenn es sich um einen Write-Request handelt. Das SDRAM liefert seine Lesedaten nur dann, wenn es in ausreichendem Zeitabstand vorher ein read-Kommando erhalten hat. Das liefert die Determinierungsannahmen

```
determined (request);
if request = 1 then determined(rw) end if;
if request = 1 then determined(address) end if;
```

```
if request = 1 and rw = 0 then determined(wdata) end if;
if prev(sd_ctrl, 2) = read then determined (sd_rdata) end if;
```

Dass die Eigenschaft read_new_row aus Abschnitt 3.2.5 das sichtbare Register *last_row* determinieren soll, wird durch die lokale Determinierungsbedingung

```
at t+9:     determined(last_row)
```

verlangt.

3.4.3 Vollständigkeitsprüfer

Zentraler Bestandteil dieser Arbeit ist ein zum Patent angemeldeter Vollständigkeitsprüfer [Bormann/Busch 2005], der einen Operationsautomaten analysiert und darin Verifikationslücken aufdeckt oder die Abwesenheit von Verifikationslücken bestätigt. Das Benutzungskonzept entspricht dem eines Property Checkers. Wie ein Property Checker vom Benutzer Annahmen und Beweisziele in Form einer Eigenschaft erhält und diese gegen ein Modell prüft, so erhält der Vollständigkeitsprüfer Determinierungsannahmen und Determinierungsforderungen, und prüft diese gegen einen Operationsautomaten. Dabei wird untersucht, ob die Eigenschaften nur Ausgabetraces akzeptieren, die den Determinierungsforderungen entsprechen, sofern die Eingabetraces den Determinierungsannahmen entsprechen.

Die Untersuchung wird häufig auf Eingabetraces beschränkt, die gewisse Umgebungsannahmen erfüllen. Die Umgebung kann durch Constraints beschrieben sein, oder Teil der Verifikation sein. In letzterem Fall werden diese Umgebungsannahmen durch Assertions beschrieben. Als Sammelbegriff für die Umgebungsannahmen beim Beweis der Vollständigkeit einer Menge von Operationseigenschaften wird im Folgenden der Begriff lokaler Constraint verwandt.

Technisch stellt der Vollständigkeitsprüfer zunächst sicher, dass zu jedem Eingabetrace eine Folge von Operationen gefunden werden kann, die den Eingabetrace überdeckt. Diese Folge wird Kette genannt. Entlang jeder Kette stellt der Vollständigkeitsprüfer in weiteren Arbeitsschritten fest, dass die Ausgabesignale entsprechend der Determinierungsforderungen determiniert sind.

Der Vollständigkeitsprüfer wurde unter Verwendung eines Eigenschaftsprüfers implementiert und speziell für die Prüfung eines Operationsgraphen bzw. einer Menge von Operationseigenschaften entwickelt. Auf diese Weise können Operationsgraphen von Schaltungen industriell relevanter Größe überprüft werden. Eine detaillierte Beschreibung des Vollständigkeitsprüfers findet sich in Abschnitt 0 zusammen mit einer detaillierten Würdigung des Stands der Technik.

In der Praxis gibt der Vollständigkeitsprüfer Gegenbeispiele aus, die den Benutzer auf Verifikationslücken hinweisen. In vielen Fällen werden zwei Abläufe gezeigt, die die Determinierungsannahmen erfüllen, aber die Determinierungsforderungen verletzen. In anderen Fällen besteht das Gegenbeispiel nur aus einem Trace. Dieser zeigt, dass die Operationen bzw. Operationseigenschaften zulassen, dass die Schaltung in einen konzeptionellen Zustand gerät, in dem es keine Fortsetzung der Kette der Operationen gibt. Das wird durch Angabe eines Eingabetraces demonstriert, der zwar die Constraints der Verifikation erfüllt, aber die Eingabebedingungen aller Operationen verletzt, die in diesem konzeptionellen Zustand starten. Der Vollständigkeitsprüfer erzeugt dabei recht detaillierte Fehlerbilder, weil er mit den Operationen bzw. Operationseigenschaften verschiedene Tests durchführt, deren Verletzung auf unterschiedliche Fehlerursachen hindeuten.

Die in den Gegenbeispielen angezeigten Verifikationslücken leiten den Benutzer an, die Verifikation zu vervollständigen. In vielen Fällen gehen die Verifikationslücken aber auch auf Spezifikationslücken zurück, sodass nicht nur die Qualität der Schaltung, sondern auch die Qualität der Spezifikation von der Vollständigkeitsprüfung profitiert.

Wenn der Vollständigkeitsprüfer die Determinierungsforderungen unter Voraussetzung der Determinierungsannahmen beweisen kann, wird die Menge von Eigenschaften als vollständig bezeichnet. Dieses Vollständigkeitskriterium löst das von Anwendern und in der Literatur immer wieder benannte Problem [Mitra 2008], die Qualität einer Menge von Eigenschaften zu bestimmen.

Im Kontext einer Verifikation liefert der Vollständigkeitsprüfer ein logisches Kriterium dafür, dass eine Schaltungsverifikation abgeschlossen wurde. Ein derartiges Kriterium wurde vorher nie in der industriellen Schaltungsverifikation angewandt. Die Erfüllung des Kriteriums kann detailliert dokumentiert werden und ist jederzeit nachvollziehbar. Dies liefert einen wesentlich belastbareren Qualitätsnachweis für die Schaltung, als mit simulationsbasierten Verfahren möglich.

3.4.4 Verifikation komplexer Assertions durch den Vollständigkeitsprüfer

In Abschnitt 2.8 wurde bereits herausgestellt, dass Assertions auch für die vollständige Verifikation eine wichtige Bedeutung haben, weil damit punktuelle Verifikationsziele und wichtige Zwischenergebnisse während der vollständigen Verifikation ausgedrückt werden. In diesem Abschnitt wurde auch bereits darauf hingewiesen, dass Assertions für die Untersuchung mit ABFV-Werkzeugen zu komplex werden können. In Abschnitt 3.3.5 wurde dargestellt, dass der direkte Beweis von Assertions per IPC die Bestimmung von Erreichbarkeitsbedingungen erfordert, und dies manchmal prohibitiv aufwändig sein kann.

Ein Ausweg ergibt sich bei Vorliegen eines vollständigen Satzes von Operationseigenschaften. Diese akzeptieren zu jedem Eingabetrace nur eine beschränkte Menge von Traces der internen und Ausgabesignale. Wenn die Assertion auf all diesen Traces gilt, und die Eigenschaften auf der Schaltung gelten, dann gilt die Assertion auch auf der Schaltung.

Auf dieser Einsicht beruht eine Erweiterung des Vollständigkeitsprüfers. Die Grundidee besteht darin, die Assertion wie ein zusätzliches Ausgabesignal zu behandeln, von dem nachgewiesen wird, dass es von den Operationseigenschaften immer determiniert wird und dabei immer anzeigt, dass die Assertion erfüllt ist. Entsprechend bildet die vollständige Menge der Operationseigenschaften ein der Schaltung angepaßtes Induktionsschema zur Verifikation der Assertion [Sheeran/Singh/Stalmarck 2000].

Mit diesem Analysator für komplexe Assertions wird die mit dem Assertionbeweis verbundene, schwierige Erreichbarkeitsanalyse auf die für den Beweis von Operationseigenschaften erforderliche Erreichbarkeitsanalyse zurückgeführt, die nach den Überlegungen aus Abschnitt 3.3 deutlich einfacher ist.

3.5 Kompositionale Vollständige Verifikation

Die kompositionale vollständige Verifikation [Beyer/Bormann 2008] geht von mehreren Eigenschaftsmengen aus, die jeweils einen Teil einer Gesamtschaltung vollständig verifizieren, und untersucht, ob deren Vereinigung die Gesamtschaltung vollständig verifiziert. Diese Aufgabe erscheint nur auf den ersten Blick trivial. Ein zweiter Blick enthüllt, dass die Aufgabe im

Allgemeinen nur demjenigen trivial erscheint, der zur Begründung der Trivialität einen Zirkelschluss benutzt. Wenn nämlich ein Modul unter dem Constraint α eine Assertion β und ein anderes Modul unter dem Constraint β die Assertion α erfüllt, darf im Allgemeinen nicht geschlossen werden, dass die Gesamtschaltung die Assertions α und β erfüllt. Ein Gegenbeispiel dazu wird in Abschnitt 6.4.3 vorgestellt.

Bei den Untersuchungen spielen Integrationsbedingungen genannte Beschreibungen der Interfaces der Eigenschaftsmengen eine zentrale Rolle, die in Abschnitt 3.5.1 eingeführt werden. Integrationsbedingungen gibt es sowohl für solche Eigenschaftsmengen, deren Vollständigkeit mit dem Vollständigkeitsprüfer nachgewiesen wurde, als auch für Eigenschaftsmengen, die nach den Regeln der kompositionalen vollständigen Verifikation gebildet wurden. Entsprechend wird hier ein hierarchisches Verfahren zur vollständigen Verifikation beliebig großer Schaltungen entwickelt.

Damit sich die vollständigen Eigenschaftsmengen der Teilschaltungen zu einer vollständigen Eigenschaftsmenge der Gesamtschaltung vereinigen lassen, müssen die Eigenschaften auf geeigneten Schaltungsmodellen bewiesen worden sein und die Integrationsbedingungen der Teilschaltungen und der Gesamtschaltung gewisse Forderungen erfüllen. Dazu gehören Forderungen an die einzelnen Integrationsbedingungen und Forderungen an das Zusammenspiel der Integrationsbedingungen.

Da jede Schaltung üblicherweise Teil eines größeren Systems ist, sollten die Forderungen an die einzelnen Integrationsbedingungen auch dann erfüllt werden, wenn gar keine kompositionale Verifikation durchgeführt werden soll. Entsprechende Tests bieten sich also als Plausibilitätstests während jeder vollständigen Verifikation an und stellen eine attraktive Alternative zu der in der ABFV üblichen Prüfung von Constraints auf Widerspruchsfreiheit dar. Die Tests werden übersichtsweise in Abschnitt 3.5.3 und detailliert in Abschnitt 6.2 vorgestellt.

Die Forderungen an das Zusammenspiel der Integrationsbedingungen ergeben sich exemplarisch aus zwei Anwendungsgebieten der kompositionalen vollständigen Verifikation. Beim IP-basierten Entwurf eines System-On-Chip (SoC) wird damit die Kommunikation zwischen den IP-Blöcken verifiziert. Dies wird in Abschnitt 3.5.5 vorgestellt.

Die vollständige Verifikation von Modulen, wie z.B. den IP-Blöcken selbst, muss häufig mit mehreren nebenläufig arbeitenden Operationsautomaten umgehen, die durch verschiedene Schaltungsteile implementiert werden. Diese Schaltungsteile werden Cluster genannt und in Abschnitt 3.5.6 beschrieben. Die Verifikation einer aus mehreren Clustern bestehenden Schaltung wird in Abschnitt 3.5.7 dargestellt.

Kompositionale vollständige Verifikation wird detailliert in Kapitel 6 behandelt. Die zugehörige mathematische Theorie wird im Anhang dargelegt.

3.5.1 Integrationsbedingungen

Eine vollständige Verifikation wird unter Constraints und Determinierungsannahmen geführt. Zu den Constraints gehören die Voraussetzungen über das Eingabeverhalten einer Schaltung, unter denen die einzelnen Eigenschaften bewiesen werden und die Voraussetzungen, die als Teil der Dependencies bei der Vollständigkeitsprüfung eingesetzt werden. Beide zusammen legen Rechenschaft darüber ab, unter welchen Annahmen an die Protokolle der Interfaces die vollständige Verifikation der Schaltung durchgeführt wurde. Entsprechend werden diese Informationen Integrationsannahmen genannt.

Über die Ausgangssignale einer Schaltung werden Assertions und Determinierungsforderungen bewiesen. Diese Angaben beschreiben also, welches Protokoll die Ausgangssignale der Schaltung einhalten. Entsprechend werden diese Informationen Integrationszusicherungen genannt.

Integrationsannahmen und –zusicherungen werden zu Integrationsbedingungen zusammengefasst. Diese stellen formale Beschreibungen des von einer Schaltung unterstützen Protokolls dar. Eine Besonderheit dieses Ansatzes besteht darin, dass durch Integrationsannahmen adäquat ausgedrückt werden kann, wann ein Kommunikationspartner Daten und Adressen von seinen Eingangssignalen liest, bzw. wann er sie auf den Ausgabesignalen versendet. Dies schafft die Voraussetzung für die Prüfung, dass jeder Kommunikationspartner in einem SOC nur dann Daten und Adressen liest, wenn sie auch vom anderen Kommunikationspartner gesendet werden. Diese Besonderheit ist für die Prüfung der Kommunikation zwischen den IP-Blöcken fundamental und wurde bei keinem anderen Verfahren zur formalen Verifikation der Kommunikation zwischen IP-Blöcken angetroffen.

3.5.2 Beispiel

Die Integrationsbedingungen des SDRAM-Interfaces aus Abschnitt 3.2.1 ergeben sich aus den Constraints und den Determinierungsannahmen bei seiner vollständigen Verifikation und haben die Form

```
integration_condition of sdram_if is

assume:
      reset = '0';

      -- constraint processor_protocol
      if request = '1' and ready = '0' then
            next(request) = '1' and
            next(address) = address and
            next(rw) = rw and
            if rw = '0' then next(wdata) = wdata;
      end if;

      -- Determinierungsannahmen des SDRAM-IFs
      determined (request);
      if request = 1 then determined(rw) end if;
      if request = 1 then determined(address) end if;
      if request = 1 and rw = 0 then determined(wdata) end if;
      if prev(sd_ctrl, 2) = read then determined (sd_rdata) end if;

guarantee:
      -- Eine mögliche Assertion über Pulse auf dem Ready-Signal
      if prev(ready) = '1' then ready = '0'; end if;

      -- Determinierungsforderungen des SDRAM-IFs
      if rw = 1 and ready = 1 then determined(rdata) end if;
      if sd_ctrl = write then determined(sd_wdata) end if;
      if sd_ctrl = read or sd_ctrl = write or sd_ctrl = activate
            then determined(sd_addr) end if;

end integration_condition;
```

Die Integrationsannahme zeigt, welches Protokoll das SDRAM-Interface unterstützt. Dies wird mit den Constraints aus Abschnitt 3.2.5 und den Determinierungsannahmen aus Ab-

schnitt 3.4.2 ausgedrückt. Das Prozessorprotokoll verlangt im Wesentlichen, dass der Prozessor seine Anforderung aufrecht erhält, bis das Peripheral anzeigt, dass es den Request bedient hat. Die Determinierungsannahme drückt aus, wann die Eingangssignale vom SDRAM-Interface ausgewertet werden.

Als Beispiel wurde in die Integrationszusicherung eine Assertion darüber eingefügt, dass das SDRAM-Interface auf dem ready-Signal Pulse erzeugt, dass also ready niemals zwei Takte hintereinander aktiv ist. Eine solche Assertion bestätigt, dass die Spezialität des Prozessorprotokolls, auch sofortige (kombinatorische) Reaktionen eines Peripherals zuzulassen, hier nicht genutzt wird. Die Assertion könnte wichtig werden, wenn beispielsweise eine Prozessorversion zum Einsatz kommt, die mit sofortigen Reaktionen von Peripherals nicht umgehen kann.

Weitere Assertions könnten das Protokoll zum SDRAM selbst betreffen und etwa die Korrekte Abfolge von Kommandos auf sd_ctrl oder auch die Einhaltung von Zeitbedingungen zwischen Kontrollkommandos zusichern.

3.5.3 Plausibilitätstests

Integrationsannahmen sollten nur solche Eingabetraces beschreiben, die die Schaltung auch wirklich empfangen kann, wenn sie Teil eines größeren Systems ist. Das ist völlig unproblematisch, wenn in der Integrationsannahme keine Ausgangs- und internen Signale der untersuchten Schaltung vorkommen.

Wenn sie aber doch vorkommen, wird die Integrationsannahme reaktiv genannt. Reaktive Integrationsannahmen können Verhalten postulieren, das kein umgebendes System einhalten kann. Beispielsweise könnte eine reaktive Integrationsannahme nicht nur die Eingangssignale der zu untersuchenden Schaltung beschränken, sondern gleich auch noch gewisse Werte auf Ausgangssignalen verbieten, obwohl es dazu gar keine Entsprechung in der Physik einer Digitalschaltung gibt. Außerdem können die Integrationsannahmen ein Verhalten des umgebenden Systems postulieren, das es nur aufweisen würde, wenn es nicht kausal wäre, d.h. wenn es hellseherische Fähigkeiten hätte. Drittens könnte die Integrationsannahme einen kombinatorischen Pfad im umgebenden System erforderlich machen, der mit einem kombinatorischen Pfad der untersuchten Schaltung zusammen eine kombinatorische Schleife bildet.

Eine der Voraussetzungen für die kompositionale Verifikation ist, dass die Integrationsannahmen auf keine der oben genannten Weisen pathologisch sein dürfen. Für die Formulierung dieses Kriteriums wurde eine Fassung gefunden, die auch für Nicht-Logiker intuitiv ist: Es muss mindestens eine Ersatzschaltung geben, die die Integrationsannahme implementiert, und die sich mit der zu untersuchenden Schaltung verbinden lässt, ohne dass es dabei zu zyklischen kombinatorischen Abhängigkeiten kommt. Details werden in Abschnitt 6.2 vorgestellt.

Das Kriterium besteht demnach aus zwei Teilen. Einerseits wird verlangt, dass es überhaupt eine solche Ersatzschaltung gibt. Wenn es sie gibt, wird die Integrationsannahme implementierbar genannt. Die Forderung nach Implementierbarkeit einer Integrationsannahme stellt zum einen sicher, dass sie alle Werte der Ausgangs- und internen Signale der untersuchten Schaltung akzeptiert. Ferner wird sichergestellt, dass der Constraint keine Abhängigkeiten zwischen aktuellen Werten der Eingangssignale und Werten der Ausgangs- oder internen Signale zu zukünftigen Taktzyklen verlangt.

Im zweiten Kriterium wird verlangt, dass sich keine kombinatorischen Schleifen bilden dürfen, wenn diese Ersatzschaltung mit der zu untersuchenden Schaltung verbunden wird. Dieses Kriterium wird strukturelle Kompatibilität genannt. Strukturelle Kompatibilität kann als Ver-

feinerung der Forderung nach Kausalität verstanden werden. Diese Verfeinerung betrifft kombinatorische Abhängigkeiten zwischen Signalen. Solange es keine kombinatorischen Schleifen in einer Schaltung gibt, gibt es eine Reihenfolge, in der sie innerhalb eines Taktes auseinander berechnet werden können. Es gibt also auch zwischen den kombinatorisch abhängigen Signalen eine Kausalitätsbeziehung und die Forderung nach struktureller Kompatibilität sorgt nur dafür, dass die Integrationsannahme zu dieser Kausalitätsbeziehung passt.

In Abschnitt 6.2 werden Tests vorgestellt, die hinreichende Bedingungen dafür implementieren, dass eine Integrationsannahme implementierbar und strukturell kompatibel ist zu der untersuchten Schaltung. Diese Tests werden regelmäßig als Plausibilitätskriterien angewandt, mit denen der Entwertung einer Verifikation durch ungeeignete Integrationsannahmen vorgebeugt werden soll.

Sie übernehmen damit eine Rolle, die bei der ABFV durch Plausibilitätstests eingenommen wird, bei denen Constraints mit oder ohne Berücksichtigung des Modells auf Widerspruchsfreiheit untersucht werden. Im Vergleich sind die Plausibilitätstests der ABFV wesentlich unschärfer. Letztere schlagen beispielsweise erst an, wenn der Constraint alle Ausgabewerte verbietet, und sie sind blind für nicht-kausale Abhängigkeiten zwischen Ausgangs- und Eingangssignalen der untersuchten Schaltung.

3.5.4 Beispiele zurückgewiesener Integrationsannahmen

Um den Wert der Tests auf Implementierbarkeit und strukturelle Kompatibilität zu demonstrieren, werden nachfolgend Integrationsannahmen präsentiert, die davon zurückgewiesen werden:

```
integration_condition of example1 is
assume:
      if next(outsig) then determined(insig) end if;
guarantee: …
end integration_condition;
```

würde zurückgewiesen werden, weil die Determinierungsannahme nicht kausal ist.

Die Integrationsannahme

```
integration_condition of example2 is
assume:
      determined(insig);
      insig = outsig+1;
guarantee: …
end integratin_condition;
```

würde zurückgewiesen werden, wenn insig und outsig gleich breite Bitvektoren sind. Dann verbietet die Integrationsannahme nämlich, dass outsig jemals "1111...1111" wird, weil ITL die Addition numerisch ausführt und das Additionsergebnis nicht mehr in insig gespeichert werden kann.

Für die Schaltung nach Abbildung 10 wird die Integrationsbedingung

```
integration_condition of example3 is
assume:
      determined(in1);
      determined(in2);
      in2 = sum;
```

```
guarantee: …
end integratin_condition;
```

zurückgewiesen, weil die Integrationsannahme nicht strukturell kompatibel ist mit dem untersuchten Modell. Beim Zusammenfügen jeder möglichen Ersatzschaltung mit dem Halbaddierer entsteht nämlich die in Abbildung 10 gestrichelt angedeutete zusätzliche kombinatorische Abhängigkeit, die eine kombinatorische Schleife verursacht. Dies aufzudecken ist vorteilhaft, weil sich unter dem Constraint die doch ungewöhnliche Assertion

```
assertion unusual := out_sig = '0';
```

beweisen lässt. Denn aus $sum = in1\ xor\ in2$ ergäbe sich mit $in1 = 1$ die Beziehung $sum = not(in1)$, was dem Constraint widerspricht. Also muß $in1 = 0$ sein, wodurch die Assertion bewiesen wird. Der Beweis dieser Assertion erscheint aber widersinnig, weil es keine Schaltungsumgebung gibt, die nur mit den Signalen sum und in2 verbunden ist, und die damit das Signal in1 oder out_sig beeinflußt.

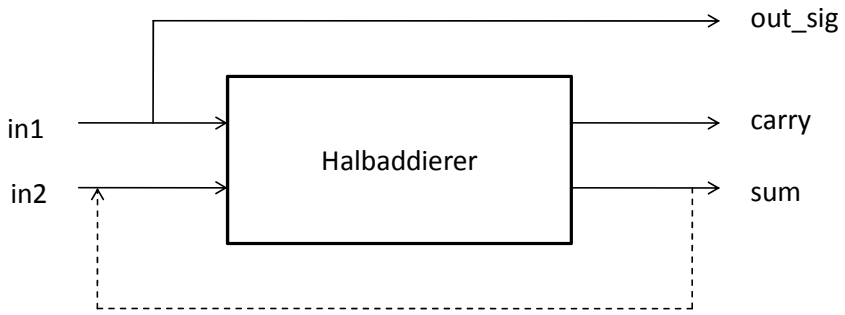

Abbildung 10: Strukturell pathologischer Fall

3.5.5 IP-basierter SoC-Entwurf

Eine Strategie zur schnellen Entwicklung von integrierten Schaltungen ist der IP-basierte Entwurf eines System-on-Chip (SoC). Hier werden Schaltungsblöcke vorab entwickelt, verifiziert und in einer IP-Bibliothek zur Verfügung gestellt, ohne dass es dabei endgültige Pläne für die Entwicklung eines speziellen ASIC geben muss. Solche Schaltungsblöcke sind beispielsweise Prozessoren, Arbiter, Speichercontroller oder Bussysteme. Für derartige Module hat sich der Begriff IP-Block (IP = intellectual property) eingebürgert, der darauf verweist, dass das Endprodukt ihrer Entwicklung nicht mehr Siliziumschaltungen sind, sondern lediglich die Fertigungsunterlagen für diese IP-Blöcke. Gefertigt werden diese Blöcke erst als Teil eines SoC, das irgendwann später von einem anderen Entwicklerteam aus den Fertigungsunterlagen verschiedener IP-Blöcke zusammengestellt wird.

Die IP-Entwickler werden neben den Fertigungsunterlagen des IP, d.h. dem RTL, Synthesescripten usf. viele weitere Informationen anbieten. Darunter sind Integrationsinformationen darüber, wie sich die Eingabesignale des IP-Blocks verhalten dürfen, damit er richtig funktioniert oder Beschreibungen über das Verhalten der Ausgabesignale. Häufig bestehen die Integrationsinformationen darin, die Protokolle zu benennen, die das Modul unterstützt. Möglicherweise wird zusätzlich angegeben, dass die Schaltung bestimmte Spezialitäten des Protokolls entweder nicht versteht oder nicht ausnutzt. Die Integrationsinformationen enthalten damit ähnliche Informationen wie die sich aus vollständigen Verifikationen ergebenden Integrationsbedingungen.

Im IP-basierten SoC-Entwurf ist die Verifikation des IP-Blockes selbst Sache der IP-Entwickler. Diese Modulverifikation muss neben der korrekten Funktion auch die Korrektheit der Integrationsinformationen nachweisen. Dies geschieht vor allem mit den Mitteln der Simulation und ist daher entsprechend unsicher.

Die IP-Nutzer treffen anhand der Integrationsinformationen eine Vorauswahl über die in Frage kommenden IP-Blöcke. Nach der Verschaltung der IP-Blöcke (durch entsprechenden RTL-Code) wird aber simuliert, um einerseits die problemlose Interaktion mit den Nachbarmodulen zu bestätigen, und um andererseits sicherzustellen, dass das Zusammenwirken der IP-Blöcke tatsächlich die von der Spezifikation verlangte Funktion hat.

Die Integrationsbedingungen ergeben sich wie oben beschrieben direkt aus der vollständigen Verifikation. Sie sind präzise und formal bewiesene Beschreibungen der unterstützten Protokolle, die sich automatisch miteinander in Beziehung setzen lassen, bevor viel Aufwand in den Aufbau des SoC geflossen ist. Auf diese Weise ist eine formale Verifikation der Kommunikation im SoC möglich, die wie bei der Simulation auf die IP- und die SoC-Entwickler verteilt ist: Die IP-Entwickler stellen einerseits mit einer vollständigen Verifikation die Korrektheit der Integrationsbedingungen sicher. Dabei prüfen sie auch, dass die Integrationsannahmen implementierbar und mit ihren Schaltungen strukturell kompatibel sind.

Die SoC-Entwickler stellen sicher, dass die Integrationsannahmen der Gesamtschaltung implementierbar und mit der Gesamtschaltung strukturell kompatibel sind. Sie müssen ferner die strukturelle Kompatibilität der Integrationsannahmen der Teilblöcke mit der Gesamtschaltung sicherstellen. Und sie müssen natürlich feststellen, ob die beteiligten Integrationsbedingungen zusammenpassen. Dazu müssen einerseits die Integrationsannahmen aller Blöcke unter Berücksichtigung der Verschaltung durch Integrationszusicherungen ihrer Nachbarblöcke oder durch die Integrationsannahmen der Gesamtschaltung erfüllt werden. Andererseits müssen die Integrationszusicherungen der Gesamtschaltung durch die Integrationsannahmen der Gesamtschaltung und die Integrationszusicherungen der Blöcke gerechtfertigt werden. Die mathematische Formulierung dieser Forderungen wird in Abschnitt 6.3.2 präsentiert.

Wenn das SDRAM-Interface, dessen Integrationsbedingung in Abschnitt 3.5.2 vorgestellt wurde, den Prozessor mit Instruktionen versorgen soll und daher keine Daten schreiben muss, könnte der Prozessor etwa die folgenden Integrationszusicherungen haben:

```
integration_condition of processor is
assume:
      …
guarantee:
      p_rw = '1' and
      if p_request = '1' and p_ready = '0' then
            next(p_request) = '1' and
            next(p_address) = p_address
      end if;

      if p_request = '1' then determined(p_address) end if;
      …
end integration_condition;
```

Das Präfix p_ vor den Signalen soll daran erinnern, dass die Ausgangssignale des Prozessors zunächst nichts mit den Eingangssignalen des SDRAM-Interfaces zu tun haben. Die Prozessorsignale müssen also geeignet substituiert werden, bevor die Integrationstests durchgeführt

werden können. Nach dieser Substitution lässt sich der Test erfolgreich durchführen. Dabei erscheint interessant, dass die Determinierungsannahmen des SDRAM Interfaces über rw durch die Assertion über p_rw entlastet wird.

3.5.6 Cluster

Zur vollständigen Verifikation einer größeren Schaltung kann es erforderlich werden, Schaltungsteile separat zu verifizieren. Das ist immer dann der Fall, wenn die Gesamtschaltung nicht eine Folge von Operationen nacheinander abarbeitet, sondern wenn Schaltungsteile nebenläufig Operationen abarbeiten. In diesem Fall muss die Schaltung durch mehrere Operationsautomaten verifiziert werden.

Die Schaltungsteile, die durch einen Operationsautomaten verifiziert werden, heißen Cluster. Zwei Schaltungselemente gehören unterschiedlichen Clustern an, wenn sie z.B. Events aus zwei unabhängigen Quellen bearbeiten.

Viele Schaltungen, z.B. die IP-Blöcke aus dem vorherigen Abschnitt 3.5.5, bestehen aus mehreren Clustern. Prozessorperipherals haben z.B. häufig einen Konfigurationsblock, der aus den Konfigurationsregistern besteht und aus einem Slaveinterface zum Beschreiben oder Auslesen dieser Konfigurationsregister. Dieser Block wird häufig in einem eigenen Cluster verifiziert, während die eigentliche Funktion des Peripherals in einem oder mehreren weiteren Clustern verifiziert wird.

Bei Prozessoren wird häufig die gesamte Pipeline in einem Cluster untersucht. Wenn es aber eine etwas anspruchsvollere Prefetcheinheit gibt, so arbeitet die nicht nach dem Rhythmus der Pipeline, sondern sie führt ein Eigenleben, das z.B. von der Zuteilung des Instruktionsbusses gesteuert wird. Entsprechend gibt es dann Fifos zwischen der Prefetcheinheit und der Decode-Stufe der Prozessorpipeline, mit denen die unterschiedlichen Rhythmen ausgeglichen werden. In diesem Fall werden die Prefetcheinheit und die Prozessorpipeline ab der Decode-Stufe in unterschiedlichen Clustern verifiziert.

Ein weiterer Grund zur Aufteilung einer Schaltung in Cluster kann auch die Schaltungskomplexität sein, doch kann IPC recht große Cluster behandeln, wie etwa eine komplette Prozessorpipeline, sodass die Aufteilung der Schaltung in Cluster meist den funktionalen Erfordernissen folgt und nicht durch Komplexitätsprobleme verursacht wird.

3.5.7 Vollständige Verifikation von Schaltungen mit mehreren Clustern

Eine Schaltung mit mehreren Clustern wird vollständig verifiziert, indem die vollständige Verifikation auf die Cluster aufgeteilt wird. Es ist naheliegend, für jeden Cluster ein eigenes Modell zu bauen, dort die Verifikation durchzuführen und die Cluster anschließend wie zuvor zusammenzufassen. Diese Vorgehensweise erfordert aber detaillierte Integrationsannahmen, die alles Verhalten von Nachbarclustern erfassen, das zur korrekten Funktion des untersuchten Clusters erforderlich ist. Entsprechend detailliert müssen auch die Integrationszusicherungen sein. Die passende Aufstellung dieser Angaben ist aufwändig.

Um Aufwand zu sparen, wurde eine Variante der kompositionalen Verifikation entwickelt, bei der die Verifikationen der einzelnen Cluster auf einem Modell durchgeführt wird, in dem manche Clustereingänge interne Signale bleiben dürfen. Dadurch werden diese Clustereingänge durch diejenigen Schaltungsteile beschränkt, die sie erzeugen. Die entsprechenden Einschränkungen müssen dann nicht explizit in der Integrationsannahme des untersuchten Clusters auftauchen. Sie können ganz eingespart werden, oder es sind nur Teilaspekte ihrer Funktion zu beschreiben.

Zur Durchführung des Verfahrens wird zunächst der Clustergraph analysiert. Die Knoten dieses Graphen sind Cluster, und wenn ein Cluster ein Signal produziert, das in einem anderen Cluster weiterverarbeitet wird, gibt es in diesem Clustergraphen eine Kante von dem einen zum anderen Cluster.

Im Clustergraph müssen Kanten bestimmt werden, durch deren Elimination der Clustergraph azyklisch wird. Die Verifikation muss auf einem Modell durchgeführt werden, bei dem diese Signale durchgeschnitten sind. Das hier erforderliche Schneiden entspricht anschaulich der entsprechenden Operation mit der Kneifzange. Die durch das Schneiden entstandenen Enden des Signals werden entsprechend dem Informationsfluss zu neuen primären Aus- und Eingängen des zu untersuchenden Modells.

Die vollständige Verifikation wird dann für jeden Cluster separat durchgeführt, aber auf dem oben beschriebenen Modell, das möglicherweise immer noch alle Cluster enthält. Dabei werden viele Clustereingänge interne Signale bleiben, die von anderen Clustern erzeugt werden. Die vollständige Verifikation jedes Clusters ergibt Integrationsbedingungen für diesen Cluster. Wenn in diesen Integrationsbedingungen ungeschnittene Clustereingänge durch reaktive Constraints, d.h. unter Verwendung von Ausgangssignalen des Clusters beschrieben werden, müssen diese Clustereingänge auch noch aufgeschnitten werden, und die entsprechenden Integrationsannahmen ggfs. verstärkt werden. Die Integrationsannahmen über neue, durch Schneiden entstandene Eingangssignale müssen dabei implementierbar und strukturell kompatibel sein.

Anschließend wird wiederum festgestellt, ob die Integrationsannahmen aller Cluster durch Integrationszusicherungen ihrer Nachbarcluster oder durch die Integrationsannahmen der Gesamtschaltung erfüllt werden. Dabei werden alle durchs Schneiden entstandenen neuen Ausgangssignale durch die neuen Eingangssignale substituiert.

Außerdem müssen die Integrationszusicherungen der Gesamtschaltung durch die Integrationsannahmen der Gesamtschaltung und die Integrationszusicherungen der Cluster gerechtfertigt werden.

Die Besonderheit bei dieser Vorgehensweise ist die Verwendung eines Modells, in dem einige Signale zwischen Clustern zwar durchgeschnitten sind, aber andere intakt bleiben können. Diese Schnitte sind erforderlich, um den Nachweis führen zu können, dass die vollständige Verifikation der Gesamtschaltung tatsächlich auf die vollständige Verifikation ihrer Cluster aufgeteilt wurde. Es ist aber ein Vorteil der hier entwickelten Theorie, dass die Verifikation auf einem Modell durchgeführt werden kann, in dem noch viele andere Eingangssignale von Clustern interne Signale sind. Denn das Verhalten solcher interner Signale wird durch den Nachbarcluster beschränkt, durch den sie erzeugt werden, auch wenn die Eigenschaften gar nicht über den Nachbarcluster sprechen. Eine solche Beschränkung wird die Verifikation im Allgemeinen vereinfachen.

Dies wird im Vergleich zum Beispiel in Abschnitt 3.5.2 deutlich, wenn angenommen wird, dass das SDRAM-Interface zum Prozessor dazugehört und mit ihm entwickelt und verifiziert wird. Prozessor und SDRAM-Interface bilden dann zwei Cluster und ihr Clustergraph ist zyklisch. Es bietet sich an, die Kante vom Prozessor zum SDRAM-Interface zu eliminieren, um den Clustergraphen azyklisch zu machen. Dann wird die Verifikation von Prozessor und SDRAM-Interface auf einem gemeinsamen Modell durchgeführt, in dem die Signale ready und rdata vom SDRAM-Interface zum Prozessor interne Signale sind. Die Verifikation wird

nach wie vor clusterweise durchgeführt, aber eben auf dem gemeinsamen Modell mit den wenigen Schnitten. Dadurch ist die Bedingung über die Pulse auf dem ready-Signal in den Integrationsbedingungen nicht erforderlich, die in Abschnitt 3.5.2 notwendig war, um die korrekte Prozessorfunktion sicherzustellen.

3.6 Vollständige Verifikation im Industriellen Einsatz

Operationsautomaten und -eigenschaften, IPC und der Vollständigkeitsprüfer gehen in diesem Ansatz eine vorteilhafte Symbiose ein, die sich über fast 10 Jahre in der industriellen Praxis bewährt hat und in dem Produkt OneSpin 360 MV [OneSpin o.J.] verfügbar gemacht wurde. Der Vollständigkeitsprüfer kann durch die Konzentration auf Operationen und ihre Eigenschaften einfachere Beweisziele prüfen, und damit auch Eigenschaften über große Schaltungen untersuchen. IPC ist besonders geeignet für Operationseigenschaften, insbesondere auch deshalb, weil dabei relativ wenige Erreichbarkeitsbedingungen erforderlich werden. Wenn aber solche Bedingungen gebraucht werden, so erlaubt der Ansatz auch Benutzervorgaben, durch die ein höheres Verständnis über die Schaltung in die Verifikation eingebracht werden kann. Bei benutzerdefinierten Erreichbarkeitsbedingungen sorgt der Vollständigkeitsprüfer dafür, dass sie nicht ungeprüft in die Verifikation eingebracht werden. Alles in allem ermöglicht die Symbiose die eigenständige Untersuchung von Modulen, auf die sonst die simulationsbasierte Modulverifikation angewandt wird – mit den im Folgenden zusammengefassten Leistungsmerkmalen:

3.6.1 Höchste Qualität der Implementierung

Die in dieser Arbeit mit ihren Grundlagen beschriebene Vorgehenswese ist bereits in vielen Verifikationsprojekten eingesetzt worden. Damit wurden höchste Schaltungsqualitäten erreicht – mit einem deutlich einfacheren Vorgehen als mit Theorembeweisern und mit einer viel höheren Produktivität. In dieser Beziehung ist die vollständige Verifikation die erste ihrer Art. Denn eine korrekt aufgesetzte vollständige Verifikation lässt im Gegensatz zur Simulation keine funktionalen Fehler mehr unentdeckt [Büttner 2007]: Simulation hingegen übersieht Fehler in nicht stimulierten Situationen; aber auch stimulierte Fehler werden von der Simulation nur entdeckt, wenn der Simulationsstimulus das Fehlverhalten bis zum Interface eines Checkers propagiert, der überdies auch noch dafür empfindlich sein muss. Zum Abmildern solcher Schwachstellen der simulationsbasierten Verifikation werden Verfahren [Grosse/Hampton 2005, Certess o.J.] angeboten, die auf Fehlerinjektion und Mutation Analysis [Offutt/Untch 2000] beruhen.

Die vollständige Verifikation mildert solche Schwachstellen nicht nur ab, sondern sie verhindert sie zuverlässig und automatisch. Der vollständigen Verifikation entgehen nur dann Fehler, wenn die Spezifikation in der selben falschen Weise formalisiert wurde, der auch die Schaltungsbeschreibung folgt, oder wenn die Constraints über die primären Eingänge zu restriktiv formuliert wurden. Dieses Problem teilen alle Verifikationsverfahren, solange von informellen Spezifikationen ausgegangen wird. Bei der vollständigen Verifikation wird dieser Fall durch den systematischen Abgleich zwischen informeller Spezifikation, Formalisierung und Implementierung extrem unwahrscheinlich.

Durch Eigenschaften formalisierte Spezifikationen stehen für die vollständige Verifikation einerseits in Form wiederverwendbarer Assertions über Protokolle zur Verfügung, und andererseits etwa als formalisierte Architekturbeschreibungen für Prozessoren [Bormann/Beyer/Skalberg 2006], von denen ausgehend auch schnelle Simulatoren entwickelt wurden. Wenn von derartigen formalisierten Spezifikationen ausgegangen wird, stellt sich das Problem fälschlich akzeptierter Fehler gar nicht mehr.

3.6.2 Spezifikationsqualität

Bei sorgfältiger Übersetzung der Spezifikation in die Operationseigenschaften identifiziert der Vollständigkeitsprüfer auch Spezifikationslücken und erlaubt damit die Vervollständigung der Spezifikation.

Überhaupt ist der Ansatz recht tolerant gegenüber der Qualität der Spezifikation, was im industriellen Einsatz vorteilhaft ist. In verschiedenen Beispielanwendungen stand außer dem Modulnamen nur wenig niedergeschriebene Information über die Schaltung zur Verfügung. Selbst damit lässt sich noch effizient arbeiten, sofern eine Autorität wie etwa ein Systemarchitekt verbindliche Aussagen über das erwartete Verhalten machen kann. Gibt es eine solche Autorität nicht, reduziert sich die Verifikation auf Plausibilitätstests wie z.B. über die Synchronisation der Schaltung mit ihrer Umgebung. Selbst diese Plausibilitätstests sind wegen der geschlossenen Darstellung von Operationen der Schaltung noch recht mächtige Verifikationsinstrumente.

Die am Ende erstellte Schaltungsbeschreibung durch Operationen manifestiert präzise und auf einem Abstraktionsniveau deutlich oberhalb von RTL, wie die Spezifikation interpretiert wurde.

3.6.3 Constraining

Die vollständige Verifikation ist so angelegt, dass jede Zwischenbehauptung bewiesen werden muss. Constraints werden daher nur über die primären Eingangssignale der vollständig zu verifizierenden Gesamtschaltung benötigt, wo sie auch für simulationsbasierte Verfahren bekannt sein müssen. Die Anwendung der vollständigen Verifikation erlaubt dem Designer, sich auf eine Operation nach der anderen zu konzentrieren, und sich dabei auch immer auf einige wenige Constraints und ihre exakte Ausformulierung zu kümmern. Schon dadurch ist korrektes Constraining für vollständige Verifikation einfacher als für andere formale Verfahren. Darüber hinaus bietet vollständige Verifikation kritischere Plausibilitätsprüfungen über Constraints an, als andere formale Verfahren. Bei der Integration von IP-Blöcken zu SoCs prüft die kompositionale vollständige Verifikation, dass die Constraints der einzelnen IP-Blöcke in der Tat durch die Nachbarblöcke gerechtfertigt werden.

3.6.4 Verifikationsprozess

Vollständige Verifikation ist ein eigenständiges, hochautomatisiertes Verifikationsverfahren, das geeignet ist, weit bessere Qualität zu liefern als simulationsbasierte Verfahren. Formale Vollständige Verifikation ist daher eine Alternative zur Simulation, sobald die eigentliche Verifikationsphase beginnt. Vorher erscheint Simulation unverzichtbar, um Designern ein Gefühl für die Schaltung und ihr Verhalten zu vermitteln.

Anders als bei ABFV addiert sich der Aufwand also nicht zum Aufwand für die simulationsbasierte Verifikation hinzu, sondern die vollständige Verifikation kann das Aufwandsbudget der Simulationsverifikation übernehmen.

Verifikationsplanung [Bormann et al. 2007] für vollständige Verifikation ist eine Projektplanung mit dem Ziel eines Netzplans für die Teilaufgaben, die sich meist an Clustern orientieren. Für die vollständige Abdeckung der Spezifikation sorgt der Vollständigkeitsprüfer während der Ausführung der Verifikation. Der Netzplan ist die Grundlage für das Berichtswesen im Verifikationsprojekt. Verifikationsplanung für Simulation erfordert hingegen viel Umsicht [Bergeron et al 2005], viele Reviews und damit mehr Aufwand.

Bei der Durchführung der vollständigen Verifikation konzentriert man sich auf einzelne Operationen und ihre Interaktion und damit auf ein Abstraktionsniveau ähnlich dem der transaktionsbasierten Verifikation per Simulation. In dieser Beziehung gibt es also den Bruch nicht, der zwischen ABFV und Simulationsverifikation besteht. Die Operationen werden der Spezifikation entnommen, und die Freiheitsgrade der Spezifikation werden ausgefüllt entsprechend den Festlegungen in der Implementierung. Ebenfalls wird der Implementierung entnommen, wie die Operationen miteinander interagieren.

Die Untersuchung der Schaltungsbeschreibung ist damit ein integraler Bestandteil der vollständigen Verifikation. Der Prozess verfolgt dabei das Vorgehen, das ein Ingenieur sowieso anwendet, wenn er oder sie eine Schaltung ausgehend von ihrer Beschreibung kennenlernen und verstehen soll: Man beginnt mit der zentralen Kontrolleinheit der Schaltung und arbeitet sich von dort aus zu einem Verständnis darüber vor, wie die Eingabesignale diese Kontrolleinheit beeinflussen und die Kontrolleinheit den Datenpfad. Dieses Verständnis wird überprüft, indem Zug um Zug die Operationseigenschaften entwickelt, Zwischenstände bewiesen, und die Vollständigkeit der Eigenschaften überprüft wird.

Erfahrungsgemäß ist dabei viel weniger Interaktion mit den Designern vonnöten als in simulationsbasierten Verifikationsprojekten oder bei Anwendung von ABFV [Mitra 2008]. Und trotzdem fallen Fehlerlokalisationen präzise aus bis hin zu funktional geprüften Korrekturvorschlägen.

Mehrjährige Anwendung des Ansatzes [Bormann/Spalinger 2001, Bormann 2003, Winkelmann et al. 2004, Thomas et al. 2004, Bormann/Blank/Winkelmann 2005, Bormann et al. 2007, Loitz et al. 2008] zeigt, dass die Gefahr gering ist, Fehler fälschlicherweise zu akzeptieren, und dass die Sicherheit groß ist, alle Fehler in der Schaltungsbeschreibung zu finden, und damit natürlich viel mehr Fehler als bei simulationsbasierter Verifikation.

3.6.5 Terminierungskriterium

Das Ende einer Verifikation ist erreicht, wenn

- alle Eigenschaften erfolgreich gegen die Schaltungsbeschreibung bewiesen werden,
- wenn der Vollständigkeitsprüfer für jeden Cluster anzeigt, dass die Verifikation vollständig ist,
- wenn sich die vollständigen Verifikationen der einzelnen Cluster zu einer vollständigen Verifikation der Gesamtschaltung zusammenfügen lassen
- und wenn alle vom Designer möglicherweise vorgegebenen Assertions bewiesen sind.

Dies wird in OneSpin 360 MV automatisch geprüft. Damit ist es das erste Verifikationswerkzeug mit einem nicht-heuristischen Terminierungskriterium. Wenn dieses Terminierungskriterium erreicht ist, formen die Eigenschaften und insbesondere die zugehörigen Timingdiagramme eine lesbare und bewiesenermaßen aktuelle Designdokumentation.

Ob die Verifikation dann wirklich Fehlerfreiheit zusichert, hängt davon ab, ob die Eigenschaften tatsächlich die Spezifikation widergeben, und ob die Constraints tatsächlich die Umgebungsbedingungen modellieren, unter denen die Schaltung arbeiten soll. Ersteres kann durch Reviews abgesichert werden, letzteres wird per Simulation oder formal nachgewiesen, wenn die Einbettung der Schaltung in ein größeres System untersucht wird.

3.6.6 Produktivität

Geübte Verifikationsingenieure, die bereit und in der Lage sind, Details von RT-Beschreibungen zu verstehen, erreichen dieses höchste Qualität versprechende Terminierungskriterium deutlich schneller als mit simulationsbasierter Verifikation [Bormann 2003, Bormann/Blank/Winkelmann 2005], die dazu noch geringere Qualität liefert. In den vielen Anwendungsbeispielen wurden von erfahrenen Verifikateuren durchschnittlich 2000 bis 4000 Zeilen RTL-Code pro Personenmonat verifiziert, in Rekordfällen auch 8000 Zeilen pro Personenmonat. Dabei ist auch der Aufwand für die Verifikationsumgebung und die Rechenleistung für die Verifikation weit geringer als bei der Simulation.

Der Ansatz verzichtet auf komplexere benutzerdefinierte Abstraktionen und vermeidet damit einen Grund für die unvorhersehbaren Aufwände, über die R. Mitra [Mitra 2008] klagt. Der Ansatz wird langsamer, wenn die Schaltung konzeptionell unklar, unstrukturiert, und insgesamt schwer verständlich, d.h. im Sinne von Softwareengineering schlecht beschrieben ist. Solch schlechter RTL-Code ergibt sich von vorn herein, wenn die Schaltung unzureichend konzipiert wurde und anschließend aufgrund von Rückmeldungen aus der Verifikation inkrementell verbessert wurde. Schlechter RTL-Code ergibt sich insbesondere dann, wenn eine zur Wiederverwendung bestimmte Schaltungsbeschreibung über die Zeit mehrere Abänderungen durch mehrere Designer erfahren hat. Gerade aber bei schlechtem RTL-Code ist eine vollständige Verifikation besonders nützlich, weil inkrementelle Änderungen besonders anfällig für Corner-Case-Fehler sind und weil die vollständige Verifikation ein Fundament für die Wartung oder Ergänzung der Schaltungsbeschreibung schafft.

3.6.7 Integration mit Simulation

Wenn vollständig verifizierte Module in ein größeres System eingebettet werden, das per Simulation überprüft werden soll, brauchen im Verifikationsplan keine Verifikations- und Coverageziele für diese Module definiert werden. Es muss lediglich hinreichend intensiv geprüft werden, dass das System die Constraints der vollständigen Verifikation einhält. Dies liefert ein pragmatisches Vorgehen für eine technologieübergreifende Verifikationsplanung und Coveragebewertung.

Systemweite Verifikationsziele erfordern eine Kontrolle der Qualität der Simulation durch Coverage. Für die vollständig verifizierten Module bietet sich an, funktionale Coverage auf der Basis der Operationen zu definieren: Die entsprechenden Coveragebedingungen sind erfüllt, wann immer die entsprechenden Operationen ausgeführt werden, d.h. die Annahmen der entsprechenden Operationseigenschaften erfüllt sind. Auf diese Weise kann einerseits verhindert werden, dass die Operationen selbst durch die Systemsimulation ein weiteres Mal intensiv untersucht werden, und andererseits die Definition von funktionaler Coverage über Abläufe, an denen mehrere Operationen beteiligt sind, erleichtert werden.

3.6.8 White-Box-Verifikation

Als Schwachpunkt der vollständigen Verifikation sehen Praktiker die Abhängigkeit von der konkreten Implementierung der Schaltung an. In der Tat müssen Änderungen an der Schaltung meist von Änderungen in den Eigenschaften begleitet werden, weil sonst die Prüfung irgendeiner Eigenschaft fehlschlägt. Diese Anpassung hat meist weitere zur Folge, damit der Eigenschaftssatz wieder vollständig wird. Der Anpassungsaufwand ist bei aufeinanderfolgenden Versionen im Rahmen einer Entwurfsphase meist gering. Er steigt nur an, wenn Schaltungsteile verworfen und einem ganz anderen Konzept folgend implementiert werden.

Ein anderes Bild ergibt sich bei Schaltungs-IP, d.h. wiederverwendbaren Modulen, die über Jahre und viele Revisionen hinweg von verschiedenen Verantwortlichen gepflegt werden. Hier möchte man die Simulation nicht oder nur wenig anpassen, weil dies Aufwand einspart. Diese Einsparung schlägt sich aber als Altern der simulationsbasierten Verifikation nieder, das dadurch gekennzeichnet ist, dass die Verifikation immer weniger kritische Situationen der internen modifizierten Abläufe untersucht und daher immer wahrscheinlicher Fehler übersieht. Wenn dann Fehler vom Produkttest oder gar aus dem Feld gemeldet werden, wird die Unachtsamkeit der Modulverifikation mit der veralteten Verifikationstechnik entschuldigt, obwohl diese bei der ersten Version des Moduls durchaus in der Lage war, eine vernünftige Qualität zu sichern.

Vollständige Verifikation verhindert das Altern der Verifikation. Allerdings erfordert ihre Integration Eingriffe in die Entwicklungs- und Pflegeprozesse der Schaltungs-IP, weil die Änderung der Schaltungsbeschreibung und die der Operationseigenschaften verschränkt werden müssen. Das bietet aber den Vorteil, dass vor jeder Revision eine aktuelle Designdokumentation in Form der Operationseigenschaften vorhanden ist, mit deren Hilfe sich ein (möglicherweise neuer) verantwortlicher Entwickler schnell einarbeiten kann, die vorzunehmenden Änderungen planen kann und die Änderung der Schaltungsbeschreibung verifizieren kann. Experimenten zufolge scheint dies nicht nur zu höchster Qualität der Schaltungs-IP zu führen. Es reduziert auch den Umfang der Änderungen der Schaltungsbeschreibung gegenüber der bisherigen Praxis und trägt zur konzeptionellen Klarheit der geänderten Schaltungsbeschreibung bei, was seinerseits wieder bei erneuten Revisionen hilft.

3.6.9 Hilft vollständige Verifikation gegen das Verification Gap?

Erfüllt vollständige Verifikation den Traum, mit formaler Verifikation das Verification Gap zu verringern? Die in vielen erfolgreichen Anwendungsprojekten nachgewiesenen Sprünge in Qualität und Aufwand legen es nahe. Aber es ist ein disruptiver Ansatz, der eine ganz andere Ausbildung der Verifikationsingenieure benötigt, der bestehenden Verifikationscode nicht nutzen kann, für den es noch wenig Verifikations-IP gibt, und für den Entwurfs- und Verifikationsprozesse verändert werden müssen.

4 Grundlagen

4.1 Synchrone Schaltungen

Ziel aller Bemühungen beim synthesebasierten Entwurf sind digitale Schaltungen, die in FPGAs oder ASICs implementiert werden. Ohne Einschränkung der Allgemeinheit konzentriert sich diese Arbeit auf synchrone Schaltungen mit einem Takt. Solche Schaltungen bestehen aus kombinatorischen Gattern und Flip-Flops. Dabei sollen die kombinatorischen Gatter zyklenfrei verschaltet sein und damit nicht selbst speicherndes Verhalten implementieren. In dieser Arbeit werden nur Schaltungen mit Flip-Flops betrachtet, die bei der steigenden Taktflanke schalten.

Auch bezüglich des Zeitverhaltens der Eingabesignale sollen Besonderheiten ausgeschlossen sein. Daher wird bei diesen Untersuchungen auch angenommen, dass die Schaltung ihrerseits in eine größere Schaltung eingebettet sei, die den vorstehenden Annahmen genügt.

Wie im synchronen Entwurf üblich, wird angenommen, dass die Latenz kombinatorischer Pfade vom Ausgang eines Flip-Flops zum Eingang eines anderen klein genug ist, dass das zweite Flip-Flop den entlang dieses Pfades nach einer Taktflanke neu berechneten Wert an seinem Eingangssignal mit der nächsten Taktflanke aufnehmen kann. Unter diesen Umständen sind sowohl die im synthesebasierten Entwurf üblichen Zero-Delay-Beschreibungen der Schaltung zu rechtfertigen als auch die Abbildung dieser Beschreibungen auf einen Mealy-Automaten.

4.1.1 Automaten

Dieser Mealy-Automat repräsentiert die Werte der Signale der Schaltung kurz vor jeder steigenden Taktflanke [Bormann et al. 1995]. Er wird charakterisiert durch

- eine Menge $\mathbb{i}$ von Eingabevariablen und einer Wertemenge $\mathcal{W}(\mathbb{i})$, die diese Variablen annehmen können. Diese Variablen sollen die Eingabesignale der Schaltung sein.
- eine Menge $\mathbb{s}$ von Zustandsvariablen und eine Wertemenge $\mathcal{W}(\mathbb{s})$ von Zuständen, die diese Variablen annehmen können. Diese Variablen beschreiben die Werte der Flip-Flops in der Schaltung.
- eine Menge $\mathbb{z}$, die aus der Menge der internen Signale $\mathbb{u}$ als auch der Menge der Ausgangssignale $\mathbb{o}$ besteht, und die vom automatentheoretischen Standpunkt her Ausgabevariablen des Automaten sind. Die Wertemengen werden mit $\mathcal{W}(\mathbb{z})$, $\mathcal{W}(\mathbb{u})$ und $\mathcal{W}(\mathbb{o})$ bezeichnet. Es ist also $\mathbb{z} = \mathbb{u} \cup \mathbb{o}$ und $\mathcal{W}(\mathbb{z}) = \mathcal{W}(\mathbb{u}) \times \mathcal{W}(\mathbb{o})$.[1]
- eine Zustandsübergangsfunktion Δ: $\mathcal{W}(\mathbb{i}) \times \mathcal{W}(\mathbb{s}) \rightarrow \mathcal{W}(\mathbb{s})$,
- eine Funktion Λ_{out}: $\mathcal{W}(\mathbb{i}) \times \mathcal{W}(\mathbb{s}) \rightarrow \mathcal{W}(\mathbb{o})$ zur Bestimmung der Werte der Ausgabesignale
- eine Funktion Λ_{int}: $\mathcal{W}(\mathbb{i}) \times \mathcal{W}(\mathbb{s}) \rightarrow \mathcal{W}(\mathbb{u})$ zur Bestimmung der Werte der internen Signale
- einer Bedingung ϱ: $\mathcal{W}^r(\mathbb{i}) \rightarrow \mathcal{B}$ über Resetsequenzen[2]. Diese Resetsequenzen sind Eingabetraces der Länge r, mit der die Schaltung in ihren Resetzustand gebracht wird. Es muss dafür gesorgt werden, dass die Schaltung immer erst eine der Resetsequenzen abarbeitet, bevor sie normale Eingaben verarbeitet. Diese Bedingung ersetzt die in Automaten sonst üblichen Bedingungen über die Initialzustände.

[1] Flip-Flops treten in dieser Formalisierung tatsächlich doppelt auf, einmal als Teil des Zustands $\mathbb{s}$ und einmal als Teil der internen Signale $\mathbb{u}$.

[2] Dabei steht $\mathcal{B}$ für die Menge $\{true, false\}$ von Wahrheitswerten.

Die Schaltungsfunktion ergibt sich also, indem zunächst eine Resetsequenz $(i_{-r}, i_{-r+1}, i_{-r+2}, \dots, i_{-1})$ von einem beliebigen Zustand der Schaltung aus bearbeitet wird, wodurch die Schaltung eine Folge $(s_{-r}, s_{-r+1}, s_{-r+2}, \dots, s_{-1})$ von Zuständen durchläuft, die mit einem beliebigen Zustand $s_{-r} \in \mathcal{W}(\mathbb{s})$ beginnt und an deren Ende sich ein Resetzustand s_0 einstellt. Davon ausgehend wird eine Folge $(i_0, i_1, i_2, \dots)$ von Eingabewerten von der Maschine bearbeitet. Dabei ergeben sich Werte $(s_0, s_1, s_2, \dots)$ interner speichernder Signale. Aus dem die Resetsequenz enthaltenden Eingabestimulus ergeben sich also Folgen $(u_{-r}, u_{-r+1}, u_{-r+2}, \dots, u_{-1}, u_0, u_1, u_2, \dots)$ und $(o_{-r+1}, o_{-r+2}, \dots, o_{-1}, o_0, o_1, o_2, \dots)$ von Werten der internen und Ausgabesignale, die bestimmt werden gemäß der sogenannten Übergangsgleichungen

$$s_{j+1} = \Delta(\mathrm{s_j}, \mathrm{i_j})$$

und

$$o_j = \Lambda_{\mathrm{out}}(\mathrm{s_j}, \mathrm{i_j}) \text{ bzw. } u_j = \Lambda_{\mathrm{int}}(\mathrm{s_j}, \mathrm{i_j})$$

wobei $j \geq -r$ ist. Δ wird als Zustandsübergangsfunktion des Automaten bezeichnet und Λ_{out} als Ausgabefunktion.

4.1.2 Traces

Eine Wertefolge $(X^{(j)})$ auf einer Menge von Signalen wird Trace X dieser Signale genannt. Typischerweise zählt dabei der Index j die Zyklen eines Taktes. Das einzelne $X^{(j)}$ wird j-tes Element des Traces X genannt. Traces können endlich oder unendlich sein. Zu einer Signalmenge $\mathbb{x}$ wird mit $\mathcal{W}^{\infty}(\mathbb{x})$ die Menge aller Traces dieser Signale bezeichnet. Dabei werden die Signale wie freie Variable behandelt, d.h. $\mathcal{W}^{\infty}(\mathbb{x})$ soll auch solche Traces enthalten, die bei Berücksichtigung der Schaltung gar nicht vorkommen können.

Ein endliches Anfangsstück $(X^{(0)}, X^{(1)}, X^{(2)}, \dots, X^{(n)})$ eines Traces X wird Präfix des Traces genannt. n wird als größter Index des Präfixes bezeichnet. Zwei Traces können einen gemeinsamen Präfix besitzen. Als Separierungsindex $SEP(X, Y)$ wird der größte Index des größten gemeinsamen Präfixes von X und Y bezeichnet. Wenn $X = Y$, wird $SEP(X, Y) = \infty$.

Ein aus zwei Traces X und Y zusammengefügter Trace (X, Y) besteht aus den Paaren $\left((X^{(j)}, Y^{(j)})\right)$ der Elemente der beiden Traces. Es wird dabei nicht zwischen dem Paar der Traces und dem Trace aus den Paaren der Elemente unterschieden.

Ein Automat M kann gemäß Abschnitt 4.1.1 mit einem Prädikat M auf Traces identifiziert werden. $M(I, S, U, O)$ nimmt genau für die Eingabetraces I, Ausgabetraces O, und Traces der Zustandsvariablen S und der internen Signale U den Wert 1 an. Der Trace der Zustandsvariablen spielt dabei häufig keine selbständige Rolle, weil diese Information bereits im Trace der internen Signale U enthalten ist. Daher wird der Automat meistens als Prädikat $M(I, U, O)$ aufgefaßt. Ein Trace (I, U, O) der das Prädikat $M(I, U, O)$ erfüllt, heißt Trace des Automaten M. Die Traces beginnen mit dem Index $(-r)$, damit Rechenschaft über die Resetsequenz abgelegt wird.

4.2 Assertions, Constraints und Eigenschaften

4.2.1 Rollen von Assertions, Constraints und Eigenschaften

Auf den Traces I, U und O eines Automaten werden Bedingungen $\beta(I, U, O)$ definiert, mit denen pragmatische Einschränkungen des Verhaltens des Automaten oder Erwartungen an sein Verhalten ausgedrückt werden. Diese Bedingungen ergeben sich häufig aus Formulierungen $\hat{\beta}(t, I, U, O)$, die relativ zu einem beliebigen aber festen Zeitpunkt t aufgestellt werden und für alle $t \geq 0$ gelten sollen:

$$\beta(I, U, O) = \bigwedge_{t \geq 0} \hat{\beta}(t, I, U, O)$$

Die Bedingungen $\hat{\beta}$ können Assertions, Constraints oder Operationseigenschaften sein. Constraints beschreiben Verhalten, das nicht weiter hinterfragt werden soll, weil entweder angenommen werden soll, dass sich die Umgebung der zu untersuchenden Schaltung entsprechend verhält, oder weil eine Verifikation auf die im Constraint beschriebene Situation beschränkt werden soll. Constraints werden beim Beweis von Assertions, von Eigenschaften, oder bei der Vollständigkeitsprüfung vorausgesetzt und beschränken den Beweis entsprechend. Constraints gelten global, d.h. sie werden für die gesamte Verifikation vorausgesetzt.

Assertions enthalten entweder Verifikationsziele (siehe Abschnitt 2.8), Erreichbarkeitsbedingungen (Abschnitt 3.3.3) oder lokale Constraints (Abschnitt 3.4.3).

Assertions und Operationseigenschaften formalisieren Beweisaufgaben entsprechend Kapitel 3.

4.2.2 Dependencies

Bei der Anwendung des hier vorgestellten Verfahrens wird bei vielen Beweisen die Gültigkeit anderer Objekte vorausgesetzt. So werden Operationseigenschaften und viele Assertions unter Voraussetzung von anderen Assertions oder Constraints mit IPC bewiesen. Vollständigkeit und komplexe Assertions werden hingegen mit dem Vollständigkeitsprüfer unter Voraussetzung von Assertions, Constraints und Operationseigenschaften bewiesen (siehe Abschnitt 4.3.7). Die vorausgesetzten Objekte werden Dependencies des zu beweisenden Objekts genannt. Der Benutzer trifft die Auswahl der Dependencies entsprechend der für den Beweis benötigten Information. Irrelevante Information als Dependencies anzugeben, kann zu höheren Beweislaufzeiten führen. Dependencies bilden einen Graphen, anhand dessen Zirkelschlüsse erkannt und vermieden werden können.

4.2.3 Reaktive Constraints und Assertions

Constraints beschreiben das Verhalten der Eingabesignale einer Schaltung, indem die entsprechenden Bedingungen wahr werden für einen Eingabetrace, der beim Betrieb der Schaltung vorkommen kann und andernfalls falsch wird. Constraints lassen sich nur in einfachen Fällen durch Bedingungen beschreiben, die nur von den Eingabesignalen der Schaltung abhängen. In den allermeisten Fällen ist die Schaltung Teil eines größeren Systems, das die Ausgabesignale der Schaltung zu den Eingabesignalen rückkoppelt. Diese Effekte der Schaltungsumgebung wird mit Constraints formalisiert, die auch von Ausgangs- oder internen Signalen abhängen. Solche Constraints werden reaktiv genannt.

In Verbindung mit kompositionaler Verifikation kann es auch zu reaktiven lokalen Constraints kommen, die in der globalen Sicht Assertions sind. Solche Assertions und auch die

Assertions, die die Abhängigkeit der Ausgabesignale von Eingabesignalen beschreiben, werden als reaktive Assertions bezeichnet.

4.2.4 Safety- und Liveness-Bedingungen

Eine Safety-Bedingung beschreibt, dass ein unerwünschter Sachverhalt nie eintrifft, eine Liveness-Bedingung wird erfüllt, wenn ein erwünschter Sachverhalt irgendwann in der Zukunft auftreten wird.

Eine Bedingung ist genau dann eine Safety-Bedingung, wenn sie auf endlichen Präfixen der Traces widerlegt werden kann. Wenn also eine Safety-Bedingung $B(L)$ auf einem Trace L widerlegt wird, muss es einen Index N geben, sodass $B(L')$ verletzt ist für alle Traces L', die mit L einen gemeinsamen Präfix der Länge N haben. Typische Safety-Bedingungen verlangen, dass eine Auswahl von Signalen immer die gleiche boolesche Bedingung erfüllt, oder auch, dass nach einem ersten Ereignis ein zweites Ereignis in einer beschränkten Zeit eintritt.

Liveness-Bedingungen erfordern die Untersuchung des gesamten Traces. Eine typische Livenessbedingung fordert, dass nach einem Startereignis irgendwann eine Reaktion eintrifft, ohne dass eine maximale Wartezeit vorgegeben wird. Die Untersuchung von Liveness-Assertions ist normalerweise sehr aufwändig. Die vollständige Verifikation verzichtet daher an allen zentralen Stellen auf die Untersuchung solcher Assertions. Das ist zulässig, weil Liveness-Bedingungen bei der Schaltungsverifikation in zwei Situationen auftauchen, für die es Alternativlösungen gibt:

Die erste Situation besteht darin, dass die Wartezeit zwischen dem Startereignis und der Reaktion nur von der Schaltung bestimmt wird. In diesem Fall verlangt die Methodik entsprechend Abschnitt 3.2.3 und in Folge dessen der Vollständigkeitsprüfer, dass die Wartezeit exakt bestimmt wird und in die entsprechenden Operationseigenschaft kodiert wird, die dadurch eine Safety-Eigenschaft wird und daher mit den Techniken dieses Ansatzes verifiziert werden kann.

Im zweiten Fall erfolgt die Schaltungsantwort erst nach einem weiteren externen Ereignis, das nach dem Startereignis eintreffen muss. In diesem Fall beschreibt eine Operationseigenschaft unter anderen, wie die Schaltung in einer Wartestellung verharrt, bis das zusätzliche Ereignis eintrifft. Wenn es nie eintreffen sollte, beschreibt die Operationseigenschaft, wie die Schaltung für immer in dieser Wartestellung verharrt. Für den Zusammenhang zwischen dem externen Ereignis und der Reaktion gilt im übrigen das zuvor gesagte. Damit ist auch diese entsprechende Operationseigenschaft eine Safety-Eigenschaft.

Durch diese Behandlung ist es möglich, die vollständige Verifikation nur auf Safety-Eigenschaften und –Assertions zu begründen. Liveness-Assertions können nur als von außen vorgegebene Beweisziele auftreten. Sie werden dann auf Basis der Operationseigenschaften und der anderen Safety-Assertions in einem separaten Verifikationsschritt bewiesen.

4.2.5 Behandlung unendlicher Wartezeiten

Die aktuelle Implementierung der vollständigen Verifikation kann innerhalb einer Eigenschaft keine unendlichen Wartezeiten zwischen einem Start- und einem externen Ereignis im Sinne des vorigen Abschnitts 4.2.4 verifizieren. Möglich sind nur endliche Wartezeiten, wobei die obere Schranke aus Komplexitätsgründen recht klein gewählt werden sollte. Diese Beschränkung betrifft die Eigenschafts- genauso wie die Vollständigkeitsprüfung.

Der Benutzer kann trotzdem vollständige Verifikation in Allgemeinheit durchführen, muss dann allerdings jede Warteperiode aufteilen in eine Phase, in der die Schaltung die Warteperiode betritt und nach einer meist kleinen Zahl von Takten in einen der Wartezustände gelangt, eine weitere Phase, in der die Schaltung ohne Eintreffen des externen Ereignisses in der Menge der Wartezustände verharrt, und in eine dritte Phase, wo nach dem Eintreffen des externen Ereignisses die Warteperiode verlassen wird und die Schaltung weiterarbeitet. Für jede Phase müssen dann separate Operationen und ihre Eigenschaften definiert werden, und die Menge der Wartezustände wird zum konzeptionellen Zustand.

Diese Vorgehensweise ist manchmal sehr aufwändig, weil hier meistens eine Operation an einem recht unnatürlichen Zeitpunkt in drei Operationen zerlegt wird. Für diesen Zeitpunkt gelten die Argumente aus Abschnitt 3.3.5 nicht, die erläutern, warum Erreichbarkeitsbedingungen aufgrund des operationsbasierten Vorgehens relativ einfach bleiben. Entsprechend kann die Bestimmung der Erreichbarkeitsbedingungen in den Wartezuständen aufwändig werden. Ferner reduziert die Zerlegung die Verständlichkeit der Verifikation.

Das alternative Vorgehen besteht in der Beschränkung der Wartezeit auf ein Maximum, das vom IPC-Beweiser noch gut behandelt werden kann. Diese Vorgehensweise lässt eine Verifikationslücke, deren Irrelevanz aber häufig leicht aus dem Schaltungsverhalten zu begründen ist. Der Vorteil der Vorgehensweise besteht in dem geringeren Aufwand bei der Verifikation und in den umfassenderen Beschreibungen der Operationen, durch die das Schaltungsverhalten offener zu Tage tritt.

In der nachfolgenden Beschreibung wird die Theorie soweit entwickelt, dass sie auch auf Operationen mit unbeschränkten Wartezeiten anwendbar ist.

4.3 Eigenschaftssprache ITL

Eine formale Variante der funktionalen Verifikation wird umso mehr akzeptiert, je besser das zu beweisende Verhalten in der Eigenschaftssprache des Property Checkers ausgedrückt werden kann. Ziel bei der Entwicklung einer Eigenschaftssprache sollte also sein, dass ein Benutzer seine Energie auf die Identifizierung des zu beweisenden Verhaltens konzentriert, und möglichst wenig Aufwand braucht für die Formalisierung dieses Beweiszieles und das Finden sowie die Korrektur von Abweichungen zwischen intendiertem und tatsächlich beschriebenem Verhalten. Ferner muss die Eigenschaftssprache in der Lage sein, die Ursache-Wirkungs-Zusammenhänge an den Interfaces einer Schaltung darzustellen, die zur Erzeugung bzw. Bearbeitung einer Transaktion gehören. Unter diesen Forderungen wurde ITL entwickelt [Siegel et al. 1999] ausgehend von Erfahrungen, die mit einer Vorgängerversion [Bormann 1995] gemacht wurden.

Zu ITL wurde die bereits eingeführte graphische Repräsentation entwickelt [Bormann 2001, Paucke 2003], die sich eng an Timingdiagramme anlehnt und wesentlich intuitiver ist als andere graphische Formalismen zur Beschreibung von Schaltungsverhalten [Schlör 2001, Peukert et al. 2001]. Darüber hinaus wurde ein Verfahren zur Übersetzung von ITL-Eigenschaften in synthetisierbare Simulationsmonitore entwickelt [Bormann 2003, Beuer 2005].

4.3.1 Zeitbehandlung

Eine erste Beispieleigenschaft wurde bereits in Abschnitt 3.2.5 vorgestellt. Sie zeigt, dass das Hauptcharakteristikum der Sprache die explizite Behandlung von Zeit ist, die in Takten gemessen wird. Alle Operationseigenschaften werden relativ zu einem beliebigen aber festen

Zeitpunkt formuliert, der durch das Schlüsselwort "t" repräsentiert wird. Endliche Offsets von t werden durch $t + n$ oder $t - n$ ausgedrückt, wobei n eine ganze Zahl ist.

Variable Offsets sind beispielsweise erforderlich, um zu beschreiben, wie während der Abarbeitung einer Transaktion auf ein Synchronisationsereignis vom Kommunikationspartner gewartet wird. Sie werden durch Zeitvariablen repräsentiert, die relativ zu t definiert werden. Um die Verhältnisse darzustellen, wenn das Synchronisationsereignis gar nicht eintrifft, können die Zeitvariablen den Wert unendlich annehmen, der durch "$" repräsentiert wird.

Zeitvariablen, die unendlich werden können, werden als unendliche Zeitvariablen bezeichnet. Andernfalls heißen sie endliche Zeitvariablen.

4.3.2 Zeitbehaftete Bedingungen

Die Eigenschaften werden durch Bedingungen der Form

```
at <time point>: <Boolean condition>;
during [<time point1>, <time point2>]: <Boolean condition>;
within [<finite time pt1>, <finite time pt2>]: <Boolean condition>;
```

gebildet. Die erste Bedingung verlangt, dass die boolesche Bedingung am vorgegebenen Zeitpunkt gilt, die zweite Bedingung verlangt, dass sie für alle Zeitpunkte im angegebenen Intervall gilt, und die dritte, dass sie mindestens einmal im angegebenen Intervall gilt. Die Intervalle enthalten jeweils ihre Grenzen. Wenn die linke Intervallgrenze größer ist als die rechte, ist das Intervall leer. Dann ist die "during"-Bedingung trivialerweise erfüllt und die "within"-Bedingung trivialerweise verletzt.

Die Zeitangaben haben die Form $T + n$ oder $T - n$, wobei T für t oder für eine Zeitvariable steht. In within-Bedingungen dürfen keine unendlichen Zeitvariablen auftauchen. Wenn die Zeitangabe hinter "at" eine Zeitvariable enthält und diese Zeitvariable den Wert unendlich annimmt, ist die gesamte Bedingung trivialerweise erfüllt. Wenn beide Grenzen des during-Intervalls unendlich sind, ist die entsprechende Bedingung trivialerweise erfüllt, wenn nur die rechte Grenze unendlich ist, muss die boolesche Bedingung ab der linken Intervallgrenze für alle Zeiten erfüllt sein.

Die booleschen Bedingungen haben eine Syntax, die wahlweise ähnlich zu Verilog oder VHDL ist. Je nachdem wird ITL auch als VLI (Verilog Interval Language) oder VHI (VHD Interval Language) bezeichnet. Die booleschen Bedingungen werden zu den jeweils angegebenen Zeitpunkten ausgewertet. Die Rückgabewerte arithmetischer Bitvektoroperationen sind immer breit genug, um das exakte Ergebnis zu repräsentieren. Durch Funktionen prev und next können Ausdrücke zu anderen Zeitpunkten als den angegebenen ausgewertet werden.

4.3.3 Grundlegende Implikation

Die Grundstruktur einer ITL-Eigenschaft ist gegeben durch verschiedene Blöcke. Zwei davon sind besonders wichtig: Der eine Block wird mit dem Schlüsselwort "assume:" eingeleitet und beschreibt den Annahmenteil der Eigenschaft. Der andere Block wird mit dem Schlüsselwort "prove:" eingeleitet und beschreibt den Beweiszielteil der Eigenschaft.

Der Annahmenteil beschreibt die Situation, die mit der Eigenschaft geprüft werden soll und der Beweiszielteil das in dieser Situation erwartete Verhalten.

4.3.4 Annahmen- und Beweiszielteil einer Eigenschaft

Der Annahmen- und Beweiszielteil einer Eigenschaft werden durch eine Anzahl von Bedingungen gebildet, die im vorangegangenen Abschnitt 4.3.2 beschrieben wurden. Wenn diese Bedingungen nur durch Semikolon separiert hintereinander geschrieben werden, wird die Konjunktion gebildet.

Disjunktion wird durch das Konstrukt

```
either
        Bedingung
or
        Bedingung
or
        ...
end either;
```

ausgedrückt.

Annahmen- und Beweiszielteil einer Eigenschaft sind Funktionen auf den Traces $L = (I, U, O)$ des aus der Schaltung gewonnenen Automaten M mit den Werten 0 (falsch) oder 1 (wahr). Neben dem Trace sind t und alle Zeitvariablen T_i Argumente dieser Funktion. Das Annahmenprädikat $\hat{A}(t, L, T_1, T_2, \dots)$ sagt aus, ob die Annahme auf dem Trace L für eine gegebene Wahl von Zeitpunkten $t, T_1, T_2, \dots$ erfüllt ist. Analog dazu sagt das Beweiszielprädikat $\hat{C}(t, L, T_1, T_2, \dots)$, ob der Beweiszielteil erfüllt ist.

4.3.5 Zeitvariablen

Zeitvariablen werden in einem eigenen Block definiert, der mit den Schlüsselworten "for timepoints:" eingeleitet wird. Eine Zeitvariable wird durch den Ausdruck

```
Tj = Bj + nj .. mj awaits wj;
```

vereinbart. Dabei ist B_j entweder t oder eine vorher bereits eingeführte Zeitvariable. n_j ist eine ganze Zahl und m_j entweder eine ganze Zahl $\geq n_j$ oder \$, d.h. unendlich. w_j ist eine Wartebedingung über Eingabe-, Ausgabe-, oder interne Signale der Schaltung. Der Ausdruck legt fest, dass T_j bei endlichem B_j auf den ersten Zeitpunkt im Intervall $[B_j + n_j, B_j + m_j]$ gesetzt wird, an dem w_j erfüllt ist. Wenn w_j zu keiner Zeit in diesem Intervall erfüllt ist, wird $T_j = B_j + m_j$ gesetzt, falls m_j eine ganze Zahl ist und $T_j = \$$, falls $m_j = \$$. Auch wird $T_j = \$$, falls B_j bereits unendlich ist.

Aufgrund der Abhängigkeit von der Wartebedingung können die T_j damit als Funktionen $T_j(t, L)$ angesehen werden, die abhängig von t und dem Trace L den Wert der entsprechenden Zeitvariablen angeben. Wenn diese Funktionen in das Annahmen- oder Beweiszielprädikat substituiert werden, entstehen $A(t, L) = \hat{A}(t, L, T_1(t, L), T_2(t, L), \dots)$ und $C(t, L) = \hat{C}(t, L, T_1(t, L), T_2(t, L), \dots)$.

4.3.6 Freeze-Variablen

Zur Darstellung von Datentransport können Werte in einer eigenen Variable eingefroren werden. Sie stehen dann in der gesamten Eigenschaft zur Verfügung. Freeze-Variablen werden in

einem eigenen Block vereinbart, der mit dem Schlüsselwort "freeze:" eingeleitet wird. Diese werden durch

F_j = $expression_j$ @ $timepoint_j$;

vereinbart. Semantisch verhalten sich Freeze-Variablen so, als ob sie überall durch die rechte Seite der Vereinbarung substituiert würden.

Eine Freeze-Variable, die relativ zu einer unendlichen Zeitvariablen T definiert ist, darf nur in Ausdrücken verwendet werden, die irrelevant sind, falls T den Wert unendlich annimmt. Solche Ausdrücke sind at-Konstrukte, die relativ zu T oder relativ zu einer Zeitvariablen definiert sind, die von T abhängt, oder during-Konstrukte, bei denen die linke Intervallgrenze diese Gestalt hat.

4.3.7 Dependencies

Assertions und Eigenschaften verfügen über einen optionalen Dependencies-Block, mit dem ihre Dependencies im Sinne von Abschnitt 4.2.2 durch Auflistung von Namen angegeben werden können.

Eine Dependency ist ein Prädikat $d(t, L)$ mit der Definition einer Assertion oder eines Constraints. Alle Dependencies müssen auf jedem Trace des unterliegenden Modells zu jedem Zeitpunkt ≥ 0 erfüllt sein. IPC nimmt die Dependencies auf dem Untersuchungsfenster der Eigenschaft oder der Assertion an.

Alle Dependencies werden zu einer Bedingung

$$D(L) = \bigwedge_{t \geq 0} d_1(t, L) \wedge d_2(t, L) \wedge d_3(t, L) \wedge \ldots$$

über den Trace zusammengefasst.

4.3.8 Assertions und Constraints

Die Syntax von Assertions und Constraints im Sinne von 4.2 ist

```
constraint <name>;
        <boolesche Bedingung>;
end constraint;

assertion <name>;
        <boolesche Bedingung>;
        dependencies: <Liste von Assertion- und Constraintnamen>;
end assertion;
```

mit den booleschen Bedingungen aus Abschnitt 4.3.2. Diese booleschen Bedingungen dürfen mit $prev(\ldots)$ und $next(\ldots)$ Signalwerte am vorangegangenen bzw. nachfolgenden Zeitpunkt referenzieren und sollen für alle Zeitpunkte nach der Resetsequenz gelten. In ITL können aus den in Abschnitt 4.2.4 genannten Gründen nur Safety-Assertions und –Constraints formalisiert werden.

4.3.9 Eigenschaften

Die Syntax einer Eigenschaft ist:

property <name>;
dependencies: <list of assertions and constraints>;
for timepoints: <time variable declaration>;
freeze: <freeze variable declaration>;
assume:
<assume part>;
prove:
<prove part>;
end property;

Dabei sind alle Blöcke bis auf den Beweiszielteil optional und wurden in vorangegangenen Abschnitten beschrieben.

Eine Eigenschaft drückt die Beziehung

$$P(t,I,U,O) = (A(t,I,U,O) \Rightarrow C(t,I,U,O))$$

aus. Sie gilt auf einem Trace (I,U,O) des Automaten, wenn sie zu jedem Zeitpunkt $t \geq 0$ gilt. Eine Eigenschaft gilt insgesamt, wenn auf allen Traces (I,U,O) des Automaten, die zudem noch die Dependencies erfüllen, die durch D zusammengefaßt sind, die folgende Formel erfüllt ist

$$\bigwedge_{I,U,O} \left((M(I,U,O) \wedge D(I,U,O)) \Rightarrow \bigwedge_{t \geq 0} P(t,I,U,O) \right)$$

Da sich alle Dependencies aus dem Automaten und der Konjunktion der Constraints $Constr$ herleiten lassen, ist mit dem Beweis einer Eigenschaft auch nachgewiesen, dass alle Traces (I,U,O) des Automaten die Eigenschaft erfüllen, sofern sie nur auch alle Constraints erfüllen:

$$\bigwedge_{I,U,O} \left((M(I,U,O) \wedge Constr(I,U,O)) \Rightarrow \bigwedge_{t \geq 0} P(t,I,U,O) \right)$$

4.3.10 Beispiel

Als Beispiel soll eine Eigenschaft über die Ausführung einer Leseoperation in einem Prozessor vorgestellt werden. Der Prozessor sei durch eine Pipeline implementiert, deren Stufen nacheinander mit dec (decode), ex1 (execute 1), ex2 und wb (write back) bezeichnet seien. Jede Stufe kann mit den Signalen dec_stall, $ex1\_stall$, $ex2\_stall$ and wb_stall angehalten werden. Dann wird kein Datum von der angehaltenen Stufe an die nächste weitergegeben. Entsprechend werden in der Eigenschaft die Zeitvariablen t_ex1, t_ex2, und t_wb definiert, mit denen die Zeitpunkte bezeichnet sind, bis zu denen sich die Instruktion in der entsprechende Stufe der Pipeline aufhält.

Dabei unterliegen die Stall-Signale gewissen Forderungen, ohne die die Pipeline nicht funktionieren würde. Es sei angenommen, dass dieses Forderungen in dem Constraint stall_relation ausgedrückt seien.

Am Beginn der Eigenschaft soll eine Leseinstruktion in der nicht angehaltenen dec-Stufe warten. Dies legen die drei Bedingungen fest, die im Annahmenteil zum Zeitpunkt t verlangt werden. Zum Zeitpunkt t wird auch die Kodierung der Instruktion auf dem Signal instr in der Freeze-Variable instr_dec eingefroren, damit jederzeit auf die darin enthaltene Information zugegriffen werden kann. Die Leseadresse wird in der ex1-Stufe ausgegeben. Es wird angenommen, dass der Datenspeicher reagiert hat, wenn die ex2-Stufe weiterarbeiten kann. Die Reaktion des Speichers sei entweder eine Meldung über verletzte Zugriffsrechte auf dem Signal *violation*, oder die Lesedaten selbst. Im Fall dieser Eigenschaft seien die Zugriffsrechte erfüllt. Deshalb kommen die Daten zum Zeitpunkt t_ex2 und werden zu diesem Zeitpunkt auch in der Freeze-Variable read_val_ex2 gespeichert, damit sie mit dem Wert des Signals result_wb während des Aufenthalts der Instruktion in der Pipelinestufe wb verglichen werden können. Es wird nachgewiesen, dass der Ausgang der wb-Stufe die gelesenen Daten beispielsweise zum Eintrag in das Registerfile verfügbar macht.

Der Eigenschaftstext ist:

```
property read_no_violation is
dependencies: no_reset, stall_relation;

for timepoints: t_ex1 = t     + 1 .. $ awaits ex1_stall = '0',
                t_ex2 = t_ex1 + 1 .. $ awaits ex2_stall = '0',
                t_wb  = t_ex2 + 1 .. $ awaits wb_stall = '0';

freeze: instr_dec = instr @ t,
        read_val_ex2 = read_val @ t_ex2;

assume:
at    t:               opcode(ip_instr) == READ_INSTRUCTION;
at    t:               dec_valid = '1';
at    t:               dec_stall = '0';

at t_ex2:              violation = '0';

prove:
during [t+1, t_ex1]:    adr_out == address(instr_dec);
during [t_ex2+1, t_wb]: result_wb == read_val_ex2;
end property;
```

4.3.11 Graphische Darstellung

Eine wesentliche Untermenge von ITL-Eigenschaften, nämlich Eigenschaften ohne either-oder within-Statements lässt sich graphisch darstellen durch farbkodierte Timingdiagramme [Bormann 2001, Paucke 2003]. Diese Repräsentation ist insbesondere für protokolldominierte Schaltungen sehr hilfreich. Die Eigenart der graphischen Repräsentation lässt sich aus dem Beispiel in Abbildung 11 entnehmen, mit dem die Eigenschaft aus Abschnitt 4.3.10 visualisiert wird. Die Einfachheit dieser Darstellung ist ein großer Vorteil gegenüber anderen graphischen Formalismen [Schlör 2003, Peukert et al. 2001].

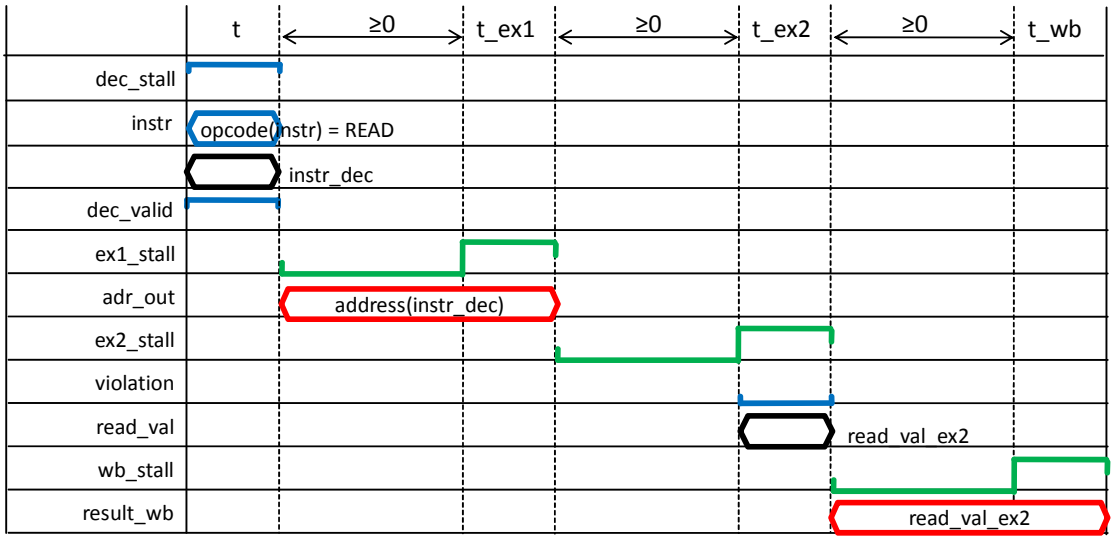

Abbildung 11: Graphische Darstellung der Eigenschaft read_no_violation

Die Farbcodes in den verallgemeinerten Timingdiagrammen sind

- blau für den Annahmenteil
- rot für den Beweiszielteil
- grün für die Wartebedingungen der Zeitvariablen und
- schwarz für die Zeitpunkte, an denen andere Bedingungen auf den Signalwert zugreifen. Wenn über Freeze-Variablen zugegriffen wird, werden ihre Namen eingetragen. Wird anders zugegriffen, d.h. etwa durch $prev$ oder $next$, wird kein Name eingetragen.

Der Zeitbereich, der von einer Zeitvariablen überstrichen werden kann, wird für jedes Signal durch eine breitere Zelle und durch einen Pfeil in der Kopfzeile des Timingdiagramms angedeutet. Der Pfeil soll andeuten, dass die Breite dieser Zelle eigentlich variabel ist. Im Extremfall kann die Zelle die Breite 0 erhalten und ganz verschwinden, wenn nämlich die Wartebedingung sofort erfüllt ist. Die Zelle kann andernfalls so viele Takte breit werden, wie es der Längenbereich angibt, der an dem Pfeil in der Kopfzeile annotiert ist. Wenn ein Signal verzö-

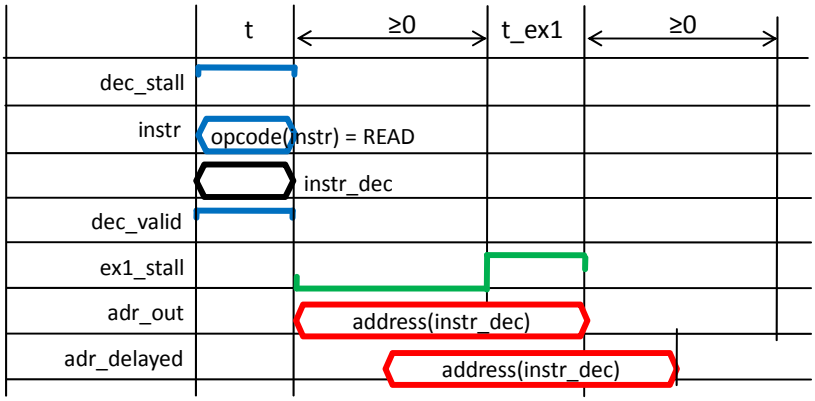

Abbildung 12: Verzögerte Reaktion auf Synchronisationsereignis

gert auf ein Synchronisationssignal reagiert, verschiebt sich auch die breitere Zelle entsprechend der Verzögerung. Zu diesem Detail zeigt Abbildung 12 ein Beispiel.

4.3.12 Diskussion

ITL ist eine proprietäre Sprache mit Leistungsmerkmalen, die sie für die vollständige formale Verifikation besonders geeignet machen. Besonders wichtige Leistungsmerkmale sind die boolesche Implikation und die explizite Bezeichnung von Zeit. Unter boolescher Implikation wird eine Implikation verstanden, bei der sich Annahme- und Beweiszielteil zeitlich überlappen dürfen. Das ist wichtig, denn nur so lassen sich nämlich ganze Transaktionen auch dann noch untersuchen, wenn sich darin Forderungen über das protokollgerechte Verhalten an Schaltungsausgängen mit Annahmen über das protokollgerechte Verhalten an Schaltungseingängen mehrfach abwechseln.

Boolesche Implikation wird aber nur sinnvoll genutzt werden können, wenn in Annahme- und Beweiszielteil leicht über den gleichen Zeitpunkt gesprochen werden kann. Das ist in ITL mit seinen expliziten Zeitausdrücken der Fall, auch dann noch, wenn der gemeinsame Zeitpunkt variabel ist und durch eine Zeitvariable festgelegt wird, deren Wartebedingung das Eintreffen eines Synchronisationsereignisses ist.

Ein drittes wesentliches Leistungsmerkmal von ITL ist seine graphische Darstellung. Anhand dieser Darstellung lassen sich gerade Eigenschaften über protokolldominierte Schaltungen leicht aufstellen, verstehen und kontrollieren.

Diese drei Leistungsmerkmale sind es, warum ITL noch immer Bestand hat neben den beiden standardisierten Eigenschaftssprachen, PSL [PSL 2004] und der Teilsprache von System Verilog [System Verilog 2005] zur Beschreibung von Assertions, kurz SVA. In der normalen Benutzung von PSL und SVA kommen die oben genannten Leistungsmerkmale nicht vor: Statt boolescher Implikation wird normalerweise Sequence Implication eingesetzt, bei der der Antezedent (vergleichbar mit dem Annahmeteil in ITL) und der Sukzedent (der Beweiszielteil in ITL) sich zeitlich nicht überlappen dürfen und alle Voraussetzungen zeitlich vor allen Forderungen gemacht werden müssen.

Statt expliziter Zeitangabe bieten SVA und PSL die Möglichkeit, Sequenzen von Bedingungen aufzustellen, die nacheinander auf einem Trace gelten müssen. Diese Sequenzen werden durch reguläre Ausdrücken gebildet. Damit lässt sich protokollgerechtes Verhalten komfortabel ausdrücken, allerdings nur dann, wenn es sich um einen internen Bus handelt, der auf protokollgerechtes Verhalten geprüft werden soll, und alle an den Bus angeschlossenen Module Teil der zu untersuchenden Schaltungen sind. Dies ist bei Simulationen häufig der Fall, aber nicht bei formaler Verifikation, weil solche Schaltungen normalerweise zu groß sind für formale Werkzeuge. Bei formaler Verifikation wird protokollgerechtes Verhalten meistens an primären Interfaces untersucht. Um dann Transaktionen geschlossen verifizieren zu können, müssen kompliziertere Ursache-Wirkungs-Zusammenhänge ausgedrückt werden als mit der Sequence Implication von SVA oder PSL möglich ist. In dieser Situation wird die Untersuchung für PSL oder SVA normalerweise phasenweise in mehrere Eigenschaften aufgeteilt, die sich mit den Einschränkungen der Sequence Implication beschreiben lassen. Dies stört die Lesbarkeit und Verständlichkeit der Verifikation, kommt aber den niedrigen Komplexitätsschranken der ABFV-Werkzeuge entgegen. Alles in allem sind aber mit SVA und PSL zwei Eigenschaftssprachen standardisiert worden, deren hauptsächlich verwandte Sprachmittel der Untersuchung von Transaktionen entgegenstehen, obwohl Transaktionen sonst für die funktionale Verifikation fundamental sind.

Bei Verwendung weniger prominenter Sprachmittel können die oben beschriebenen Leistungsmerkmale aber zumindest auch in SVA genutzt werden. Dies wird durch die Bibliothek TiDAL [Bormann 2007] ermöglicht.

Neben den drei oben genannten wesentlichen Leistungsmerkmalen hat ITL (und TiDAL gleichermaßen) auch einige Vorteile in Bezug auf die Benutzbarkeit. Am auffälligsten dürfte die klare Aufteilung von Ursache, Wirkung und zeitlicher Struktur in den assume-, prove-, und for-timepoint-Blöcken der Eigenschaftsbeschreibung sein. In SVA und PSL wird die zeitliche Strukturierung mit der Beschreibung von Ursache und Wirkung vermischt. Da die zeitliche Strukturierung meist durch externe Events vorgegeben wird, führt dies zu schwer verständlichen Sukzedenten. Insgesamt erscheinen SVA und PSL reicher an Fallstricken, sodass Benutzer mehr Aufwand in die Klärung der Frage investieren müssen, ob die intendierte Eigenschaft tatsächlich durch eine SVA-Beschreibung dargestellt wird.

5 Vollständigkeit und ihre Prüfung

Der Vollständigkeitsprüfer ist für die Automatisierung der vollständigen Verifikation von zentraler Bedeutung. Er wird nachfolgend dargestellt und mit anderen Ansätzen zur Bestimmung der Qualität einer Verifikation verglichen.

5.1 Verfahren zur Messung der Qualität einer simulationsbasierten Verifikation

Um ein Verifikationsprojekt durchführen zu können, ist es wichtig, auch schon lange vor dem geplanten Abschluss die jeweils bereits erreichte Qualität der Verifikation zu erfassen [Carter/Hemmady 2007]. Dieser Qualitätsstand trifft dabei eine Aussage darüber, wie intensiv die Verifikation die einzelnen Schaltungsteile untersucht hat, und wie groß die Gefahr ist, dass sich noch Fehler in der Schaltung befinden.

5.1.1 Coverage

Für simulationsbasierte Verifikation wird die Qualität meist mit der Input Coverage der Simulationen in Beziehung gesetzt, die häufig einfach als Coverage bezeichnet wird. Die Input Coverage trifft eine Aussage darüber, inwieweit die Eingabetraces der Simulation die Schaltung in jeder möglichen Situation untersucht haben. Das Resultat wird zu einer Zahl, dem Coveragemaß, verdichtet.

Verfahren zur Messung von Input Coverage wurden zuerst für die Softwareverifikation entwickelt [Hirsch 1967, Beizer 1990], und dann im Laufe der 90er Jahre in der Hardwareverifikation eingeführt. Etliche dieser Verfahren gehören nun zur Grundausstattung einer Verifikationsumgebung. Es gibt verschiedene Ansätze dafür, die Input Coverage zu bestimmen [Stuart/Dempster 2000]. Allen gemeinsam ist es, eine Menge von Coveragebedingungen zu definieren und ein Coverageziel, das zu jeder Coveragebedingung angibt, wie häufig sie erfüllt werden soll. Als Maßzahl wird dann ein unter Umständen gewichtetes Verhältnis zwischen dem Coverageziel und seinem Erfüllungsgrad während einer Verifikation ausgegeben.

Die verschiedenen Verfahren zur Bestimmung von Input Coverage unterscheiden sich bezüglich der Definition der Coveragebedingungen und der Coverageziele. Eine große Gruppe von Verfahren orientiert sich am RTL-Code. Diese Verfahren heißen Code Coverage. Das einfachste Verfahren, die Line oder Statement Coverage, definiert zu jeder Code-Zeile die Coveragebedingung, dass diese Zeile ausgeführt wird und das Coverageziel, dass jede Zeile mindestens einmal ausgeführt wird.

Bei der Branch Coverage werden die Bedingungen der Kontrollstatements (z.B. if und case) untersucht. Für jeden Wert, den die Kontrollbedingung annehmen kann, gibt es eine Coveragebedingung, die erfüllt ist, wenn die Kontrollbedingung den Wert angenommen hat. Das Coverageziel ist wiederum, dass jede Coveragebedingung einmal erfüllt werde. Davon abgeleitete Verfahren (Condition Coverage, Expression Coverage, Focussed Expression Coverage) untersuchen nicht die Kontrollbedingung als ganzes, sondern ihre Teilausdrücke. Bei Path Coverage werden Pfade durch den Kontrollgraphen des RTLs vorab bestimmt und als Coveragebedingungen aufgestellt, dass solch ein Pfad ausgeführt wird.

Bei anderen Coveragemetriken wird von den Signalen der Schaltung ausgegangen. Bei der Toggle Coverage beschreiben die Coveragebedingungen Änderungen der Signale und Coverageziel ist, dass in der Simulation für jedes Signal eine Änderung seines Wertes beobachtet wird. Bei der Triggering Coverage wird analysiert, welches Signal durch eine Wertänderung die Ausführung eines Prozesses veranlasst.

Coverage ist ein probates Mittel, die beschränkten Fähigkeiten der Simulation wenigstens optimal zu nutzen. Aber alle oben beschriebenen Coveragemetriken sind zufrieden, lange bevor die Schaltung unter allen Situationen untersucht wurde. Coverage hat dabei beispielsweise dann Schwächen, wenn es darum geht, dass parallel Prozesse gemeinsam eine Funktion implementieren und mit jedem möglichen Zeitversatz untersucht werden müssen.

Neben der Code-Coverage gibt es deshalb die funktionale Coverage. Dabei werden die einzelnen Coveragebedingungen und -ziele explizit von einem Verifikationsingenieur vorgegeben und so als kritisch angesehene Situationen speziell untersucht, die von oben genannten Coveragekriterien nicht scharf genug auseinandergehalten werden. Offensichtlich ist die Aussagekraft der funktionalen Coverage abhängig von der Definition der Coveragebedingungen. Wenn eine Coveragebedingung vergessen wurde, wird man sich während der Verifikation nicht anstrengen, die Schaltung in die darin kodierte Situation zu bringen.

5.1.2 Coverage verteilt Aufmerksamkeit der Simulation

Eine Verifikation strebt nach 100% Input Coverage. Die Verifikationsprojektleiter sind daran gewöhnt, dass sich die Input Coverage in einem gut verlaufenden Verifikationsprojekt über die Zeit entlang einer Saturierungskurve bewegt, und dass deshalb ein Coveragemaß von 98% gegen Ende eines Verifikationsprojekts einen dramatisch niedrigen Wert darstellen kann. Ziel sind häufig Werte im Bereich von 99.9%. Bei 100% Input Coverage sind Verifikationsteams üblicherweise glücklich, trotz der Unzulänglichkeiten des Ansatzes. Die Suche nach Fehlern aufgrund paralleler Ausführung von Prozessen wird durch die oben genannten Coveragemetriken z.B. nicht strukturiert.

Simulation ist bei etwas größeren Schaltungen prinzipiell nicht in der Lage, jeden Stimulus zu prüfen. Daher teilen die Coveragebedingungen die gesamte Verifikationsaufgabe in ein Raster ein. Jedes Element dieses Rasters enthält eine Vielzahl von Abläufen, mit denen jeweils eine Coveragebedingung erfüllt wird. Zur Erfüllung der Coveragebedingung ist es egal, welcher der Abläufe wirklich stattgefunden hat. Es wird angenommen, dass die Schaltung unter den anderen Abläufen so ähnlich funktionieren würde und dass deswegen ein Fehler an jedem dieser Abläufe erkannt werden könnte. Welche Unterschiede zwischen den Abläufen als irrelevant gelten, wird von der Auswahl der Coveragebedingungen festgelegt. Dabei wird durch die Auswahl der Verfahren zur Code-Coverage eine erste Aufteilung gemacht, die durch die Einzelanfertigung von Bedingungen der funktionalen Coverage angepasst wird.

Die Verifikationsqualität ist bei einem feineren Raster, d.h. bei detaillierten Coveragebedingungen höher als bei einem groben Raster. Dennoch darf für eine Verifikation auch kein beliebig feines Raster gewählt werden, denn damit geht die Aufgabe einher, jede einzelne Coveragebedingung auch zu erfüllen. Gibt es zu viele Coveragebedingungen, lassen sich Abläufe für alle Coveragebedingungen nur mit großem Aufwand finden. Daher muss die Konfigurierung der Coverageziele auch darauf abzielen, das Raster der Coveragebedingungen nicht zu eng zu gestalten.

Wenn Anwender formaler Verifikation die Vorteile ihrer Methode gegenüber von Coverage geleiteten Simulationen herausstellen wollen, bauen sie ein Gedankenmodell auf. Sie postulieren ein Coveragemaß, in dem es zu jedem Eingabetrace einer hinreichend groß zu wählenden Länge[3] eine Coveragebedingung gibt, die erfüllt ist, wenn genau dieser Eingabetrace abgearbeitet wurde. Zu jeder Coveragebedingung postulieren sie ferner ein Coverageziel, sie auch

[3] Die Länge könnte etwa aus dem Durchmesser der Schaltung entsprechend [Biere et al. 1999] bestimmt werden.

mindestens einmal zu erfüllen. Nur wenn diese Coveragemetrik 100% anzeigt, fand eine Verifikation mit allen Eingabetraces statt, wie sie für formale Verifikation typisch ist. Für Schaltungen üblicher Größe und realistische Simulationen wird diese Coveragemetrik aber nie auch nur ein Promille Coverage erreichen, weil es einfach so viele Coveragebedingungen gibt, dass sie während einer Simulation unmöglich alle erfüllt werden können.

Aber natürlich würde niemand ein solches Input-Coverageverfahren für Simulation ernsthaft in Betracht ziehen, sondern eben ein gröberes Raster wählen. Hierbei wird deutlich, dass Coverage eben auch zum Ziel hat, die beschränkte Untersuchungskraft der Simulation möglichst sinnvoll auf die Verifikationsaufgabe zu verteilen, und auf heikle Schaltungsteile zu konzentrieren. Coverage ist nicht in der Lage, die prinzipielle Untersuchungskraft der Simulation zu erhöhen, sondern sie erlaubt nur einen besseren Einsatz der beschränkten Kräfte.

5.1.3 Output Coverage

Doch die Qualität einer Verifikation wird nicht nur durch Input Coverage bestimmt, sondern auch durch die Frage, ob genug Prüfer in die Simulation eingebaut worden sind, seien es nun Assertions, Monitore, Checker, oder andere Mechanismen zur Erkennung von Fehlern. Input Coverage beeinflusst ja nur die Chancen, dass ein Fehler im Code wirklich ausgeführt wird und damit zu einem Fehlverhalten der Schaltung führt, anhand dessen der Fehler erkannt werden könnte. Ob dieses Fehlverhalten wirklich erkannt wird, hängt von den der Schaltung hinzugefügten Prüfern ab. Wie präzise diese das Schaltungsverhalten überwachen, geht nicht in die Bestimmung der Input Coverage ein. Im Extremfall könnte eine Simulation beste Werte für die Input Coverage liefern, obwohl sie keinen einzigen Prüfer enthält und so die Antwort der Schaltung auf den Eingabetrace gar nicht untersucht wird. Offensichtlich haben also auch die Prüfer wesentlichen Anteil an der Qualität der Verifikation.

Im Rahmen dieser Arbeit wird der Begriff Output Coverage einer Verifikation genutzt, um die Qualität der Prüfer zu beschreiben. Input und Output Coverage liefern orthogonale Beiträge zu der Qualität einer Verifikation. An der Zusammenfassung dieser eigentlich unabhängigen Werte zu einer Aussage über die Verifikationsqualität wird gearbeitet [Bailey 2007].

Für die Aussagekraft der Output Coverage werden meist am Anfang eines Verifikationsprojekts wichtige Weichenstellungen getroffen, nämlich bei der Festlegung des Verifikationsplans, und dabei speziell während der Festlegung der Verifikationsziele. Während dieser Phase wird informell gearbeitet: Die Spezifikation wird analysiert und in einzelne Verifikationsteilaufgaben aufgeteilt. Dieser Schritt ist kritisch, denn wenn dabei Verifikationsziele nicht identifiziert werden, werden diese Verifikationsziele später nicht geprüft, und entsprechendes Fehlverhalten wird nicht erkannt.

Weil dieser Prozess informell ist, lässt er sich nicht grundlegend durch ein Werkzeug unterstützen. Es gibt Hilfestellungen, um die Beziehung zwischen Spezifikation und Verifikationsplan durch Querverweise zu dokumentieren und dadurch Textstellen der Spezifikation zu identifizieren, die noch nicht durch einen Eintrag im Verifikationsplan abgedeckt sind. Aber ob die solcherart unterversorgten Textstellen wirklich einen Eintrag im Verifikationsplan gebraucht haben, können diese Werkzeuge nicht erkennen, weil sich ihnen der Inhalt der Textstellen nicht erschließt. Eine wichtige Qualitätssicherungsmaßnahme während der Erstellung des Verifikationsplans sind Reviews durch alle Verantwortlichen für Spezifikation und Schaltungsentwurf.

In vielen Designteams sind diese Reviews die Hauptmaßnahmen zur Qualitätssicherung in Bezug auf Output Coverage. Automatische Werkzeuge zur Prüfung der Output Coverage von

simulationsbasierten Verifikationen [Certess o. J., Grosse/Hampton 2005] werden nur von einigen Designteams eingesetzt. Diese Werkzeuge können erst verwandt werden, wenn die Testbench mit allen ihren Tests und Prüfern weitgehend fertiggestellt ist. Die Untersuchungen basieren auf Mutation Analysis [Offutt/Untch 2000], d.h. sie injizieren Fehler in die Schaltung und untersuchen, ob diese Fehler durch die Simulation gefunden werden. Wenn die injizierten Fehler nicht gefunden werden, wurden zumeist Defizite in der Output Coverage erkannt. Wenn dann das Verifikationsprojekt noch genug Zeit zum Verbessern der Simulation hat, kann die Verifikationsqualität aufgrund dieser Rückmeldungen erhöht werden.

5.2 Output Coverage in der formalen Verifikation

Das Problem unzulänglicher Input Coverage gibt es im Zusammenhang mit formaler Verifikation nur im Umfeld von semiformaler Verifikation, von der im Folgenden abgesehen werden soll. Ein echter Beweis wirkt, als ob die in Frage stehende Eigenschaft oder Assertion mit allen möglichen Eingabetraces untersucht wird. Diese mächtige Untersuchungsmethode prüft allerdings häufig recht einfache Aussagen. Es scheint, dass besonders die Assertions der ABFV viel Fehlverhalten nicht identifizieren können, weil sie nur einen sehr engen Bereich von Funktionalität tatsächlich untersuchen. Bei undifferenzierter Betrachtungsweise drängt sich der Eindruck auf, dass der Vorteil der optimalen Input Coverage formaler Verifikation aufgezehrt wird durch den Nachteil sehr unzulänglicher Output Coverage.
Dass dem nicht so sein muss, sollen verschiedene Arbeiten zeigen, die sich mit der Bestimmung der Output Coverage von formaler Verifikation beschäftigen. Einige dieser Ansätze werden im Folgenden besprochen.

5.2.1 Fehlerinjektionsverfahren

Der in [Hoskote et al. 1999, Hojati 2003] beschriebene Ansatz beruht ähnlich der oben für Simulation vorgestellten Mutation Analysis darauf, Fehler in die Schaltung zu injizieren und dann zu prüfen, ob mindestens eine der Eigenschaften durch einen Eigenschaftsprüfer auf der so modifizierten Schaltung widerlegt wird. Wenn jeder injizierte Fehler zu einer Widerlegung führt, wird die Qualität des Eigenschaftssatzes als ausreichend hoch bezeichnet. Ansonsten wird eine Maßzahl über den Anteil derjenigen injizierten Fehler bestimmt, die von dem Eigenschaftssatz aufgedeckt werden.

Dieses Verfahren ist heuristisch. Es gibt keine Garantie dafür, dass andere als die injizierten Fehler von dem Eigenschaftssatz identifiziert werden. Das Verfahren benötigt neben der Eigenschaftsmenge auch die Schaltungsbeschreibung.

5.2.2 Vollständigkeit einer Spezifikation für eine vorgegebene Schaltung

Der Ansatz in [Katz et al. 1999] basiert auf einer simulation preorder genannten Relation $\leq$ zwischen sog. Kripkestrukturen, die im Rahmen dieser Arbeit als Automaten verstanden werden können. Für zwei Schaltungen und ihre Automaten M und $\bar{M}$ gilt $M \leq \bar{M}$, wenn $\bar{M}$ alle Eingangssignale hat, die auch M hat, und wenn eventuell zusätzliche Eingänge von $\bar{M}$ immer so gewählt werden können, dass $\bar{M}$ sequentiell äquivalent ist zu M. Wenn $M \leq \bar{M}$ und $\bar{M} \leq M$ gilt, sind M und $\bar{M}$ sequentiell äquivalent, d.h. sie haben gleiches Ein- und Ausgabeverhalten.

Zu den bei [Katz et al. 1999] betrachteten temporallogischen Formeln φ lassen sich (unter Umständen indeterministische) Automaten $T(\varphi)$ generieren, die als Tableau von φ bezeichnet werden. Das Tableau $T(\varphi)$ erfüllt die Eigenschaft φ, und weist ein so reichhaltiges Verhalten auf, dass sich jede Schaltung M, die φ ebenfalls erfüllt, im Sinne von $M \leq T(\varphi)$ einbetten lässt.

Wenn ψ die Spezifikation einer Schaltung M ist, die ggf. durch Konjunktion von Teilspezifikationen zusammengesetzt ist, wird zunächst durch normales Property Checking überprüft, ob M die Spezifikation erfüllt. Wenn das so ist, gilt $M \leq T(\psi)$. Die Untersuchung der Output Coverage von ψ beruht dann auf dem Versuch, auch

$$T(\psi) \leq M$$

zu zeigen. Bei dieser Untersuchung werden die Zustände und die Transitionen von $T(\psi)$ und M einander zugeordnet. Wenn M Transitionen oder Zustände hat, die keinen Zuordnungspartner in $T(\psi)$ finden, dann werden durch ψ gewisse Anteile des Verhaltens von M nicht beschrieben und ψ ist keine für M vollständige Spezifikation.

Die Spezifikation ψ wird als vollständig für M bezeichnet, wenn $T(\psi) \leq M$ tatsächlich nachgewiesen werden kann. In diesem Fall sind $T(\psi)$ und M sequentiell äquivalent. Offensichtlich ist eine weitere Schaltung $\bar{M}$ sequentiell äquivalent zu M, sofern ψ auch eine vollständige Beschreibung von $\bar{M}$ ist. [Katz et al. 1999] geht aber nicht darauf ein, ob auf den Nachweis von $T(\psi) \leq \bar{M}$ verzichtet werden könnte. Ohne diese Überlegung kann aber nur davon gesprochen werden, dass ψ eine vollständige Spezifikation für eine gegebene Implementierung ist. Ob ψ unabhängig von einer Implementierung vollständig ist, wird in [Katz et al. 1999] nicht untersucht.

Im Vergleich dazu ist der Vollständigkeitsbegriff dieser Arbeit eine von der konkreten Schaltung unabhängige Eigenart einer Menge von Eigenschaften. Daneben gibt es anwendungsorientierte Einwände gegen [Katz et al. 1999]: Zur Diagnose werden die Tableauautomaten $T(\psi)$ mit der Schaltung M verglichen, sodass die Diagnose nur dann verstanden werden kann, wenn der Benutzer etwas über Tableauautomaten weiß und wie sie mit der ursprünglichen Eigenschaft ψ in Beziehung stehen. Selbst wenn das BDD-basierte Verfahren in [Katz et al. 1999] heute möglicherweise durch leistungsfähigere Basisalgorithmen abgelöst werden könnte, so erscheint doch die Tableaugenerierung selbst als ein recht aufwändiger Schritt, der die Komplexität der Schaltungen beschränken dürfte, die mit dieser Methode untersucht werden können.

5.2.3 Vollständigkeitskriterium

In dieser Arbeit wird Output Coverage daran gemessen, ob der Eigenschaftssatz vollständig ist. Das Vollständigkeitskriterium wurde in Abschnitt 3.4.2 und 3.4.3 informell eingeführt: Eine Menge von Operationseigenschaften heißt vollständig, wenn sie zu Eingabetraces, die die Determinierungsannahmen erfüllen, nur Ausgabetraces zulässt, die ihrerseits die Determinierungsforderungen erfüllen. Dabei darf nicht auf die konkrete Schaltung zurückgegriffen werden.

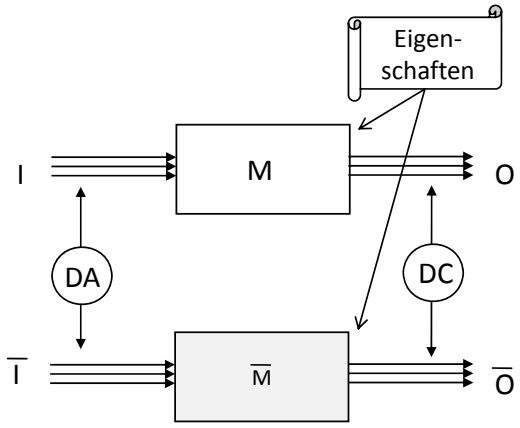

Abbildung 13: Vollständigkeitsprüfung

Dabei lässt sich die Determinierungsannahme wie in Abschnitt 3.4.2 vorgeführt auf die Syntax

```
if g then determined(e) end if;
```

zurückführen, wobei g eine Bedingung und e ein ITL-Ausdruck ist, der zu jedem Zeitpunkt τ den Informationsgehalt des Eingabetraces I extrahiert. Bedingung und Extraktion darf von Ausgabe- und internen Signalen abhängen. Wenn zwei Tripel (I, U, O) und $(\bar{I}, \bar{U}, \bar{O})$ von Traces gegeben sind, so erfüllen diese die Determinierungsannahme $DA(I, U, O, \bar{I}, \bar{U}, \bar{O})$, wenn gilt

$$DA(I,U,O,\bar{I},\bar{U},\bar{O}) = \left(\bigwedge_{t\geq 0}\left(\sim g(t,I,U,O) \wedge \sim g(t,\bar{I},\bar{U},\bar{O})\right) \vee e(t,I,U,O) = e(t,\bar{I},\bar{U},\bar{O})\right)$$

wobei $\sim$ die Negation bezeichnet. Abkürzend wird

$$da(t,I,U,O,\bar{I},\bar{U},\bar{O}) = \left(\left(\sim g(t,I,U,O) \wedge \sim g(t,\bar{I},\bar{U},\bar{O})\right) \vee e(t,I,U,O) = e(t,\bar{I},\bar{U},\bar{O})\right)$$

als lokale Determinierungsannahme eingeführt.

Analog ist die Determinierungsforderung $DC(I, U, O, \bar{I}, \bar{U}, \bar{O})$ und ihr lokales Gegenstück $dc(t, I, U, O, \bar{I}, \bar{U}, \bar{O})$ definiert.

Der Vollständigkeitsprüfer untersucht die Situation, dass zwei beliebige Schaltungen M und $\bar{M}$ gegeben sind, die beide alle Operationseigenschaften erfüllen. Diese beiden Schaltungen sollen Eingabetraces I und $\bar{I}$ erhalten, und dazu Ausgabetraces O und $\bar{O}$, sowie Traces U und $\bar{U}$ interner Signale produzieren. Die Traces sollen die Dependencies $D(I, U, O)$, $D(\bar{I}, \bar{U}, \bar{O})$ und die Determinierungsannahme $DA(I, U, O, \bar{I}, \bar{U}, \bar{O})$ erfüllen. Der Vollständigkeitsprüfer untersucht, ob die Ausgabetraces in dieser Situation die Determinierungsforderungen $DC(I, U, O, \bar{I}, \bar{U}, \bar{O})$ erfüllen. Wenn dies für alle möglichen Eingabestimuli I und $\bar{I}$ gilt, heißt der Eigenschaftssatz vollständig. M und $\bar{M}$ haben dann gleiches Ein-/Ausgabeverhalten, jedenfalls im Rahmen der Determinierungsannahmen und -forderungen. Die Verhältnisse werden in Abbildung 13 dargestellt.

So ergibt sich letztendlich folgende Bedingung für die Vollständigkeit einer Menge $\mathcal{P}$ von Eigenschaften, in der DA statt $DA(I,U,O,\bar{I},\bar{U},\bar{O})$, DC statt $DC(I,U,O,\bar{I},\bar{U},\bar{O})$, D statt $D(I,U,O)$, $\bar{D}$ statt $D(\bar{I},\bar{U},\bar{O})$, P statt $P(t,I,U,O)$ und $\bar{P}$ statt $P(t,\bar{I},\bar{U},\bar{O})$ geschrieben wird:

5-1

$$\bigwedge_{\substack{I,\bar{I}\in\mathcal{W}^{\infty}(\mathbb{i}),\\ U,\bar{U}\in\mathcal{W}^{\infty}(\mathbb{u}),\\ O,\bar{O}\in\mathcal{W}^{\infty}(\mathbb{o})}} DA \wedge \left(D \wedge \bigwedge_{P\in\mathcal{P}} \bigwedge_{t\geq 0} P \right) \wedge \left(\bar{D} \wedge \bigwedge_{P\in\mathcal{P}} \bigwedge_{t\geq 0} \bar{P} \right) \Rightarrow DC$$

5.2.4 Alternative Formulierungen des Vollständigkeitskriteriums

Falls die Determinierungsannahmen einfach die Gleichheit von I und $\bar{I}$ und die Determinierungsforderungen einfach die Gleichheit von O und $\bar{O}$ verlangen, vereinfacht sich das Kriterium zu

5-2

$$\bigwedge_{\substack{I,\in\mathcal{W}^{\infty}(\mathbb{i}),\\ U,\bar{U}\in\mathcal{W}^{\infty}(\mathbb{u}),\\ O,\bar{O}\in\mathcal{W}^{\infty}(\mathbb{o})}} \left(D(I,U,O) \wedge \bigwedge_{\substack{P\in\mathcal{P},\\ \mathrm{t}\geq 0}} P(t,I,U,O) \right) \wedge \left(D(I,\bar{U},\bar{O}) \wedge \bigwedge_{\substack{P\in\mathcal{P},\\ \mathrm{t}\geq 0}} P(t,I,\bar{U},\bar{O}) \right) \Rightarrow O = \bar{O}$$

Dies entspricht der unabhängig von dieser Arbeit entstandenen Formulierung in [Claessen 2006]. Das hier verfolgte, bereits 2005 zum Patent [Bormann/Busch 2005] angemeldete Vollständigkeitskriterium ist durch die Determinierungsannahmen und -forderungen etwas allgemeiner. Die vorliegende Arbeit geht auch insofern über [Claessen 2006] hinaus, als dass das Vollständigkeitskriterium nicht direkt implementiert wird, weil das zu recht hohen Komplexitäten führen würde. Statt dessen wird es in einer auf Operationseigenschaften spezialisierten Form angewandt und kann deswegen durch weniger komplexe Tests geprüft werden, die auch eine detailliertere Gegenbeispielsanalyse ermöglichen.

Eine weitere Abwandlung des Kriteriums aus 5-2 besteht darin, eine der beiden Annahmen über die Anwendbarkeit aller Eigenschaften durch das Modell selbst zu ersetzen, das ja sowieso gegen die Eigenschaften geprüft wurde, sodass leicht einzusehen ist, dass die vorgenannte Formulierung äquivalent ist zu

5-3

$$\bigwedge_{\substack{I,\in\mathcal{W}^{\infty}(\mathbb{i}),\\ U,\bar{U}\in\mathcal{W}^{\infty}(\mathbb{u}),\\ O,\bar{O}\in\mathcal{W}^{\infty}(\mathbb{o})}} (D(I,U,O) \wedge M(I,U,O)) \wedge \left(D(I,\bar{U},\bar{O}) \wedge \bigwedge_{\substack{P\in\mathcal{P},\\ \mathrm{t}\geq 0}} P(t,I,\bar{U},\bar{O}) \right) \Rightarrow O = \bar{O}$$

Eine Implementierung eines Vollständigkeitsprüfers entsprechend dieser Umformulierung wird bei [Große et al. 2008] angestrebt.

5.3 Der Algorithmus

Der Vollständigkeitsprüfer erhält eine Reihe von Eingaben, die in Abschnitt 5.3.1 beschrieben werden. Hauptbestandteil dieser Eingaben sind die Operationseigenschaften, und davon ausgehend führt der Vollständigkeitsprüfer eine Reihe von Tests durch. Es gibt mehrere Klassen von Tests, die jeweils durch eine Formel charakterisiert sind. Die Formeln werden entsprechend der Benutzereingaben instanziiert und anschließend mit einem Eigenschaftsprüfer bewiesen. Sie werden nicht auf der eigentlichen Schaltung verifiziert, sondern auf einem Modellpaar, in dem es für jedes Signal der Schaltung zwei typkorrekte freie Variable gibt, entsprechend der Schaltungen M und $\overline{M}$ des Vollständigkeitskriteriums 5-1 aus Abschnitt 5.2.3. Dass das Zusammenspiel der Tests wirklich das Vollständigkeitskriterium prüft, wird in Abschnitt 5.3.8 bewiesen.

Wie die Formeln zu instanziieren sind, wird jeweils an dem schon vertrauten Beispiel aus Abschnitt 3.2.1 dargestellt, von dem die Eingaben für den Vollständigkeitsprüfer entsprechend Abschnitt 3.4.2 abgeleitet werden.

5.3.1 Eingaben

Zur Durchführung eines Vollständigkeitstests muss dem Vollständigkeitsprüfer der Operationsautomat mit den Operationseigenschaften zur Verfügung gestellt werden. Damit wird die Menge $\mathcal{P}$ der Eigenschaften zusammen mit einer Nachfolgerrelation $\gg$ bekannt gemacht. Die Nachfolgerrelation bestimmt, welche Eigenschaften entsprechend des Operationsautomaten aufeinanderfolgen dürfen. Mit jeder Eigenschaft P sind entsprechend Abschnitt 3.2.3 der Referenzzeitpunkt $T_{ref}^{P}(t, I, U, O)$ und entsprechend Abschnitt 4.3 die Zeitvariablen $T_i^P(t, I, U, O)$, der Annahmenteil $A^P(t, I, U, O)$ und der Beweiszielteil $C^P(t, I, U, O)$ gegeben. Der Referenzzeitpunkt T_{ref}^{P} habe die Form $T + n$, wobei T entweder t oder eine der Zeitvariablen $T_i^P(t, I, U, O)$ ist.

Die Reseteigenschaft R ist ein besonders herausgehobenes Element von $\mathcal{P}$, das die Behandlung des Resetverhaltens entsprechend Abschnitt 4.1.1 sicherstellen soll. Der Annahmeteil der Reseteigenschaft besteht aus der Bedingung $\varrho(I_{t-r}, I_{t-r+1}, I_{t-r+2}, \ldots I_{t-1})$, mit der die Resetsequenzen beschrieben werden, sodass $R(0, I, U, O)$ das Resetverhalten der Schaltung beschreibt.

Die Determinierungsforderungen werden entsprechend der einzelnen $determined$-Festlegungen durch lokale Determinierungsforderungen $dc_\nu(\tau, I, U, O, \bar{I}, \bar{U}, \bar{O})$ formalisiert.

Der Benutzer gibt für jede Eigenschaft P und für jede Determinierungsforderung an, in welchem sogenannten Determinierungszeitraum diese Determinierungsforderung von der Eigenschaft erfüllt sein soll. Der Determinierungszeitraum wird durch seinen ersten Zeitpunkt dl_k^P und seinen letzten Zeitpunkt dh_k^P charakterisiert. dl_k^P ist von der Form $t + n$ und dh_k^P hat die Form $T + n$, wobei t der Anfangszeitpunkt der Eigenschaft und T entweder t oder eine der Zeitvariablen $T_k^P(t, I, U, O)$ ist. Entsprechend ist der Determinierungszeitraum eine Funktion $[dl_k^P(t, I, U, O), dh_k^P(t, I, U, O)]$, die von t und den Traces der Eingabe-, Ausgabe- und der internen Signale abhängt. Die Determinierungsforderungen für die Reseteigenschaft werden alle für Zeitpunkte ≥ 0, d.h. für die Zeiten nach der Resetsequenz verlangt.

Um Zwischenergebnisse über die Determinierung der in Abschnitt 3.2.2 eingeführten sichtbaren Register formulieren zu können, kann der Benutzer jeder Eigenschaft P lokale Determinierungsforderungen $Dloc^P(t, I, U, O, \bar{I}, \bar{U}, \bar{O})$ mitgeben. Sie umfasst unter Umständen mehre-

re Einzelbedingungen in der Gestalt entsprechend Abschnitt 3.4.2, die allerdings zu einzelnen Zeitpunkten relativ zu t oder einer der Zeitvariablen T_i^P angenommen werden.

Zur Schreibvereinfachung wird geschrieben

5-4

$$Det^P(t) = Dloc^P(t, I, U, O, \bar{I}, \bar{U}, \bar{O}) \wedge \bigwedge_i \bigwedge_{\tau \in [dl_i^P(t,I,U,0), dh_i^P(t,I,U,O)]} dc_i(\tau, I, U, O, \bar{I}, \bar{U}, \bar{O})$$

Die Assertions und Constraints unter den Dependencies des Vollständigkeitsbeweises bilden die Bedingung $D(I, U, O)$ auf den Traces der Eingabe-, internen und Ausgabesignale der Schaltung.

Verschiedene Vorgaben sind redundant. Die Nachfolgerrelation ≫, die Referenzzeitpunkte T_{ref}^P, die Determinierungszeiträume $[dl_i^P(t, I, U, O), dh_i^P(t, I, U, O)]$ und eigentlich sogar die lokalen Determinierungsbedingungen $Dloc^P(t, I, U, O, \bar{I}, \bar{U}, \bar{O})$ lassen sich alle aus einer vorgegebenen Menge von Operationseigenschaften automatisch ableiten. Tatsächlich zielten erste Arbeiten [Busch 2005] auch auf einen Vollständigkeitsprüfer, der diese Vorgaben selbst automatisch bestimmt. In diesen Arbeiten wurden die Determinierungs- und die Fallunterscheidungstests entwickelt. Die Arbeiten ließen aber sichtbare Register außer Acht, verfehlten damit den Nachweis des Vollständigkeitskriteriums und erkannten nicht die Notwendigkeit eines Nachfolgertests. Ferner waren die Algorithmen so aufwändig, dass nur recht kleine Schaltungen untersucht werden konnten.

Aus heutiger Sicht hat es sich nicht nur aus Komplexitätsgründen bewährt, den Benutzer diese an sich redundanten Informationen separat vorgeben zu lassen: Sie spiegeln sowieso die Intentionen des Verifikateurs, gehören in den meisten Fällen auf natürliche Weise zu den Schaltungsbeschreibungen durch Operationsautomaten und können daher leicht vom Benutzer eingegeben werden. Sie erlauben außerdem im Falle von Unvollständigkeit aussagekräftigere Diagnosen.

5.3.2 Beispiel

Der Algorithmus wird illustriert an dem Beispiel aus Abschnitt 3.2.1 mit dem Transaktionsautomaten entsprechend Abbildung 4. Die Eigenschaft read_new_row aus Abschnitt 3.2.5 wird geschrieben als

```
property read_new_row is
dependencies: no_reset, processor_protocol;

assume:
at    t:                state = row_act;
at    t:                request = '1';
at    t:                rw = '1';
at    t:                address /= last_row;

prove:
at    t+9:              state = row_act;
at    t+9:              last_row = prev(row(address));
during [t+1, t+7]:      ready = '0';
at    t+8:              ready = '1';
at    t+9:              ready = '0';
at    t+8:              rdata = prev(sd_rdata);
```

```
do_read(t, sd_ctrl, sd_addr, address);

end property;
```

Diese Darstellung ist äquivalent mit der aus Abschnitt 3.2.5. Es wurde nur ein Makro do_read eingeführt, um sie zu kompaktieren:

```
macro do_read(tt: timepoint; sd_ctrl: bit;
      sd_addr, address: bit_vector): temporal :=
at    t+1:              sd_ctrl = prech;
at    t+2:              sd_ctrl = nop;
at    t+3:              sd_ctrl = activate;
at    t+3:              sd_addr = row(address);
at    t+4:              sd_ctrl = nop;
at    t+5:              sd_ctrl = read;
at    t+5:              sd_addr = col(address);
at    t+6:              sd_ctrl = stop;
during [t+7, t+9]:      sd_ctrl = nop;
end macro;
```

Der Referenzzeitpunkt $TRef\_R$ ist $t + 9$. Zu der Eigenschaft gehört die lokale Determinierungsforderung

```
at t+9: determined(last_row);
```

Daneben werde auch eine Eigenschaft über Schreibzugriffe betrachtet:

```
property write_old_row is
dependencies: no_reset, processor_protocol;

assume:
at t:       state = row_act;
at t:       request = '1';
at t:       rw = '0';
at t:       address = last_row;

prove:
at t+2:     state = row_act;
at t+2:     last_row = prev(last_row, 2);
at t+1:     ready = '1';
at t+2:     ready = '0';
at t+1:     sd_ctrl = write;
at t+1:     sd_addr = col(address);
at t+1:     sd_wdata = wdata;
at t+2:     sd_ctrl = stop;

end property;
```

Während dieser Operation soll das Register last_row nicht mit einem neuen Wert belegt werden, daher muss nachgewiesen werden, dass es bei Abschluss der Operation noch den selben Wert hat wie bei ihrem Beginn.

Für diese Eigenschaft ist der Referenzzeitpunkt $TRef\_W = t + 2$. Zu der Eigenschaft gehört die lokale Determinierungsforderung

```
at t+2: determined(last_row);
```

Der Beweis der Eigenschaften lässt sich unter den Dependencies

```
constraint no_reset :=
      reset = '0';
end constraint;

constraint processor_protocol :=
      if request = '1' and ready = '0' then
            next(request) = '1' and
            next(address) = address and
            next(rw) = rw and
            if rw = '0' then next(wdata) = wdata;
      end if;
end constraint;
```

führen. Die Vollständigkeitsprüfung wird unter der Dependency $processor\_protocol$ geführt.

Die Eigenschaften können entsprechend Abbildung 4 in beliebiger Reihenfolge auftreten, daher gilt für die Nachfolgerrelation sowohl $read\_new\_row \gg write\_old\_row$, $write\_old\_row \gg read\_new\_row$, $read\_new\_row \gg read\_new\_row$ als auch $write\_old\_row \gg write\_old\_row$.

In Abschnitt 3.4.2 wurden bereits die Determinierungsforderungen

```
determined(sd_ctrl);
determined(ready);
if rw = '1' and ready = '1' then determined (rdata) end if;
if sd_ctrl = write then determined (sd_wdata) end if;
if sd_ctrl = read or sd_ctrl = write or sd_ctrl = activate
            then determined(sd_addr) end if;
```

und die Determinierungsannahmen

```
determined (request);
if request = '1' then determined(rw) end if;
if request = '1' then determined(address) end if;
if request = '1' and rw = '0' then determined(wdata) end if;
if prev(sd_ctrl, 2) = read then determined (sd_rdata) end if;
```

vorgestellt. Die Determinierungszeiträume sind $[t+1, TRef\_R] = [t+1, t+9]$ bzw. $[t+1, TRef\_W] = [t+1, t+2]$.

5.3.3 Ketten von Operationseigenschaften

Die Grundidee des Vollständigkeitsprüfers besteht darin, Ketten von Operationseigenschaften

5-5

$$\prod_{j \geq 0} P_j(t_j, I, U, O)$$

zu bilden, wobei $t_0 = 0$, $t_{j+1} = T_{ref}^{P_j}$ und dann nachzuweisen, dass jedes Verhalten der vollständig zu verifizierenden Schaltung durch mindestens eine Kette im Rahmen der Determinierungsbedingungen eindeutig beschrieben wird. P_0 ist dabei die Reseteigenschaft R und beschreibt das Verhalten der Schaltung nach dem Reset. Die Ketten brechen ab, sobald ein t_{j+1} unendlich wird.

Ein erstes Teilziel des Vollständigkeitsprüfers ist der Nachweis, dass sich für jeden Eingabestimulus, den die Assertions und Constraints unter den Dependencies der Vollständigkeitsprüfung zulassen, eine Eigenschaftskette finden lässt, in der jeder Annahmeteil durch den Eingabestimulus erfüllt wird. Dafür führt der Vollständigkeitsprüfer die Fallunterscheidungstests durch, die in Abschnitt 5.3.4 beschrieben werden.

Es soll dann überprüft werden, dass diese Kette den Ausgabetrace O determiniert. Dabei ist jede Eigenschaft P_j alleine dafür verantwortlich, jede Determinierungsbedingung in ihrem jeweiligen Determinierungszeitraum zu erfüllen. Dies wird vom Determinierungstest geprüft, zusammen mit der Frage, ob die Zeitbereiche aufeinanderfolgender Eigenschaften auch nahtlos aneinander angrenzen. Der Determinierungstest wird in Abschnitt 5.3.6 vorgestellt.

Zusätzlich muss sichergestellt werden, dass jede Vorgängereigenschaft P_{j-1} genug über die internen Signale beweist, dass die Anwendbarkeit der Nachfolgereigenschaft P_j determiniert ist, d.h. eindeutig sichergestellt ist. Dies prüft der Nachfolgertest aus Abschnitt 5.3.5.

5.3.4 Fallunterscheidungstest

Die Fallunterscheidungstests stellen die Existenz von Ketten zu jedem Eingabetrace sicher. Für sich genommen, untersucht jeder Fallunterscheidungstest die Situation, dass an einem beliebig gewählten Zeitpunkt t die Eigenschaft $P(t, I, U, O)$ gehalten hat. Der Fallunterscheidungstest prüft, ob der Annahmenteil mindestens einer der möglichen Nachfolgereigenschaften erfüllt werden kann. Es wird verlangt, dass mindestens ein Annahmeteil der möglichen Nachfolgereigenschaften im Anschluss daran gilt. Zur Schreibvereinfachung wird D statt $D(I, U, O)$ und $|P| = A^P \wedge C^P$ geschrieben. Dann hat der Fallunterscheidungstest die Form

5-6

$$\bigwedge_{I,U,O} \bigwedge_{t \geq 0} D \wedge |P| \Rightarrow \bigvee_{Q \in \mathcal{P}: P \gg Q} A^Q\left(T_{ref}^P(t, I, U, O), I, U, O\right)$$

Der Test wird auf einem trivialen Modell durchgeführt, in dem die Signale aus den Mengen $\mathbb{i}, \mathbb{u}, \mathbb{o}$ freie Variablen sind.

Die aktuelle Implementierung des Tests beschränkt sich auf Eigenschaften mit endlichen Zeitvariablen entsprechend Abschnitt 4.3.1.

Wenn der Fallunterscheidungstest für alle Eigenschaften inklusive der Reseteigenschaft gilt, dann folgt per Induktion, dass es zu jedem Eingabetrace eine Kette (P_i) gibt, in der jeder Annahmenteil in der Formel 5-5 durch den Eingabetrace erfüllt wird.

Die typische Verifikationslücke, die durch den Fallunterscheidungstest aufgedeckt wird, ist ein durch keine Eigenschaft abgedecktes Verhalten des Eingabetraces nach der Eigenschaft P. Die Diagnose für einen solchen Fehler listet alle Annahmenteile A^Q auf und legt Rechenschaft darüber ab, warum der jeweilige Annahmenteil nicht erfüllt wird.

Der Vollständigkeitsprüfer generiert die entsprechenden Beweisaufgaben aus dem Text der ursprünglichen Eigenschaften. Im Beispiel aus Abschnitt 5.3.2 ergibt sich als Fallunterscheidungstest

```
property case_split;
dependencies: processor_protocol;

assume:
-- Annahmenteil von write_old_row
at t:       state = row_act;
at t:       request = '1';
at t:       rw = '0';
at t:       address = last_row;
-- Beweiszielteil von write_old_row
at t+2:     state = row_act;
at t+2:     last_row = prev(last_row, 2);
at t+1:     ready = '1';
at t+2:     ready = '0';
at t+1:     sd_ctrl = write;
at t+1:     sd_addr = col(address);
at t+1:     sd_wdata = wdata;
at t+2:     sd_ctrl = stop;

prove:
either
      -- Annahmenteil von write_old_row, verschoben um TRef_W
      at TRef_W:        state = row_act;
      at TRef_W:        request = '1';
      at TRef_W:        rw = '0';
      at TRef_W:        address = last_row;
or
      -- Annahmenteil von read_new_row, verschoben um TRef_W
      at TRef_W:        state = row_act;
      at TRef_W:        request = '1';
      at TRef_W:        rw = '1';
      at TRef_W:        address /= last_row;
or
      … (verschobene Annahmen weiterer Nachfolgeoperationen)
end either;
```

5.3.5 Nachfolgertest

Der Nachfolgertest stellt fest, ob die Anwendbarkeit der Nachfolgereigenschaft Q nach einer Vorgängereigenschaft P determiniert ist. Es darf also nur vom Eingabetrace I und von den in $Dloc^P$ determinierten sichtbaren Registern abhängen, ob der Annahmeteil $A^Q(T^P_{ref}(t,I,U,O),I,U,O)$ erfüllt und damit die Eigenschaft Q eine Nachfolgereigenschaft von P ist. Im Folgenden werden jeweils offensichtliche Argumentlisten fortgelassen: Es wird DA statt $DA(I,U,O,\bar{I},\bar{U},\bar{O})$ geschrieben, D ersetzt $D(I,U,O)$, $\bar{D}$ ersetzt $D(\bar{I},\bar{U},\bar{O})$, A^Q ersetzt $A^Q\left(T^P_{ref}(t,I,U,O),I,U,O\right)$, $\bar{A}^Q$ ersetzt $A^Q\left(T^P_{ref}(t,I,U,O),\bar{I},\bar{U},\bar{O}\right)$, T_i^Q ersetzt $T_i^Q\left(T^P_{ref}(t,I,U,O),I,U,O\right)$ und $\bar{T}_i^Q$ ersetzt $T_i^Q\left(T^P_{ref}(t,I,U,O),\bar{I},\bar{U},\bar{O}\right)$. Ferner wird $|P| = A^P(t,I,U,O) \wedge C^P(t,I,U,O)$ und $|\bar{P}| = A^P(t,\bar{I},\bar{U},\bar{O}) \wedge C^P(t,\bar{I},\bar{U},\bar{O})$ geschrieben. Damit hat der Nachfolgertest die Gestalt

5-7

$$\bigwedge_{I,\bar{I},U,\bar{U},O,\bar{O}} \bigwedge_{t\geq 0} (DA \wedge D \wedge \bar{D} \wedge |P| \wedge |\bar{P}| \wedge Det^P(t) \wedge A^Q) \Rightarrow \left(\bar{A}^Q \wedge \bigwedge_i T_i^Q = \bar{T}_i^Q\right)$$

Der Test wird wiederum auf einem trivialen Modell durchgeführt, in dem die Signale aus den Mengen $\mathbb{i}, \mathbb{u}, \mathbb{o}, \bar{\mathbb{i}}, \bar{\mathbb{u}}$ und $\bar{\mathbb{o}}$ freie Variablen sind.

Eine Situation, in der der Nachfolgertest verletzt wird, wird in Abbildung 14 beschrieben: Die Vorgängeroperation P hat einen konzeptionellen Endzustand $state = idle$, der aber von den Anfangszuständen der Nachfolgeoperationen Q_1 und Q_2 jeweils nicht vollständig abgedeckt wird. Statt dessen wird aufgrund eines in P nicht erwähnten internen Zählers cnt über die Nachfolgeoperation entschieden. Da die vorhergehende Operation nicht beschreibt, wie dieses Signal gebildet wird, können zwei unterschiedliche Schaltungen das Signal cnt unterschiedlich bilden und deswegen zu den gleichen Eingabetraces unterschiedliche Traces interner Signale bilden. Dann könnten beide Schaltungen unterschiedliche Nachfolgeoperationen mit unterschiedlichem Ausgabeverhalten ausführen, und das steht im Widerspruch zur Determiniertheit der Ausgangssignale. Es kann auch nur eines der beiden Verhalten vom Benutzer erwünscht sein, sodass es sich bei dieser Situation auch wirklich um eine Verifikationslücke

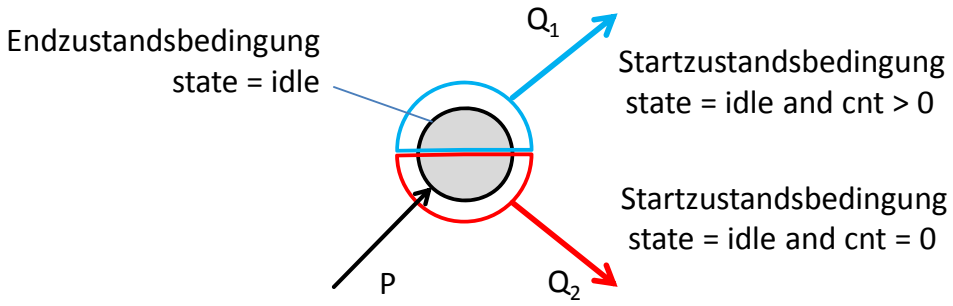

Abbildung 14: Gegenbeispiel zum Nachfolgertest

handelt.

Im Gegenbeispiel zum Nachfolgetest über z.B. die Paarung P als Vorgängereigenschaft und Q_1 als Nachfolgereigenschaft schlägt sich dies dadurch nieder, dass ein Eingabetrace angegeben wird, der die Voraussetzungen von P und Q_1 erfüllt und zu dem zwei Traces interner Signale bestimmt, die zwar beide den Annahmenteil A^P und den Beweiszielteil C^P erfüllen, von denen aber nur einer die Startzustandsbedingung von Q_1 erfüllt, und der andere die Startzustandsbedingung von Q_1 verletzt. Zur Diagnose wird der Vollständigkeitsprüfer den Eingabetrace und die Traces interner Signale ausgeben, und zeigen, dass nur der eine dazu führt, dass die Nachfolgeoperation tatsächlich ausgeführt wird.

Im Beispiel aus Abschnitt 5.3.2 ergibt sich als Nachfolgetest für die Abfolge der Eigenschaften $write\_old\_row$ und $read\_new\_row$ die folgende Eigenschaft, bei der die Objekte der zweiten Schaltung $\bar{M}$ durch das Suffix _c gekennzeichnet sind.

```
property successor_test;
dependencies: processor_protocol, processor_protocol_c;

-- Determinierungsannahmen
during [t, TRef_W+9]:   request = request_c;
during [t, TRef_W+9]:   (request = 0 and request_c = 0) or rw = rw_c;
during [t, TRef_W+9]:   (request = 0 and request_c = 0)
                        or address = address_c;
during [t, TRef_W+9]:   (
                              (request = 0 or rw = 1) and
                              (request_c = 0 and rw_c = 1)
                        ) or wdata = wdata_c;
during [t, TRef_W+9]:   (
```

```
                              prev(sd_ctrl, 2) /= read and
                              prev(sd_ctrl_c, 2) /= read
                        ) or sd_rdata = sd_rdata_c;

-- Determinierungsforderungen von write_old_row
during [t+1, t+2]:      sd_ctrl = sd_ctrl_c;
during [t+1, t+2]:      ready = ready_c;
during [t+1, t+2]:      (
                              (rw = 0 or ready = 0) and
                              (rw_c = 0 and ready_c = 0)
                        ) or rdata = rdata_c;
during [t+1, t+2]:      (sd_ctrl /= write and sd_ctrl_c /= write) or
                        sd_wdata = sd_wdata_c;
during [t+1, t+2]:      (
                              sd_ctrl /= read and
                              sd_ctrl /= write and
                              sd_ctrl /= activate and
                              sd_ctrl_c /= read and
                              sd_ctrl_c /= write and
                              sd_ctrl_c /= activate
                        ) or sd_addr = sd_addr_c;

-- lokale Determinierungsbedingung von write_old_row
at t+2:     last_row = last_row_c;

-- Annahmenteil von write_old_row
at t:       state = row_act;
at t:       request = '1';
at t:       rw = '0';
at t:       address = last_row;
-- Beweiszielteil von write_old_row
at t+2:     state = row_act;
at t+2:     last_row = prev(last_row, 2);
at t+1:     ready = '1';
at t+2:     ready = '0';
at t+1:     sd_ctrl = write;
at t+1:     sd_addr = col(address);
at t+1:     sd_wdata = wdata;
at t+2:     sd_ctrl = stop;

-- kopierter Annahmenteil von write_old_row
at t:       state_c = row_act;
at t:       request_c = '1';
at t:       rw_c = '0';
at t:       address_c = last_row_c;
-- kopierter Beweiszielteil von write_old_row
at t+2:     state_c = row_act;
at t+2:     last_row_c = prev(last_row_c, 2);
at t+1:     ready_c = '1';
at t+2:     ready_c = '0';
at t+1:     sd_ctrl_c = write;
at t+1:     sd_addr_c = col(address);
at t+1:     sd_wdata_c = wdata;
at t+2:     sd_ctrl_c = stop;

-- Annahmenteil von read_new_row, verschoben
at TRef_W:        state = row_act;
at TRef_W:        request = '1';
at TRef_W:        rw = '1';
at TRef_W:        address /= last_row;

prove:
```

```
-- kopierter Annahmenteil von read_new_row, verschoben
at TRef_W:        state_c = row_act;
at TRef_W:        request_c = '1';
at TRef_W:        rw_c = '1';
at TRef_W:        address_c /= last_row_c;
end property;
```

5.3.6 Determinierungstest

Wenn durch den Nachfolgertest sichergestellt ist, dass die Anwendbarkeit einer Nachfolgereigenschaft Q nach der Vorgängereigenschaft P determiniert ist, wird die Frage interessant, ob Q auch wirklich alle Ausgabesignale und sichtbaren Register zu den richtigen Zeitpunkten determiniert. Dies prüft der Determinierungstest. Unter Verwendung der Schreibvereinfachungen aus dem vorangegangenen Abschnitt und den Abkürzungen $T_{ref}^{P} = T_{ref}^{P}(t, I, U, O)$, $|Q| = A^{Q}\left(T_{ref}^{P}, I, U, O\right) \wedge C^{Q}\left(T_{ref}^{P}, I, U, O\right)$, $|\bar{Q}| = A^{Q}\left(T_{ref}^{P}, \bar{I}, \bar{U}, \bar{O}\right) \wedge C^{Q}\left(T_{ref}^{P}, \bar{I}, \bar{U}, \bar{O}\right)$ $dh_i^P = dh_i^P(t, I, U, O)$, $dl_i^Q = dl_i^Q\left(T_{ref}^{P}, I, U, O\right)$, $T_i^Q = T_i^Q\left(T_{ref}^{P}, I, U, O\right)$ und $\bar{T}_i^Q = T_i^Q\left(T_{ref}^{P}, \bar{I}, \bar{U}, \bar{O}\right)$ hat er folgende Gestalt:

5-8

$$\bigwedge_{I,U,O,\bar{I},\bar{U},\bar{O}} \bigwedge_{t \geq 0} \left(DA \wedge D \wedge \bar{D} \wedge |P| \wedge |\bar{P}| \wedge |Q| \wedge |\bar{Q}| \wedge Det^{P}(t)\right)$$
$$\Rightarrow \left(Det^{Q}(T_{ref}^{P}) \wedge \left(\bigwedge_{i} \bigwedge_{\tau \in \left[dh_i^P + 1, dl_i^Q - 1\right]} dc_i(\tau, I, U, O, \bar{I}, \bar{U}, \bar{O}) \right) \right)$$

Der erste Teil des Sukzedenten stellt sicher, dass die Eigenschaft Q ihre Determinierungsaufgaben erfüllt. Der zweite Teil prüft etwaige Lücken zwischen den Determinierungsbereichen der Eigenschaften P und Q.

Der Test wird wiederum auf einem trivialen Modell durchgeführt. Die Dependencies D und $\bar{D}$ können in diesem Zusammenhang zusätzliche Information über das Verhalten von Signalen zur Verfügung stellen, die für den Nachweis der Determinierung erforderlich sind. So können z.B. Lücken in der Beschreibung des Verhaltens eines Ausgangssignals aufgefüllt werden, die z.B. in Operationseigenschaften über Pipelines entstehen können, deren einzelne Stufen durch ein Stall-Signal angehalten werden können (siehe z.B. [Bormann/Beyer/Skalberg 2006]).

Die typische von einem Determinierungstest aufgedeckte Verifikationslücke ist eine Nachfolgereigenschaft Q, die eine Determinierungsbedingung über den Wert eines Ausgabesignals nicht erfüllt. O und $\bar{O}$ können dann zu diesem Zeitpunkt unterschiedliche Werte haben, was zwar $|Q|$ und $|\bar{Q}|$ erfüllt, nicht aber die Determinierungsbedingung.

Die dem Beispiel aus Abschnitt 5.3.2 entsprechende Eigenschaft wird relativ lang. Deshalb werden Teile ausgeblendet, die wortgleich aus dem Nachfolgertest des Abschnitts 5.3.5 entnommen werden können.

```
property determination_test;
dependencies: processor_protocol, processor_protocol_c;

-- Determinierungsannahmen
... (siehe Abschnitt 5.3.5)
```

```
-- Determinierungsforderungen von write_old_row
... (siehe Abschnitt 5.3.5)
-- lokale Determinierungsbedingung von write_old_row
at t+2:           last_row = last_row_c;

-- Annahmenteil von write_old_row
... (siehe Abschnitt 5.3.5)
-- Beweiszielteil von write_old_row
... (siehe Abschnitt 5.3.5)

-- kopierter Annahmenteil von write_old_row
... (siehe Abschnitt 5.3.5)
-- kopierter Beweiszielteil von write_old_row
... (siehe Abschnitt 5.3.5)

-- Annahmenteil von read_new_row, verschoben
at TRef_W:        state = row_act;
at TRef_W:        request = '1';
at TRef_W:        rw = '1';
at TRef_W:        address /= last_row;
-- Beweiszielteil von read_new_row, verschoben
at TRef_W+9:      state = row_act;
at TRef_W +9:     last_row = prev(row(address));
during [TRef_W +1, TRef_W +7]:      ready = '0';
at TRef_W +8:     ready = '1';
at TRef_W +9:     ready = '0';
at TRef_W +8:     rdata = prev(sd_rdata);
do_read(TRef_W, sd_ctrl, sd_addr, address);

-- kopierter Annahmenteil von read_new_row, verschoben
at TRef_W:        state_c = row_act;
at TRef_W:        request_c = '1';
at TRef_W:        rw_c = '1';
at TRef_W:        address_c /= last_row_c;
-- kopierter Beweiszielteil von read_new_row, verschoben
at TRef_W+9:      state_c = row_act;
at TRef_W +9:     last_row_c = prev(row(address_c));
during [TRef_W +1, TRef_W +7]:      ready_c = '0';
at TRef_W +8:     ready_c = '1';
at TRef_W +9:     ready_c = '0';
at TRef_W +8:     rdata_c = prev(sd_rdata_c);
do_read(TRef_W, sd_ctrl_c, sd_addr_c, address_c);

prove:
-- Determinierungsforderungen
during [t+3, tRef_W+9]: sd_ctrl = sd_ctrl_c;
during [t+3, tRef_W+9]: ready = ready_c;
during [t+3, tRef_W+9]: (
                              (rw = 0 or ready = 0) and
                              (rw_c = 0 and ready_c = 0)
                         ) or rdata = rdata_c;
during [t+3, tRef_W+9]: (sd_ctrl /= write and sd_ctrl_c /= write) or
                         sd_wdata = sd_wdata_c;
during [t+3, tRef_W+9]: (
                              sd_ctrl /= read and
                              sd_ctrl /= write and
                              sd_ctrl /= activate and
                              sd_ctrl_c /= read and
                              sd_ctrl_c /= write and
                              sd_ctrl_c /= activate
                         ) or sd_addr = sd_addr_c;
```

```
-- lokale Determinierungsbedingung
at TRef_W+9:            last_row = last_row_c;
end property;
```

5.3.7 Tests über die Reseteigenschaft

Die aufzubauenden Ketten beginnen immer mit der Reseteigenschaft R. Diese Eigenschaft hat keine Vorgängereigenschaft. Daher müssen die entsprechenden Tests leicht abgewandelt werden.

Um nachzuweisen, dass die Anwendung der Reseteigenschaft determiniert ist, wird eine Abwandlung des Nachfolgertests eingesetzt.

5-9

$$\bigwedge_{I,\bar{I},U,\bar{U},O,\bar{O}} \bigwedge_{t\geq 0} (DA \wedge D \wedge \bar{D} \wedge A^R) \Rightarrow \left(\bar{A}^R \wedge \bigwedge_i T_i^R = \bar{T}_i^R \right)$$

Dabei werden die bereits eingeführten Schreibabkürzungen verwandt, sowie $A^R = A^R(0, I, U, O)$, $\bar{A}^R = A^R(0, \bar{I}, \bar{U}, \bar{O})$, $T_i^R = T_i^R(0, I, U, O)$ und $\bar{T}_i^R = T_i^R(0, \bar{I}, \bar{U}, \bar{O})$.

Zu guter Letzt muss nachgewiesen werden, dass die Reseteigenschaft ihre Ausgaben determiniert. Dies wird mit

5-10

$$\bigwedge_{I,U,O,\bar{I},\bar{U},\bar{O}} \bigwedge_{t\geq 0} (DA \wedge D \wedge \bar{D} \wedge |R| \wedge |\bar{R}|) \Rightarrow Det^R(t)$$

geprüft, wobei die Schreibabkürzungen $|R| = A^R(0, I, U, O) \wedge C^R(0, I, U, O)$ und $|\bar{R}| = A^R(0, \bar{I}, \bar{U}, \bar{O}) \wedge C^R(0, \bar{I}, \bar{U}, \bar{O})$ beachtet werden.

Beide Resettests werden wiederum auf freien Variablen $\mathbb{i}, \mathbb{u}, \mathbb{o}, \bar{\mathbb{i}}, \bar{\mathbb{u}}$ und $\bar{\mathbb{o}}$ durchgeführt.

5.3.8 Beweis

Es soll nachgewiesen werden, dass die Vollständigkeitsbedingung 5-1 eingehalten ist, wenn auf einer Menge von Operationseigenschaften die Fallunterscheidungs-, Nachfolger-, Determinierungs- und Resettests halten.

Seien zwei Eingabetraces I und $\bar{I}$ gegeben. Sie enthalten jeweils Resetsequenzen $\left(I^{(-r)}, I^{(-r+1)}, I^{(-r+2)}, \dots, I^{(-1)}\right)$ bzw. $\left(\bar{I}^{(-r)}, \bar{I}^{(-r+1)}, \bar{I}^{(-r+2)}, \dots, \bar{I}^{(-1)}\right)$ und setzen sich danach so fort, dass die synchronen Schaltungen M und $\bar{M}$ durch $M(I, U, O)$ und $\bar{M}(\bar{I}, \bar{U}, \bar{O})$ Traces $U, O, \bar{U}$ und $\bar{O}$ von internen und Ausgangssignalen erzeugen. Für diese Traces sei die Determinierungsannahme $DA(I, O, U, \bar{I}, \bar{O}, \bar{U})$ erfüllt. Über die beiden Schaltungen ist nur bekannt, dass sie die Eigenschaften der Menge $\mathcal{P}$ erfüllen. Es ist zu beweisen, dass die Determinierungsforderung $DC(I, O, U, \bar{I}, \bar{O}, \bar{U})$ ebenfalls erfüllt ist.

Der Beweis wird durch vollständige Induktion geführt. Die Induktionshypothese ist, dass eine endliche Kette $(P_0, P_1, P_2, \dots, P_n)$ von Eigenschaften und dazu eine endliche Folge

$(t_0, t_1, t_2, \ldots, t_n)$ mit $t_0 = 0$, $t_{j+1} = T_{ref}^{P_j}$ gefunden wird, sodass durch $\prod_{j=0}^{n} P_j(t_j, I, U, O)$ und $\prod_{j=0}^{n} P_j(t_j, \bar{I}, \bar{U}, \bar{O})$ interne und Ausgabetraces bestimmt werden, für die

$$\bigvee_j \bigvee_{\tau \in \left[0, dh_j^{P_n}\right]} DC_j(\tau, I, U, O, \bar{I}, \bar{U}, \bar{O})$$

und $Dloc^{P_n}(t_n)$ gilt. Ferner gilt $T_{ref}^{P_n} = \bar{T}_{ref}^{P_n}$.

Induktionsbasis: Für $n = 0$ ist die Reseteigenschaft R zu betrachten, angewandt für $t = 0$. Deren Annahmenteil ist aufgrund der Forderungen an die Reseteigenschaft sowohl durch I als auch durch $\bar{I}$ erfüllt. Aufgrund des ersten Tests 5-9 über die Reseteigenschaften sind daher eventuell auftretende Zeitvariablen der Reseteigenschaft für die gestrichenen und die ungestrichenen Traces gleich. Daraus ergibt sich auch für die Referenzzeitpunkte T_{ref}^{R} und $\bar{T}_{ref}^{R}$ Gleichheit und der entsprechende Teil der Induktionshypothese ist nachgewiesen. Weil die Annahme der Reseteigenschaft sowohl für die gestrichenen als auch für die ungestrichenen Traces erfüllt ist, verhalten sich M und $\bar{M}$ für $t = 0$ entsprechend der Reseteigenschaft, sodass Formel 5-10 anwendbar ist und $Det^{R}(0)$ sicherstellt. Aufgrund der Formel 5-4 und den Festlegungen über die rechten Grenzen der Determiniertheitsbereiche der Reseteigenschaft folgt daraus der Rest der Induktionshypothese.

Schritt von n nach $n + 1$: Die Induktionshypothese gelte für n. Insbesondere gilt also $A^{P_n}(t_n, I, U, O)$ und $C^{P_n}(t_n, I, U, O)$. Der Fallunterscheidungstest 5-6 liefert also für $t = t_n$ die Existenz einer Eigenschaft P_{n+1}, deren Annahmenteil für $t = T_{ref}^{P_n}(t_n, I, U, O)$ erfüllt ist. Damit ist t_{n+1} entsprechend der Induktionshypothese festgelegt. Aus dem Nachfolgertest 5-7 folgt, dass auch $A^{P_{n+1}}(t_{n+1}, \bar{I}, \bar{U}, \bar{O})$ erfüllt ist. Damit verhalten sich M und $\bar{M}$ für $t = t_{n+1}$entsprechend der Eigenschaft P_{n+1}. Der Nachfolgertest 5-7 zeigt auch, dass etwaige Zeitvariablen auf den gestrichenen und den ungestrichenen Traces die gleichen Werte annehmen und damit $T_{ref}^{P_{n+1}} = \bar{T}_{ref}^{P_{n+1}}$ ist. Außerdem ist der Determinierungstest auf das Eigenschaftspaar P_n und P_{n+1} sowie auf $t = t_{n+1}$ anwendbar und beweist aufgrund von Formel 5-4 zunächst

$$Dloc^{P_{n+1}}(T_{ref}^{P_n}) \wedge \left(\bigwedge_j \bigwedge_{\tau \in \left[dh_j^{P_n}+1, dh_j^{P_{n+1}}\right]} dc_j(\tau, I, U, O, \bar{I}, \bar{U}, \bar{O}) \right)$$

Wenn dieser Ausdruck mit der Induktionshypothese für n zusammengebracht wird, ergibt sich die Induktionshypothese für $n + 1$.

Damit ist die Gültigkeit der Induktionshypothese für alle n gezeigt, für die t_n endlich bleibt. Das sind entweder alle natürlichen Zahlen, oder es gibt eine größte Zahl N, für die $T_{ref}^{P_N}$ unendlich ist. Im ersten Fall wächst $dh_i^{P_n}$ über alle Grenzen, denn es wird relativ zu t_n bestimmt und t_n wächst über alle Grenzen, weil für jedes $P \in \mathcal{P}$ vorausgesetzt wurde, dass $t < T_{ref}^{P}$ gilt. Im zweiten Fall ist $dh_j^{P_N} = \infty$ für alle j zu denen es Determinierungsforderungen DC_j gibt. Im Endeffekt gilt

$$\bigvee_{j}\bigvee_{\tau\geq 0} dc_j(\tau, I, U, O, \bar{I}, \bar{U}, \bar{O})$$

und das ist nach Abschnitt 5.3.1 die Definition von $DC(I, U, O, \bar{I}, \bar{U}, \bar{O})$. Damit ist der Beweis erbracht.

5.4 Diskussion

Die prominente Rolle des Vollständigkeitsprüfers zur Sicherstellung einer vollständigen Verifikation und zum Beweis ihrer Terminierung wurde in Abschnitt 3.4.3 ausführlich dargestellt und soll hier nicht wiederholt werden. Statt dessen soll auf einige Besonderheiten der Implementierung des Vollständigkeitsprüfers hingewiesen werden.

5.4.1 Fallunterscheidungstests und reaktive Constraints

Abbildung 15 zeigt eine leicht veränderte Version der Eigenschaft aus Abbildung 7, bei der zusätzliche Annahmen über $request$ und rw für die Zeitpunkt $t + 1$ bis $T - 1$ gemacht wurden. Diese zusätzlichen Annahmen sind eigentlich redundant, denn sie sind in der Situation der Eigenschaft eine Folge der zwei Protokollconstraints

```
if ready = '0' and request = '1' then next(request = '1') end if;
if ready = '0' and request = '1' then rw = next(rw) end if;
```

und der Tatsache, dass alle Operationen im Zustand $state = row\_act$ das ready-Signal deaktivieren.

Wenn alle anderen Eigenschaften analog geändert würden, würde der Fallunterscheidungstest ein Gegenbeispiel liefern, das im Prinzip fragt, wo denn die Eigenschaften über den Fall sind, dass sich der Prozessor bezüglich der Signal $request$ und rw nicht protokollgerecht verhält. Diese verblüffende Antwort ist eine Folge der Formulierung 5-6 des Fallunterscheidungstests, denn dort werden die Beweiszielteile C^Q der Nachfolgereigenschaft gar nicht untersucht. Entsprechend nutzlos sind die reaktiven Constraints.

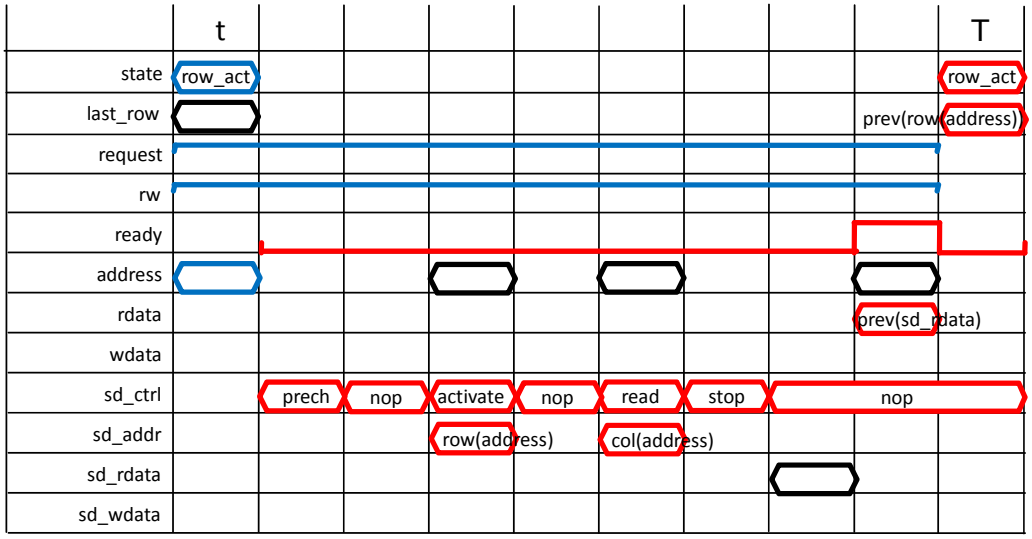

Abbildung 15: Veränderte SDRAM-Eigenschaft (mehr Annahmen über request und rw)

In der Verifikationspraxis werden reaktive Protokollconstraints beim Beweis der Eigenschaften eingesetzt und Annahmenteile, die zu den Protokollconstraints redundant sind, aus der Eigenschaft gelöscht. Daher ist die Eigenschaft aus Abbildung 7 für die Verwendung des Fallunterscheidungstests geeigneter.

5.4.2 Vollständigkeitsbeweis auf Operationsautomaten

Der Vollständigkeitsbeweis für Operationsautomaten wird mit den selben Tests durchgeführt wie der Vollständigkeitsbeweis für Operationseigenschaften. Zwar fehlen in den Annahmen der Operationen die wichtigen Zustände, aber ihre Aufgabe im Vollständigkeitstest wird von der Nachfolgerrelation $\gg$ und den Referenzzeitpunkten T_{ref}^{P}, übernommen. Auf diese Weise ist der Vollständigkeitsprüfer auch einsetzbar als Plausibilitätsprüfung für einen Operationsautomaten, für den es noch keine Implementierung gibt. Diese Plausibilitätsprüfung lässt die Frage der Implementierbarkeit des Operationsautomaten offen. Diese Frage klärt sich erst, wenn es zum Operationsautomaten eine Implementierung gibt, die gegen alle Transitionen des Operationsautomaten verifiziert wurde.

5.4.3 Wichtige Zustände

Der Vollständigkeitsprüfer identifiziert die konzeptionellen Zustände selbständig. Es konnte vermieden werden, dass der Benutzer die konzeptionellen Zustände etwa unter Verwendung einer besonderen Syntax markieren muss. Tatsächlich sind in der Praxis teilweise recht ungewöhnliche Formulierungen konzeptioneller Zustände aufgetreten. Die konzeptionellen Zustände können Eingabe- oder Ausgabesignale enthalten, sie können größere Anteile von Ausgabeverhalten über einen längeren Zeitraum beschreiben, und es können darin Zeitvariablen vorkommen. All dies ist nicht sofort mit dem Begriff eines Zustands vereinbar und würde eine zusätzliche Schwierigkeit bei der Verwendung der vollständigen Verifikation darstellen. Die Tatsache, dass der Vollständigkeitsprüfer die konzeptionellen Zustände selbständig identifiziert, bewahrt also einen Benutzer davor, sich genaue Gedanken darüber machen zu müssen, ob ein Teil seiner Eigenschaft nun für den Anschluss der Nachfolgereigenschaft oder für die Beschreibung des Ausgabeverhaltens oder gar für beides erforderlich ist.

Dabei ist der Vollständigkeitsprüfer auf den Kontrollanteil der konzeptionellen Zustände, also auf die wichtigen Zustände nicht wirklich angewiesen. Er prüft nur im Nachfolgertest, ob die konzeptionellen Zustände aufeinanderfolgender Eigenschaften auch untereinander und im Zusammenhang mit Referenzzeitpunkten und Nachfolgerelation konsistent definiert worden sind. Der Vollständigkeitsprüfer kann aber zufriedengestellt werden, wenn Operationen nur durch ihr Ein- und Ausgabeverhalten charakterisiert werden und wenn die Aufeinanderfolge von Operationen durch die Nachfolgerrelation und die Referenzzeitpunkte festgelegt wird.

6 Kompositionale vollständige Verifikation

Kompositionale Verifikation [Beyer/Bormann 2008] wird in diesem Kapitel in zwei Facetten beschrieben. Das Zusammensetzen vollständiger Verifikationen wird exemplarisch am Beispiel des Entwurfs eines System on Chip (SoC) aus IP-Blöcken vorgestellt, der in Abschnitt 6.1 eingeführt wird, und resultiert in einer effizienten Methode zur Verifikation der Kommunikation der IP-Blöcke auf einem SoC, die in Abschnitt 6.3 behandelt wird.

In Abschnitt 6.4 geht es um das Zerlegen einer vollständigen Verifikation, das bei der Verifikation vieler Module wie z.B. der IP-Blöcke für den SoC-Entwurf notwendig wird. Anders als beim Zusammensetzen vollständiger Verifikationen liegt hier also bei Beginn der Verifikation der gesamte RTL-Code zumindest in einer ersten vorläufigen Version vor und soll vollständig verifiziert werden. Der Ansatz besteht darin, die Schaltung in Cluster genannte Teile (s. Abschnitt 3.5.6) zu zerlegen, die dann jeder für sich vollständig verifiziert werden. Das Vorhandensein des gesamten Verifikationscodes erlaubt einige Vereinfachungen bei der Durchführung der vollständigen Verifikation.

Vom logischen Ablauf her werden natürlich IP-Blöcke, deren vollständige Verifikation mit den Mitteln von Abschnitt 6.4 verifiziert wurden, erst danach zu SoCs kombiniert, deren vollständige Verifikation ihrerseits mit den Mitteln aus Abschnitt 6.3 zusammengesetzt werden. Da aber die Theorie der Zerlegung von vollständigen Verifikationsaufgaben auf der Theorie des Zusammensetzens basiert, wird die Zusammensetzung vor der Zerlegung behandelt. Beim ersten Lesen von Abschnitt 6.3 können alle IP-Blöcke als aus einem Cluster bestehend angesehen werden, dessen Verifikation mit den in den vorherigen Kapiteln beschriebenen Methoden durchgeführt wurde.

Die kompositionale vollständige Verifikation basiert auf so genannten Integrationsbedingungen. Ob Teilschaltungen in einem SoC richtig kommunizieren, kann abschließend dadurch geprüft werden, dass die Integrationsbedingungen interagierender Module oder Cluster mit Hilfe eines IPC-Beweisers abgeglichen werden. Diese Integrationsbedingungen gibt es zu jeder vollständigen Verifikation, unabhängig davon, ob sie auf einem einzelnen Cluster durchgeführt wurde, oder ob sie bereits mit den Mitteln der kompositionalen Verifikation durchgeführt wurde. Daher ebnet dieses Kapitel auch die Bahn für eine hierarchische Anwendung der vollständigen Verifikation auf ganze SoCs.

Hinter der kompositionalen vollständigen Verifikation steckt eine Abwandlung der Theorien über Modelle digitaler Systeme und das zugehörige Assume-Guarantee-Reasoning, mit dem Fragestellungen über die Integration von Komponenten geklärt werden, die entsprechend dieser Theorien modelliert wurden. Die einschlägige Mathematik wird im Anhang präsentiert.

6.1 Kompositionale vollständige Verifikation im IP-basierten SoC-Entwurf

6.1.1 Verifikationsaufgaben beim IP-basierten SoC-Entwurf

Erster Schritt bei einem SoC-Entwurf ist die Partitionierung der Gesamtfunktion in Teilaufgaben. Diese Teilaufgaben werden häufig von IP-Blöcken, d.h. vorentwickelten und vorverifizierten Schaltungen, übernommen. Meist kann unter einer größeren Anzahl verschiedener IP-Blöcke ähnlicher Funktionalität ausgewählt werden. Kriterien bei dieser Auswahl sind natürlich funktionale Aspekte, aber auch die Einschätzung der RTL-Qualität, Nützlichkeit beigepackter Informationen, Lizenzgebühren, Chipfläche, Stromverbrauch, usf.

Zu den funktionalen Aspekten gehört die eigentliche Funktion und abstrakte Maßzahlen beispielsweise über Bearbeitungszeiten und Durchsatz und die an den Interfaces unterstützten Protokolle. Die Frage nach den Protokollen ist entscheidend, um eine Grundvoraussetzung des SoC-Entwurfs zu gewährleisten, weil nur solche Blöcke miteinander kommunizieren können, die das selbe Protokoll unterstützen. Protokolle sind standardisiert worden, um mit einem Wort sagen zu können, welches Protokoll an einem Interface unterstützt wird.

Standardisierung ist aber unflexibel. Sie nimmt einem SoC-Entwickler die Möglichkeit der Abwägung zwischen Leistungsanforderungen an das Protokoll und Kosten in Form von Siliziumfläche bzw. Stromverbrauch. Die heutige SoC-Entwurfspraxis geht deswegen einen Mittelweg, bei dem bestimmte Leistungsmerkmale von Protokollen als optional behandelt werden. Teilweise sind diese Leistungsmerkmale in den Protokollspezifikationen auch tatsächlich als optional bezeichnet, teilweise hat sich aber auch nur die optionale Behandlung eigentlich als verpflichtend spezifizierter Leistungsmerkmale eingebürgert.

Auf diese Optionen muss bei der Auswahl von IP-Blöcken geachtet werden, denn Missachtung führt im einfachen Fall dazu, dass Teile der IP-Blöcke brach liegen, die Leistungsmerkmale des Protokolls implementieren, die vom Kommunikationspartner nicht benutzt werden. In diesem Fall kostet die Missachtung nur Strom und Siliziumfläche, stellt aber das SoC-Projekt nicht generell in Frage. Schlimmer sind Fehler der Art, dass ein Kommunikationspartner Leistungsmerkmale nutzt, die der andere nicht versteht, denn dann kommt es zu Kommunikationsfehlern. Wenn diese Kommunikationsfehler auf Bussen auftreten, ziehen sie häufig alle an diesem Bus angeschlossenen Komponenten in Mitleidenschaft. Kommunikationsfehler sind daher häufig schwere Funktionsfehler des SoCs, die nicht leicht umgangen werden können.

Auf diese möglichen Inkompatibilitäten muss bereits während der Auswahl der IP-Blöcke geachtet werden. Zum Zeitpunkt der Auswahl ist der eigentliche RTL-Code des IP-Blocks meist noch unbekannt, weil entweder die Entwickler keine Zeit hatten, sich mit dem Code zu beschäftigen, oder weil die geschäftlichen Verhandlungen über die Lizensierung des IP-Blocks noch nicht in einem Zustand sind, der die Übergabe des hoch sensiblen RTL-Codes erlauben. Der SoC-Entwickler verlässt sich dann auf die Dokumentation der IP-Blöcke und wird die Dokumentationen über das Interfaceverhalten von IP-Blöcken vergleichen, die im noch zu entwickelnden SoC miteinander kommunizieren sollen. Dieser Abgleich ist fehleranfällig. Die Dokumentation kann unvollständig sein, sie muss nicht notwendigerweise genau zum IP-Block passen und sie kann vom SoC-Entwickler falsch verstanden werden.

Heute werden Inkompatibilitäten, die beim Abgleich der Dokumentation über das Interfaceverhalten übersehen werden, frühestens bei der Systemsimulation des weitgehend fertig entwickelten SoC entdeckt. Und selbst dies ist nicht sicher, denn eine RTL-Systemsimulation läuft so langsam, dass nur wenige Szenarien simuliert werden können. So können solche Kommunikationsfehler unter Umständen erst nach der Produktion der ersten Muster im Produkttest auffallen.

Bis die Systemsimulation durchgeführt werden kann, ist schon viel Aufwand in den neuen SoC investiert worden, und all dieser Aufwand wurde auf Basis der anfänglichen Auswahl der IP-Blöcke durchgeführt. Zu diesem Aufwand kann die Entwicklung der Kommunikationsinfrastruktur des SoC gehören, sofern diese nicht selbst ein IP-Block ist. Häufig fängt auch die Softwareentwicklung parallel mit der Hardwareentwicklung an. Änderungen an der ursprünglichen Auswahl der IP-Blöcke führen dann zu Zusatzaufwänden zur Anpassung der Kommunikationsinfrastruktur und zur Anpassung der bereits entwickelten Software.

Es ist also für den SoC-Entwurf sehr hilfreich, wenn schon während der Auswahl der IP-Blöcke verlässliche Aussagen über die Kompatibilität potentieller Kommunikationspartner gemacht werden können.

6.1.2 Formale Verifikation der Kommunikation von IP-Blöcken im SoC

Die hier vorgestellte Technik erlaubt früh belastbare Aussagen über die Kompatibilität des Interfaceverhaltens von IP-Blöcken. Diese Aussagen können zudem in Minutenschnelle gewonnen werden. Die Vorgehensweise ergibt sich sehr natürlich aus der Fragestellung, unter welchen Umständen die vollständige Verifikation von separaten Teilschaltungen eine vollständige Verifikation der Gesamtschaltung ist.

Voraussetzung ist eine Art Interfacebeschreibung der vollständigen Verifikation des IP-Blockes - durchaus im Sinne von [Broy/Rumpe 2007]. Diese besteht aus einer Integrationsannahme, in der die Annahmen über Verhalten und Determiniertheit der Eingabesignale zusammengefasst werden, und aus einer Integrationszusicherung, in der Aussagen über Verhalten und Determiniertheit der Ausgangssignale vereinigt werden. Integrationsannahme und –zusicherung werden kollektiv als Integrationsbedingungen bezeichnet. Sofern der IP-Block vollständig verifiziert ist, liegen die Integrationsbedingungen automatisch vor. Es lohnt sich aber auch für einen IP-Anbieter, die Integrationsbedingungen ohne eine formale Verifikation aufzustellen. Dies lässt dann zwar noch Raum für Diskrepanzen zwischen dem RTL-Code und den Integrationsbedingungen, ermöglicht aber die in diesem Kapitel beschriebene Kompatibilitätsprüfung.

Anhand dieser Integrationsbedingungen kann durch einige formale Tests effizient festgestellt werden, dass IP-Blöcke bezüglich ihres Kommunikationsverhaltens kompatibel sind. Es ist vorstellbar, dass diese Integrationsbedingungen für jeden IP-Block frei verfügbar gemacht werden. Auf diese Weise kann der hier vorgeschlagene Test durchgeführt werden, ohne dass ein potentieller Käufer der IP bereits Einblick in den RTL-Code nehmen muss.

6.1.3 Protokollbeschreibungen durch Integrationsbedingungen

Ein Beispiel für eine Integrationsbedingung wurde in Abschnitt 3.5.2 präsentiert. Grundform einer Integrationsbedingung ist die Syntax

```
integration_condition of <module_name> is
assume:
      <Determinierungsannahmen und Constraints>;
guarantee:
      <Determinierungsforderungen und Assertions>;
end integration_condition;
```

Mit den Integrationsbedingungen ist es zum ersten Mal im Kontext der formalen Verifikation möglich, den Hauptdaseinszweck von Protokollen überhaupt zu beschreiben, nämlich sicherzustellen, dass ein Modul, das ein Datum empfangen soll, auch in dem Moment zum Empfang bereit ist, an dem das sendende Modul das Datum abschickt.

ABFV hat keine Möglichkeit, solches auszudrücken, und beschränkt sich deshalb auf die Verhaltensaspekte sowohl der Kontroll- als auch der Datensignale. Wenn ein Datensignal in einer gewissen Situation unverändert bleiben soll, kann ABFV dies komfortabel ausdrücken und prüfen. Ob dieses Datensignal dann aber einen Wert trägt, der vom Empfänger ausgewertet werden soll, kann mit den Mitteln der ABFV nicht ausgedrückt werden, wohl aber mit dem Schlüsselwort "determined" der Integrationsbedingungen.

Besonders augenfällig werden die beschränkten Möglichkeiten der ABFV beim Lesedatensignal rdata des Prozessorinterfaces des SDRAM Interfaces. Dieses Signal unterliegt nämlich keinerlei Verhaltenseinschränkungen, wie sie in einer Assertion ausgedrückt werden könnten. Das Signal selbst darf sich beliebig verhalten, es trägt nur manchmal die gelesenen Daten, nämlich genau dann, wenn $ready\_o = '1'$ ist. In Assertions über das Protokoll des Prozessorinterfaces würde rdata entsprechend überhaupt nicht vorkommen, denn ABFV kann darüber nichts prüfen. In den Integrationsbedingungen hingegen wird die Determinierungszusicherung

```
if ready_o = '1' then determined(rdata) end if;
```

formuliert und während der Vollständigkeitsprüfung auch überprüft.

Bei der simulationsbasierten Verifikation identifizieren die Transaktionsextraktoren aus Abschnitt 3.1.2, wann ein Signal einen zu übertragenden Wert führt, und extrahieren daraus einen Parameter der entsprechenden Transaktion. Beim Vergleich dieser extrahierten Parameter mit den Werten, die ein transaktionsbasiertes Referenzmodell ausrechnet, kann festgestellt werden, dass verschaltete Module Daten und Adressen tatsächlich konsistent weitergeben und entsprechend ihrer Funktion verarbeiten. Auf diese Weise wird die Kommunikation aber eben erst während der Systemverifikation geprüft.

6.1.4 Ein einfacher kompositionaler Vollständigkeitsbeweis

Um ein Gefühl für die beabsichtigte Schlussweise bei kompositionaler vollständiger Verifikation zu vermitteln, sei angenommen, dass zwei Blöcke M_1 und M_2 hintereinandergeschaltet seien. M_1 sei durch eine Eigenschaftsmenge $\mathcal{P}_1$ vollständig verifiziert. Die vollständige Verifikation habe die Integrationsannahme IA_1 angenommen und die Integrationszusicherung IC_1 bewiesen. Entsprechendes gelte für M_2.

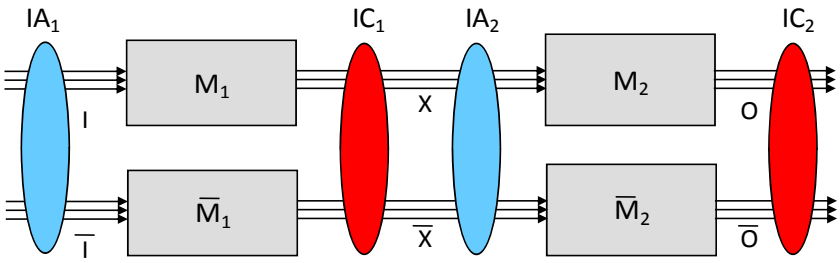

Abbildung 16: Zyklenfreier Clustergraph

Um die Gesamtschaltung vollständig zu verifizieren, muss offensichtlich nur sichergestellt werden, dass IA_2 erfüllt ist, wann immer IC_1 erfüllt ist. Wenn das der Fall ist, dann ist die Schaltung vollständig durch die Eigenschaftsmenge $\mathcal{P}_1 \cup \mathcal{P}_2$ bewiesen. Die Integrationsannahme der vollständigen Verifikation der Gesamtschaltung ist IA_1 und die Integrationszusicherung IC_2.

Zum Nachweis, dass die Gesamtschaltung durch $\mathcal{P}_1 \cup \mathcal{P}_2$ vollständig bewiesen ist, müssen entsprechend Abschnitt 5.2.3 zwei beliebige Schaltungen M und $\overline{M}$ verglichen werden, die beide $\mathcal{P}_1 \cup \mathcal{P}_2$ erfüllen. Unter der Integrationsannahme IA_1 beweist $\mathcal{P}_1$ die Integrationszusi-

cherung IC_1 und identifiziert so Schaltungsteile M_1 und $\bar{M}_1$ in M und $\bar{M}$, die durch $\mathcal{P}_1$ vollständig beschrieben sind und daher gemäß IA_1 und IC_1 gleiches Ein-/Ausgabeverhalten aufweisen. Weil IC_1 erfüllt ist, ist auch IA_2 erüllt und daher identifiziert auch $\mathcal{P}_2$ unter der Voraussetzung IA_2 Schaltungsteile M_2 und $\bar{M}_2$, deren vollständige Verifikation IC_2 rechtfertigt. Damit ist gezeigt, dass M und $\bar{M}$ gemäß Abbildung 16 aus M_1 und M_2 bzw. $\bar{M}_1$ und $\bar{M}_2$ und durch $\mathcal{P}_1 \cup \mathcal{P}_2$ funktional vollständig beschrieben sind.

6.1.5 Problem bei zyklischer Abhängigkeit der IP-Blöcke

Die zuvor geschilderte Argumentation kann allerdings nur so lange angewandt werden, wie es zwischen den IP-Blöcken keine zyklischen Abhängigkeiten gibt und das beschränkt ihre An-

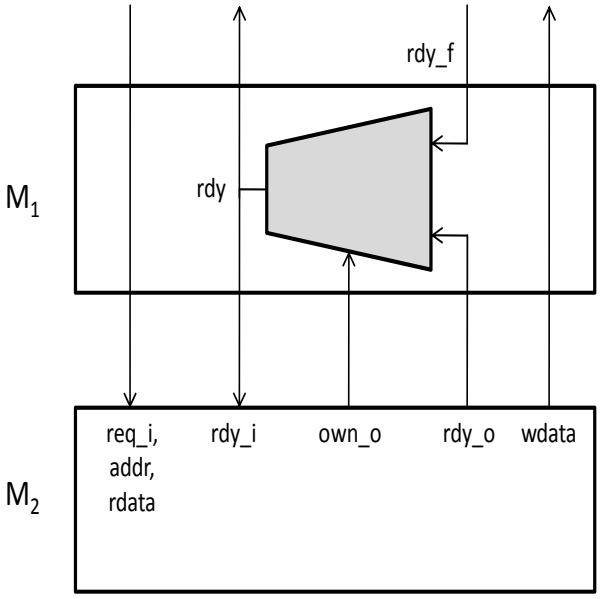

Abbildung 17: Problematische Vollständigkeitsprüfung

wendung auf künstliche Beispiele. Dass zyklische Abhängigkeiten zwischen den Modulen in der Tat zu Problemen führen, soll an dem folgenden Szenario demonstriert werden, das eine verfremdete, fokussierte und vereinfachte Version eines Kundenproblems [Beyer 2008] ist und in Abbildung 17 dargestellt wird.

Die zu verifizierende Schaltung bestand aus einem Bussystem M_1, von dem hier nur ein Multiplexer interessiert, und aus einem Peripheral M_2. Das Protokoll benötigt ein Synchronisationssignal rdy, das an alle Peripherals verteilt wird. Das gerade aktive Peripheral muss das Synchronisationssignal steuern, indem es mit dem Signal own_o den Multiplexer ansteuert und mit dem Signal rdy_o den Wert des Signals festlegt. Wenn $own\_o = 0$ ist, steuert das Peripheral M_2 das Synchronisationssignal nicht, sondern es wird durch ein anderes Peripheral festgelegt, was in dieser vereinfachten Version durch einen freien primären Eingang rdy_f von M_1 modelliert wird.
Zu M_1 wurde eine vollständige Verifikation durchgeführt mit den Integrationsbedingungen (für das Beispiel irrelevante Informationen wurden weggelassen)

```
integration_condition of bus_system is
```

```
assume:
      determined(own_o);
      if own_o = '1' then determined(rdy_o) end if;
      if own_o = '0' then determined(rdy_f) end if;

guarantee:
      if own_o = '1' then rdy_i = rdy_o end if;
      determined(rdy_i);

end integration_condition;
```

Das Peripheral wurde unter den Integrationsbedingungen

```
integration_condition of peripheral is

assume:
      if own_o = '1' then rdy_i = rdy_o end if;
      determined(rdy_i);

guarantee:
      determined(own_o);
      if own_o = '1' then determined(rdy_o) end if;

end integration_condition;
```

vollständig verifiziert. Dadurch sah es zunächst so aus, als wäre das Gesamtsystem für die Integrationsbedingung

```
integration_condition system is

assume:
      if own_o = '0' then determined(rdy_f) end if;

guarantee:
      determined(own_o);
      if own_o = '1' then determined(rdy_o) end if;

end integration_condition;
```

vollständig bewiesen.

Die Verifikation des Peripherals konnte jedoch als vollständig bewiesen werden, obwohl in allen Eigenschaften des Peripherals jegliches Beweisziel über rdy_o vergessen wurde. Die Eigenschaften verifizierten rdy_o also offensichtlich nicht und trotzdem zeigte der Vollständigkeitsprüfer keine Verifikationslücke an. Fehlverhalten von rdy_o hätte so nicht erkannt werden können.

Dieses zunächst unerwartete Verhalten des Vollständigkeitprüfers ergab sich durch die Determinierungsannahme über das Signal rdy_i in der Vollständigkeitsprüfung für M_2 und durch den Constraint

```
if own_o = '1' then rdy_i = rdy_o end if;
```

im Determinierungstest 5-8, wo der Constraint zu den Dependencies D und $\overline{D}$ gehört. Sofern eine Eigenschaft $own\_o = '0'$ vorsah, verlangt die Determinierungsforderung nichts über

das Signal rdy_o. Wenn die Eigenschaft $own\_o = '1'$ vorsah, war der Constraint anwendbar und sicherte $rdy\_o = rdy\_i$ und $\overline{rdy\_o} = \overline{rdy\_i}$ zu. Wegen der unbedingten Determinierungsannahme über rdy_i ist also $rdy\_i = \overline{rdy\_i}$ und daher konnte auch $rdy\_o = \overline{rdy\_o}$ bewiesen werden, ohne dass es dazu einen Beitrag aus den Eigenschaften erforderte.

Der Fehler ergab sich, weil aus den Integrationsbedingungen der Teilmodule die Integrationsbedingungen der Gesamtschaltung nur unter Verwendung eines Zirkelschlusses abgeleitet werden konnte. Der Zirkelschluss ergab sich dadurch, dass der Vollständigkeitstest über M_2 die bedingte Determinierungsforderung über rdy_o nur unter der Annahme verifizierte, dass rdy_i determiniert sei, und der Vollständigkeitstest über M_1 die Determinierung von rdy_i nur unter der Annahme der bedingten Determinierung von rdy_o bewies.
In diesem Kapitel soll beschrieben werden, unter welchen Bedingungen auch die vollständigen Eigenschaftssätze zyklisch verschalteter IP-Blöcke zu einer Eigenschaftsmenge vereinigt werden dürfen, die die Gesamtschaltung vollständig prüft.

6.2 Plausibilitätstests über Integrationsannahmen

Zu den Bedingungen, unter denen vollständige Eigenschaftssätze von Teilschaltungen zu einem vollständigen Eigenschaftssatz der Gesamtschaltung vereinigt werden dürfen, gehören Forderungen an die Gestalt der Integrationsannahmen. Diese Forderungen ergeben sich auf natürliche Weise aus dem Wunsch, dass es überhaupt Schaltungen geben möge, die sich einerseits so verhalten können, wie die Integrationsannahme vorsieht und gleichzeitig mit der untersuchten Schaltung keine kombinatorischen Schleifen bilden dürfen. Dass dies nicht selbstverständlich ist, zeigen die Beispiele aus Abschnitt 3.5.3.

Die Forderungen nach Implementierbarkeit der Integrationsannahme und nach struktureller Kompatibilität zwischen Integrationsannahme und Schaltung sind diesem Wunsch geschuldet. Auf der Basis einer Formalisierung der Integrationsannahmen in Abschnitt 6.2.1 werden die Kriterien in den Abschnitten 6.2.2 und 6.2.4 detailliert dargestellt.

Dass nicht immer leicht zu erkennen ist, ob Integrationsannahmen tatsächlich realistische Abbilder möglicher Schaltungsumgebungen sind, zeigt das Beispiel aus Abschnitt 6.1.5. Es wird im Verlauf der folgenden Diskussionen gezeigt werden, dass die Integrationsannahme des Peripherals nicht implementierbar ist.

Diese Bedingungen sind nicht nur dann wichtig, wenn vollständige Verifikationen zusammengefügt werden sollen. Sie sind auch als Plausibilitätskriterium über die Constraints und Determinierungsannahmen bei der vollständigen Verifikation eines einzelnen Clusters sehr hilfreich, weil sie Benutzer davor bewahren, Aufwand durch die Verwendung ungeeigneter Constraints oder Determinierungsannahmen zu vergeuden.

In ihrer Rolle als Plausibilitätskriterien stehen die hier entwickelten Prüfungen in Konkurrenz mit der Prüfung der Constraints auf Widerspruchsfreiheit, dem in der ABFV üblichen Mittel, Benutzer auf ungeeignete Constraints hinzuweisen, mit denen sie Verifikationsaufwand vergeuden würden. Ein Vergleich zwischen den Verfahren wird am Ende dieses Abschnitts gezogen.

6.2.1 Formalisierung von Integrationsbedingungen

Obwohl die Syntax der Integrationsbedingungen scheinbar nur über einen Variablensatz des Modells spricht, muss die Formalisierung der Integrationsbedingungen der Tatsache Rechnung tragen, dass diese Bedingungen vollständige Verifikationen charakterisieren. Bei diesen

vollständigen Verifikationen geht es um den Vergleich zweier beliebiger Schaltungen, die beide die entsprechenden Eigenschaften erfüllen sollen. Daher müssen die Integrationsannahmen zwei Traces I und $\bar{I}$ von Eingabesignalen beschränken und die Integrationszusicherungen Aussagen über zwei Traces O und $\bar{O}$ formulieren.

Mit dieser Vorrede definieren sich die Integrationsannahmen entsprechend der Notation aus Kapitel 5 durch

$$IA(I,U,O,\bar{I},\bar{U},\bar{O}) = \alpha(I,U,O) \wedge \alpha(\bar{I},\bar{U},\bar{O}) \wedge DA(I,U,O,\bar{I},\bar{U},\bar{O})$$

Dabei soll α alle Constraints umfassen, die bei der vollständigen Verifikation eines Moduls vorausgesetzt wurden, d.h. einerseits die Constraints aus den Dependencies D der Vollständigkeitsprüfung, aber andererseits auch die Constraints, unter denen die einzelnen Eigenschaften gegen den RTL-Code verifiziert wurden. Welcher Anteil einer Determinierungsannahme entsprechend der in Abschnitt 6.1.3 eingeführten Syntax dabei zum Constraint α und welcher zur Determinierungsbedingung DA gehört, wird anhand einer Zerlegung in Konjunkte und dann anhand des Schlüsselwortes determined erkannt.

Auf entsprechende Weise werden Integrationszusicherungen durch

$$IC(I,U,O,\bar{I},\bar{U},\bar{O}) = \zeta(I,U,O) \wedge \zeta(\bar{I},\bar{U},\bar{O}) \wedge DC(I,U,O,\bar{I},\bar{U},\bar{O})$$

formalisiert, wobei ζ die Konjunktion der Assertions über die Ausgabesignale bezeichnen soll.

Die Formalisierung der Integrationsannahmen erlaubt reaktive Formulierungen, doch sind gerade die reaktiven Formulierungen fehleranfällig und bedürfen deswegen der hier vorgestellten Plausibilitätstests.

Mit Ausnahme der Integrationsbedingungen treten alle an einem Vollständigkeitstest beteiligten Objekte paarweise auf. Dem soll eine Schreibvereinfachung Rechnung tragen, bei der Doppelbuchstaben jeweils auf das Paar zusammengehörender Objekte verweisen. Es sei also $II = (I,\bar{I})$, $OO = (O,\bar{O})$, $UU = (U,\bar{U})$, $MM(II,UU,OO) = M(I,U,O) \wedge \bar{M}(\bar{I},\bar{U},\bar{O})$. Leicht abweichend davon sei

$$PP(II,UU,OO) = \bigwedge_{P\in\mathcal{P}} \bigwedge_{t\geq 0} P(t,I,U,O) \wedge \bigwedge_{P\in\mathcal{P}} \bigwedge_{t\geq 0} P(t,\bar{I},\bar{U},\bar{O})$$

definiert und wird im Folgenden anstelle der Eigenschaftsmenge $\mathcal{P}$ verwendet. Das Ergebnis einer Vollständigkeitsprüfung läßt sich damit aus Formel 5-1 und unter Berücksichtigung des Beweises der Assertion ζ aus den Eigenschaften umschreiben zu

6-1

$$IA(II,UU,OO) \wedge PP(II,UU,OO) \Rightarrow IC(II,UU,OO)$$

6.2.2 Implementierbare Integrationsannahmen

Um nachzuweisen, dass eine Integrationsannahme implementierbar ist, muss eine Ersatzschaltung N^{IA}dieser Integrationsannahme gefunden werden, die als Zeuge dafür dient, dass die

Integrationsannahme zu vorgegebenen Traces UU und OO der internen und Ausgabesignale nur Eingabetraces II zuläßt, die auch von einer Umgebungsschaltung erzeugt werden können. Wie N^{IA} mit dem Paar MM der untersuchten Schaltungen verbunden werden soll, zeigt Abbildung 18.

N^{IA} erhält die Traces U, $\bar{U}$, O und $\bar{O}$ der internen und Ausgangssignale beider an der Vollständigkeitsprüfung beteiligten Schaltungen M und $\bar{M}$ als Eingaben. Ferner darf die Ersatzschaltung noch beliebig viele weitere Eingabesignale haben, denen der Trace F zugeordnet

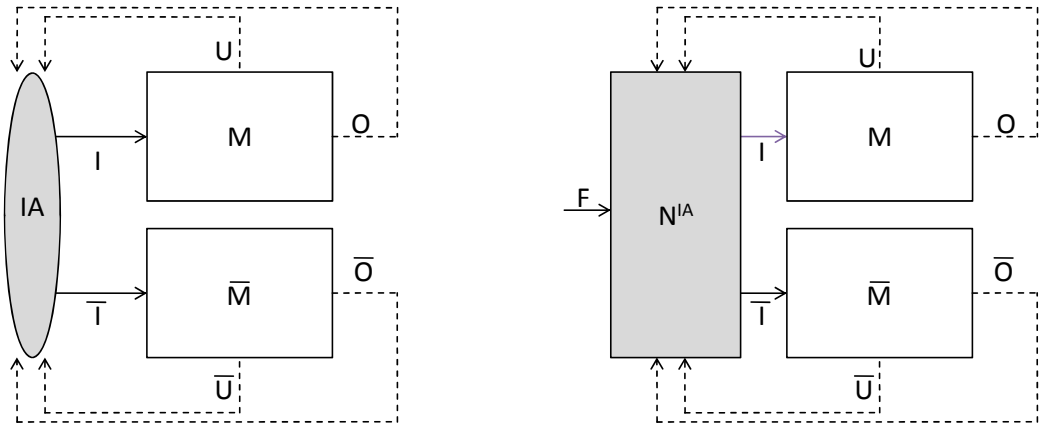

Abbildung 18: Implementierbare Constraints

sei. N^{IA} liefere an seinem Ausgang die Eingabetraces I und $\bar{I}$ für die Schaltungen M und $\bar{M}$. Wie für Schaltungen üblich, werden Ausgänge ganz rechts geschrieben, sodass sich als Parameterliste $N^{IA}((F, UU, OO), II)$ ergibt. Falls N^{IA} Flip-Flops enthält, seien diese zum Zeitpunkt 0 auf einen geeigneten Resetwert zurückgesetzt. Wenn N^{IA} separat betrachtet wird, und damit UU und OO freie Variablen sind, dann muss es für jeden Trace UU, OO und II, der $IA(II, UU, OO)$ erfüllt, einen Trace F geben, sodass $N^{IA}((F, UU, OO), II)$ gilt, d.h. dass N^{IA} den Trace II ausgibt, wenn F, UU und OO eingegeben werden[4]. Umgekehrt muss N^{IA} für jede Kombination von Traces UU, OO und F einen Trace II erzeugen, der $IA(II, UU, OO)$ erfüllt. Mögliche Kandidaten für Ersatzschaltungen ergeben sich aus [Schickel et al. 2006, Schickel et al. 2007].

Zu den Konsequenzen der Definition gehört, dass nicht reaktive Integrationsannahmen immer implementierbar sind.

Ferner werden die Traces UU und OO der internen und Ausgangssignale durch eine implementierbare Integrationsannahme nicht beschränkt. Das rechtfertigt die Sichtweise, dass die Integrationsannahme eigentlich nur eine Bedingung über die Eingangssignale ist, die gewissermaßen durch die Ausgangs- und internen Signale parametrisiert wird. Mit anderen Worten heißt dies, dass es zu jedem Trace UU und OO von internen und Ausgabesignalen immer einen Eingabetrace II gibt, der $IA(II, UU, OO)$ erfüllt. Dieser Eingabetrace muss taktweise in Abhängigkeit von Vergangenheit und Gegenwart bestimmt werden können, aber nicht von der Zukunft: $II^{(0)}$ wird aus der Resetsequenz, sowie $UU^{(0)}$ und $OO^{(0)}$ bestimmt, $II^{(1)}$ passend zu diesen drei Werten, sowie zu $UU^{(1)}$ und $OO^{(1)}$, $II^{(2)}$ passend zu $II^{(0)}$, $II^{(1)}$, $UU^{(0)}$, $UU^{(1)}$, $UU^{(2)}$, $OO^{(0)}$, $OO^{(1)}$ und $OO^{(2)}$ und so fort.

[4] Der Trace der internen Signale von N^{IA} interessiert nicht.

Bei der Untersuchung der Implementierbarkeit einer Integrationsannahme müssen im allgemeinen alle Zeilen in ihrem Zusammenwirken beachtet werden. So lässt sich beispielsweise leicht eine Schaltung finden, die zu den Eingabesignalen own_o, rdy_o und einem weiteren freien Eingang alle beliebigen Traces von rdy_i ausgibt, die

```
if own_o = '1' then rdy_i = rdy_o end if;
```

erfüllen. Genauso leicht lässt sich eine Schaltung angeben, die einen freien Eingang hat und alles Verhalten von rdy_i und $\overline{rdy\_i}$ erzeugt, das

```
determined(rdy_i);
```

erfüllt. Aber dennoch ist die Integrationsannahme aus Abschnitt 6.1.5, die aus diesen beiden Zeilen besteht

```
integration_condition of peripheral is

assume:
      if own_o = '1' then rdy_i = rdy_o end if;
      determined(rdy_i);

guarantee:
      …

end integration_condition;
```

ihrerseits nicht implementierbar. Denn die Integrationsannahme steht für die Bedingung

$$(own\_o = '1' \Rightarrow rdy\_i = rdy\_o) \wedge \left(\overline{own\_o} = '1' \Rightarrow \overline{rdy\_i} = \overline{rdy\_o}\right) \wedge \left(rdy\_i = \overline{rdy\_i}\right)$$

und diese beschränkt im Falle von $own\_o = \overline{own\_o} = '1'$ den Ausgabetrace auf $rdy\_o = \overline{rdy\_o}$ im Widerspruch dazu, dass implementierbare Integrationsannahmen die Ausgangssignale nicht einschränken dürfen. Entsprechend würde schon die Forderung nach Implementierbarkeit der Integrationsannahme das in Abschnitt 6.1.5 beschriebene Problem aufgedeckt haben.

6.2.3 Implementierbarkeitskriterien für Integrationsannahmen

Dieser Abschnitt stellt Kriterien für die Implementierbarkeit einer Integrationsannahme vor. Der Übersichtlichkeit halber wird der Trace UU interner Signale im Folgenden nicht mehr geschrieben. Er kann als Teil des Traces OO angesehen werden.

Zur Prüfung der Implementierbarkeit einer Integrationsannahme muss nachgewiesen werden, dass zu jedem Trace OO von Ausgabe- und internen Signalen ein Eingabetrace II existiert, der $IA(II, OO)$ erfüllt. Dieser Existenznachweis lässt sich mit SAT-Beweisern nur unter zusätzlichen Bedingungen an die Struktur von IA führen. Diese ergeben sich aus der Formulierung der Integrationsannahmen in ITL, weil dort alle Integrationsannahmen so beschaffen sind, dass daraus eine Bedingung $\widetilde{IA}\left(II^{(t)}, II^{[0,t-1]}, OO^{[0,t]}\right)$ abgelesen werden kann, die vergangenen Werten der Eingabetraces und vergangenen oder aktuellen Werten der internen und Ausgabetraces den aktuellen Wert des Eingabetraces zuordnet. Die eigentliche Integrationsannahme ergibt sich aus $\widetilde{IA}$ durch

6-2

$$IA(II, OO) = \bigwedge_{t \geq 0} \widetilde{IA}\left(II^{(t)}, II^{[0,t-1]}, OO^{[0,t]}\right)$$

ist. Dabei bezeichnet $II^{[n,m]}$ die Liste der Werte $II^{(n)}$, $II^{(n+1)}$, $II^{(n+2)}$ bis $II^{(m)}$.

Dieses $\widetilde{IA}$ gehört zu einer implementierbaren Integrationsbedingung, wenn jedes Element von II taktweise bestimmt werden kann, wenn es also zu jedem Zeitpunkt t einen Wert ii gibt, mit dem $\widetilde{IA}\left(ii, II^{[0,t-1]}, OO^{[0,t]}\right)$ erfüllt werden kann. Aber auch diese Forderung läßt sich noch nicht mit einfachen Mitteln prüfen. Abhilfe verspricht hier die weitere Aufteilung von $\widetilde{IA}$ in Teilausdrücke $\widetilde{IA}_l$, die jeweils für die Festlegung von nur einigen wenigen Eingangsvariablen zuständig sind, deren Trace mit II_l bezeichnet wird, wobei die $\widetilde{IA}_l$ nur von solchen aktuellen Werten von Eingabesignalen abhängen dürfen, die von einem $\widetilde{IA}_\nu$ mit kleinerem Index festgelegt wurden. Es sei also $II = (II_0, II_1, II_2, \ldots II_n)$ und

6-3

$$\widetilde{IA}\left(II^{(t)}, II^{[0,t-1]}, OO^{[0,t]}\right) = \bigwedge_{l=0}^{n} \widetilde{IA}_l\left(II_l^{(t)}, II_{[0,l-1]}^{(t)}, II^{[0,t-1]}, OO^{[0,t]}\right)$$

Dabei bezeichnet $II_{[0,l-1]}^{(t)}$ die Liste $II_0^{(t)}, II_1^{(t)}, II_2^{(t)}, \ldots II_{l-1}^{(t)}$.

Auf der Basis dieser Zerlegung lässt sich ein erstes Implementierbarkeitskriterium angeben. Dazu sei eine Konstante m defniert, die z.B. die größte Verzögerung bezeichnet, mit der $\widetilde{IA}$ noch Signale auswertet. Bei der in Abschnitt 6.1.3 beispielhaft dargestellten Syntax der Integrationsannahmen entspräche dies der maximalen Schachtelungstiefe der prev-Operatoren. Das Kriterium muss für alle $l \in [0, n]$ geprüft werden.

Es besteht aus einem Basisfall für $t \in [0, m-1]$:

6-4

$$\left(\varrho\varrho(II) \wedge \bigwedge_{k=0}^{t-1} \widetilde{IA}\left(II^{(k)}, II^{[0,k-1]}, OO^{[0,k]}\right) \wedge \bigwedge_{k=0}^{l-1} \widetilde{IA}_k\left(II_k^{(t)}, II_{[0,k-1]}^{(t)}, II^{[0,t-1]}, OO^{[0,t]}\right)\right) \Rightarrow \bigvee_{ii_l} \widetilde{IA}_l\left(ii_l, II_{[0,l-1]}^{(t)}, II^{[0,t-1]}, OO^{[0,t]}\right)$$

und einem Fall für $t \geq m$:

6-5

$$\left(\bigwedge_{k=t-m}^{t-1} \widetilde{IA}\left(II^{(k)}, II^{[0,k-1]}, OO^{[0,k]}\right) \wedge \bigwedge_{k=0}^{l-1} \widetilde{IA}_k\left(II_k^{(t)}, II_{[0,k-1]}^{(t)}, II^{[0,t-1]}, OO^{[0,t]}\right)\right) \Rightarrow \bigvee_{ii_l} \widetilde{IA}_l\left(ii_l, II_{[0,l-1]}^{(t)}, II^{[0,t-1]}, OO^{[0,t]}\right)$$

Diese Formeln können mit SAT bewiesen werden, wenn die Signale von II_l nur aus wenigen Bits bestehen, weil dann die Disjunktion auf der rechten Seite der Implikation überschaubar viele Terme enthält. Für breitere Variablen wird nachfolgend ein alternatives Implementierbarkeitskriterium angegeben.

Zur Prüfung der Bedingung 6-5 werden die Variablen relativ zu t angelegt, sodass es einen Variablensatz für $OO^{(t)}$ und $II^{(t)}$ gibt, einen anderen für $OO^{(t-1)}$ und $II^{(t-1)}$ usf. bis zum letzten Variablensatz, der für $OO^{(t-2m)}$ und $II^{(t-2m)}$ steht. Frühere Elemente von II und OO werden in Bedingung 6-5 nicht ausgewertet.

Im Anhang wird nachgewiesen, dass dieses Kriterium für Integrationsannahmen in ITL tatsächlich die Existenz einer Ersatzschaltung sicherstellt.

In der Praxis haben viele Integrationsannahmen eine Struktur, aus der sich eine einfachere Prüfung auf die Implementierbarkeit einer Integrationsannahme ergibt, die insbesondere auch dann anwendbar ist, wenn das Auffalten von $\bigvee_{ii_l} \widetilde{IA}_l \left(ii_l, II^{(t)}_{[0,l-1]}, II^{[0,t-1]}, OO^{[0,t]}\right)$ aus Bedingung 6-4 oder 6-5 zu einer Disjunktion zu groß würde. Die Anwendung des Kriteriums nach 6-4 und 6-5 kann mit dem alternativen Kriterium vermischt werden.

Diese vereinfachte Prüfung ist auf solche $\widetilde{IA}_l$ anwendbar, die aus der Konjunktion eines Constraints der Form

$$\texttt{if } c_l\left(I^{(t)}_{[0,l-1]}, I^{[0,t-1]}, O^{[0,t]}\right) \texttt{ then } I^{(t)}_l = F_l\left(I^{(t)}_{[0,l-1]}, I^{[0,t-1]}, O^{[0,t]}\right) \texttt{ else } R_l\left(V^{(t)}_l\right) \texttt{ end if;}$$

und einer Determinierungsannahme der Form

$$\texttt{if } g_l\left(I^{(t)}_{[0,l-1]}, I^{[0,t-1]}, O^{[0,t]}\right) \texttt{ then determined}\left(I^{(t)}_l\right) \texttt{ end if;}$$

bestehen. Dabei steht I wie immer für den Trace der Eingabesignale des Moduls und O für den Trace der internen und Ausgabesignale. R_l, c_l und g_l beschreiben Bedingungen, F_l einen Ausdruck, dessen Ergebnistyp mit dem Typ von I_l übereinstimmt, sodass die Gleichung im then-Zweig des Constraints immer erfüllt werden kann. Von welchen Signalwerten die Bedingungen und der Ausdruck abhängen dürfen, ist durch die Parameterlisten charakterisiert.

Für die nachfolgenden Betrachtungen werden Indices und Parameterlisten von g, c und F unterdrückt, es wird lediglich $\bar{g}$, $\bar{c}$ und $\bar{F}$ geschrieben, wenn die Bedingungen auf $\bar{I}$ und $\bar{O}$ angewandt werden. Dann besteht die Prüfung auf Implementierbarkeit wiederum aus einem Basisfall für $t \in [0, m-1]$:

6-6

$$\left(\varrho\varrho(II) \wedge \bigwedge_{k=0}^{t-1} \widetilde{IA}\left(II^{(k)}, II^{[0,k-1]}, OO^{[0,k]}\right) \wedge \bigwedge_{k=0}^{l-1} \widetilde{IA}_k\left(II^{(t)}_k, II^{(t)}_{[0,k-1]}, II^{[0,t-1]}, OO^{[0,t]}\right)\right) \Rightarrow \left(R(F) \wedge ((g \vee \bar{g}) \wedge c \wedge \bar{c} \Rightarrow F = \bar{F})\right)$$

und einem Fall für $t \geq m$:

6-7

$$\left(\bigwedge_{k=t-m}^{t-1} \widetilde{IA}\left(II^{(k)}, II^{[0,k-1]}, OO^{[0,k]}\right) \wedge \bigwedge_{k=0}^{l-1} \widetilde{IA}_k\left(II_k^{(t)}, II_{[0,k-1]}^{(t)}, II^{[0,t-1]}, OO^{[0,t]}\right)\right)$$
$$\Rightarrow \left(R(F) \wedge \left((g \vee \bar{g}) \wedge c \wedge \bar{c} \Rightarrow F = \bar{F}\right)\right)$$

Diese Tests lassen sich mit SAT-Algorithmen prüfen, ohne dass die Formeln wie zuvor durch das Auffalten der Existenzbedingung anschwellen. Die zuvor gemachten Bemerkungen zur Wahl der Variablen gelten auch für Gleichung 6-7. Auch diese Implementierbarkeitsbedingung wird im Anhang bewiesen.

Als Beispiel soll das zweite Kriterium auf die ursprüngliche Integrationsannahme des Peripherals des Beispiels aus Abschnitt 6.1.5 angewandt werden. Diese lautete

```
integration_condition of peripheral is

assume:
      if own_o = '1' then rdy_i = rdy_o end if;
      determined(rdy_i);

guarantee:
      …

end integration_condition;
```

Es ist also rdy_i die Variable, die von der Integrationsannahme festgelegt wird, $R = 1$, $g = 1$, $c = (own\_o = 1)$ und $f = rdy\_o$. Die Prüfung von $R(F)$ ist trivial, aber $(g \vee \bar{g}) \wedge c \wedge \bar{c} \Rightarrow F = \bar{F}$ wird zu

$$(own\_o = 1 \ \wedge \overline{own\_o} = 1) \Rightarrow rdy\_o = \overline{rdy\_o}$$

und diese Bedingung lässt sich auf freien Variablen nicht beweisen. Dies ist ein Hinweis dafür gewesen, dass die ursprüngliche Integrationsannahme aus Abschnitt 6.1.5 nicht implementierbar ist und so die vollständige Verifikation nicht geeignet ist, mit anderen vollständigen Verifikationen zusammengesetzt zu werden.

Mit der verbesserten Integrationsannahme

```
integration_condition of peripheral is

assume:
      if own_o = '1' then rdy_i = rdy_o end if;
      if own_o = '0' then determined(rdy_i) end if;

guarantee:
      …

end integration_condition;
```

ergibt sich als Hauptteil der Tests 6-6 und 6-7

$$\left((own\_o = 0 \vee \ \overline{own\_o} = 0) \wedge own\_o = 1 \ \wedge \overline{own\_o} = 1\right) \Rightarrow rdy\_o = \overline{rdy\_o}$$

Dieser Test ist erfüllt, weil die linke Seite der Implikation immer verletzt ist. Daher ist die verbesserte Determinierungsannahme für die Verifikation des Beispiels in Abschnitt 6.1.5 geeignet.

6.2.4 Strukturelle Kompatibilität

Die zweite Forderung an die Integrationsannahme verlangt strukturelle Kompatibilität mit der Schaltung. Zur Prüfung der strukturellen Kompatibilität wird zunächst der Graph K^M der von den Registerausgängen und Eingangssignalen ausgehenden kombinatorischen Abhängigkeiten einer Schaltung M folgendermaßen bestimmt: Zunächst werden durch eine syntaktische Analyse Kandidaten $s \rightarrow u$ von potentiellen kombinatorischen Abhängigkeiten zwischen einem Registerausgang oder Eingangssignal s und einem Ausgangs- oder internen Signal u bestimmt. Diese werden dann auf einem durch zwei Instanzen M und $\overline{M}$ der zu untersuchenden Schaltung gebildeten Modell daraufhin untersucht, ob die Eigenschaft

```
property check_dependency is
assume:
      at t:       <v = v̄> für alle Eingangssignale und
                  Zustandssignale v mit Ausnahme von s;
prove:
      at t:       u = ū;
end property;
```

fehlschlägt. Wenn sie fehlschlägt, gehört die kombinatorische Abhängigkeit $s \rightarrow u$ zu K^M.

Ferner wird der Graph K^{IA} der kombinatorischen Abhängigkeiten in einer Ersatzschaltung von IA folgendermaßen abgeschätzt: Wenn im ITL-Text von $\widetilde{IA}_l\left(II_l^{(t)}, II_{[0,l-1]}^{(t)}, II^{[0,t-1]}, OO^{[0,t]}\right)$ ein Ausgangs- oder internes Signal o zum Zeitpunkt t referenziert wird, wird für jedes Signal i, das durch $II_l^{(t)}$ beschrieben wird, die kombinatorische Abhängigkeit $o \rightarrow i$ zu K^{IA} hinzugefügt. Ferner wird auch für jedes Signal $\hat{\imath}$, das durch den Trace $II_{[0,l-1]}$ beschrieben wird, und das in $\widetilde{IA}_l\left(II_l^{(t)}, II_{[0,l-1]}^{(t)}, II^{[0,t-1]}, OO^{[0,t]}\right)$ ebenfalls zum Zeitpunkt t ausgewertet wird, und für jedes Signal i, das durch $II_l^{(t)}$ beschrieben wird, die kombinatorische Abhängigkeit $\hat{\imath} \rightarrow i$ zu K^{IA} hinzugefügt. K^{IA} besteht letztendlich aus allen kombinatorischen Abhängigkeiten, die sich aufgrund dieser Überlegungen für eines der $\widetilde{IA}_l$ ergeben. In K^{IA} kann es aufgrund der Abhängigkeiten $\hat{\imath} \rightarrow i$ zwischen Eingangssignalen durchaus längere Pfade aus mehreren kombinatorischen Abhängigkeiten geben und damit mehr kombinatorische Abhängigkeiten von Ausgabe- zu Eingabesignalen, als nur die aufgesammelten oben dargestellten $o \rightarrow i$.

Die kombinatorischen Abhängigkeiten in K^{IA} spiegeln die Situation der Ersatzschaltung wider, die im Anhang zum Nachweis der Implementierbarkeit von IA angegeben ist. Dabei ist es ausreichend, wenn K^{IA} nur Abhängigkeiten zwischen Signalen einer der beiden Instanzen des Modellpaares MM enthält. Dies ist aus Symmetriegründen genug, denn mit jeder kombinatorischen Abhängigkeit $\bar{a} \rightarrow \bar{b}$ ist auch $a \rightarrow b$ eine kombinatorische Abhängigkeit in der Ersatzschaltung. Ferner ist jede gemischte Abhängigkeit in der Ersatzschaltung von der Form $\bar{a} \rightarrow b$ oder $a \rightarrow \bar{b}$ die Folge einer Determinierungsannahme, die dann auch zur Abhängigkeit $a \rightarrow b$ führt.

Daher ist es ausreichend, bei der Prüfung der strukturellen Kompatibilität zwischen einer Schaltung und seiner Integrationsannahme nur den einfachen Variablensatz in K^M und K^{IA} zu

untersuchen. Für die Untersuchung wird der Graph mit der Kantenmenge $K = K^{IA} \cup K^{M}$ auf Zyklen untersucht. Wenn er zyklenfrei ist, sind IA und M strukturell kompatibel.

6.2.5 Plausibilitätskriterien über Constraints und Determinierungsannahmen in ABFV und vollständiger Verifikation

Dieser Abschnitt stellt zwei Forderungen an Integrationsbedingungen auf, die erfüllt sein müssen, damit die Integrationsbedingungen zur untersuchten Schaltung passen. Die eine Bedingung fordert die Implementierbarkeit der Integrationsbedingung, die andere fordert struktureller Kompatibilität zwischen der Integrationsbedingung und der Schaltung. Die Implementierbarkeit der Integrationsbedingung lässt sich unabhängig von der Schaltung prüfen, die Strukturbedingung erfordert die Kenntnis der Schaltung, allerdings in einem recht geringen Rahmen.

Plausibilitätsbedingungen über die Voraussetzungen der Verifikation spielen auch bei der ABFV eine Rolle. Dort sind diese Voraussetzungen nur durch die Menge der Constraints gegeben, unter denen verifiziert wird. Determinierungsannahmen gehören nicht zum Arsenal der ABFV.

Bei der ABFV sind die Plausibilitätsbedingungen als Hilfestellung dagegen gedacht, dass ungeeignete Constraints eine Verifikation entwerten. Durch eigentlich kleine Fehler im Constraint werden weniger Eingaben zugelassen als der Benutzer eigentlich will. Dadurch vereinfacht sich die Verifikation: Die Assertions lassen sich leichter beweisen, und meist wiegt sich der Benutzer im falschen Glück. Denn die Entwertung der Verifikation fällt nicht auf, weil Constraints die einzigen Objekte bei einer Verifikation sind, die nicht hinterfragt werden. Um dem zu begegnen, sind Plausibilitätstests über die Constraints entwickelt worden. Diese Plausibilitätstests prüfen, ob die Constraints in sich selbst widersprüchlich sind. Diese Prüfung besteht in dem Beweis einer Implikation der Form

$$\prod Constraints \Rightarrow false$$

Wenn sich dieser Beweis führen lässt, sind die Constraints in sich widersprüchlich und daher ungeeignet. Wenn der Beweis zu Gegenbeispielen führt, werden die Constraints akzeptiert.

Dabei werden die Variablen der Constraints zunächst als freie Variablen betrachtet. In diesem Fall werden Widersprüche entdeckt, die sich ausschließlich aus dem Zusammenspiel der Constraints ergeben. In einem zweiten Durchgang wird dem Plausibilitätstest die zu untersuchende Schaltung unterlegt. Wenn dann die Implikation bewiesen wird, ist ein Widerspruch zwischen den Constraints und der Schaltung identifiziert worden.

Wenn ein Plausibilitätstest fehlschlägt, wünschen Benutzer Diagnoseunterstützung. Diese ist bei dem in der ABFV üblichen Plausibilitätstest nicht leicht zu generieren, weil sich das Problem ja gerade aus dem Zusammenwirken von Einzelbedingungen ergibt, die jede für sich genommen meist harmlos sind.

Die Prüfung der Constraints auf Widerspruchsfreiheit ist aber nur ein grober Test, der nur besonders krasse Fälle schlechter Constraints identifiziert. Dieser Test gibt sich schon zufrieden, wenn die Constraints im Verein mit der Schaltung auch nur einen einzigen Eingabetrace zulassen. Selbst wenn ein reaktiver Constraint das Ausgabeverhalten einer Schaltung versehentlich sehr stark einschränkt, wird dies von diesem groben Test meist nicht identifiziert.

Die hier vorgestellten Tests auf Implementierbarkeit und strukturelle Kompatibilität sind hingegen viel kritischer. Ihre Formulierung hier für Integrationsannahmen ist dem Fokus dieser Arbeit geschuldet. Sie lassen sich aber ohne weiteres auch an die Bedürfnisse von ABFV anpassen, dann müssen eben allein die Constraints implementierbar sein und mit einer Instanz der Schaltung strukturell kompatibel sein.

Damit werden zumindest alle Situationen identifiziert, in denen es auch nur einen Trace der Ausgabesignale gibt, zu dem der Constraint keinen Eingabetrace zulässt. Ein solche Ausgabetrace kann vorteilhaft als Gegenbeispiel ausgegeben werden.

Damit sind Constraints, die die hier vorgeschlagenen Tests erfüllen, bereits frei von Widersprüchen in sich. Der vorgeschlagene Test identifiziert also alle fehlerhaften Constraints, die auch die übliche Widerspruchsuntersuchung auf freien Variablen zurückweisen würde.

Aber es kommt noch besser. Durch den strukturellen Abgleich wird nämlich auch sichergestellt, dass sich zu jedem Zeitpunkt ein Eingabewert finden lässt, der den Constraint erfüllt und in die Schaltung eingegeben werden kann. Folglich ist sogar sichergestellt, dass Constraints, die die hier vorgeschlagenen Tests erfüllen, auch unter Berücksichtigung der Schaltung widerspruchsfrei sind. Bei den hier vorgeschlagenen neuen Plausibilitätstests wird die Schaltung selbst zwar nur im Hinblick auf ihre kombinatorischen Pfade untersucht. Aber diese Untersuchung ist ausreichend, um jeden Constraint als ungeeignet zurückzuweisen, der von der üblichen Untersuchung unter Berücksichtigung der Schaltungsfunktion auch zurückgewiesen würde.

Alles in allem sind die hier vorgestellten Tests attraktive Alternativen zu heute durchgeführten Plausibilitätstests auf Constraints. Über die detailliertere Untersuchung hinaus erlauben sie auch wesentlich detailliertere Diagnoseverfahren, weil einerseits die ITL-Zeilen farblich markiert werden können, bei denen ein Problem auftritt, und andererseits auch ein Ausgabetrace berechnet werden kann, bei dem sich das Problem ergibt. Wenn sogar die detaillierteren Tests im zweiten Teil von Abschnitt 6.2.3 durchgeführt werden, ergeben sich noch wesentlich genauere Diagnosemöglichkeiten.

6.3 Zusammenfügen vollständiger Verifikationen

6.3.1 Kompositionale vollständige Verifikation und Assume-Guarantee-Reasoning

Die Verschaltung von Komponenten wird in vielen Ansätze zur Modellierung digitaler Funktionalität behandelt, wie etwa die ASMs [Börger/Stärk 2003], TLT [Cuellar/Huber 1995], TLA[Lamport 1994], oder UML [Broy/Rumpe 2007]. Entsprechende Arbeiten werden unter dem Schlagwort Assume-Guarantee-Reasoning zusammengefasst [Abadi/Lamport 1995]. Sie haben auch Konsequenzen für reaktive Constraints über die Umgebung des Modells [Broy 1994].

Assume-Guarantee-Reasoning geht üblicherweise von Moore-Automaten aus, d.h. Automaten ohne direkte Abhängigkeit zwischen Ausgängen und Eingängen. Um dann zu begründen, dass auch der Verbund von richtig funktionierenden Moore-Automaten M_1 und M_2 ebenfalls immer richtig funktioniert, wird angenommen, dass er irgendwann falsch funktioniere. Dann muss es einen ersten Zeitpunkt geben, zu dem die Fehlfunktion in einem der beiden Komponenten auftritt. Die Fehlfunktion kann aber nur auftreten, wenn sich vorher die andere Komponente nicht mehr richtig verhalten hat, und das steht dann im Gegensatz zu der Annahme

darüber, dass der früheste Zeitpunkt der Fehlfunktion betrachtet worden war. Also funktioniert der Verbund aus M_1 und M_2 korrekt.

Im Unterschied dazu betrachtet die vollständige Verifikation hauptsächlich Operationsautomaten, für die es geradezu ein Hauptmerkmal ist, dass die Ausgangssignale während einer Operation durch das Eingabeverhalten gesteuert werden. Die Operations- und Transaktionsautomaten sind also gerade keine Moore-Automaten, sodass auf sie die übliche Argumentation des Assume-Guarantee-Reasonings nicht direkt anwendbar ist. Trotzdem kommt die nachfolgend vorgestellte Theorie ohne Einschränkungen über die Gestalt der Operations- und Transaktionsautomaten aus, weil die Problematik auf die der Verifikation unterliegenden Schaltungen zurückgespielt werden konnte. Deren übliche synchrone digitale Automatendarstellungen sind zwar auch noch keine Moore-Automaten, denn das würde kombinatorische Abhängigkeiten zwischen Ein- und Ausgangssignalen verbieten. Aber unter Ausnutzung der Forderung, dass die Schaltungen keine kombinatorischen Schleifen enthalten dürfen, lässt sich ein auf Kausalität beruhendes verfeinertes Zeitmodell definieren, auf das dann endlich die übliche Assume-Guarantee-Argumentation angewandt werden kann. Diese kann ihrerseits wieder auf die Operationsautomaten übertragen werden, weil diese gegen die RTL-Beschreibung bewiesen wurden.

Überhaupt ergeben sich Unterschiede zu den oben erwähnten digitalen Modellierungsverfahren, weil hier Schaltungen verifiziert werden sollen. Dafür muss natürlich auch die Funktionalität modelliert werden, wofür bei den Transaktionsautomaten eine Zeitabstraktion gewählt wurde, die von der transaktionsbasierten Simulation inspiriert ist, durch die mehrere Takte zu einem abstrakten Zeitschritt zusammengefasst werden. Über die Modellierung hinausgehend gibt es aber Integrationsbedingungen, die nach der Kategorisierung bei [Broy/Rumpe 2007] als Interface-Beschreibungen anzusehen sind, und eben mit den Integrationsbedingungen auch Auskunft über den Verifikationscode geben.

Eine weitere Eigenart der hier vorgestellten Theorie ist es, dass sie angewendet werden kann, wenn eine Verifikationsaufgabe in die Verifikation mehrerer Cluster zerlegt werden soll, und wenn diese Cluster auch noch zyklisch untereinander verbunden sind. Dabei wird nicht gefordert, dass das Modell in die einzelnen Cluster zerfallen muss, wie es etwa [Abadi/Lamport 1995] nahelegt. Statt dessen müssen nur einzelne Signale geschnitten werden, d.h. durch primäre Ein- und Ausgänge ersetzt werden. Dabei soll ein zyklenfreier Abhängigkeitsgraph der Cluster entstehen. Die Cluster dürfen aber sozusagen in der Gegenrichtung verbunden bleiben. Dies vereinfacht die Verifikation teilweise erheblich, wie Abschnitt 3.5.7 verdeutlicht.

6.3.2 Voraussetzungen für das Zusammensetzen vollständiger Verifikationen

Ausgangspunkt der Untersuchungen ist wie in Abbildung 19 eine Schaltung M, die aus Blöcken M_j zusammengesetzt ist, wobei jeder Block für sich vollständig verifiziert ist mit lokalen Integrationszusicherungen IC_j und lokalen Integrationsannahmen IA_j.

M habe eine eigene globale Integrationszusicherung GC, sowie eine eigene implementierbare globale Integrationsannahme GA. M und GA seien auch strukturell kompatibel entsprechend Abschnitt 6.2.4.

Dieses Kapitel soll Auskunft darüber geben, unter welchen Bedingungen M durch die Vereinigungsmenge der Eigenschaften der M_j vollständig verifiziert ist mit der globalen Integrationsannahme GA und der globalen Integrationszusicherung GC.

Die Eingangssignale der M_j seien aufgeteilt in primäre Eingangssignale der Gesamtschaltung, die den Trace I bilden und Eingangssignale, die von Ausgabesignalen anderer Blöcke versorgt werden. Diese Verbindungsstruktur sei durch Signale gegeben, die im Trace X zusammengefasst werden, und von denen nur eine Auswahl zum Block M_j führt. Falls ein Block mehrere Eingänge anderer Blöcke treibt, gibt es in X entsprechend mehrere Signale und die Integrationszusicherung des treibenden Blocks wird durch eine Bedingung ergänzt, die verlangt, dass die Signale gleich sind.

Die Ausgabesignale der M_j seien entsprechend aufgeteilt in primäre Ausgabesignale, die zusammen den Trace O bilden, und in eine Auswahl der Verbindungsstruktursignale von X. Solange die Schaltung entlang der in Abbildung 19 angedeuteten Linie zerschnitten ist, wird jedes M_j separat betrachtet, und die Verbindungsstruktursignale bilden einen Trace $X°$ von Ausgangssignalen und einen Trace X' von Eingangssignalen.

Durch die Verschaltung werden die lokalen Ein- und Ausgabesignale von $X°$ und X' zu internen Signalen, die den Trace X bilden. Die Gesamtfunktion von M ergibt sich dann durch

$$M(I,U,X,O) = \bigwedge_j M_j(I,X,U,O,X)$$

Dabei soll U der Trace aus allen internen Signalen der M_j sein.

Auf die oben beschriebenen Weise sind also die Signale in den Eigenschaftsmengen $\mathcal{P}_j$ der M_j bereits so umbenannt, dass sie auch die Verbindungsstruktur repräsentieren. Die M_j seien ent-

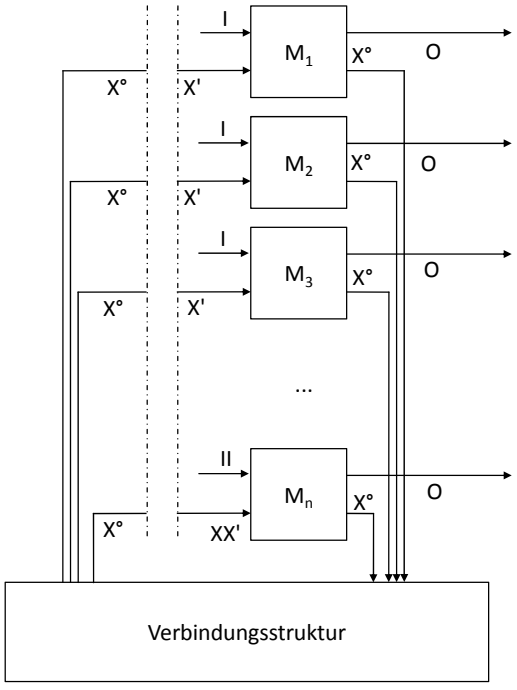

Abbildung 19: Zusammensetzen vollständiger Verifikationen

sprechend der Integrationsannahmen IA_j und der Integrationszusicherungen IC_j vollständig verifiziert.

Damit die Gesamtschaltung vollständig durch $\bigcup_j \mathcal{P}_j$ verifiziert ist unter der Integrationsannahme $GA(II, UU, XX, OO)$ und mit der Integrationszusicherung $GC(II, UU, XX, OO)$ müssen folgende Voraussetzungen eingehalten werden:

Voraussetzung 1 verlangt, dass der Gesamtgraph aller kombinatorischen Abhängigkeiten der Gesamtschaltung und der lokalen sowie der globalen Integrationsannahmen $K^{GA} \cup K^{M} \cup \bigcup K^{IA_j}$ frei ist von kombinatorischen Schleifen.

Voraussetzung 2 verlangt, dass die globale Integrationsannahme $GA(\boldsymbol{II}, UU, XX, OO)$ implementierbar ist und II beschränkt. Außerdem müssen die $IA_j(\boldsymbol{II}, \boldsymbol{XX'}, UU, OO, XX°)$ implementierbare Integrationsannahmen sein, die II und XX' beschränken. Der Fettdruck bei implementierbaren Integrationsannahmen soll die Traces herausheben, die von der Integrationsannahme eingeschränkt werden, die also Ausgangssignale der Ersatzschaltungen sein dürfen.

Voraussetzung 3 verlangt, dass die globale Integrationsannahme $GA(\boldsymbol{II}, UU, XX, OO)$ auf freien Variablen nur solche Traces II zulassen darf, die auch von den lokalen Integrationsannahmen IA_j zugelassen werden, d.h.

$$GA(\boldsymbol{II}, UU, XX, OO) \Rightarrow \bigvee_{XX'} \bigwedge_{j} IA_j(\boldsymbol{II}, \boldsymbol{XX'}, UU, OO, XX°)$$

Voraussetzung 4 verlangt, dass auf freien Variablen die lokalen Integrationsannahmen durch die lokalen Integrationszusicherungen und die globale Integrationsannahme durch

$$GA(\boldsymbol{II}, UU, XX°, OO) \wedge GA(\boldsymbol{JJ}, UU, XX', OO) \wedge \bigwedge_{j} IC_j(II, XX', UU, OO, XX°)$$
$$\Rightarrow \bigwedge_{j} IA_j(\boldsymbol{JJ}, \boldsymbol{XX}°, UU, OO\ XX')$$

gerechtfertigt werde, d.h. dass die Integrationszusicherungen die Integrationsannahmen implizieren müssen. Dabei bezeichnet JJ einen Trace von Kopien der Variablen aus II.

Voraussetzung 5 verlangt, dass die globale Integrationszusicherung eine Folge der lokalen Integrationszusicherungen und der globalen Integrationsannahme sein muss, dass also wiederum auf freien Variablen

$$GA(\boldsymbol{II}, UU, XX, OO) \wedge \bigwedge_{j} IC_j(II, XX, UU, OO, XX) \Rightarrow GC(II, UU, XX, OO)$$

gelten soll.

Wenn die Voraussetzungen 1 bis 5 erfüllt sind, ist die Vereinigungsmenge $\bigcup_j \mathcal{P}_j$ der Eigenschaftsmengen der Teilblöcke vollständig und verifiziert M. Dies wird im Anhang in voller Allgemeinheit bewiesen.

6.3.3 Prüfung der Voraussetzungen

Wie die Voraussetzungen 1 und 2 geprüft werden, wurde im Abschnitt 6.2 detailliert beschrieben. Die Voraussetzungen 4 und 5 sind offensichtliche SAT-Anwendungen.

Es bleibt zu klären, wie die Existenzaussage der Voraussetzung 3 zu untersuchen ist. In der Praxis sind die einzelnen Interfaces eines Blocks meist unabhängig voneinander, und die Signale von II bilden ein solches Interface. Es steht also zu erwarten, dass sich von jedem IA_j ein Konjunkt IA_j^{II} über die Signale von II abspalten läßt, das unabhängig ist von XX', d.h.

$$IA_j(\boldsymbol{II}, \boldsymbol{XX'}, UU, OO, XX°) = IA_j^{II}(\boldsymbol{II}, UU, OO, XX°) \wedge IA_j^{XX'}(II, \boldsymbol{XX'}, UU, OO, XX°)$$

In diesem Fall reduziert sich dann die Prüfung der Voraussetzung 3 auf die durch SAT leicht zu bewältigende Implikation

$$GA(\boldsymbol{II}, UU, XX, OO) \Rightarrow \bigwedge_j IA_j^{II}(\boldsymbol{II}, UU, OO, XX°)$$

6.3.4 Rolle der Kommunikationsinfrastruktur

Bei der Auswahl von IP-Blöcken steht meist noch nicht fest, wie genau die Blöcke verschaltet werden sollen. Entsprechend wird nur für ausgewählte Verbindungen geprüft, ob die beteiligten Blöcke miteinander kommunizieren können. Welche Blöcke dabei in die Untersuchung einbezogen werden, hängt von der beabsichtigten Kommunikation ab. Punkt-zu-Punkt-Verbindungen benötigen nur zwei Blöcke, wenn aber Blöcke über ein Bussystem kommunizieren sollen, dann werden alle Blöcke des Bussystems verschaltet werden. Wenn das Bussystem später noch passend zu den Blöcken entwickelt werden soll, kann es zunächst durch eine einfache Struktur ersetzt werden. Diese Struktur muss nicht besonders effiziente Kommunikation ermöglichen, sondern nur die protokollgerechte Kommunikation aller Beteiligter. Eine Busmatrix mit mehreren parallelen Möglichkeiten zum Verbindungsaufbau muss beispielsweise nicht entwickelt werden, bevor die Kompatibilität von IP-Blöcken geprüft werden kann, es reicht auch eine einfache Verbindungsstruktur. Genauso wenig ist es erforderlich, Flip-Flops zur Laufzeitsynchronisation bei physikalisch langen Bussen schon eingeführt zu haben. Sofern eine leistungsfähige Businfrastruktur erst nach der Auswahl der IP-Blöcke entwickelt werden soll, reicht für den Kompatibilitätstest eine einfache Struktur. Wenn sich die Blöcke als kompatibel erweisen, wird sich die Businfrastruktur später passend dazu entwickeln lassen.

Vorteilhaft ist es, wenn die Businfrastruktur selbst ein IP-Block ist. In diesem Fall ist dieser IP-Block genauso eines der M_j wie jeder andere IP-Block auch. In diesem Fall wird es besonders einfach, wie die Variablen XX im Abschnitt 6.3.2 die Verschaltung der Blöcke darstellen müssen. Denn durch die Verwendung eines IP-Blocks als Kommunikationsinfrastruktur reduziert sich die Verdrahtung, die beim formalen Kompatibilitätstest untersucht werden muss, auf Punkt-zu-Punkt-Verbindungen jedes IP-Blocks zu dem IP-Block für die Kommunikationsinfrastruktur.

6.4 Zerlegen vollständiger Verifikationen

6.4.1 Zerlegen einer Verifikationsaufgabe in Cluster

In diesem Abschnitt geht es um die vollständige Verifikation einer im Ganzen konzipierten und entwickelten Schaltung, die beispielsweise einer der Blöcke ist, die mit den zuvor dargestellten Verfahrensweisen zu einem SoC zusammengefügt werden.

Wie bereits in Abschnitt 3.5.6 angesprochen, lassen sich solche Schaltungen in vielen Fällen nicht durch einen einzelnen Operationsautomaten beschreiben. Statt dessen müssen die Schaltungen in Teilen betrachtet werden, die Cluster genannt werden, und die ihrerseits durch jeweils einen Operationsautomaten beschrieben werden. Die Gesamtfunktion solcher Schaltungen ergibt sich also durch mehrere Ketten von Operationseigenschaften, die nebenläufig nebeneinander herlaufen, und über Signale miteinander verbunden sind. Entsprechend haben die einzelnen Cluster Eingangs- und Ausgangssignale, die aber im Kontext der Gesamtschaltung interne Signale sind.

Im Prinzip wäre es möglich, die Cluster jeweils isoliert zu untersuchen. Dazu könnten aus dem RTL Automaten gebildet werden, deren Eingangs- und Ausgangssignale jeweils mit den Ein- und Ausgängen der Cluster übereinstimmen. Die Schaltung zerfällt damit gleich einem Puzzle in Einzelteile, die ihrerseits vollständig verifiziert werden, und deren Verifikationen nach den Regeln aus Abschnitt 6.3.2 zu einer vollständigen Verifikation der Ausgangsschaltung zusammengefügt werden.

6.4.2 Vorteile von Modellen mit mehreren Clustern

In vielen Fällen ist es aber nicht erforderlich, die Cluster vollständig zu isolieren. Ein Beispiel dafür ist in Abbildung 20 angegeben. In diesem Beispiel können die Beweise der Operationseigenschaften für M_1 und M_2 auf dem Gesamtmodell der Schaltung durchgeführt werden. Es braucht nur $IC_1 \Rightarrow IA_2$ vorausgesetzt zu werden, damit die Argumentation aus Abschnitt 6.1.4 angewendet werden kann, um nachzuweisen, dass die Eigenschaftsmengen für M_1 und M_2 zusammen eine vollständige Verifikation der Gesamtschaltung darstellen. Es ist dabei nicht einmal erforderlich, dass IA_2 implementierbar ist oder dass IA_2 und M_2 strukturell kompatibel sind.

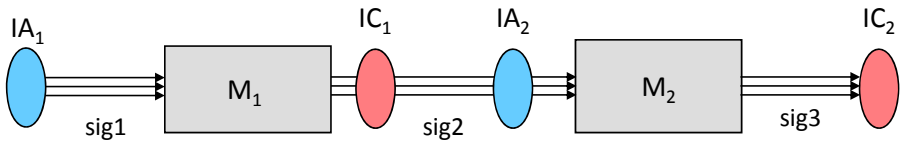

Abbildung 20: Eingangssignale eines Clusters als interne Signale

Wenn die Cluster nicht komplett separiert werden, gibt es Ein- und Ausgangssignale eines Clusters, die interne Signale des untersuchten Modells sind. Solche Ein- und Ausgangssignale werden als intern bezeichnet, im Gegensatz zu den echten Ein- und Ausgangssignalen, die als primär bezeichnet werden. Wenn das Interface eines Clusters im Mittelpunkt steht, ohne dass zwischen primären und internen Signale unterschieden wird, wird von lokalen Ein- und Ausgängen des Clusters gesprochen. In Abbildung 20 sind die Signale $sig2$ interne Eingangssignale des Clusters M_2 und interne Ausgangssignale von M_1. Die lokalen Ein- und Ausgabesignale von M_2 sind $sig2$ und $sig3$.

Als lokale Constraints werden Bedingungen über die lokalen Eingangssignale bezeichnet, die für die Verifikation des einzelnen Clusters die Rolle von Constraints einnehmen, also nicht während der vollständigen Verifikation des Clusters hinterfragt werden. Aus dem Blickwinkel der gesamten Verifikation werden die lokalen Constraints durch Assertions der Nachbarmodule und globale Constraints der Gesamtverifikation entlastet. Analog zu den lokalen Constraints gibt es lokale Determinierungsannahmen über die lokalen Eingangssignale. Diese lokalen Constraints und Determinierungsannahmen formen die Integrationsannahme eines Clusters. In Abbildung 19 besteht IA_2 aus lokalen Constraints und Determinierungsannahmen, aber die Constraints in IA_2 sind eigentlich Assertions, die unter Verwendung von M_1 bewiesen werden können.

Es ist vorteilhaft, die Cluster einer vollständigen Verifikation nicht zu separieren, denn das spart Verifikationsaufwand. Der Cluster M_1 in Abbildung 19 sorgt beispielsweise dafür, dass M_2 nur solche Traces erhält, die M_1 tatsächlich produzieren kann. Da die Verifikation mit IPC durchgeführt wird, gehören dazu zwar auch solche Traces, die sich von unerreichbaren Zuständen in M_1 aus ergeben, aber dennoch bedeutet dies eine Einschränkung des Verhaltens von $sig2$. Diese Einschränkung macht die Verifikation von M_2 leichter, ohne dass diese Einschränkung in IC_1 oder IA_2 explizit gemacht werden müßte.

Wären M_1 und M_2 nicht verbunden, und $sig2$ primäre Eingänge eines Modells, das nur M_2 umfaßt, dann müßten diese Einschränkungen explizit formuliert werden. Dabei ist das Protokoll des internen Interfaces zwischen M_1 und M_2 meist nicht standardisiert, sodass die Identifikation der Protokollbedingungen eine aufwändige Arbeit werden kann. Immerhin müssen Bedingungen identifiziert werden, die stark genug sind, dass die Beweise der Operationseigenschaften über M_2 geführt werden können und schwach genug sind, um selbst auf M_1 bewiesen zu werden. Diesen Kompromiss zu finden, kann bereits dann schwer sein, wenn M_1 und M_2 fehlerfrei sind. Wenn aber zudem auch noch M_2 einen Schaltungsfehler hat, der nur bei Nutzung eines bestimmten Protokollfeatures auftritt, dann wird die Situation schwer durchschaubar und der Benutzer wird versucht sein, das Protokollfeature abwechselnd ein- und auszubauen, je nachdem, ob gerade M_1 oder M_2 verifiziert wird. Solange M_1 und M_2 verbunden sind, brauchen viele Bedingungen des Protokolls nicht explizit formuliert werden, und das spart Aufwand.

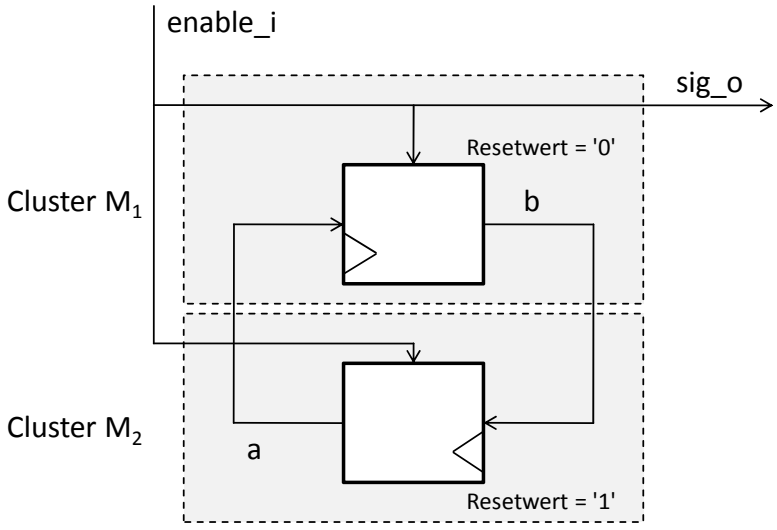

Abbildung 21: Zyklische Abhängigkeit von Clustern

6.4.3 Notwendigkeit primärer Ein- und Ausgangssignale

Leider lässt sich die Begründung für die Vollständigkeit der Verifikation der Schaltung aus Abbildung 19 nicht ohne weiteres auf Schaltungen übertragen, bei denen die Cluster zyklisch verschaltet sind. Ein Beispiel für die Verifikationslücken, die dann auftreten können, zeigt das Beispiel in Abbildung 21. Dort sind zwei Cluster gegeben, die jeweils vor allem aus einem Flip-Flop mit Enable bestehen. Wenn das Enable-Signal den Wert '1' hat, speichern die Flip-Flops den Wert des Dateneingangssignals ab, andernfalls behalten sie den bereits gespeicherten Wert.

Cluster M_1 läßt sich vollständig verifizieren unter der Integrationsbedingung

```
integration_condition of M1 is

assume:
      determined(a);
      determined(enable_i);
      prev(reset) or prev(a) /= a;

guarantee:
      determined(b);
      prev(reset) or prev(b) /= b;
      determined(sig_o);
      sig_o = '1';

end integration_condition;
```

Die Operationseigenschaft dazu ist

```
property m1_op is
dependencies: constraints_from_integration_assumption, no_reset;
assume:
      at t:                  b = '1';
prove:
      at t+1:                b = '0';
      at t+2:                b = '1';
      during [t+1, t+2]:     sig_o = '1';
end property;
```

Diese Eigenschaft lässt sich auf dem Gesamtmodell beweisen. Weil der Constraint den Ausgang a des Flip-Flops aus M_2 beschränkt, verlangt der Constraint implizit, dass das Enable-Signal '1' sein soll, sobald der Reset inaktiv wird. Daher werden die Registerwerte tatsächlich immer weitergeschoben, sodass sich der mit der Operationseigenschaft behauptete Wechsel der Werte tatsächlich einstellt und somit auch die Assertion der Integrationszusicherung bewiesen ist.

Analog dazu kann M_2 unter der Integrationsbedingung

```
intergration condition of M2 is

assume:
      determined(b);
      determined(enable_i);
      prev(reset) or prev(b) /= b;

guarantee:
```

```
        determined(a);
        prev(reset) or prev(a) /= a;

    end integration_condition;
```

vollständig bewiesen werden.

Aber aus diesen beiden vollständigen Verifikationen darf nicht gefolgert werden, dass nun die Gesamtschaltung die Integrationsbedingung

```
    integration_condition of M is

    assume:
        determined(enable_i);

    guarantee:
        determined(sig_o);
        sig_o = '1';

    end integration_condition;
```

erfüllt, denn dies wäre offensichtlich falsch: sig_o ist direkt mit enable_i verbunden und enable_i ist nicht beschränkt. Daher kann sig_o auch '0' werden.

Dabei erfüllen die Integrationsbedingungen von M_1 und M_2 sogar fast alle Voraussetzungen für das Zusammensetzen vollständiger Verifikation aus Abschnitt 6.3.2. Die einzige Ausnahme besteht darin, dass in Abschnitt 6.3.2 vorausgesetzt war, dass die IP-Blöcke auf separaten Modellen vollständig verifiziert wurden. Tatsächlich lassen sich die Eigenschaften nicht mehr beweisen, wenn die Cluster M_1 und M_2 in separaten Modellen verifiziert werden, in denen a und b primäre Ein- und Ausgangssignale sind. Entsprechend stimmen die Integrationsbedingungen von M_1 und M_2 nicht mehr und die globale Integrationsbedingung kann nicht mehr daraus gefolgert werden.

6.4.4 Anpassung des Modells auf der Basis des Clustergraphen

Die für diese Arbeit entwickelte Theorie der kompositionalen vollständigen Verifikation weist einen Mittelweg zwischen dem Wunsch, Aufwand dadurch zu sparen, dass mehrere Cluster im - der jeweiligen Verifikation unterliegenden - Modell der Schaltung verbunden bleiben, und der Notwendigkeit, die Cluster in separaten Modellen zu untersuchen, die durch das Gegenbeispiel im vorigen Abschnitt 6.4.3 illustriert wurde.

Dieser Mittelweg besteht darin, Modelle zu erlauben, in denen mehrere Cluster durch interne Signale verbunden sein dürfen, in denen die Cluster aber nicht zyklisch voneinander abhängen dürfen.

Um eine solche Verifikation einzurichten, wird zunächst der Clustergraph aufgestellt. In diesem Graph sind die Cluster Knoten, die durch gerichtete Kanten verbunden werden. Die Kanten führen von einem Cluster zu einem anderen, wenn es ein lokales Ausgabesignal des ersten Clusters gibt, das lokales Eingabesignal des zweiten Clusters ist. Im allgemeinen wird dieser Clustergraph Zyklen haben. Diese Zyklen müssen aufgebrochen werden. Dies geschieht, indem eine Anzahl von Kanten aus dem Clustergraph eliminiert wird.

Diese Kanten entsprechen Signalen, die bei der Modellgenerierung geschnitten werden. Durch das Schneiden wird der lokale Eingang des von dem Signal versorgten Clusters zu einem primären Eingang, und der lokale Ausgang des anderen Clusters zu einem primären Ausgang.

Auf diese Weise entsteht ein Modell mit einigen zusätzlichen primären Eingängen und Ausgängen, in dem aber Cluster verbunden sein können. Auf diesem Modell werden die vollständigen Verifikationen für alle Cluster durchgeführt. Wenn bei diesen Verifikationen reaktive Integrationsannahmen über lokale Clustereingänge erforderlich werden, müssen diese lokalen Clustereingänge nachträglich noch geschnitten werden und die Verifikationen evt. nachgezogen werden.

Wenn es also Cluster gibt, die in einer Weise zusammenarbeiten, dass immer ein Cluster Informationen an den nächsten weiterreicht, braucht deren Interaktion nicht aufgeschnitten werden und es kann auf diese Weise Aufwand gespart werden.

Doch kann es auch Gründe geben, mehr als die minimale Anzahl von Kanten des Clustergraphen zu eliminieren und die entsprechenden Signale zu schneiden. Wenn etwa das Modell zu groß wird und zu erheblichen Beweiserlaufzeiten führt, kann es in wenige Teilmodelle zerlegt werden, indem die Kanten so eliminiert werden, dass der Clustergraph in unabhängige Teilgraphen zerfällt.

Im Extremfall führt die Elimination aller Kanten zu einer Verifikation, bei der jeder Cluster auf seinem eigenen separaten Modell verifiziert wird. Diese Verifikation unterscheidet sich dann nicht mehr von der Situation beim Zusammenfügen vollständiger Verifikationen aus Abschnitt 6.3.

6.4.5 Forderungen an die Integrationsannahmen

Auf diese Weise wird die ursprüngliche Aufgabe der vollständigen Verifikation der Gesamtschaltung M in Teilaufgaben zerlegt, die jede für sich die vollständige Verifikation eines Clusters M_j erfordert, aber auf einem Modell M' durchgeführt wird, das immer noch mehrere Cluster und viele, aber nicht mehr alle Verbindungen zwischen den Clustern enthält. Der Aufwand der Verifikation der M_j auf dem Modell M' ist höher als wenn er auf M hätte durchgeführt werden können, aber diese Aufwandserhöhung hält sich häufig bei geeigneter Wahl geschnittenen Signale in Grenzen und bleibt unterhalb des Zusatzaufwands, der entstünde, wenn auf separaten Modellen der M_j verifiziert würde.

Wie zuvor in Abschnitt 6.3.2 müssen die Integrationsbedingungen der Teilaufgaben und die globalen Integrationsbedingungen außerdem einige Voraussetzungen erfüllen, damit die Gesamtschaltung durch die Menge aller Eigenschaften der Cluster vollständig verifiziert wird.

Um diese Voraussetzungen zu formalisieren, muss beachtet werden, dass es für jeden Cluster nun drei Kategorien von lokalen Eingangssignalen gibt. Diese sind die primären Signale der Gesamtschaltung, deren Trace nachfolgend weiterhin mit I bezeichnet werden soll, die durch das Schneiden neu entstandenen Eingangssignale mit dem Trace X' und die internen Eingangssignale mit dem Trace Y. Analog gibt es dazu primäre Ausgangssignale mit dem Trace O, durch Schneiden neu entstandene Ausgangssignale mit dem Trace $X°$ und die internen Ausgangssignale mit dem Trace Y.

Die folgenden Voraussetzungen sind abgewandelte Versionen derjenigen in Abschnitt 6.3.2, vornehmlich ergänzt durch die Behandlung des Traces Y der internen Ein- und Ausgänge der Cluster.

Voraussetzung 1 verlangt, dass sich die Integrationsannahme IA_j jedes Clusters schreiben läßt durch einen Teil IA_j^{int} über die internen Ein- und Ausgangssignale, und einen Teil IA_j^{prim} über die primären Signale des Modells, d.h. die primären Signale der Schaltung und alle geschnittenen Signale. Es muss gelten

$$IA_j(II, XX', YY, UU, OO, XX°) = IA_j^{int}(YY, UU, OO) \wedge IA_j^{prim}(\boldsymbol{II}, \boldsymbol{XX}', YY, UU, OO, XX°)$$

Die kombinatorischen Abhängigkeiten von M, der Integrationsannahmen IA_j^{prim} und der globalen Integrationsannahme GA müssen sich wiederum zu einem zyklenfreien Graph $K^M \cup K^{GA} \cup K^{IA_j^{prim}}$ zusammenfügen.

Voraussetzung 2 verlangt, dass die globale Integrationsannahme $GA(\boldsymbol{II}, UU, XX, YY, OO)$ eine implementierbare Integrationsannahme ist und dass der Eingabetrace II von MM' durch GA beschränkt wird. Außerdem müssen die Integrationsannahmen $IA_j^{prim}(\boldsymbol{II}, \boldsymbol{XX}', YY, UU, OO, XX°)$ implementierbare Integrationsannahmen sein, die II und XX' beschränken.

Voraussetzung 3 verlangt, dass $GA(\boldsymbol{II}, UU, XX, YY, OO)$ auf freien Variablen nur solche Traces II zulassen darf, die auch von den Integrationsannahmen IA_j^{prim} zugelassen werden, d.h.

$$GA(\boldsymbol{II}, UU, XX, YY, OO) \Rightarrow \bigvee_{XX'} \bigwedge_j IA_j^{prim}(\boldsymbol{II}, \boldsymbol{XX}', YY, UU, OO, XX°)$$

Voraussetzung 4 verlangt, dass die lokalen Integrationszusicherungen jedes Clusters und die globale Integrationsannahme gemeinsam die lokalen Integrationsannahmen der Cluster erfüllen müssen. Entsprechend muss auf freien Variablen gelten

$$GA(\boldsymbol{II}, UU, XX°, YY, OO) \wedge GA(\boldsymbol{JJ}, UU, XX', YY, OO) \wedge \bigwedge_j IC_j(II, XX', YY, UU, OO, XX°)$$
$$\Rightarrow \bigwedge_j IA_j(JJ, XX°, YY, UU, OO\ XX')$$

Dabei bezeichnet JJ einen Trace von Kopien der Variablen aus II.

Voraussetzung 5 verlangt, dass die globale Integrationszusicherung eine Folgerung der lokalen Integrationszusicherungen und der globalen Integrationsannahme sein muss, dass also wiederum auf freien Variablen

$$GA(\boldsymbol{II}, UU, XX, YY, OO) \wedge \bigwedge_j IC_j(II, XX, YY, UU, OO, XX) \Rightarrow GC(II, UU, XX, YY, OO)$$

gelten soll.

Wenn die Voraussetzungen 1 bis 5 erfüllt sind, ist die Vereinigungsmenge $\bigcup_j \mathcal{P}_j$ der Eigenschaftsmengen der Cluster vollständig und verifiziert M. Dies wird im Anhang detailliert bewiesen.

Zur Prüfung der Voraussetzungen sei auf Abschnitt 6.3.3 verwiesen.

6.4.6 Beispiel

Zum Abschluss dieser Arbeit soll die Zerlegung einer vollständigen Verifikation anhand einer Schaltung demonstriert werden, die aus dem Prozessor und dem SDRAM-Interface besteht. Beide sollen gemeinsam entwickelt worden sein und das SDRAM-Interface als exklusives Interface zu einem Befehlsspeicher dienen. Das Szenario sieht vor, dass noch keine Verifikation durchgeführt wurde. Das Protokoll zwischen SDRAM-Interface und Prozessor soll für ein ad-hoc-Protokoll stehen. Das bedeutet insbesondere, dass zum Beginn der Verifikation keine Protokollbeschreibung über das Prozessorprotokoll vorliegt, weder informell durch eine Spezifikation, noch gar durch die Integrationsbedingungen aus Abschnitt 3.5.2.

Es sollen SDRAM-Speicherblöcke von Drittanbietern eingesetzt werden. Diese seien durch Datenblätter charakterisiert, aus denen sich unter Berücksichtigung der Taktfrequenzen folgende Integrationsbedingungen ablesen lassen:

```
integration_condition of sdram is
assume:
      determined(sd_ctrl);
      if sd_ctrl = read or sd_ctrl = write or sd_ctrl = activate then
            determined(sd_addr);
      end if;
      if sd_ctrl = precharge or sd_ctrl = activate then
            next(sd_ctrl = nop) end if;
      end if;
      if sd_ctrl = write then determined(sd_wdata) end if;
```

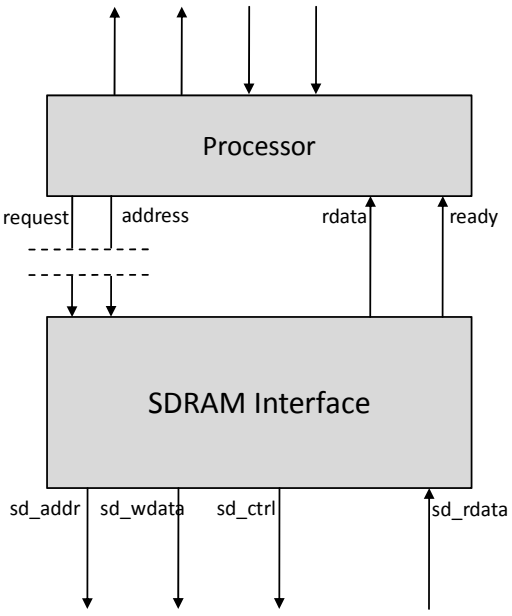

Abbildung 22: Zur Verifikation einer Gesamtschaltung aus Prozessor und SDRAM Interface

```
guarantee:
      if prev(sd_ctrl, 2) = read then determined(sd_rdata) end if;
end integration_condition;
```

Der Clustergraph aus Prozessor und SDRAM-Interface ist zyklisch. Da das SDRAM-Interface zur korrekten Funktion fordert, dass request und address-Signale während eines Requests konstant sein müssen, bis das ready-Signal aktiviert wurde, gibt es hier eine reaktive Abhängigkeit und deshalb werden diese Signale geschnitten. Deshalb wird die Verifikation auf einem geschnittenen Modell entsprechend Abbildung 22 durchgeführt. Dieses Bild entspricht Abbildung 3, allerdings mit ein paar Unterschieden. Insbesondere wird das rw-Signal behandelt, als sei es immer auf '1' festgelegt, weil das SDRAM-Interface nur Instruktionen lesen soll. Daher ist auch das Schreibdatensignal überflüssig[5].

Der Prozessor funktioniere nur korrekt, wenn das ready-Signal Pulse liefert und niemals zwei Takte nacheinander aktiv ist. Diese Voraussetzung müsste explizit Teil der Integrationsbedingungen sein, wenn Prozessor und SDRAM-Interface separat verifiziert und danach die Verifikationen entsprechend Abschnitt 6.3 zusammengesetzt würden. Durch die Verschaltung beider Module in einem Modell kann auf diese Bedingung verzichtet werden. Die Integrationsbedingungen ergeben sich zu

```
integration_condition of processor is
assume:
      determined(ready);
      if ready = '1' then determined(rdata); end if;

      <Annahmen über Dateninterface>
guarantee:
      determined(request);
      if request = '1' then determined(address); end if;
      if prev(request) = 1 and prev(ready) = 0 then
            request = '1' and
            address = prev(address);
      end if;

      <Zusicherung über Dateninterface>
end integration_condition;
```

und

```
integration_condition of sdram_if is
assume:
      determined(request);
      if request = '1' then determined(address); end if;
      if prev(request) = 1 and prev(ready) = 0 then
            request = '1' and
            address = prev(address);
      end if;

      if prev(sd_ctrl, 2) = read then determined(sd_rdata) end if;
guarantee:
      determined(ready);
      if ready = 1 then
            determined(rdata);
      end if;
```

[5] Es soll einfach angenommen werden, dass bereits Programmcode im Speicher steht. Die Mechanismen zum Laden des Speichers mit dem Programmcode bleiben unberücksichtigt.

```
      determined(sd_ctrl);
      if sd_ctrl = precharge or sd_ctrl = activate then
            next(sd_ctrl = nop) end if;
      if sd_ctrl = read or sd_ctrl = activate then
            determined(sd_addr)
      end if;
      sd_ctrl /= write;
end integration_condition;
```

In den Integrationsbedingungen wird nicht zwischen den durch das Schneiden entstandenen neuen Ein- und Ausgängen unterschieden, weil die jeweiligen Signale automatisch zugeordnet werden können.

Es soll nun geprüft werden, ob diese beiden Integrationsbedingungen verifizieren, dass die Gesamtschaltung unter der globalen Integrationsbedingung

```
integration_condition of full_circuit is
assume:
      if prev(sd_ctrl, 2) = read then determined(sd_rdata) end if;
      <Annahmen über Dateninterface>
guarantee:
      determined(sd_ctrl);
      if sd_ctrl = precharge or sd_ctrl = activate then
            next(sd_ctrl = nop) end if;
      if sd_ctrl = read or sd_ctrl = activate then
            determined(sd_addr)
      end if;
      sd_ctrl /= write;
      <Zusicherung über Dateninterface>
end integration_condition;
```

vollständig verifiziert ist. Dazu werden die Voraussetzungen aus Abschnitt 6.4.5 geprüft.

Zunächst ist der Clustergraph aus Abbildung 22 zyklenfrei. Der SDRAM-Cluster hat keine lokalen Clustereingänge, daher ist $IA_{sd}^{int} = true$. IA_{sd} sieht auch keine kombinatorischen Abhängigkeiten vor.

Beim Prozessorinterface wird IA_p^{int} durch die gesamte Integrationsannahme gegeben, wobei vom Dateninterface abgesehen werden soll. Hier ist $IA_p^{prim} = true$, sodass es auch hier keine kombinatorischen Abhängigkeiten gibt.

Es gibt auch in GA keine kombinatorischen Abhängigkeiten. Damit ist die Voraussetzung 1 erfüllt.

IA_{sd}^{prim}, IA_p^{prim} und GA sind implementierbar. Daher ist Voraussetzung 2 erfüllt.

Durch Nachrechnen lässt sich erkennen, dass auch die Voraussetzungen 3, 4 und 5 erfüllt sind. Damit ist gezeigt, dass die Eigenschaftsmengen des Prozessors und des SDRAM-Interfaces gemeinsam die Gesamtschaltung vollständig verifizieren.

7 Nachwort

Diese Arbeit beschreibt einen Durchbruch bei der funktionalen Verifikation digitaler Schaltungsentwürfe. Sie überträgt zunächst das Paradigma der transaktionsbasierten Verifikation aus der Simulation in die formale Verifikation. Ein Ergebnis dieser Übertragung ist eine bestimmte Form von formalen Eigenschaften, die Operationseigenschaften genannt werden. Schaltungen werden mit Operationseigenschaften untersucht durch Interval Property Checking, einer besonders leistungsfähigen SAT-basierten funktionalen Verifikation. Dadurch können Schaltungen untersucht werden, die sonst als zu komplex für formale Verifikation gelten. Ferner beschreibt diese Arbeit ein für Mengen von Operationseigenschaften geeignetes Werkzeug, das alle Verifikationslücken aufdeckt, komplexitätsmäßig mit den Fähigkeiten der IPC-basierten Schaltungsuntersuchung Schritt hält und als Vollständigkeitprüfer bezeichnet wird. Die Methodik der Operationseigenschaften und die Technologie des IPC-basierten Eigenschaftsprüfers und des Vollständigkeitsprüfers gehen eine vorteilhafte Symbiose zum Vorteil der funktionalen Verifikation digitaler Schaltungen ein. Darauf aufbauend wird ein Verfahren zur lückenlosen Überprüfung der Verschaltung derartig verifizierter Module entwickelt, das aus den Theorien zur Modellierung digitaler Systeme abgeleitet ist.

Der in dieser Arbeit vorgestellte Ansatz hat in vielen kommerziellen Anwendungsprojekten unter Beweis gestellt, dass er den Namen "vollständige funktionale Verifikation" zu Recht trägt, weil in diesen Anwendungsprojekten nach dem Erreichen eines durch die Vollständigkeitsprüfung wohldefinierten Abschlusses keine Fehler mehr gefunden wurden. Der Ansatz wird von OneSpin Solutions GmbH unter dem Namen "Operation Based Verification" und "Gap Free Verification" vermarktet.

Aufgrund der Bedeutung der vollständigen Verifikation für das Geschäft der OneSpin Solutions GmbH sollen hier keine konkreten offenen Aufgaben und nächsten Schritte beschrieben werden. Ich meine jedoch, dass die Beziehung zwischen dem weiten Feld digitaler Modellierung und der vollständigen Verifikation sich nicht auf die kompositionale vollständige Verifikation beschränken sollte. Ich denke, dass in den vielen Arbeiten zur digitalen Modellierung ein Schatz wichtiger Ideen schlummert, der die vollständige Verifikation entscheidend befruchten kann, und noch längst nicht gehoben ist. Umgekehrt würde ich mich freuen, wenn einige der hier vorgestellten Gedanken auch Anstöße für die Arbeit an der Modellierung digitaler Systeme bereithalten.

8 Anhang: Die Mathematik der kompositionalen vollständigen Verifikation

8.1 Implementierbarkeit

8.1.1 Erstes Implementierbarkeitskriterium

Dieser Abschnitt soll zeigen, dass es tatsächlich eine Ersatzschaltung gibt, wenn das erste Implementierbarkeitskriterium aus Abschnitt 6.2.3 erfüllt ist. Wie dort werden die internen Signale als Teil des Traces OO behandelt. Die Integrationsannahme sei durch einen ITL-Ausdruck gegeben, der entsprechend der Formeln 6-2 und 6-3 durch die Beziehung

$$IA(II, OO) = \bigwedge_{t \geq 0} \widetilde{IA}\left(II^{(t)}, II^{[0,t-1]}, OO^{[0,t]}\right)$$

und

$$\widetilde{IA}\left(II^{(t)}, II^{[0,t-1]}, OO^{[0,t]}\right) = \bigwedge_{l=0}^{n} \widetilde{IA}_l\left(II_l^{(t)}, II_{[0,l-1]}^{(t)}, II^{[0,t-1]}, OO^{[0,t]}\right)$$

in Teilausdrücke $\widetilde{IA}_l$ partitioniert sei, die jeweils einen Anteil von Eingangssignalen beschreiben, die durch die Traces $II_l^{(t)}$ beschränkt werden. In Abschnitt 6.2.3 wird behauptet, dass die Integrationsannahme IA implementierbar ist, wenn für alle $l \in [0, n]$ und alle $t \in [0, m-1]$ die Gleichungen 6-4

$$\left(\varrho\varrho(II) \wedge \bigwedge_{k=0}^{t-1} \widetilde{IA}\left(II^{(k)}, II^{[0,k-1]}, OO^{[0,k]}\right) \wedge \bigwedge_{k=0}^{l-1} \widetilde{IA}_k\left(II_k^{(t)}, II_{[0,k-1]}^{(t)}, II^{[0,t-1]}, OO^{[0,t]}\right)\right) \Rightarrow \bigvee_{ii_l} \widetilde{IA}_l\left(ii_l, II_{[0,l-1]}^{(t)}, II^{[0,t-1]}, OO^{[0,t]}\right)$$

sowie für alle $l \in [0, n]$ und alle $t \geq m$ die Gleichung 6-5

$$\left(\bigwedge_{k=t-m}^{t-1} \widetilde{IA}\left(II^{(k)}, II^{[0,k-1]}, OO^{[0,k]}\right) \wedge \bigwedge_{k=0}^{l-1} \widetilde{IA}_k\left(II_k^{(t)}, II_{[0,k-1]}^{(t)}, II^{[0,t-1]}, OO^{[0,t]}\right)\right) \Rightarrow \bigvee_{ii_l} \widetilde{IA}_l\left(ii_l, II_{[0,l-1]}^{(t)}, II^{[0,t-1]}, OO^{[0,t]}\right)$$

erfüllt sind. Diese Behauptung soll nachfolgend durch Angabe einer Ersatzschaltung bewiesen werden. Die Ersatzschaltung erfüllt keinerlei Optimierungskriterien des Schaltungsentwurfs. Das ist aber auch nicht erforderlich, weil nur die Existenz der Ersatzschaltung interessiert.

Eine Skizze der Ersatzschaltung zeigt Abbildung 23. Zunächst sei bemerkt, dass die Integrationsannahme als ITL-Ausdruck gegeben sein soll. Daher ist gibt es eine maximale Verzögerung m, mit der Schaltungssignale ausgewertet werden. Dies erlaubt die Verwendung von Schieberegistern, die in der Skizze unten rechts dargestellt sind, um auf verzögerte Signalwerte zuzugreifen.

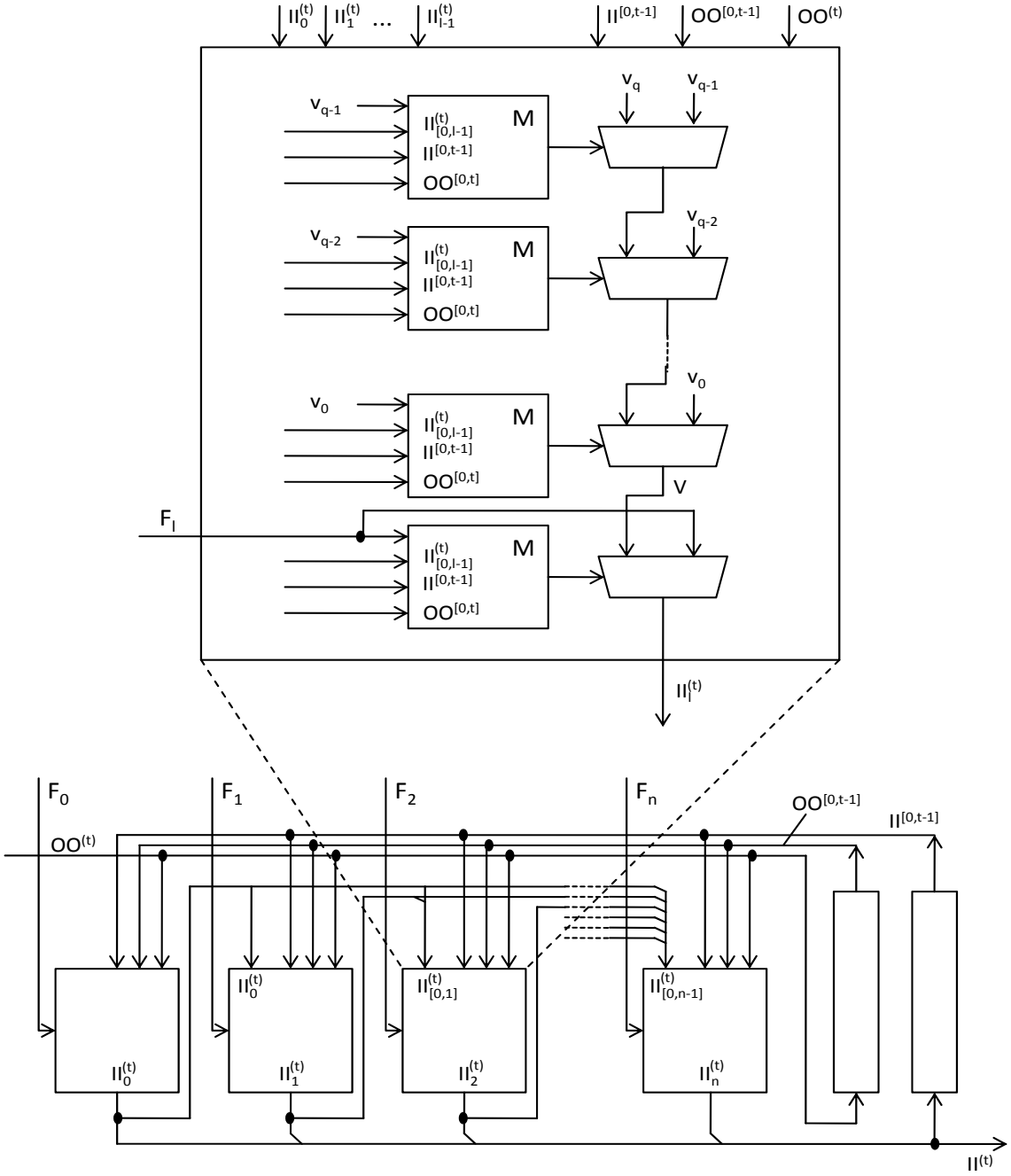

Abbildung 23: Ersatzschaltung für Integrationsannahmen (ausgefüllte Kreise bezeichnen Leitungsverbindungen, schräge Linien die Hinzufügung eines Signals zu einem Leitungsbündel)

Das Modul M generiere ein Ausgangssignal, das anzeigt, ob $\widetilde{IA}_l\left(ii, II^{(t)}_{[0,l-1]}, II^{[0,t-1]}, OO^{[0,t]}\right)$ erfüllt ist, wobei ii eines seiner Eingangssignale ist. Die Listen $II^{[0,t-1]}$ und $OO^{[0,t-1]}$ verzögerter Eingabe- und Ausgabewerte lassen sich durch Schieberegister versorgen, da eine maximale Verzögerung vorausgesetzt ist. $v_0, v_1, v_2, \ldots, v_q$ sei eine Enumeration aller Werte, die ii_l annehmen kann. Dann gibt es im oberen Teil der Abbildung 23 für jedes $v \in [0, q]$ einen Schaltungsteil, der mit einem Ausgangssignal anzeigt, ob $\widetilde{IA}_l\left(v_v, II^{(t)}_{[0,l-1]}, II^{[0,t-1]}, OO^{[0,t]}\right)$

erfüllt ist. Die Multiplexerstruktur bis zum Eingang des letzten Multiplexers bestimmt damit eine Art Rückfallwert v, für den $\widetilde{IA}_l\left(v, II^{(t)}_{[0,l-1]}, II^{[0,t-1]}, OO^{[0,t]}\right)$ erfüllt ist, sofern es ein solches v überhaupt gibt. Durch den letzten Multiplexer und den freien Eingang wird dafür gesorgt, dass der obere Teil von Abbildung 23 alle möglichen Werte ausgibt, die $\widetilde{IA}_l$ erfüllen können. Sofern also die Existenz von v immer sichergestellt ist, ist dieser Schaltungsteil eine Ersatzschaltung für $\widetilde{IA}_l$.

Die Ersatzschaltungen zur Bestimmung von, $II^{(t)}_0, II^{(t)}_1, II^{(t)}_2$, ... werden so verschaltet, wie es der untere Teil des Diagramms in Abbildung 23 angibt. Um nachzuweisen, dass dies tatsächlich eine Ersatzschaltung für IA darstellt, muss also nur nachgewiesen werden, dass es in jeder Teilschaltung für $\widetilde{IA}_l$ immer den jeweiligen Wert v gibt. Dies läßt sich durch vollständige Induktion nachweisen. Für $t = 0$ befinden sich in dem Schieberegister zur Verzögerung der Eingabewerte die Resetsequenz und im Schieberegister für OO beliebige Werte. Nach Bedingung 6-4 existiert also zunächst ein Wert v für den Schaltungsteil zu $\widetilde{IA}_0$, dann ein Wert für den Schaltungsteil $\widetilde{IA}_1$, usf. Insgesamt kann die Schaltung also alle $II^{(0)}$ produzieren, die $\widetilde{IA}$ zum Zeitpunkt 0 erfüllen.

Zum Zeitpunkt $t = 1$ befindet sich eines dieser $II^{(0)}$ im Schieberegister und erfüllt die zweite der drei Bedingungen auf der linken Seite der Implikation 6-4. Der älteste Wert der Resetsequenz wurde aus dem Schieberegister hinausgeschoben. Auf diesen Wert kommt es bei der Entscheidung über $\widetilde{IA}_l\left(ii_l, II^{(1)}_{[0,l-1]}, II^{(0)}, OO^{[0,1]}\right)$ aber auch nicht mehr an, weil er nicht mehr referenziert wird. Forderung 6-4 sichert also wiederum die Existenz von Werten, mit denen sichergestellt wird, dass die oberen Schaltungsteile in Abbildung 23 alle möglichen Werte $II^{(1)}$ generieren, die $\widetilde{IA}$ zum Zeitpunkt 1 erfüllen.

Diese Argumentation kann entsprechend bis zum Zeitpunkt $t = m - 1$ wiederholt werden. Danach befinden sich in den Schieberegistern nur noch die Werte $II^{[t-m,t-1]}$ und die zugehörigen Schaltungsantworten. Über jeden im Schieberegister enthaltenen Wert ist bekannt, dass er $\widetilde{IA}$ zum jeweiligen Zeitpunkt erfüllt. Damit sichert die Implikation 6-5 wiederum von den Schaltungsteilen für $\widetilde{IA}_l$ geforderten Werte, sodass wiederum alle $II^{(t)}$ erzeugt werden können, die die Bedingung $\widetilde{IA}\left(II^{(t)}, II^{[0,t-1]}, OO^{[0,t]}\right)$ erfüllen.

Damit produziert diese Schaltung alle Traces II, die $\bigwedge_{t\geq 0} \widetilde{IA}\left(II^{(t)}, II^{[0,t-1]}, OO^{[0,t]}\right)$ und damit nach 6-2 auch $IA(II, OO)$ erfüllen und bildet daher eine Ersatzschaltung für IA.

8.1.2 Begründung der zweiten Implementierbarkeitsbedingung

In Abschnitt 6.2.3 wird eine zweite, einfachere Implementierbarkeitsbedingung angegeben, die voraussetzt, dass $\widetilde{IA}_l$ aus einem Constraint der Form

$$\texttt{if } c_l\left(I^{(t)}_{[0,l-1]}, I^{[0,t-1]}, O^{[0,t]}\right) \texttt{ then } I^{(t)}_l = V_l\left(I^{(t)}_{[0,l-1]}, I^{[0,t-1]}, O^{[0,t]}\right) \texttt{ else } R_l\left(I^{(t)}_l\right) \texttt{ end if;}$$

und einer Determinierungsannahme der Form

$$\texttt{if } g_l\left(I^{(t)}_{[0,l-1]}, I^{[0,t-1]}, O^{[0,t]}\right) \texttt{ then determined}\left(I^{(t)}_l\right) \texttt{ end if;}$$

besteht. R_l, c_l $g_l$und beschreiben Bedingungen, V_l einen Ausdruck, dessen Ergebnistyp mit dem Typ von I_l übereinstimmt, sodass die Gleichung im then-Zweig immer erfüllt werden

kann. Von welchen Signalwerten die Bedingungen und der Ausdruck abhängen dürfen, ist durch die Parameterlisten charakterisiert. Für die nachfolgenden Betrachtungen werden Indices und Parameterlisten von g, c und V unterdrückt, es wird lediglich $\bar{g}$, $\bar{c}$ und $\bar{V}$ geschrieben, wenn der Variablensatz von $\bar{M}$ angewandt wird.

In Abschnitt 6.2.3 wird behauptet, dass eine solche Integrationsannahme implementierbar ist, wenn für $l \in [0,n]$ und $t \in [0,m-1]$ die Bedingung 6-6

$$\left(\varrho\varrho(II) \wedge \bigwedge_{k=0}^{t-1} \widetilde{IA}\left(II^{(k)}, II^{[0,k-1]}, OO^{[0,k]}\right) \wedge \bigwedge_{k=0}^{l-1} \widetilde{IA}_k\left(II_k^{(t)}, II_{[0,k-1]}^{(t)}, II^{[0,t-1]}, OO^{[0,t]}\right)\right) \Rightarrow \left(R(F) \wedge ((g \vee \bar{g}) \wedge c \wedge \bar{c} \Rightarrow V = \bar{V})\right)$$

und für $l \in [0,n]$ und $t \geq m$ die Bedingung 6-7

$$\left(\bigwedge_{k=t-m}^{t-1} \widetilde{IA}\left(II^{(k)}, II^{[0,k-1]}, OO^{[0,k]}\right) \wedge \bigwedge_{k=0}^{l-1} \widetilde{IA}_k\left(II_k^{(t)}, II_{[0,k-1]}^{(t)}, II^{[0,t-1]}, OO^{[0,t]}\right)\right) \Rightarrow \left(R(F) \wedge \left((g \vee \bar{g}) \wedge c \wedge \bar{c} \Rightarrow V = \bar{V}\right)\right)$$

auf freien Variablen erfüllt ist.

Zum Beweis soll nachgewiesen werden, dass unter den gegebenen Umständen die Existenzaussage in 6-4 und 6-5 erfüllt ist, wenn $R(V) \wedge \left((g \vee \bar{g}) \wedge c \wedge \bar{c} \Rightarrow V = \bar{V}\right)$ gilt.

Dazu sei zunächst bemerkt, dass die Determinierungsannahme nur eine andere Schreibweise für

$$(\sim g \wedge \sim \bar{g}) \vee I_l^{(t)} = \bar{I}_l^{(t)}$$

ist. Wenn also sowohl g als auch $\bar{g}$ nicht erfüllt sind, können also die Werte von $I_l^{(t)}$ und $\bar{I}_l^{(t)}$ unabhängig voneinander entsprechend des Constraints ausgewählt werden. Dass dieser mindestens einen Wert zulässt, ergibt sich für den then-Zweig des Constraints trivialerweise. Für den else-Zweig ergibt es sich, weil $R(V)$ gefordert wird, aus dem sich per Symmetrie auch $R(\bar{V})$ ergibt.

Sei nun $g \vee \bar{g}$ erfüllt. Dann müssen $I_l^{(t)}$ und $\bar{I}_l^{(t)}$ gleich sein. Wenn unter dieser Bedingung c erfüllt ist, aber nicht $\bar{c}$, so können $I_l^{(t)}$ und $\bar{I}_l^{(t)}$ durch V bestimmt werden. Wegen $R(\bar{V})$ gilt auch $R(\bar{I}_l^{(t)})$, sodass das Paar aus $I_l^{(t)}$ und $\bar{I}_l^{(t)}$ den Constraint für den ungestrichenen und den gestrichenen Variablensatz erfüllt. Symmetrische Überlegungen ergeben sich für den Fall, dass $\bar{c}$ erfüllt ist, aber nicht c.

Wenn $c \wedge \bar{c}$ erfüllt ist, darf $I_l^{(t)} = V$ und $\bar{I}_l^{(t)} = \bar{V}$ gewählt werden, denn nach den Forderungen in 6-6 und 6-7 ist $V = \bar{V}$ und erfüllt auch die Determinierungsbedingung.

Dies zeigt, dass die rechten Seiten der Implikationen 6-6 und 6-7 die in den rechten Seiten der Implikationen 6-4 und 6-5 geforderten Existenzaussagen sicherstellen. Damit ist diese Bedin-

gung als Spezialfall der ersten Implementierbarkeitsbedingung identifiziert, die im vorigen Abschnitt 8.1.1 begründet wurde.

8.2 Hauptsatz

8.2.1 Grundidee des Hauptsatzes

Das Problem der kompositionalen vollständigen Verifikation wird zunächst zurückgeführt auf einen Hauptsatz darüber, wann – plakativ formuliert – eine Assertion α dadurch bewiesen werden kann, dass sie gleichzeitig vorausgesetzt wird. Der Hauptsatz zeigt, dass dies möglich ist, wenn die Assertion Aussagen über Signale macht, die vor der Untersuchung geschnitten werden. Abbildung 24 verdeutlicht die Situation. Beim Schneiden eines Signals wird das zu Grunde liegende Schaltungsmodell M verändert, indem die Quelle des Signals zu einem primären Ausgang wird und die Senken des Signals danach statt von der Quelle von einem primären Eingang versorgt werden. Das geschnittene Modell soll mit M' bezeichnet werden, die zu schneidenden Signale in M seien in der Menge $\mathbb{x}$ versammelt und die Menge der durch das Schneiden entstandenen, neuen Eingangssignale werden mit $\mathbb{x}'$, die der Ausgangssignale mit $\mathbb{x}°$ bezeichnet.

Die Grundidee des Beweises ist am einfachsten durch Widerlegung einer falschen Annahme zu verstehen. Unter der der Annahme, dass α auf dem ursprünglichen Modell M irgendwann nicht erfüllt ist, gibt es einen Eingabetrace I und einen zugehörigen Trace X auf den Signalen von $\mathbb{x}$ und einen frühesten Zeitpunkt t, an dem α durch X zum ersten Mal nicht erfüllt ist. Wenn man diesen Trace auf die neuen Eingabesignale überträgt und die Traces I und X' in das geschnittene Modell M' eingibt, entsteht an den neuen Ausgabesignalen der Trace $X°$, d.h. die entsprechende Übertragung des Traces X. $X°$ verletzt $\alpha°$ zum Zeitpunkt t. $X°$ kann aufgrund der Vorausssetzungen des Satzes nur dann $\alpha°$ verletzen, wenn α' durch X' verletzt wurde. Der Zeitpunkt, an dem α' durch X' verletzt wird, liegt vor t, weil die Werte von X' erst die Schaltung durchlaufen müssen. Dies steht aber im Widerspruch zu der Annahme, dass schon der erste Zeitpunkt betrachtet wurde, an dem die Assertion α verletzt war.

Um die Theorie auch auf Protokollbeschreibungen anzuwenden, muss α durch entsprechende Bedingungen instanziiert werden können. Da die Protokollbeschreibungen häufig Einschränkungen über das Eingabeverhalten in Abhängigkeit von den Ausgabewerten als Protokollconstraints zusichern, muss der Hauptsatz so weit gefasst werden, dass α' ein reaktiver Constraint sein kann, der die neuen Eingabesignale in Abhängigkeit von I, U, O und den neuen Ausgangssignalen beschränkt.

Der hier vorzustellende Ansatz soll außerdem mit kombinatorischen Pfaden zwischen den neuen Ein- und Ausgängen umgehen können. Die für synchrone Schaltungen üblichen Takte sind daher als Zeitbegriff nicht ausreichend, weil sich alle Signale längs eines kombinatorischen Pfades im selben Takt ändern könnte und eine Änderung an den neuen Ausgängen dann gleichzeitig mit der von ihr verursachten Änderung der neuen Eingänge stattfände. Damit wäre aber der Beweis nicht zu führen. Daher müssen die Takte weiter unterteilt werden.

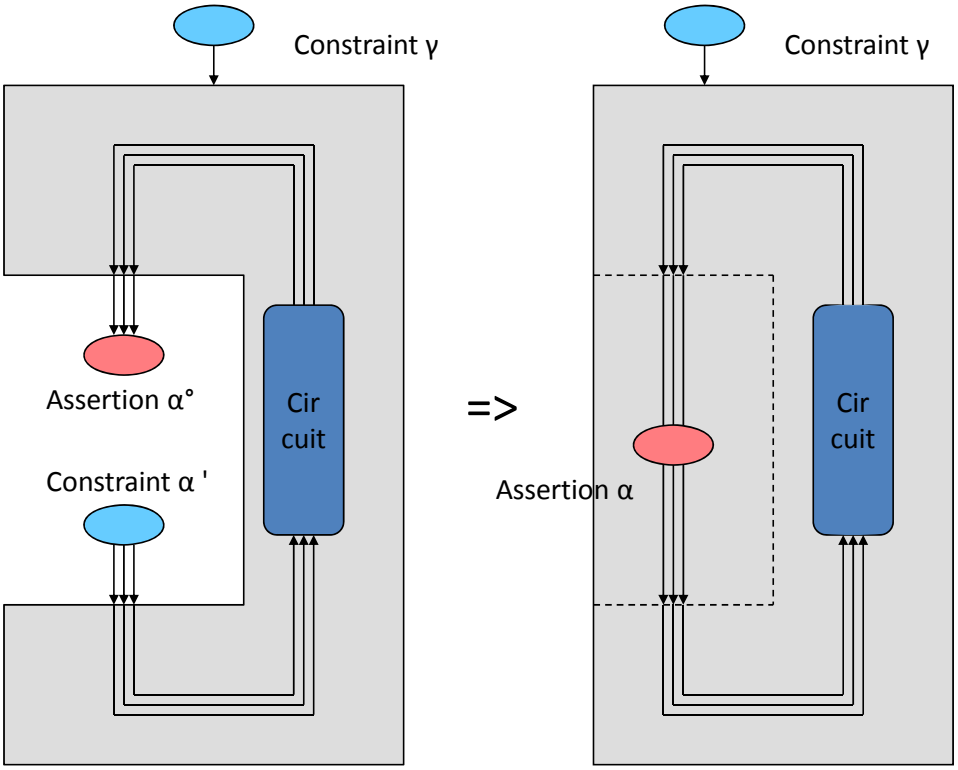

Abbildung 24: Grundidee des Ansatzes

Diese zusätzliche Unterteilung reflektiert die Kausalität der Schaltung. Bezüglich dieses verfeinerten Zeitbegriffes soll sich ein neues Ausgabesignal später ändern als alle Eingabesignale, von denen es kombinatorisch abhängt. Durch das verfeinerte Zeitmodell werden den Signalen von M' Kausalitätsindices entsprechend ihren Abhängigkeiten zugeordnet.

8.2.2 Strukturelle Kompatibilität

Für die Formulierung des Hauptsatzes sei zunächst erwähnt, dass sich der Begriff der Implementierbarkeit aus Abschnitt 6.2.2 wörtlich auf beliebige reaktive Constraints übertragen lässt. Der Fettdruck zum Herausheben der von dem implementierbaren Constraint beschränkten Traces wird beibehalten.

Auch die Forderung nach struktureller Kompatibilität zwischen Schaltung und Integrationsannahme wird auf einen beliebigen reaktiven Constraint übertragen. Dazu wird zunächst entsprechend Abschnitt 6.2.4 die Menge K^M der kombinatorischen Abhängigkeiten der Form $s \rightarrow u$ gebildet, wobei s ein Eingabesignal oder ein Registerausgang von M ist und u ein beliebiges Signal von M.

Sei $N^{\alpha'}$ eine Ersatzschaltung des Constraints $\alpha'(I, \boldsymbol{X'}, U, O, X°)$. Dann können die kombinatorischen Abhängigkeiten zwischen Eingangssignalen und Ausgangssignalen von $N^{\alpha'}$ bestimmt werden, und dann jedes x' und $x°$ durch die jeweiligen x substituiert werden. Dies ergebe die Menge $K^{N^{\alpha'}}$. α' und M heißen strukturell kompatibel, wenn es eine Ersatzschaltung $N^{\alpha'}$ gibt, sodass $K^M \cup K^{N^{\alpha'}}$ zyklenfrei ist.

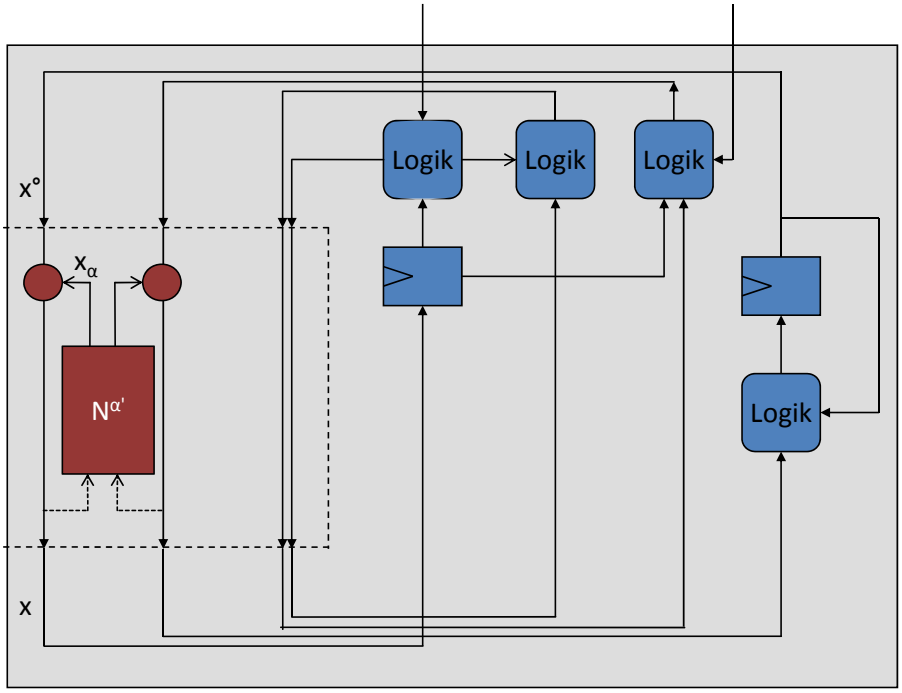

Abbildung 25: Aus Schaltung und Ersatzschaltung kombinierte Schaltung zur Bedingung über die Abwesenheit kombinatorischer Schleifen

Anschaulich gesprochen ergeben sich die kombinatorischen Abhängigkeiten aus einer Schaltung, die aus M und einer Ersatzschaltung von α' entsprechend der Abbildung 25 kombiniert wird. Die Art der Verknüpfung zwischen den Signalen aus $\mathbb{x}$ und den Ausgängen der Ersatzschaltung ist dabei nicht relevant, sie muss nur kombinatorisch sein.

8.2.3 Hauptsatz

Es sei eine Schaltung M und ein (nicht reaktiver) Constraint $\Gamma(\mathrm{I})$ über seine Eingabesignale gegeben. In M werde eine Menge $\mathbb{x}$ von internen Signalen geschnitten. Diese sollen durch den Trace X repräsentiert sein, sodass sich die Parameterliste $M(I,U,X,O)$ ergibt. Auf der durch das Schneiden erhaltenen Schaltung $M'(I,X',U,O,X°)$ werde eine Assertion $\alpha(I,X°,U,O,X')$ bewiesen unter Annahme der Constraints $\Gamma(\mathrm{I})$ und $\alpha(I,\boldsymbol{X'},U,O,X°)$, wobei letztere aus der Assertion durch Austausch der neuen Ein- und Ausgangssignale entsprechend der Parameterliste hervorgeht. Der Constraint $\alpha(I,\boldsymbol{X'},U,O,X°)$ sei implementierbar und beschränke nur X'. Zumindest eine seiner Ersatzschaltungen bilde mit $M'(I,X',U,O,X°)$ entsprechend Abschnitt 8.2.2 eine kombinierte Schaltung, die frei ist von kombinatorischen Schleifen. Dann gilt die Assertion $\alpha(I,X,U,O,X)$ auf der Schaltung $M(I,U,X,O) = M'(I,X,U,O,X)$.

8.2.4 Kausalitätsindex

Wie in Abschnitt 6.3.1 angedeutet, soll zum Beweis des Hauptsatzes ein problemangepasster Zeitbegriff entwickelt werden, der auf Kausalität beruht und die Kausalität der untersuchten Schaltung und der Assertion α widerspiegelt. Dieser Zeitbegriff basiert auf einem Kausalitätsindex, der jedem Signal s aus M' eine natürliche Zahl $k(s)$, den Kausalitätsindex, zuordnet. In dem verfeinerten Zeitbegriff werden Änderungen auf zwei Signalen s_1 und s_2 zum gleichen Takt zeitlich so auseinandergezogen, dass die Änderungen auf dem Signal mit dem kleineren

Kausalitätsindex vor den Änderungen auf dem Signal mit dem größeren Kausalitätsindex stattfinden.

Da vorausgesetzt ist, dass der Graph aus den Mengen $K^M \cup K^\alpha$ kombinatorischer Abhängigkeiten von M bzw. α entsprechend Abschnitt 6.2.4 keine Zyklen hat liefert der folgende rekursive Algorithmus deshalb einen Kausalitätsindex für jedes Signal:

Primäre Eingänge und Registerausgänge erhalten den Kausalitätsindex 0. Sei

$$Fanin_M(u) = \{i: i \to u \in K^M\}$$

und

$$Fanin_{N^{\alpha\prime}}(u) = \{i: i \to u \in K^{N^{\alpha\prime}}\}$$

Für Signale $x \in \mathbb{x}$ werde dann

$$k(x) = \max(\{k(v)|v \in Fanin_M(x)\} \cup \{k(v)|v \in Fanin_{N^{\alpha\prime}}(x)\}) + 1$$

berechnet. Für alle übrigen Signale u der Schaltung M wird

$$k(u) = max\{k(v)|v \in Fanin_M(u)\}$$

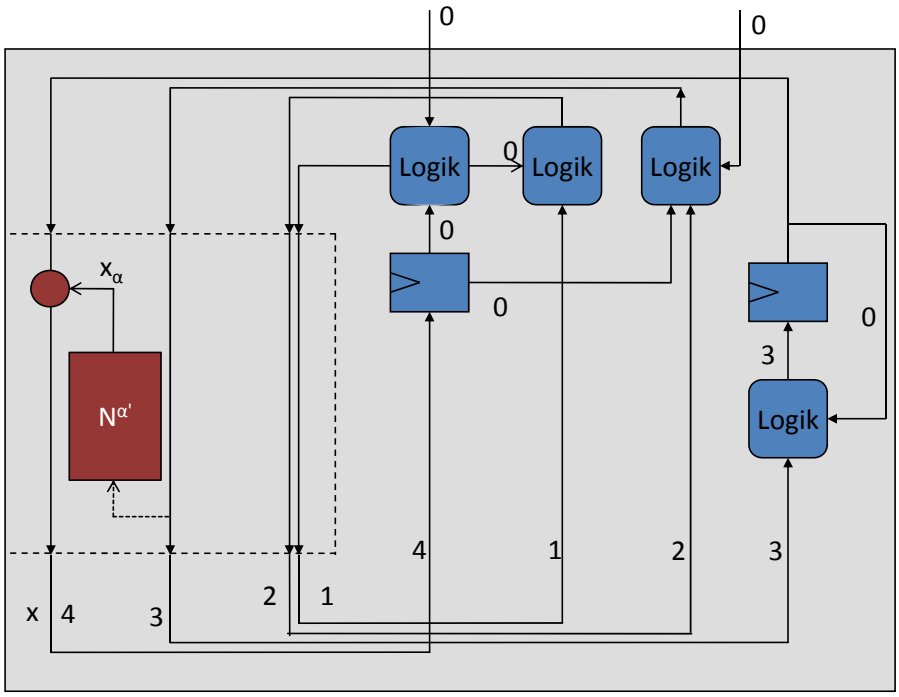

Abbildung 26: Beispiel für Berechnung von Kausalitätsindices: Ersatzschaltung $N^{\alpha\prime}$ für den implementierbare Constraint seien kombinatorisch, die Zahlen geben die Kausalitätsindices an (siehe Text)

definiert. Diese Definition ist konsistent, weil die kombinierte Schaltung voraussetzungsgemäß frei ist von kombinatorischen Schleifen.

Dies definiert die Kausalitätsindices für die ungeschnittene Schaltung M. Die Kausalitätsindices für M' sollen für gleiche Signale gleich sein. Durch das Schneiden neu entstandene Eingangs- und Ausgangssignale erhalten den Kausalitätsindex des ursprünglichen Signals.

Abbildung 26 illustriert eine kombinierte Schaltung und die Berechnung der Kausalitätsindices der Signale unter der Annahme einer kombinatorischen Ersatzschaltung.

Durch den Kausalitätsindex wird ein neuer Zeitbegriff definiert, in dem t die Takte durchnummeriert und k den Kausalitätsindex markiert. Eine Werteänderung zwischen dem Takt t und $t+1$ auf den Signalen von $\mathbb{x}$ verteilt sich dadurch ggf. auf mehrere kausale Zeitpunkte. Würden Schaltung und Bedingungen über Ein- und Ausgänge bezüglich des verfeinerten Zeitbegriffs als Automat dargestellt, ergäbe sich ein Moore-Automat und streng kausale Interfacebeschreibungen, auf die die Theorie in [Broy/Rumpe 2007] anwendbar ist.

Kausalitätszeitpunkte lassen sich vergleichen. Der Kausalitätszeitpunkt (t,k) liegt vor dem Kausalitätszeitpunkt (t',k'), wenn $t < t'$ oder $t = t'$ und $k < k'$. Es wird dann $(t,k) < (t',k')$ geschrieben. Entsprechend der Kausalitätszeitpunkte werden auch Traces verfeinert. Das Element $U_{(t,k)}$ eines Traces U enthält die Werte der Signale mit Kausalitätsindex k zum Takt t.

Zwei Traces U und V sind gleich bis zu einem Kausalitätszeitpunkt (t',k'), wenn $U_{(t,k)} = V_{(t,k)}$ gilt für alle $(t,k) \leq (t',k')$. Als Separierungszeitpunkt $sep(U,V)$ zweier Traces U und V wird der größte Kausalitätszeitpunkt (t',k') bezeichnet, für den $U_{(t',k')} = V_{(t',k')}$ gilt. Wenn U = V ist, ist der Separierungszeitpunkt unendlich. Für Separierungszeitpunkte ist die Rechenregel $sep(A,B) \geq min(sep(A,C), sep(B,C))$ nützlich.

8.2.5 Schreibweisen und Hilfssätze

Zunächst wird darauf Rücksicht genommen, dass I während der gesamten Betrachtung als beliebig aber fest angesehen werden kann, während die neuen Eingabesignale $\mathbb{x}'$ mit unterschiedlichen Traces belegt werden. Die Zusammengehörigkeit zwischen einem Trace X' und den damit in Verbindung stehenden Traces der internen und primären Ausgangssignale werde dadurch zum Ausdruck gebracht, dass X^u als Trace (I,U,O) aller Signale der geschnittenen Schaltung M' zusammengefasst wird, ferner werde $X^{ou} = (X°, X^u)$ zusammengefasst.

Vereinfachend wird $M'(X', X^{ou})$ oder $M'(X', X°, X^u)$ statt $M'(I, X', U, O, X°)$ und $\alpha(\boldsymbol{X}', X^{ou})$ oder $\alpha(\boldsymbol{X}', X°, X^u)$ statt $\alpha(I, \boldsymbol{X}', U, O, X°)$ geschrieben. Um die Rolle von $\alpha(\boldsymbol{X}', X°, X^u)$ als Constraint zu betonen, der die Werte X' auf neuen Einängen in Abhängigkeit der Werte $X°$ und X^u und auf neuen Ausgängen beschränkt, wird im Folgenden $\alpha'(\boldsymbol{X}', X^{ou})$ oder $\alpha'(\boldsymbol{X}', X°, X^u)$ geschrieben. Um die Rolle von $\alpha(X°, X', X^u)$ als Assertion zu betonen, die eine Aussage über neue Ausgangssignale in Abhängigkeit von neuen Eingabesignalen und sonstigen Signalen macht, wird im Folgenden $\alpha°(X°, X', X^u)$ geschrieben. In beiden Fällen handelt es sich aber um den selben Ausdruck, nur dass die Rolle von neuen Ein- und Ausgängen vertauscht werden.

Lemma 1: Es seien Traces X' und X^{ou} der neuen Ein- und Ausgänge gegeben, die einen implementierbaren Constraint $\alpha'(\boldsymbol{X}', X^{ou})$ erfüllen. Dann gibt es für jeden Trace Y^{ou} der neuen Ausgänge und sonstigen Signale einen Trace Y' der neuen Eingänge mit $\alpha'(\boldsymbol{Y}', Y^{ou})$ und

$sep(X', Y') > sep\ (X^{ou}, Y^{ou})$. Anschaulich besagt dieser Satz, dass der Constraint α' Änderungen an den neuen Ausgangssignalen oder an den sonstigen Signalen bezüglich des gewählten verfeinerten Zeitbegriffes verzögert an die neuen Eingaben weitergibt.

Beweis: Da α' implementierbar ist, gibt es ein F, sodass $N^{\alpha'}(X^{ou}, F, X')$. Sei Y' bestimmt durch $N^{\alpha'}(Y^{ou}, F, Y')$. Dadurch unterscheiden sich X' und Y' frühestes im ersten Takt, in dem sich X^{ou} und Y^{ou} unterscheiden. Für jeden kombinatorischen Pfad durch $N^{\alpha'}$ von einem neuen Ausgangssignal a zu einem neuen Eingangssignal b gilt jedoch $k(a) < k(b)$. Wenn X' und Y' gleich sind für alle Eingangssignale mit Indices $\leq k$, so sind die Ausgangssignale gleich bis zu größeren Indices. Das begründet die Ungleichung.

Lemma 2: Zu Traces X' und Y' werden durch $M'(X', X°, X^u)$ und $M'(Y', Y°, Y^u)$ zwei Traces $X°$ und $Y°$ bestimmt. Es ist $sep(X°, Y°) > sep(X', Y')$ und $sep(X^u, Y^u) \geq sep(X', Y')$. Anschaulich besagt dieses Lemma, dass Änderungen an den neuen Eingängen zu Änderungen an den sonstigen Signalen führen können, die auch bezüglich des verfeinerten Zeitbegriffes gleichzeitig stattfinden. Hingegen führen die Änderungen zu späteren Änderungen an den neuen Ausgängen.

Insofern ist die Aussage dieses Lemmas eine direkte Konsequenz der Definition des Kausalitätsindexes.

Lemma 3: Zu einem Trace X' seien zwei Traces X_1^{ou} und X_2^{ou} gegeben, von denen der eine $\alpha'(X', X_1^{ou})$ und der andere $M'(X', X_2^{ou})$ erfülle. Dann gibt es Traces Y' und Y^{ou}, die $M'(Y', Y^{ou})$ und $\alpha'(Y', Y^{ou})$ erfüllen, und für die $sep(X', Y') > sep(X_1^{ou}, X_2^{ou})$ gilt.

Beweis: Y' wird induktiv erzeugt, indem Folgen Y_n' und Y_n^{ou} von Traces erzeugt werden, für die jeweils $\alpha'(Y_n', Y_n^{ou})$ gilt. Die Betrachtung startet für $n = 0$ mit $Y_0' = X'$ und $Y_0^{ou} = X_1^{ou}$.

Schritt von n auf $n + 1$: Y_{n+1}^{ou} werde durch $M'(Y_n', Y_{n+1}^{ou})$ bestimmt. Sei $(t_n, k_n) = sep(Y_n^{ou}, Y_{n+1}^{ou})$. Nach Lemma 1 gibt es ein Y_{n+1}', das $\alpha'(Y_{n+1}', Y_{n+1}^{ou})$ erfüllt, und für das $sep(Y_n',\ Y_{n+1}') > sep(Y_n^{ou},\ Y_{n+1}^{ou})$ gilt. Wenn Y_{n+2}^{ou} durch $M'(Y_{n+1}', Y_{n+2}^{ou})$ bestimmt wird, erfährt M' bis $sep(Y_n',\ Y_{n+1}')$ die selben Eingaben wie bei der Bestimmung von Y_{n+1}^{ou}. Damit lässt sich unter Verwendung von Lemma 2 herleiten

$$(t_{n+1}, k_{n+1}) = sep(Y_{n+1}^{ou}, Y_{n+2}^{ou}) \geq sep(Y_n', Y_{n+1}') > sep(Y_n^{ou}, Y_{n+1}^{ou}) = (t_n, k_n)$$

(t_n, k_n) ist damit eine streng monoton wachsende Folge, die diejenigen Elemente der Traces Y_n' angibt, die sich für größere n nicht mehr ändern. Y' kann also so gewählt werden, dass es nur aus solchen Elementen besteht, die sich nicht mehr ändern. Da der Kausalitätsindex beschränkt ist, wächst t_n über alle Grenzen. Zur Bestimmung des Wertes von Y' zum Taktzeitpunkt T muss also nur ein n bestimmt werden, das so groß ist, dass $t_n > T$. Der Wert von Y_n' zum Taktzeitpunkt T ist dann der gewünschte und Y^{ou} ergibt sich daraus durch $M'(Y', Y^{ou})$. Für alle n ist dadurch $sep(Y', Y_{n+1}') \geq sep(Y_{n+1}', Y_{n+2}') > (t_n, k_n)$. Wegen Lemma 2 ist daher $sep(Y^{ou}, Y_{n+2}^{ou}) > (t_n, k_n)$.

Dass auch $\alpha'(Y', Y^{ou})$ gilt, ist folgendermaßen einzusehen: Weil α' eine Safetybedingung ist, würde sich eine Verletzung schon auf einem endlichen Anfangsstück von Y' und Y^{ou} ergeben, das bei geeigneter Wahl von n mit geeigneten Anfangsstücken von Y_n' und Y_n^{ou} übereinstimmt, für die aber $\alpha'(Y_n', Y_n^{ou})$ gilt. Daher kann $\alpha'(Y', Y^{ou})$ nicht verletzt sein.

Aus den Definitionen ergibt sich $Y'_0 = X'$, $Y^{ou}_0 = X^{ou}_1$, $Y^{ou}_1 = X^{ou}_2$, $(t_0, k_0) = (t, k)$. Entsprechend ist

$$sep(X', Y') \geq min\big(sep(X', Y'_1), sep(Y', Y'_1)\big) = min\big(sep(Y'_0, Y'_1), sep(Y', Y'_1)\big) > (t_0, k_0)$$
$$= sep(Y^{ou}_0, Y^{ou}_1) = sep(X^{ou}_1, X^{ou}_2)$$

Lemma 4: Eine besondere Konsequenz von Lemma 3 ist, dass es zu jedem implementierbaren Constraint α' und zu jedem Eingabetrace I mit $\Gamma(I)$ Traces Y' und Y^{ou} gibt, die $\alpha'(Y', Y^{ou})$ und $M'(Y', Y^{ou})$ erfüllen.

8.2.6 Hauptbeweis

Der Beweis wird wiederum durch rekursive Definition zweier Folgen von Traces X'_n und X^{ou}_n geführt. Aufgrund von Lemma 4 gibt es zunächst X'_0 und X^{ou}_0, für die $M'(X'_0, X^{ou}_0)$ und $\alpha'(X'_0, X^{ou}_0)$ gilt. Aufgrund der Voraussetzung gilt dann auch $\alpha°(X^o_0, X'_0, X^u_0)$.

Es seien nun bereits Traces X'_n und X^{ou}_n gefunden, die $M'(X'_n, X^{ou}_n)$, $\alpha'(X'_n, X^{ou}_n)$, und $\alpha°(X^o_n, X'_n, X^u_n)$ erfüllen. Sei $\hat{X}'_n$ der Trace X'_n, in dem alle neuen Eingänge durch Ausgänge ersetzt worden sind. Analog sei $\hat{X}^o_n$ der Trace X^o_n, in dem alle neuen Ausgänge durch die zugehörigen neuen Eingänge ersetzt sind.

Da $\alpha°(X^o_n, X'_n, X^u_n)$ gilt, gilt auch $\alpha'\big(\hat{X}^o_n, \hat{X}'_n, X^u_n\big)$. Nun werde Lemma 1 auf die Traces X^o_n, $X^u_n, \hat{X}'_n$ und $\hat{X}^o_n$ und $\alpha'\big(\hat{X}^o_n, \hat{X}'_n, X^u_n\big)$ angewandt. Das zeigt, dass es einen neuen Trace Y'_{n+1} gibt, der $\alpha'(Y'_{n+1}, X^o_n, X^u_n)$ erfüllt, und für den $sep\big(Y'_{n+1}, \hat{X}^o_n\big) > sep\big(X^o_n, \hat{X}'_n\big) = sep(\hat{X}^o_n, X'_n)$.

Wenn der neue Trace Y'_{n+1} in die geschnittene Schaltung eingegeben wird, wird vermittels $M'(Y'_{n+1}, Y^{ou}_{n+1})$ ein neuer Trace der neuen Ausgabesignale erzeugt. Durch Anwendung von Lemma 3 auf $\alpha'(Y'_{n+1}, X^{ou}_n)$ und $M'(Y'_{n+1}, Y^{ou}_{n+1})$ ergibt sich die Existenz von Traces X'_{n+1} und X^{ou}_{n+1}, für die $\alpha'(X'_{n+1}, X^{ou}_{n+1})$ und $M'(X'_{n+1}, X^{ou}_{n+1})$ sowie $sep(X'_{n+1}, Y'_{n+1}) > sep(X^{ou}_n, Y^{ou}_{n+1})$ gilt. Weiter kann mit Lemma 2 abgeschätzt werden

$$sep(X^{ou}_n, Y^{ou}_{n+1}) > sep(X'_n, Y'_{n+1}) \geq min\left(sep\big(X'_n, \hat{X}^o_n\big), sep\big(Y'_{n+1}, \hat{X}^o_n\big)\right) \geq sep(\hat{X}^o_n, X'_n)$$

Aus der Voraussetzung des Hauptsatzes folgt dann auch $\alpha°(X^o_{n+1}, X'_{n+1}, X^u_{n+1})$.

Mit der Rechenregel $sep(A, B) \geq min(sep(A, C), sep(B, C))$ läßt sich weiter argumentieren, dass $sep\big(X'_{n+1}, \hat{X}^o_{n+1}\big) \geq \min(sep(X^o_{n+1}, X^o_n), sep(X'_{n+1}, \hat{X}^o_n))$. Dabei gilt $sep(X^o_{n+1}, X^o_n) > sep(X'_{n+1}, X'_n) \geq \min\left(sep(X'_{n+1}, Y'_{n+1}), sep\big(Y'_{n+1}, \hat{X}^o_n\big), sep\big(X'_n, \hat{X}^o_n\big)\right) = sep\big(X'_n, \hat{X}^o_n\big)$ und $sep\big(X'_{n+1}, \hat{X}^o_n\big) \geq \min\left(sep(X'_{n+1}, Y'_{n+1}), sep\big(Y'_{n+1}, \hat{X}^o_n\big)\right) > sep\big(X'_n, \hat{X}^o_n\big)$.
Also ist $sep\big(X'_{n+1}, \hat{X}^o_{n+1}\big) > sep\big(X'_n, \hat{X}^o_n\big)$.

Die Folge von Kausalitätszeitpunkten $(t_n, k_n) = sep\big(X'_n, \hat{X}^o_n\big)$ wächst daher über alle Grenzen. Daher werde X' in derselben Weise wie in Lemma 3 definiert: Um den Wert von X' zum Taktzeitpunkt T zu definieren, wird ein n mit $T < t_n$ gesucht und der Wert von X' zu diesem Zeitpunkt durch X'_n definiert. $X°$ ergebe sich durch Eingabe von X' und I in die geschnittene Schaltung, d.h. durch $M'(I, X', U, O, X°)$. Die Gültigkeit von $\alpha'(I, X', U, O, X°)$ zeigt sich wie im Beweis von Lemma 3. Auf die selbe Weise zeigt sich, dass X' und $X°$ gleich sind modulo korrespondierender neuer Ein- und Ausgänge. Damit wurde zu dem Trace I der primären

Eingänge der Gesamtschaltung ein Trace X der Signale der Menge $\mathbb{x}$ gefunden, mit dem $\alpha(I, X, U, O, X)$ erfüllt wird. Da dies für jeden Trace I gilt, ist der Satz damit bewiesen.

8.2.7 Erweiterter Constraint Γ

Der Hauptsatz aus Abschnitt 8.2.2 ist mit einem nicht-reaktiven Constraint $\Gamma(I)$ formuliert. Das ist meist unzureichend, weil die meisten Constraints reaktiv sind. Nachfolgend soll deswegen der folgende Satz bewiesen werden:

Gegeben sei eine Schaltung $M'(I, X', U, O, X°)$ mit Eingangssignalen I und X', Ausgabesignalen O und $X°$, sowie internen Signalen U. Die X' und $X°$ zugrundeliegenden Signale sollen gleiche Typen haben. Gegeben seien zwei Constraints $\Gamma(I, U, O, X°)$ und $\alpha(I, X', U, O, X°)$. Γ sei implementierbar und beschränke I, α sei implementierbar und beschränke I und X'. Ferner gelte

$$\Gamma(I, U, O, X°) \Rightarrow \bigvee_{X'} \alpha(I, X'U, O, X°)$$

Unter den Constraints $\Gamma(I, U, O, X°), \Gamma(J, U, O, X')$ und $\alpha(I, X', U, O, X°)$ lasse sich die Assertion $\alpha(J, X°, U, O, X')$ auf $M'(I, X', U, O, X°)$ beweisen, d.h. es gelte

$$\Gamma(I, U, O, X°) \wedge \Gamma(J, U, O, X') \wedge \alpha(I, X', U, O, X°) \wedge M'(I, X', U, O, X°) \Rightarrow \alpha(J, X°, U, O, X')$$

Sei $M(I, U, X, O) = M'(I, X, U, O, X)$ die Schaltung, die aus M' durch Verbinden der Signale X' und $X°$ entsteht.

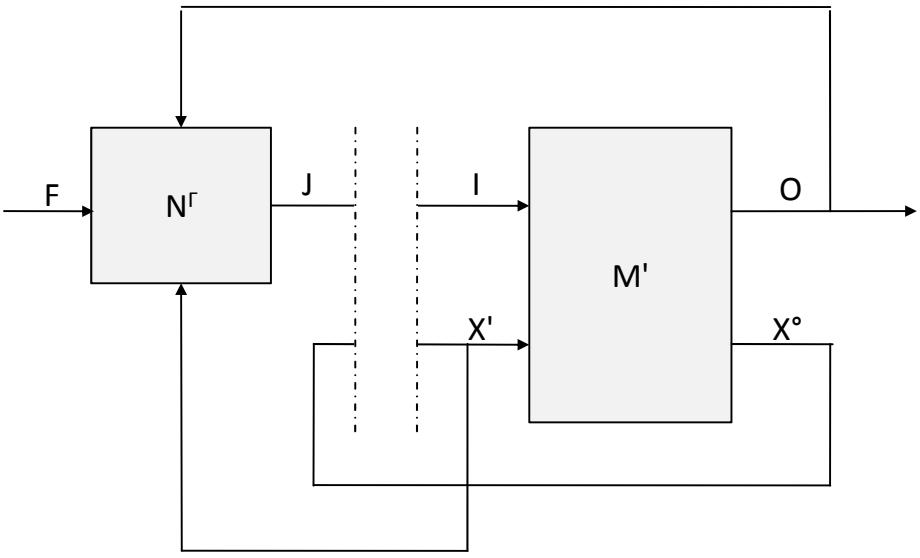

Abbildung 27: Zum Beweis der Erweiterung um einen reaktiven Constraint

Die Mengen kombinatorischer Abhängigkeiten K^{Γ}, K^{α} und K^{M} entsprechend Abschnitt 6.2.4 sollen sich zu einem zyklenfreien Graphen vereinigen.

Dann gilt auf $M(I, U, X, O)$ unter dem Constraint $\Gamma(I, U, O, X)$ die Assertion $\alpha(I, X, U, O, X)$, d.h.

$$\bigwedge_{I,U,O,X} \Gamma(I,U,O,X) \wedge M(I,U,X,O) \Rightarrow \alpha(I,X,U,O,X)$$

Zum Beweis sei $N^{\Gamma}(U,O,X,F,I)$ die Ersatzschaltung von $\Gamma(I,U,O,X)$ mit freien Eingangssignalen F und Eingangssignalen U,O,X, die aus der Schaltung abgegriffen werden, und mit dem primären Ausgang I, mit dem M versorgt wird. M und N^{Γ} seien so verschaltet, wie es Abbildung 27 zeigt.

Diese Schaltung hat als primäre Eingangssignale die Signale von F und als geschnittene Signale die Signale von I und X. F ist nicht durch irgendeinen Constraint beschränkt.

Die Schaltung aus M' und N^{Γ} hat als primäre Eingangssignale die Signale von F und als geschnittene Signale die Signale von I und X. F ist überhaupt nicht beschränkt.

Die Signale von I und X seien an der angegebenen Stelle geschnitten. Auf dem aus M' und N^{Γ} bestehenden Modell gilt nun

$$\begin{aligned}\bigwedge_{F,I,J,U,O,X',X^{\circ}} & \Gamma(I,U,O,X^{\circ}) \wedge \alpha(I,X',U,O,X^{\circ}) \wedge M'(I,X',U,O,X^{\circ}) \wedge N^{\Gamma}(U,O,X',F,J) \\ & = \Gamma(I,U,O,X^{\circ}) \wedge (X',U,O,X^{\circ}) \wedge M'(I,X',U,O,X^{\circ}) \wedge \Gamma(J,U,O,X') \\ & \Rightarrow \alpha(J,X^{\circ},U,O,X') \wedge \Gamma(J,U,O,X')\end{aligned}$$

Nun ist $\alpha(I,X',U,O,X^{\circ}) \wedge \Gamma(I,U,O,X^{\circ})$ ein implementierbarer Constraint für I und X'. Dies kann folgendermaßen eingesehen werden: Da $\alpha(I,X',U,O,X^{\circ})$ ein implementierbarer Constraint für I und X' ist, gibt es eine Ersatzschaltung N^{α}, deren Eingangssignale die Signale von U,O,X° sind, und die zusätzliche freie Eingabesignale habe, die mit Q bezeichnet seien. Die Zustandssignale von N^{α} seien durch den Trace S gegeben. $\Delta\left(U^{(t)},O^{(t)},X^{\circ(t)},Q^{(t)},S^{(t)}\right)$ $\Delta(u,o,x^{\circ},q,s)$ bestimme den Folgezustand, $\Lambda_{\mathrm{X}'}(u,o,x^{\circ},q,s)$ den Wert von $\mathrm{X}'^{(\mathrm{t})}$ und $\Lambda_{\mathrm{I}}(u,o,x^{\circ},q,s)$den Wert von $I^{(t)}$.

Vermittels $N^{\Gamma}(U,O,X^{\circ},F,I)$ sei bereits ein Trace I erzeugt worden. Es soll nun aus N^{α} eine Schaltung konstruiert werden, die I einlesen kann und alle X' ausgibt, für die $\alpha(I,X',U,O,X^{\circ}) \wedge \Gamma(I,U,O,X^{\circ})$ gilt. Zur Bestimmung dieser Schaltung werden zunächst alle Werte durchnummeriert, die q annehmen kann. Dies gibt die endliche Folge $q_0,q_1,q_2,\dots,q_n$ Dann werden alle Folgezustand- und Ausgabelogiken von N^{α} n-mal vervielfältigt und ihr q-Eingang durch alle q_k substituiert. Als nächstes wird das kleinste k bestimmt, mit dem $\Lambda_{\mathrm{I}}(u,o,x^{\circ},q_k,s) = I^{(t)}$ gilt. Ein solches existiert wegen der Voraussetzung über die Relation zwischen Γ und α. Dieses kleinste k wird mit K bezeichnet. Anschließend wird eine Schaltung gebaut, die zu den Eingaben I,U,O,X° und dem neuen freien Eingabesignal Q' alle möglichen Traces X' ausgibt. Diese Schaltung ergibt sich aus dem Pseudo-Code

```
if q' = q_0 ∧ Λ_I(u,o,x°,q_0,s) = I^(t)
            then X'^(t) = Λ_X'(u,o,x°,q_0,s) and S^(t+1) = Δ(u,o,x°,q_0,s)
else if q' = q_1 ∧ Λ_I(u,o,x°,q_1,s) = I^(t)
            then X'^(t) = Λ_X'(u,o,x°,q_1,s) and S^(t+1) = Δ(u,o,x°,q_1,s)
else if q' = q_2 ∧ Λ_I(u,o,x°,q_2,s) = I^(t)
            then X'^(t) = Λ_X'(u,o,x°,q_2,s) and S^(t+1) = Δ(u,o,x°,q_2,s)
…
else if q' = q_n ∧ Λ_I(u,o,x°,q_n,s) = I^(t)
            then X'^(t) = Λ_X'(u,o,x°,q_n,s) and S^(t+1) = Δ(u,o,x°,q_n,s)
```

```
else X'(t) = Λ_X'(u, o, x°, q_K, s) and S(t+1) = Δ(u, o, x°, q_K, s)
end if;
```

Für alle Traces Q, für die das ursprüngliche N^{α} Traces I, X' bestimmte, für die auch $\Gamma(I, U, O, X°)$ galt, liefert die neue Schaltung weiterhin die selben X'. Für solche Traces Q, bei denen N^{α} Traces I, X' bestimmte, für die $\Gamma(I, U, O, X°)$ nicht galt, verhält die neue Schaltung sich, als ob Q geeignet abgewandelt wurde. Damit leistet die neue Schaltung das gewünschte. Damit ist der Hauptsatz anwendbar und sichert auf dem aus M' und N^{Γ} bestehenden ungeschnittenen Modell, in dem I und J identifiziert sind, die Beziehung

$$M'(I, X, U, O, X) \wedge N^{\Gamma}(U, O, X, F, I) \Rightarrow \alpha(I, X, U, O, X) \wedge \Gamma(I, U, O, X)$$

zu. Da F zu gegebenen Traces U, O, X alle I mit $\Gamma(I, U, O, X)$ ausleuchtet und $M'(I, X, U, O, X) = M(I, U, X, O)$ ist, gilt daher auch

$$\Gamma(I, U, O, X) \wedge M(I, U, X, O) \Rightarrow \alpha(I, X, U, O, X)$$

und damit ist der Beweis erbracht.

8.2.8 Bemerkung zum Schließen von Schnitten

Ein Constraint $\alpha'(X', X°) = (X' = X°)$ für Traces einiger neuer Eingangssignale und der zugehörigen Ausgangssignale ist zwar implementierbar, kann aber nicht Constraint des Hauptsatzes sein, weil dann die kombinierte Schaltung kombinatorische Schleifen enthielte.

Es kann aber ohne Beschränkung der Allgemeinheit angenommen werden, dass jedes der durch neuen Ausgangssignale des Traces $X°$ von jeweils einem eindeutig zugeordneten internen Signal getrieben wird. Sei X_{int} der zugehörige Trace. Der Constraint $\alpha'(X', X_{int}) = (X' = X_{int})$ ist wiederum implementierbar. Aber er führt auch nicht zu kombinatorischen Schleifen der kombinierten Schaltung. Die zugehörige Assertion $\alpha°(X°, X_{int}) = (X° = X_{int})$ gilt trivialerweise. Daher kann der Hauptsatz aus Abschnitt 8.2.2 angewandt werden und beweist die nicht sonderlich erhellende Assertion $\alpha(X, X) = (X = X)$ über das ungeschnittene Modell M. Das Wesentliche an dieser Beobachtung ist jedoch, dass $\alpha'(X', X_{int})$ für die Verifikation auf dem geschnittenen Modell M' vorausgesetzt werden darf, und dass diese Voraussetzung den Schnitt durch alle Signale von X' wieder schließt.

In der Tat lassen sich aufgrund dieser Überlegung Schnitte in Signalen wieder schließen, aber nur dann, wenn der Constraint α' keine weiteren Einschränkungen über diese Signale fordert. Denn wenn der Constraint außer $\alpha'_1(X', X_{int}) = (X' = X_{int})$ über die selben Signale in X'weitere Einschränkungen $\alpha'_2(X', X°, X^u)$ machte, dann wäre der gesamte Constraint $\alpha' = \alpha'_1 \wedge \alpha'_2$ nicht mehr implementierbar und daher der Hauptsatz nicht mehr anwendbar. Es können also nur solche geschnittenen Signale wieder zusammengefügt werden, über die sonst keine weiteren Einschränkungen gemacht wurden.

8.3 Anwendungen des Hauptsatzes

8.3.1 Zerlegen vollständiger Verifikationen

Abschnitt 6.4.5 behandelt das Zerlegen einer vollständigen Verifikation in Cluster. Es wird herausgearbeitet, dass die Cluster auf einem Modell verifiziert werden müssen, in dem sie nicht zyklisch voneinander abhängen dürfen. Zyklische Abhängigkeiten zwischen Clustern

der eigentlich zu untersuchenden Schaltung M müssen durch Schneiden von Signalen aufgebrochen werden, sodass ein Modell M' mit geschnittenen Signalen und einem azyklischen Clustergraph entsteht. Dieses Modell sei aus den Clustern M_j gebildet.

In M' gibt es folgede Kategorien von Signalen: Primäre Ein- bzw. Ausgänge von M mit dem Trace I bzw. O, durch Schneiden entstandene neue Ein- bzw. Ausgänge mit dem Trace X' bzw. X^o, interne Signale, die lokale Eingänge einem Cluster sind, mit dem Trace Y, und interne Signale der Cluster mit dem Trace U.

Zu M gebe es eine globale Integrationsbedingung aus einer globalen Integrationsannahme GA und einer globalen Integrationszusicherung GC. mit einer Beschreibung darüber, unter welchen Bedingungen die ungeschnittene Gesamtschaltung M vollständig verifiziert sein soll. Für jeden Cluster M_j gibt es eine lokale Integrationsannahme IA_j und eine lokale Integrationszusicherung IC_j, sowie eine bezüglich der jeweiligen Integrationsbedingung vollständige Eigen-

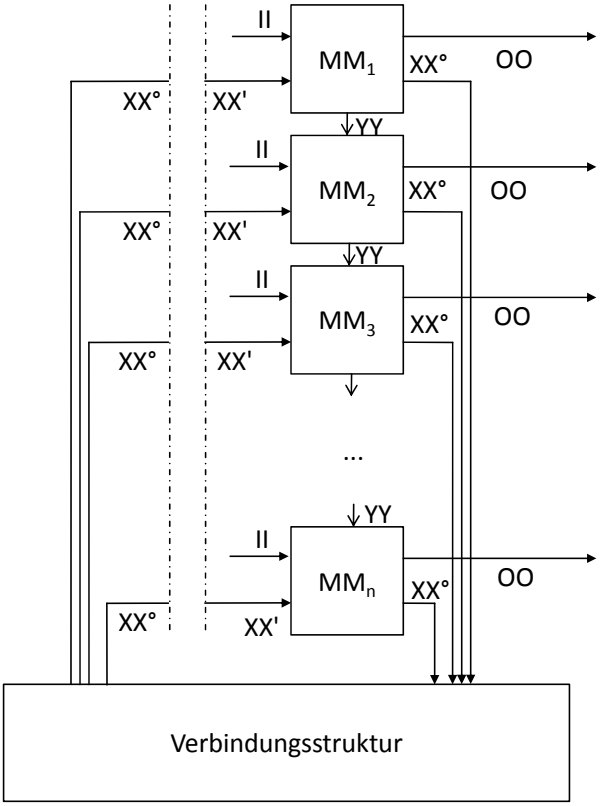

Abbildung 28: Zum Beweis der Zerlegung vollständiger Verifikationen

schaftsmenge $\mathcal{P}_j$. Es soll nachgewiesen werden, dass $\bigcup \mathcal{P}_j$ eine im Sinne von 5-1 bzw. 6-1 vollständige Eigenschaftsmenge mit der Integrationsannahme GA und der Integrationszusicherung GC, falls die Voraussetzungen 1 bis 5 aus Abschnitt 6.4.5 erfüllt sind.

Um zu beweisen, dass M durch die vollständigen Verifikationen der M_j selbst vollständig verifiziert ist, werden zwei beliebige Schaltungen M und $\overline{M}$ angenommen, die beide alle Eigenschaftssätze erfüllen, und die die Strukturbedingungen aus Voraussetzung 1 erfüllen.

Diese beiden Schaltungen bestehen aus Blöcken M_j und $\bar{M}_j$. Ihre Verbindungsstruktur ist gleich. Abbildung 28 verdeutlicht die Situation, wobei alle Objekte als Paare entsprechend Abschnitt 6.2.1 aus den entsprechenden Objekten aus M und $\bar{M}$ verstanden werden sollen. Durch den angedeuteten Schnitt der Verbindungssignale X zerfallen M und $\bar{M}$ wieder in die Cluster M_j, die aber durch die Signale von YY weiterhin verbunden bleiben. Die Cluster sind vollständig verifiziert, d.h. sie genügen

$$\bigwedge_j IA_j(II, XX', YY, UU, OO, XX^\circ) \wedge PP_j(II, XX', YY, UU, OO, XX^\circ) \Rightarrow IC_j(II, XX', YY, UU, OO, XX^\circ)$$

Im Folgenden werde die Parameterliste $(II, XX', YY, UU, OO, XX^\circ)$ nicht mehr geschrieben. Bei Hinzunahme von $GA(II, UU, XX^\circ, YY, OO)$ und $\widehat{GA} = GA(JJ, UU, XX', YY, OO)$ wird dieser Ausdruck zu

$$\bigwedge_j \left(GA \wedge \widehat{GA} \wedge IA_j \wedge PP_j \Rightarrow GA \wedge \widehat{GA} \wedge IC_j\right)$$

Der zyklenfreie Clustergraph sichert aufgrund von Voraussetzung 4 analog zu den Verhältnissen in Abschnitt 6.1.4 zu, dass die Teile $IA_j^{int}(YY, UU, OO)$ der lokalen Integrationsannahmen durch den oder die jeweiligen Vorgängercluster gerechtfertigt werden. Daher gilt

$$\bigwedge_j \left(GA \wedge \widehat{GA} \wedge IA_j^{prim} \wedge PP_j \Rightarrow GA \wedge \widehat{GA} \wedge IC_j\right)$$

Durch Umgruppieren der Konjunktionen und aufgrund der Voraussetzung 4 ist dies gleichbedeutend mit

$$\left(GA \wedge \widehat{GA} \wedge \bigwedge_j IA_j^{prim} \wedge \bigwedge_j PP_j\right) \Rightarrow \left(GA \wedge \widehat{GA} \wedge \bigwedge_j IC_j\right) \Rightarrow \bigwedge_j IA_j(JJ, XX^\circ, UU, OO, XX') \Rightarrow \bigwedge_j IA_j^{prim}(JJ, XX^\circ, UU, OO, XX')$$

Die letzte Umformung ergibt sich aus Voraussetzung 1.

Dies wird wegen der in Voraussetzung 1 verlangten Struktur zu

$$\bigwedge_{II,UU,OO,XX',XX^\circ} GA(II, UU\ XX^\circ, YY, OO,) \wedge GA(JJ, UU, XX', YY, OO) \wedge \bigwedge_j IA_j^{int}(YY, UU, OO) \wedge \bigwedge_j IA_j^{prim}(II, XX', YY, UU, OO, XX^\circ) \wedge \bigwedge_j PP_j(II, XX', YY, UU, OO, XX^\circ) \Rightarrow \bigwedge_j IA_j^{int}(YY, UU, OO) \wedge \bigwedge_j IA_j^{prim}(JJ, XX^\circ, YY, UU, OO, XX')$$

Auf diese Situation ist wegen der Voraussetzungen 1 und 3 der Satz aus Abschnitt 8.2.7 anwendbar. Daher gilt auf dem ungeschnittenen Paar MM von Schaltungen

$$\left(GA(II, UU\ XX, YY, OO) \wedge \bigwedge_j PP_j(II, XX, YY, UU, OO, XX) \right) \Rightarrow \bigwedge_j IA_j^{prim}(II, XX, YY, UU, OO, XX)$$

Weiterhin ist für alle Quellen M_j des Clustergraphen von M'

$$IA_j(II, XX, YY, UU, OO, XX) = IA_j^{prim}(II, XX, YY, UU, OO, XX)$$

und wegen

$$GA(II, UU\ XX, YY, OO) \wedge IA_j(II, XX, YY, UU, OO, XX) \wedge PP_j(II, XX, YY, UU, OO, XX) \Rightarrow IC_j(II, XX, YY, UU, OO, XX)$$

gilt auch $IA_k^{int}(YY, UU, OO)$ für alle Cluster M_k, die nur von Quellen dieses Graphen abhängen. Für diese M_k gilt damit $IA_k(II, XX, YY, UU, OO, XX)$ und die obige Argumentation läßt sich aufgrund der Zyklenfreiheit des Clustergraphen von M' solange wiederholen, bis IA_j für alle Cluster nachgewiesen ist, und sich dann aufgrund der Voraussetzung 5 ergibt

$$\begin{aligned} &\left(GA(II, UU\ XX, YY, OO) \wedge \bigwedge_j PP_j(II, XX, YY, UU, OO, XX) \right) \\ &= \left(GA(II, UU\ XX, YY, OO) \right. \\ &\left. \wedge \bigwedge_j \left(IA_j(II, XX, YY, UU, OO, XX) \wedge PP_j(II, XX, YY, UU, OO, XX) \right) \right) \\ &\Rightarrow \left(GA(II, UU\ XX, YY, OO) \wedge \bigwedge_j IC_j(II, XX, YY, UU, OO, XX) \right) \\ &\Rightarrow GC(II, UU\ XX, YY, OO) \end{aligned}$$

Damit ist gezeigt, dass $\bigcup \mathcal{P}_j$ eine im Sinne von 5-1 bzw. 6-1 vollständige Eigenschaftsmenge mit der Integrationsannahme GA und der Integrationszusicherung GC ist.

8.3.2 Zusammensetzen vollständiger Verifikationen

Das Zusammensetzen vollständiger Verifikationen entsprechend Abschnitt 6.3.2 ist ein Spezialfall des Zerlegens, bei dem es keine intern lokalen Eingänge gibt. Der Satz überträgt sich also entsprechend.

9 Literatur

Abadi/Lamport 1995
M. Abadi, L. Lamport: Conjoining Specifications, ACM Toplas 17, 3 (May, 1995) 507-534

Achterberg/Heinz/Koch 2008
T. Achterberg, S. Heinz, T. Koch: Counting solutions of integer programs using unrestricted subtree detection: The Fifth International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems CP-AI-OR 2008, http://opus.kobv.de/zib/volltexte/2008/1092

Achterberg/Wedler/Brinkmann 2008
Tobias Achterberg, Markus Wedler, Raik Brinkmann: Property Checking with Constraint Integer Programming, ICCAD 2008

AHB 1999
AMBA™ Specification Rev 2.0, ARM, Document number ARM IHI 0011A, 1999

Bailey 2008
Constrained random test struggles to live up to promises, www.scdsource.com, March 2008.

Bailey et al. 2007
Brian Bailey, Grant Martin, Andrew Piziali: ESL Design and Verification, Elsevier 2007.

Bailey 2007
B. Bailey: The Great EDA Cover-up http://electronicsystemlevel.com/EDA-Cover-up.pdf, 2007

Bergeron et al 2005
Janick Bergeron, Eduard Cerny, Alan Hunter, Andrew Nightingale: Verification Methodology Manual for System Verilog, Springer 2005.

Beizer 1990
B. Beizer: Software Testing Techniques 2nd edition, International Thomson Publishing, 1990

Beuer 2005
T. Beuer: Semiformales Property Checking mit GateProp/GateMon ITL und Modelsim PSL: Vergleich zweier Methoden zur Verifikation digitaler Systeme, Hagenberg 2005.

Beyer/Bormann 2008:
J. Bormann, S. Beyer: Erfindungsmeldung an OneSpin Solutions zur kompositionalen vollständigen Verifikation, 2008

Beyer 2008:
S. Beyer: Completeness Checker overlooks gap. Personal communication.

Beyer 2005
S. Beyer: Putting it all together – Formal Verification of the VAMP, Dissertation, Saarland University, Saarbrücken, 2005

Biere 1999
A. Biere, A. Cimatti, E. Clarke: Symbolic model checking without BDDs, TACAS, 1999

Bombana et al. 1995:
M. Bombana, P. Cavalloro, S. Conigliaro, R. B. Hughes, G. Musgrave, G. Zaza: Design-Flow and Synthesis for ASICs: A Case Study, DAC, 1995

Börger/Stärk 2003
E. Börger and R. Stärk, Abstract State Machines: A Method for High-Level System Design and Analysis, Springer-Verlag, 2003

Bormann 2007:
SVOPs – Slide show to define operation properties in SVA 2007. OneSpin Company confidential.

Bormann et al. 2007:
J. Bormann, S. Beyer, A. Maggiore, M. Siegel, S. Skalberg, T. Blackmore, F. Bruno: Complete Formal Verification of TriCore2 and Other Processors, DVCon 2007

Bormann/Beyer/Skalberg 2006:
J. Bormann, S. Beyer, S. Skalberg: Equivalence Verification between Transaction Level Models and RTL at the Example of Processors, European Patent Application, issued Dec. 22nd, 2006, Publication number EP1933245

Bormann/Blank/Winkelmann 2005
J. Bormann, C. Blank, K. Winkelmann: Technical and Managerial Data About Property Checking With Complete Functional Coverage, Best Paper Candidate, Euro DesignCon, München 2005

Bormann/Busch 2005:
J. Bormann, H. Busch: Verfahren zur Bestimmung der Güte einer Menge von Eigenschaften, verwendbar zur Verifikation und Spezifikation von Schaltungen (Method for determining the quality of a set of properties, applicable for the verification and specification of circuits). European Patent Application issued 2005. Publication number EP1764715.

Bormann/Winkelmann 2003
J. Bormann, K. Winkelmann: Verfahren und Vorrichtung zum Erstellen eines Modells einer Schaltung zur formalen Verifikation. (Method and Tool to Create a Circuit Model for Formal Verification) German Patent DE10325513

Bormann 2003:
J. Bormann: Productivity Figures for Complete Formal Block Verification, edacentrum, DATE 2003

Bormann 2003B
J. Bormann: GateProp-Spezifikation, 2003

Bormann 2001
J. Bormann: Formale Verifikation wird zum Handwerk (Formal verification becomes a craft): GI/ITG/GMM-Workshop“Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen", Meissen, 2001

Bormann/Spalinger 2001
J. Bormann, C. Spalinger: Formale Verifikation für Nicht-Formalisten (Formal Verification for Non Formalists) in: IT+TI 1/2001, Oldenbourg Verlag.

Bormann 2000
J. Bormann: From Art to Craft:: Formal Methods in Industrial ASIC Projects. Medea Conference, Munich 2000.

Bormann/Warkentin 1999
J. Bormann und P. Warkentin: Verfahren zum Vergleich elektrischer Schaltungen (Method for the comparison of electric circuits). European Patent 1068580, issued 1999

Bormann et al. 1995
J. Bormann, J.Lohse, M. Payer and G.Venzl: Model Checking in Industrial Hardware Design, DAC 1995.

Brand 199
D. Brand: Verification of Large Synthesized Designs, Proc. IEEE International Conference on Computer-Aided Design (ICCAD), Santa Clara, pp. 534-537, Nov. 1993.

Brinkmann 2003
Preprocessing for Property Checking of Sequential Circuits on the Register Transfer Level, Dissertation, Kaiserslautern 2003

Broy/Rumpe 2007
M. Broy, B. Rumpe: Modulare hierarchische Modellierung als Grundlage der Software- und Systementwicklung. Informatik-Spektrum. Springer Verlag. Band 30, Heft 1, Februar 2007.

Broy 1994:
M. Broy: A Functional Rephrasing of the Assumption/Commitment Specification Style Technical Report number TUM-I9417, Technische Universität München, 1994

Buckow/Bormann 1993
O. Buckow, J. Bormann. Formale Spezifikation und Verifikation eines SPARC-kompatiblen Prozessors mit einem interaktiven Beweissystem (Formal Specification and Verification of a SPARC Compatible Processor with an Interactive Theorem Prover), Diplomarbeit & Siemens-Report, Munich 1993.

Busch 2005:
H. Busch: TPROP - Checking Completeness and other Meta Properties, internal report, OneSpin and Infineon confidential, 2005

Busch 1991
H. Busch: Hardware Design by Proven Transformations, Dissertation, Brunel University, 1991

Büttner 2007
Wolfram Büttner: Functional Verification of IP: Quo Vadis? IP07, Keynote Talk, Grenoble 2007

Büttner/Siegel 2007
Wolfram Büttner, Michael Siegel: 'Achieving completeness in IP functional verification' posted on Feb. 12, 2007, at http://www.eetimes.com/showArticle.jhtml?articleID=197005268

Büttner 2005
Wolfram Büttner: Is Formal Verification Bound to Remain a Junior Partner of Simulation? CHARME 2005:1

Cai/Gajski 2003
Lukai Cai, Daniel Gajski: Transaction Level Modeling in System Level Design
Technical Report 2-10, Center for Embedded Computer Systems, University of California, Irvine, 2003

Carter/Hemmady 2007
Hamilton B. Carter, Shankar G. Hemmady: Metric Driven Design Verification, Springer 2007.

Case/Mishchenko/Brayton 2006
M. L.Case, A. Mishchenko, and R. K. Brayton: Inductively finding a reachable state space over-approximation, Proc. IWLS '06, pp. 172-179.

Certess o.J.
Certitude Data Sheet – www.certess.com/product/certitude_datasheet.pdf

Chauhan et al. 2002
P. Chauhan, E. Clarke, J. Kukula, S. Sapra, H. Veith, and D. Wang: Automated abstraction refinement for model checking large state spaces using SAT based conflict analysis, in Proc. International Conference on Formal Methods in Computer-Aided Design (FMCAD), 2002.

Claessen 2006
Koen Lindström Claessen - A Coverage Analysis for Safety Property Lists - Workshop on Designing Correct Circuits (DCC), Vienna March 25-26, 2006

Claessen 2007
Koen Claessen - A Coverage Analysis for Safety Property Lists - FMCAD, November 2007.

Clarke/Emerson/Sistla 1986
Clarke, E. M.; Emerson, E. A.; Sistla, A. P., "Automatic verification of finite-state concurrent systems using temporal logic specifications", ACM Transactions on Programming Languages and Systems 8: 244, 1986

Clarke et al. 2000
E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith: Counterexample-guided abstraction refinement in Proc. International Conference on Computer Aided Verification (CAV), 2000, pp. 154–169.

Conquest o.J.
Conquest – Advanced Verification Environment, Data Sheet, Real Intent

Cuéllar/Huber 1995
Jorge Cuéllar, Martin Huber: TLT. Formal Development of Reactive Systems 1995: 151-169

Dempster/Stuart 2001
D. Dempster, M. Stuart: "Verification Methodology Manual - Techniques for Verifying HDL Designs", Teamwork International, 2nd edition, 2001;

Drechsler/Hoereth 2002
R. Drechsler, S. Hoereth: Gatecomp: Equivalence Checking of Digital Circuits in an Industrial Environment, International Workshop on Boolean Problems, p. 195, Freiberg, 2002

[Een/Sörensson 2003]
N. Een, N. Sörensson: An Extensible SAT-Solver, SAT 2003.

EN61508 2001
Europäische Norm 61508: Funktionale Sicherheit elektrischer / elektronischer / programmierbarer elektronischer Systeme, siehe insbesondere Teil 2, 2001

Filkorn 1992
T. Filkorn: Symbolische Methoden für die Verifikation endlicher Zustandssysteme, Dissertation, München 1992

Foster et al 2003:
H. foster, A. Krolnik, D. Lacey: Assertion-Based Design, Kluwer Academic Publishers, 2003

Foster et al 2007
H. Foster, Lawrence Hoh, Bahman Rabii, Vigyan Singhal: Guidelines for Creating a Formal Verification Testplan, DVCon 2007

Foster/Krolnik 2008:
Foster, Krolnik: Creating Assertion Based IP, Springer 2008

Freibothe 2009:
Martin Freibothe: "Ein Ansatz für die verifikationsgerechte Verhaltensmodellierung für die semi-formale Verifikation von Mixed-Signal-Schaltungen", Dissertation, TU-Dresden 2009

Ganai/Gupta 2007:
SAT-based Scalable Formal Verification Solutions, Springer 2007.

Goldberg/Novikov 2002:
E. Goldberg, Y. Novikov: BerkMin: A Fast and Robust Sat-Solver. DATE 2002

Grosse/Hampton 2005
Grosse, Joerg and Hampton, Mark: Method for Evaluating the Quality of a Test Program, European Patent Application, publication number EP1771799, Patent applied 2005.

Große et al. 2008
D. Große, U. Kühne, R. Drechsler: Analyzing Functional Coverage in Bounded Model Checking: IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, Volume 27, Number 7, pp. 1305-1314, July 2008

Gupta et al. 2003
A. Gupta, M. Ganai, Z. Yang, and P. Ashar: Iterative abstraction using SAT-based BMC with

proof analysis, in Proc. International Conference on Computer-Aided Design (ICCAD). Washington, DC, USA: IEEE Computer Society, 2003, p. 416.

Gurevich 1995
Y. Gurevich, Evolving Algebras 1993: Lipari Guide, E. Börger (ed.), Specification and Validation Methods, Oxford University Press, 1995

Hirsch 1967
I. N. Hirsch: MEMMAP/360. Report TR-P 1168, IBM Systems Development Division, Product Test Laboratories, 1967, (zitiert nach [Beizer 1990])

Hoereth/Müller-Brahms/Rudlof 2002:
S. Hoereth, M. Müller-Brahms, T. Rudlof: Method and Device for Verifying Digital Circuits, Europäisches Patent EP1546949, eingereicht 2002

Hoereth 2003:
S. Hoereth: Method and apparatus for determining the minimum or maximum switching activity of a digital circuit, Amerikanisches Patent 6960930, eingereicht 2003

Hojati 2003
Hojati, R.: "Determining Verification Coverage Using Circuit Properties" US Patent 6594804, granted July 15th, 2003;

Hoskote et al. 1999
Hoskote, Kam, Ho, and Zhao: "Coverage Estimation for Symbolic Model Checking" Proc. of 36th Design Automation Conference, 1999;

IFV 2005
Incisive Formal Verifier, Data Sheet, Cadence 2005

Jain et al. 2008:
H. Jain, D. Kroening, N. Sharygina, and E. M. Clarke: Word-level predicate-abstraction and refinement techniques for verifying RTL Verilog, IEEE Transactions on Computer-Aided Design, vol. 27, no. 2, pp. 366–379, 2008.

Sheeran/Singh/Stalmarck 2000
M. Sheeran, S. Singh, and G. Stalmarck: Checking safety properties using induction and a SAT-solver, in Proc. International Conference on Formal Methods in Computer-Aided Design (FMCAD), 2000.

Jain et al. 2007
Alok Jain, J. Baumgarnter, R. S. Mitra, P. Dasgupta: Formal Assertion-Based Verification in Industrial Setting, Slide Show of Tutorial at DAC 2007

Jasper 2007
Jasper Gold Verification System Familiy v4.3, Data Sheet, Jasper 2007

Katz et al 1999
S. Katz, O. Grumberg, D. Geist: Have I written Enough Properties? CHARME 1999

Kranen 2008
Kathryn Kranen, Time to reconcile the design / verification divorce. www.scdsource.com, May 2008

Kroening/Seshia 2007
D. Kroening, S. A. Seshia: Formal verification at higher levels of abstraction, ICCAD 2007.

Kuehlmann/Krohm 1997
A. Kuehlmann, F. Krohm: Abstract Equivalence Checking Using Cuts and Heaps, DAC 1997

Kühne/Beyer/Pichler 2008
Paper über Prozessorsimulation. Internal report at OneSpin Solutions.

Kunz 1993
W. Kunz: HANNIBAL: An Efficient Tool for Logic Verification Based on Recursive Learning, Proc. 1993 ACM / IEEE International Conference on Computer-Aided Design (ICCAD), pp. 538 - 543, Santa Clara, November 1993

Lamport 1994
L. Lamport: Introduction to TLA, http://www.hpl.hp.com/techreports/Compaq-DEC/SRC-TN-1994-001.htm, 1994

Lohse/Warkentin 1998
J. Lohse, P. Warkentin: Method and Device for Comparing Technical Systems With Each Other, European Patent application, Publication number WO0026825, application issued 1998

Lohse/Warkentin 2001
J. Lohse, P. Warkentin: Method and configuration for comparing a first characteristic with predetermined characteristics of a technical system, US patent 6581026, filed 2001

Loitz et al. 2008
S. Loitz, M. Wedler, C. Brehm, T. Vogt, N. Wehn, W. Kunz**:** Proving Functional Correctness of Weakly Programmable IPs - A Case Study with Formal Property Checking, SASP 2008

Magellan o.J.
Magellan – Hybrid RTL Formal Verification, Data Sheet, Synopsys o.J.

Malik et. al 2001
S. Malik, M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, Chaff: Engineering an Efficient SAT Solver, DAC 2001

Marques-Silva/Sakallah 1996
J. P. Marques-Silva, K. A. Sakallah: GRASP: A New Search Algorithm for Satisfiability. In Proceedings of International Conference on Computer-Aided Design, pp. 220-227, 1996

Mayger/Harris 1991
E. Mayger, R. L. Harris: User Guide for LAMBDA version 4.1, 1991

McMillan/Amla 2003
K. L. McMillan and N. Amla: Automatic abstraction without counterexamples, in Proc. Inter-

national Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), 2003, pp. 2–17

McMillan 1992
Symbolic Model Checking, Dissertation, CMU, Pittsburgh 1992

Meredith 2004
M. Meredith: A look into Behavioral Synthesis, 2004
http://www.eetimes.com/news/design/showArticle.jhtml?articleID=18900783

Mitra 2008
Mitra, R.: Strategies for Mainstream Usage of Formal Verification, Design Automation Conference 2008

Moore 1965
G. E. Moore, Cramming more components onto integrated circuits, Electronics, Volume 38, Number 8, April 1965

Nguyen et al. 2008:
M. D. Nguyen, M. Thalmaier, M. Wedler, J. Bormann, D. Stoffel, W. Kunz: Unbounded Protocol Compliance Verification Using Interval Property Checking With Invariants: IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, vol. 27, issue 11, Nov. 2008, p 2068-2082.

Nguyen et al. 2005
Minh D. Nguyen, Dominik Stoffel, Wolfgang Kunz: Enhancing BMC-based Protocol Verification Using Transition-By-Transition FSM Traversal. GI Jahrestagung (1) 2005: 303-307

Nguyen et al. 2005a
Minh D. Nguyen, Dominik Stoffel, Markus Wedler, Wolfgang Kunz: Transition-by-transition FSM traversal for reachability analysis in bounded model checking. ICCAD 2005: 1068-1075

Novikov/Brinkmann 2005
Y. Novikov and R. Brinkmann: Foundations of Hierarchical SAT Solving. ZIB-Report 05-38 (August 2005), Zuse Institut Berlin, 2005

Novikov 2003
Y. Novikov: Local Search for Boolean Relations on the Basis of Unit Propagation. DATE 2003.

Novikov/Goldberg 2001
Y. Novikov and E. Goldberg: An efficient learning procedure for multiple implication checks. DATE 2001.

OneSpin o.J.
Data Sheet for OneSpin 360 MV, OneSpin Solutions GmbH, www.onespin-solutions.com/download.php

Offutt/Untch 2000
Offutt, Jeff and Untch, Roland: Mutation 2000: Mutation Testing in the Twentieth and the Twenty First Centuries, 45-55, San Jose, CA, October 2000.

Paucke 2003
Paucke, Marc: Formal Specification of Properties using Symbolic Timing Diagrams for Bounded Model Checking (gemeinsam mit der Infineon AG München) Diplomarbeit an AG Softwaretechnik des Instituts für Verteilte Systeme an der Otto-von-Guericke-Universität, Magdeburg, 2003

Peukert et al. 2001
R.Peukert, K. Henke, H.-D. Wuttke: A Graphical Input Method for Model Checking. Technical Report, TU Ilmenau, 2001.

PPv2 2008
Infineon Protocol Processor, Success Story,
http://www.onespin-solutions.com/download.php, 2008

PSL 2004
Property Specification Language Reference Manual, Version 1.1, Accellera, 2004

Schickel et al. 2007
M. Schickel, V. Nimbler, M. Braun, Hans Eveking: CandoGen – A Property-Based Model Generator. University Booth at DATE 2007, Nice, France, 2007.

Schickel et al. 2006
M. Schickel, V. Nimbler, M. Braun, H. Eveking: An Efficient Synthesis Method for Property-Based Design in Formal Verification. In: Sorin Huss (Ed.): Advances in Design and Specification Languages for Embedded Systems, p. 163-182, Kluwer Acad. Publishers, Boston/Dordrecht/London, 2007

Schlör 2001
R. Schlör: Symbolic Timing Diagrams: A Visual Formalism for Model Verification, Dissertation, Carl-von-Ossietzky-Universität, Oldenburg, 2001.

Schönherr et al 2008:
J. Schönherr, M. Freibothe, B. Straube, J. Bormann. Semi-formal verification of the steady state behavior of mixed-signal circuits by SAT-based property checking: Theoretical Computer Science, Elsevier Science Publishers Ltd., 2008, 404, 293-307

Shimizu et al. 2006
Shimizu, Gupta, Koyama, Omizo, Abdulhafiz, McConville, Swanson: Verification of the Cell Broadband EngineTM Processor: DAC 2006

Stoffel et al. 2004
Dominik Stoffel, Markus Wedler, Peter Warkentin, Wolfgang Kunz: Structural FSM traversal. IEEE Trans. on CAD of Integrated Circuits and Systems 23(5): 598-619 (2004)

Stuart/Dempster 2000:
Verification Methodology Manual – For Code Coverage in HDL Designs, Teamwork International, Yateley, 2000.

Siegel et al. 1999
Michael Siegel, J. Bormann, H.Busch, M. Turpin: Definition of the Interval Temporal Logic (ITL), personal communication, 1999

Solidify o.J.
Solidify Data Sheet, Averant

System Verilog 2005
IEEE Standard for System Verilog – Unified Hardware Design, Specification, and Verification Language, IEEE Computer Society, 2005

Turpin 2003
M. Turpin: The Dangers of Living with an X (bugs hidden in your Verilog): Synopsys Users Group, Boston 2003

Tarisan/Keutzer 2001:
S. Tarisan, K. Keutzer: "Coverage Metrics for Functional Validation of Hardware Designs", IEEE Design & Test of Computers. 2001.

Thomas et al. 2004:
A. Thomas, J. Becker, U. Heinkel, K. Winkelmann, J. Bormann: Formale Verifikation eines Sonet/SDH Framers. (Formal Verification of a Sonet/SDH framer) GI/ITG/GMM-Workshop "Methoden und Beschreibungssprachen zur Modellierung und Verifikation von SChaltungen und Systemen, Kaiserslautern, 2004

Wang et al. 2006
C. Wang, B. Li, H. Jin, G. Hachtel, and F. Somenzi: Improving Ariadne's bundle by following multiple threads in abstraction refinement, IEEE Transactions on Computer-Aided Design, vol. 25, pp. 2297 – 2316, 2006.

Wedler et al. 2007
Markus Wedler, Dominik Stoffel, Raik Brinkmann, Wolfgang Kunz: A Normalization Method for Arithmetic Data-Path Verification. IEEE Trans. on CAD of Integrated Circuits and Systems 26(11): 1909-1922 (2007)

Wedler et al. 2005
Markus Wedler, Dominik Stoffel, Wolfgang Kunz: Normalization at the arithmetic bit level. DAC 2005: 457-462

Wedler et al. 2004:
Markus Wedler, Dominik Stoffel, Wolfgang Kunz: Exploiting state encoding for invariant generation in induction-based property checking. ASP-DAC 2004: 424-429

Wedler et al. 2003:
Markus Wedler, Dominik Stoffel, Wolfgang Kunz: Using RTL Statespace Information and State Encoding for Induction Based Property Checking. DATE 2003: 11156-11157

Winkelmann et al. 2004:
Klaus Winkelmann, Hans-Joachim Trylus, Dominik Stoffel, Görschwin Fey: Cost-Efficient Block Verification for a UMTS Up-Link Chip-Rate Coprocessor. DATE 2004: 162-167

Wishbone 2002
WISHBONE, Revision B.3 Specification, 2002
http://www.opencores.org/projects.cgi/web/wishbone/wishbone

0-in 2005
Assertion Based Verification, Datasheet, Mentor 2005

Index