Automatisierte HW/SW Co-Verifikation von SystemC Modellen mit Hilfe von Timed Automata

Paula Herber
International Computer Science Institute
Berkeley, CA, USA

herber@icsi.berkeley.edu

Abstract: SystemC hat sich zum de-facto Standard zur Modellierung und Simulation im HW/SW Co-Design entwickelt. Existierende Verifikationstechniken für SystemC sind jedoch meist ad hoc und unsystematisch. Wir präsentieren ein formal fundiertes Framework zur systematischen und umfassenden Co-Verifikation von SystemC Modellen. Das Framework basiert auf einer von uns definierten formalen Semantik für SystemC und verwendet eine Kombination von Model Checking und Konformitätstesten zur automatisierten Qualitätssicherung während des gesamten Entwurfsablaufs. Wir demonstrieren die Leistungsfähigkeit unseres Ansatzes mit einem Packet Switch und einem System zur Anti-Blockier- und Anti-Schlupf-Regulierung.

1 Einleitung

Eingebettete Systeme werden häufig in Bereichen eingesetzt, in denen ein Fehler zu hohen finanziellen Verlusten oder sogar zu Verletzungen und Todesfällen führen kann, zum Beispiel zur Steuerung von Flugzeugen, Zügen und Satelliten. Als Folge davon wird es immer wichtiger, die Korrektheit eingebetteter Systeme mit systematischen und umfassenden Verifikationstechniken sicher zu stellen. Eine besondere Herausforderung ist dabei, dass in eingebetteten Systemen Hardware- und Software-Anteile eng miteinander verflochten sind. Um solche Systeme zu modellieren und zu simulieren wird häufig die Systembeschreibungssprache SystemC [IEE05] eingesetzt. SystemC ist eine Erweiterung von C++ zur integrierten Modellierung von Hardware und Software. Die Modelle können außerdem auf verschiedenen Abstraktionsebenen in einer Co-Simulation ausgeführt werden und damit während des gesamten Entwurfsablaufs ständig überprüft werden. Für eine umfassende Qualitätssicherung ist eine Simulation allein allerdings nicht ausreichend. Das erste Problem ist, dass mit einer Simulation nicht alle möglichen Eingabeszenarien abgedeckt werden können. Das ist insbesondere bei eingebetteten Systemen ein Problem, weil diese in der Regel nicht terminieren und eine kontinuierliche Umgebung steuern. Die Anzahl möglicher Eingabetraces ist deshalb unendlich groß und die Traces können unendlich lang sein. Ein weiteres Problem ist, dass der Automatisierungsgrad begrenzt ist. Die Simulation selbst funktioniert zwar automatisch, die Simulationsergebnisse müssen aber in der Regel manuell ausgewertet werden. Das ist auch die Ursache für das dritte Problem, dass die Konsistenz zwischen verschiedenen Abstraktionsebenen sehr schwer zu etablieren ist.

Um die genannten Probleme zu lösen haben wir das Framework VeriSTA (Framework zur **Veri**fikation von **S**ystemC Modellen mit Hilfe von **T**imed **A**utomata) entwickelt [HPG10]. Die übergeordnete Idee ist, abstrakte Modelle via Model Checking zu verifizieren und anschließend Konformitätstests zu generieren um die Konformität verfeinerter Modelle zum abstrakten Modell zu prüfen. Mit diesem Ansatz erhalten wir Garantien über bestimmte Eigenschaften des abstrakten Entwurfs und stellen gleichzeitig die Konsistenz verfeinerter Entwürfe über den Entwurfsablauf hinweg sicher. Das Ergebnis ist ein systematischer, umfassender und formal fundierter Qualitätssicherungsprozess, der den Entwurfsprozess von der abstrakten Spezifikation bis zur finalen Implementierung unterstützt. Als Basis haben wir eine formale Semantik für SystemC entwickelt, die auf einer Transformation gegebener SystemC Modelle in die wohl-definierte Sprache der UPPAAL Timed Automata (UTA) [BY04] beruht. Das resultierende formale Modell verwenden wir sowohl zum Model Checking [HFG08], als auch zur Generierung von Konformitätstests [HPG10, HFG09].

Im Folgenden geben wir zunächst einen Überblick über verwandte Arbeiten. Dann stellen wir kurz das Framework VeriSTA vor und anschließend gehen wir auf die formale Semantik für SystemC und die darauf aufbauenden Verifikationstechniken ein. Zuletzt werden wir unsere experimentellen Ergebnisse darstellen und mit einem Fazit schließen.

2 Verwandte Arbeiten

Bestehende Ansätze zur Formalisierung von SystemC sind überwiegend auf eine synchrone Untermenge des Sprachumfangs beschränkt [MRR03, GKD06] oder sie vernachlässigen das Zeitverhalten ganz [HMT06, Man05]. Außerdem werden bei fast allen Ansätzen nur einfache Kommunikationsmuster und statische Sensitivitäten berücksichtigt. Ansätze, die auch dynamische Sensitivitäten handhaben können, erfordern entweder eine manuelle Formalisierung gegebener Modelle [TCMM07] oder erzeugen sehr große Modelle [KEP06]. Wir wollen die Semantik zur HW/SW Co-Verifikation einsetzen, d. h. die Modelle, die wir betrachten beinhalten Software, synchrone und asynchrone HW-Anteile, und ggf. komplexe Kommunikationsmuster. Wir müssen also nicht nur das exakte Zeitverhalten, sondern auch dynamische Sensitivität und komplexe Kommunikationsmuster berücksichtigen. Außerdem benötigen wir eine Formalisierung die es erlaubt, formale Modelle automatisch aus gegebenen SystemC Modellen zu generieren.

Die bestehenden Ansätze zur Generierung von Konformitätstests aus Timed Automata Modellen sind im wesentlichen in den beiden Werkzeugen CoVer [HLM$^+$08] und TRON [LMN05, HLM$^+$08] umgesetzt, die UPPAAL um Möglichkeiten zur Testgenerierung erweitern. Bei CoVer ist allerdings das Problem, dass es nur mit deterministischen Spezifikationen umgehen kann. Bei TRON werden die Konformitätstests *online* während der Testausführung berechnet. Beides ist nicht akzeptabel, einerseits weil SystemC Modelle inhärent nicht-deterministisch sind und andererseits weil nur die *offline* Generierung von Testfällen es ermöglicht, sie in einem verfeinernden Entwurfsablauf in jedem Entwurfsschritt wiederholen zu können.

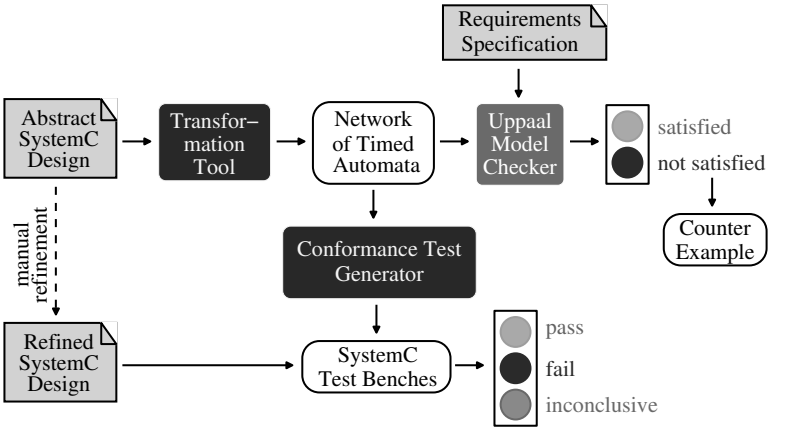

Abbildung 1: VeriSTA Framework

3 Framework zur HW/SW Co-Verifikation

Unser Framework VeriSTA ist in Abb. 1 dargestellt. Es basiert auf der Annahme, dass HW/SW Systeme in einem verfeinernden Entwurfsablauf entwickelt werden, bei dem ein abstraktes Modell schrittweise bis zur endgültigen Implementierung verfeinert wird. Ein solcher Verfeinerungsschritt ist auf der linken Seite von Abb. 1 dargestellt. Wir wollen Model Checking verwenden, um zu verifizieren, dass das abstrakte Modell eine gegebene Anforderungsspezifikation erfüllt. Um dies zu ermöglichen wird das abstrakte Modell zunächst in ein semantisch äquivalentes UTA Modell transformiert. Das so generierte UTA Modell kann direkt als Eingabe für den UPPAAL Model Checker verwendet werden und ermöglicht damit die vollautomatische und vollständige Verifikation von Sicherheits-, Lebendigkeits- und zeitlichen Eigenschaften. Die zu verifizierenden Eigenschaften müssen als Anforderungsspezifikation in temporaler Logik (UPPAAL unterstützt eine Untermenge der *Computation Tree Logic* CTL) gegeben sein. Wenn eine Eigenschaft sich als nicht erfüllt herausstellt, liefert der Model Checker zusätzlich ein Gegenbeispiel, das in der UPPAAL Werkzeugumgebung visualisiert und animiert werden kann. Zur Generierung von Konformitätstests gehen wir davon aus, dass uns eine Menge von Eingabe-Traces als SystemC Test Benches zur Verfügung stehen. Diese werden zusammen mit dem Systemmodell in ein UTA Modell transformiert. Aus diesem Modell generieren wir Konformitätstests, d. h. wir berechnen die Menge aller möglichen Ausgabe-Traces für jeden gegebenen Eingabe-Trace. Diese können zur Generierung von SystemC Test Benches verwendet werden, die automatisch prüfen, ob ein verfeinertes Modell Traces liefert, die auch im abstrakten Modell erlaubt sind. Ist das der Fall, ist das verfeinerte Modell konform zum abstrakten Modell. Insgesamt haben wir mit der Formalisierung, dem Transformationswerkzeug, dem Model Checker und der Generierung von Konformitätstests den eingangs beschriebenen Qualitätssicherungsprozess vollständig umgesetzt.

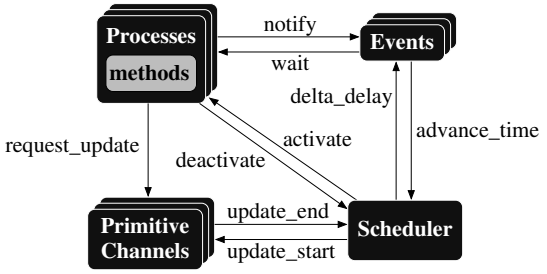

Abbildung 2: Repräsentation von SystemC Modellen in UPPAAL

4 Formale Semantik für SystemC

SystemC erweitert C++ zur Modellierung von Hardware. Dazu werden z. B. Konstrukte zur Modellierung von Zeit, Nebenläufigkeit und Reaktivität eingeführt. Die Modelle können außerdem mit Hilfe des SystemC Schedulers in einer ereignis-diskreten Simulation ausgeführt werden. Die Grundidee unserer Formalisierung ist, dass wir die Semantik aller relevanten SystemC-Konstrukte in der formalen Sprache der UTA formal definieren können. Damit können wir den gesamten Umfang der SystemC Semantik erfassen, inklusive beliebigen Zeitverhaltens, dynamischer Sensitivität und komplexer Kommunikationsmuster. Die formale Semantik ist außerdem so formuliert, dass auf Basis der Formalisierung ein gegebenes SystemC Modell automatisch in ein semantisch äquivalentes UTA Modell transformiert werden kann. Ein besonderer Vorteil ist dabei, dass die Struktur des SystemC Modells im UTA Modell erhalten bleibt. Zusammen mit der C-ähnlichen Aktionssprache von UTA führt dies dazu, dass das formale Modell eines gegebenen SystemC-Modells sehr gut nachvollziehbar ist. In der UPPAAL Werkzeugumgebung können die Modelle visualisiert und simuliert werden. Weiterhin können mit dem UPPAAL Model Checker wichtige Eigenschaften, zum Beispiel Lebendigkeit, Sicherheit oder die Einhaltung von Zeitschranken, vollautomatisch verifiziert werden. Die einzigen Bedingungen, die wir an ein SystemC Modell stellen müssen, damit es in ein UTA Modell transformiert werden kann, sind: (1) Es darf keine dynamische Speicher- oder Prozessallokation verwendet werden. Diese Bedingung ist in sicherheitskritischen eingebetteten Systemen typischerweise erfüllt. (2) Es dürfen nur Variablen verwendet werden, die auf beschränkte Integers abgebildet werden können. Dies ist akzeptabel, da die meisten Datentypen auf Integer Variablen abgebildet werden können. Wenn beide Voraussetzungen erfüllt sind, kann jedes SystemC Modell durch ein äquivalentes UTA Modell repräsentiert werden.

Ein SystemC Modell besteht aus einer Menge von Modulen, die Methoden und Prozesse enthalten. Während Methoden sequentiellen Code enthalten, werden Prozesse nebenläufig ausgeführt und ihre Ausführung wird von Ereignissen gesteuert. In Abb. 2 ist dargestellt, wie wir SystemC Modelle innerhalb von UPPAAL repräsentieren. Jede Methode wird auf einen einzelnen Automaten abgebildet. Diese Methoden-Automaten werden dann in Prozess-Automaten verpackt, die die Interaktion mit Ereignissen, primitiven Kanälen und dem Scheduler übernehmen. Die Interaktion erfolgt dabei über UPPAAL-Kanäle, zum Beispiel benachrichtigen die Prozesse die Ereignisse über einen `notify` Kanal und die Er-

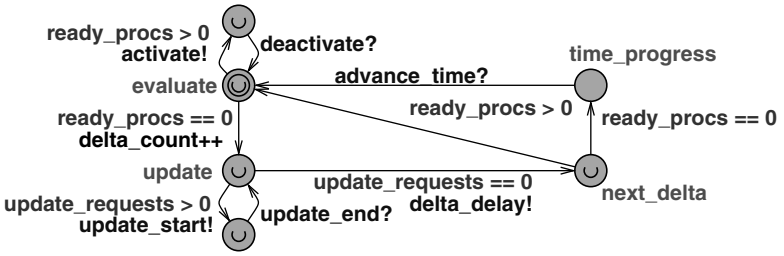

Abbildung 3: SystemC Scheduler in UPPAAL

eignisse stoßen die Prozesse über einen `wait` Kanal an. Dabei werden sowohl statische als auch dynamische Sensitivitäten berücksichtigt. Um die Ausführungssemantik von SystemC zu formalisieren haben wir vordefinierte Modelle für Ereignisse, Prozesse, primitive Kanäle und den Scheduler entwickelt, die deren Semantik formal beschreiben und damit eine formale Semantik definieren. Weiterhin können sie für die Transformation von SystemC Modellen beliebig oft instanziiert werden. Dadurch erhalten wir eine kompositionale Transformation, d. h. wir übersetzen jedes Modul einzeln und setzen das System anschließend in einer Instanziierungs- und Bindungsphase zusammen. Die vollständige formale Semantik ist in [HFG08] und [Her10] angegeben.

Als Beispiel für die konkrete Modellierung von SystemC in UTA ist in Abb. 3 das UTA Modell des SystemC Schedulers dargestellt. Der Scheduler führt SystemC-Modelle in einer ereignis-diskreten Simulation aus. Dabei werden Delta-Zyklen verwendet, um parallele Prozesse zu sequentialisieren. Jeder delta-Zyklus besteht aus einer **evaluate**- und einer **update**-Phase. In der **evaluate**-Phase werden alle Prozesse, die bereit zur Ausführung sind, mit dem Signal **activate** angestoßen. Erst wenn kein Prozess mehr bereit ist, geht der Scheduler in die **update**-Phase über. In dieser werden alle primitiven Kanäle aktualisiert, d. h. die Daten, die vorher von den Prozessen berechnet wurden, werden übernommen. Der Vorteil dieser Zweiteilung ist, dass alle parallelen Prozesse trotz Sequentialisierung auf den gleichen Daten arbeiten. Wenn ein Delta-Zyklus abgeschlossen ist, werden darüber die Ereignisse über das Signal **delta_delay** benachrichtigt, dadurch können neue Prozesse angestoßen werden. Wenn das geschieht, geht der Scheduler in **next_delta** über und startet einen neuen Delta-Zyklus. Wenn nach einem Delta-Zyklus keine Prozesse mehr bereit zur Ausführung sind geht der Scheduler in den Wartezustand **time_progress** über und wartet darauf, dass das nächstliegende zeitverzögerte Ereignis auslöst. Wenn das geschieht, erhält der Scheduler das Signal **advance_time** und startet einen neuen Delta-Zykles. Damit ist die Ausführungssemantik des Schedulers vollständig modelliert. Eine Besonderheit der Modellierung ist, dass die Prozesse nicht gezielt gestartet werden, sondern nur ein **activate** Signal über einen binären Kanal gesendet wird. Dies führt dazu, dass die Auswahl des jeweils nächsten Prozesses nicht-deterministisch erfolgt. Der Vorteil davon ist, dass beim Model Checking des resultierenden UTA Modells jede mögliche Ausführungsreihenfolge abgedeckt wird. Damit können wir auch Fehler aufdecken, die mit einer Simulation grundsätzlich nicht aufgedeckt werden können, weil sich die Simulation immer für eine Reihenfolge entscheidet.

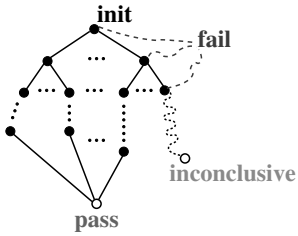

Abbildung 4: Akzeptanzgraph

5 Model Checking und Konformitätstesten

Die von uns definierte formale Semantik für SystemC ermöglicht unmittelbar die Anwendung des UPPAAL Model Checkers, wie bereits in Abb. 1 dargestellt. Weiterhin dient sie uns als Basis für Konformitätstests. Das Ziel des Konformitätstestens ist es, für eine gegebene Menge von Eingabetraces zu prüfen, ob sich das verfeinerte Modell konform zum abstrakten Modell verhält. Um dies zu erreichen, berechnen wir zunächst alle möglichen Ausgabe-Traces des abstrakten Modells für einen gegebenen Eingabetrace [HFG09]. Dazu führen wir das UTA Modell, das unsere Spezifikation darstellt, symbolisch aus. Zur formalen Fundierung der Ausführung haben wir in [Her10] eine vollständige symbolische Semantik für UTA angegeben, die die in [BY04] angegebene Semantik um Datenvariablen und binäre und broadcast- Kanäle erweitert. Die symbolische Ausführung liefert uns alle möglichen Ausgabe-Traces in Form eines Akzeptanzgraphen, der zusammen mit dem verfeinerten Modell ausgeführt werden kann um dessen Konformität zu prüfen. Im Akzeptanzgraphen (s. Abb 4) wird jeder vollständig berechnete Ausgabe-Trace mit einem **pass** abgeschlossen, jeder Ausgabe-Trace, bei dem die Berechnung wegen der Begrenzung interner Berechnungsschritte abgebrochen werden musste mit einem **inconclusive**, und implizit führen alle Traces die gar nicht enthalten sind zu einem **fail**. Der Akzeptanzgraph akzeptiert also nur solche Ausgabe-Traces, die auch im abstrakten Modell möglich sind. Wir führen damit die Konformität zwischen dem verfeinerten und dem abstrakten SystemC Modell auf die Konformität zwischen dem verfeinerten SystemC Modell und dem abstrakten UTA Modell zurück. Als formale Basis verwenden wir die *relativized timed input output conformance relation* von Larsen [LMN05]:

$$I \ rtioco \ S \ \ \textit{iff} \ \ \forall \sigma \in \mathrm{TTr}_i(\mathcal{E}) : \mathrm{TTr}_o((I,\mathcal{E}),\sigma) \sqsubseteq \mathrm{TTr}_o((S,\mathcal{E}),\sigma)$$

Die *rtioco* Relation ist eine Erweiterung von Tretmans *ioco* Relation um Zeit und eine explizite Umgebung. Eine Implementierung I ist *rtioco* konform zu einer Spezifikation S, wenn für jeden zeitbehafteten Eingabe-Trace σ, der in einer gegebenen Umgebung $\mathcal{E}$ möglich ist, die auf der Implementierung beobachteten zeitbehafteten Ausgabe-Traces $\mathrm{TTr}_o((I,\mathcal{E}),\sigma)$ in der Menge der auf der Spezifikation möglichen zeitbehafteten Ausgabe-Traces $\mathrm{TTr}_o((S,\mathcal{E}),\sigma)$ enthalten ist. Um die *rtioco*-Relation für unsere Zwecke einzusetzen, haben wir die Verfeinerung auf Mengen von Traces so umformuliert, dass sie eine explizite Verfeinerung auf symbolischen Traces erlaubt. Das hat den Vorteil, dass wir explizit ausdrücken können, dass die Implementierung das Zeitverhalten der Spezifikation verfei-

Property	1m1s	2m1s	1m2s	2m2s	satisfied
no deadlock	22.28 s	56.49 s	43.73 s	211.26 s	✓
every packet forwarded	3.02 s	3.38 s	3.30 s	4.89 s	↯
forward within time limit	129.16 s	46.63 s	298.41 s	544.88 s	✓

Tabelle 1: Aufwand des Model Checkings: Packet Switch

Property	counter-examples		verification	
no deadlock	–	–	722.54 s	✓*(maybe)*
ABS reacts within time limit	2.56 s	↯	555.56 s	✓*(maybe)*
ASR reacts within time limit	3.51 s	↯	844.15 s	✓*(maybe)*

Tabelle 2: Aufwand des Model Checkings: ABS/ASR

nern darf. Auf der formalen Grundlage der Konformitätsrelation können aus dem Akzeptanzgraphen SystemC Test Benches erzeugt werden, die verfeinerte Modelle ausführen, beobachten und die beobachteten Ausgaben mit dem Akzeptanzgraphen vergleichen. Die Besonderheit gegenüber bestehenden Arbeiten ist dabei, dass das mögliche Verhalten der Spezifikation *offline* berechnet wird und der Algorithmus auch nicht-deterministische Modell handhaben kann. Um dem Problem der Zustandsexplosion entgegenzutreten haben wir verschiedene Optimierungen entwickelt [HPG10]. Zum Beispiel nutzen wir die Besonderheiten der SystemC Semantik, um Zustände möglichst effizient zusammen zu fassen. Außerdem begrenzen wir die Anzahl interner Berechnungsschritte um die Terminierung sicher zu stellen und haben noch einige weitere Speicher- und Laufzeitoptimierungen umgesetzt, wie z.B. bit state hashing und die Auslagerung von Zuständen auf die Festplatte.

6 Experimentelle Ergebnisse

Zur Evaluierung des VeriSTA Frameworks haben wir zwei Fallstudien verwendet: das erste Beispiel ist ein Packet Switch aus der SystemC Referenzimplementierung. Das größere Beispiel ist ein System zur Anti-Blockier- und Anti-Schlupf-Regulierung (ABS/ASR). Zu diesem haben wir neben dem abstrakten Modell mit etwa 500 Zeilen Code ein verfeinertes Modell, das etwa 5000 Zeilen Code umfasst und Implementierungsdetails wie z.B. einen CAN-Bus und dessen Anbindung an die ECU enthält. Der zeitliche Aufwand zur Transformation der Modelle ist aufgrund unseres kompositionalen Ansatzes linear in der Codegröße und liegt für die genannten Beispiele unter zwei Sekunden. Für die Auswertung des Model Checking Ansatzes haben wir beim Packet Switch die Anzahl an Masters und Slaves zwischen 1 und 4 variiert (s. Tabelle 1). Für alle Variationen haben wir wichtige Eigenschaften für beliebige Eingaben geprüft, z.B. dass es keine Verklemmung geben kann und dass die Daten immer innerhalb einer gegebenen Zeitschranke weitergeleitet werden. Diese Eigenschaften konnten mit dem Model Checking Ansatz vollautomatisch und vollständig geprüft werden. Zu beachten ist, dass der Beweis der Eigenschaft,

	CPU Time (s)			**Memory Usage** (MB)		
	Base	Optim	Improv	Base	Optim	Improv
Packet Switch 1m1s	25.11	9.49	62.2%	58	5	91.4%
Packet Switch 1m2s	34.27	13.90	59.4%	98	5	94.9%
Packet Switch 2m1s	42.38	20.72	51.1%	160	5	96.9%
Packet Switch 2m2s	54.77	27.43	49.9%	275	13	95.3%
Packet Switch 4m4s	↯	443	∞	↯	302	∞
ABS/ASR System	↯	10210	∞	↯	302	∞

Tabelle 3: Generierung von Konformitätsstests

dass jedes Paket weitergeleitet wird, fehlgeschlagen ist. Dies liegt an der Verwendung von `sc_signals` in der Implementierung. Deren Semantik nach führen aufeinanderfolgende identische Pakete nicht zu einem Statuswechsel der Signale und damit auch nicht zu einer Weiterleitung im Packet Switch Beispiel. Dies ist ein typischer Randfall, der mit Hilfe von Simulationen sehr schwierig aufzudecken ist. Mit unserem Model Checking Ansatz konnte dieses Problem innerhalb von wenigen Sekunden aufgedeckt und seine Ursache dank der graphischen Animation der Gegenbeispiele leicht nachvollzogen werden. Das ABS/ASR System ((s. Tabelle 2)) kann mit dem Model Checking Ansatz nicht mehr vollständig verifiziert werden. Das Ergebnis *maybe satisfied*, d. h. für die gegebene Menge Speicher konnte kein Gegenbeispiel gefunden werden, es konnte aber auch nicht der gesamte Zustandsraum exploriert werden. Damit können zwar keine Garantien mehr über das ABS/ASR System abgegeben werden, die Experimente zeigen aber, dass die Generierung von Gegenbeispielen auch beim ABS/ASR System sehr schnell und effizient funktioniert. Der Ansatz eignet sich daher auch bei Modellen, die nicht mehr vollständig verifiziert werden können, sehr gut zur Fehlererkennung und -lokalisierung. Tabelle 3 zeigt den Aufwand der Generierung von Konformitätstests. Die Tabelle zeigt, dass unsere Optimierungen sehr erfolgreich waren, die Laufzeiten konnten um etwa 50 % reduziert werden und die Speichernutzung sogar um mehr als 90 %. Außerdem konnten wir mit dem optimierten Algorithmus auch Modelle handhaben, für die vorher der Speicher nicht ausgereicht hat.

Um die Eignung unseres Ansatzes zur Fehlererkennung zu prüfen, haben wir Fehler aus verschiedenen vordefinierten Fehlerklassen (z. B. fehlende oder falsche Bedingungen und Zuweisungen, Übertragungsverzögerungen und -verluste) in die Modelle eingestreut und diese dann mit den automatisch erzeugten Testbenches ausgeführt. Beim ABS/ASR System haben wir dabei das verfeinerte Modell verwendet, beim Packet Switch haben wir die Fehler in das abstrakte Modell eingestreut. In beiden Fällen konnten mit unserem Ansatz alle betrachteten Fehlerarten aufgedeckt werden.

7 Fazit

Das VeriSTA Framework erlaubt die systematische, umfassende, formal fundierte und automatisierte HW/SW Co-Verifikation von SystemC Modellen mit Hilfe von Timed Auto-

mata. Es basiert auf einem Qualitätssicherungsprozess, der den HW/SW Co-Design Prozess effizient und durchgängig von abstraktem Entwurf bis hin zur endgültigen Implementierung unterstützt. Die Grundidee ist es, abstrakte Modelle mit Hilfe von Model Checking zu verifizieren und anschließend Konformitätstests für verfeinerte Modelle zu generieren. Zur Etablierung einer formalen Basis haben wir die informell definierte Semantik von SystemC auf die formal wohl-definierte Semantik von UPPAAL Timed Automata abgebildet haben. Basierend auf dieser Abbildung können gegebene SystemC Modelle automatisch in semantisch äquivalente UTA Modelle transformiert werden. Damit bleibt den Entwicklern der mühsame und zeitaufwändige Prozess der formalen Spezifikation erspart. Weiterhin ermöglicht es die Anwendung des UPPAAL Model Checkers, mit dem temporallogische Eigenschaften vollautomatisch und vollständig verifiziert und Gegenbeispiele zur Fehlersuche graphisch animiert und simuliert werden können. Für die Transformation verwenden wir einen kompositionalen Ansatz, bei dem einzelne SystemC-Konstrukte und SystemC-Module separat übersetzt werden. Als Folge davon skaliert der Ansatz sehr gut und auch große SystemC Modelle können in kurzer Zeit übersetzt werden. Das informell definierte Verhalten eines SystemC Modells bleibt dabei vollständig erhalten, außerdem ist die Transformation strukturerhaltend, was die Fehlersuche bei fehlschlagenden Beweisversuchen stark vereinfacht. Darüber hinaus sind die generierten Modelle kompakt und gut verständlich und können verhältnismäßig effizient via Model Checking verifiziert werden. Unser Ansatz zur Generierung von Konformitätstests kann nicht-deterministische Modelle handhaben und berechnet *offline* alle möglichen Ausgabe-Traces für einen gegebenen Eingabe-Trace. Um dem Problem der Zustandsexplosion zu begegnen haben wir verschiedene Optimierungen entwickelt, die die Anzahl der Zustände drastisch reduzieren. Weiterhin verwenden wir die berechneten Ausgabe-Traces zur Generierung von SystemC Test Benches, mit denen verfeinerte Modelle automatisch ausgeführt und bewertet werden können. Unsere experimentellen Ergebnisse belegen die vollautomatische Anwendbarkeit, die Performanz und die Eignung zur Fehlererkennung unseres Ansatzes.

Zur Zeit arbeiten wir an verschiedenen Erweiterungen des VeriSTA Frameworks, beispielsweise an der automatischen Generierung von Eingabe-Traces und an einer Erweiterung für *Transaction Level Models*. Langfristig wollen wir den Model Checking Ansatz mit typischen Hardware-Verifikationstechniken wie *satisfiability solving* verbinden, um eine höhere Effizienz zu erzielen. Wir sind zuversichtlich, dass wir damit unseren Ansatz auch auf große und heterogene Systeme, z. B. Multiprozessor-Systeme, anwenden können.

Literatur

[BY04] Johan Bengtsson and Wang Yi. Timed Automata: Semantics, Algorithms and Tools. In *Lecture Notes on Concurrency and Petri Nets*, LNCS 3098. Springer, 2004.

[GKD06] Daniel Große, Ulrich Kühne, and Rolf Drechsler. HW/SW Co-Verification of Embedded Systems using Bounded Model Checking. In *Great Lakes Symposium on VLSI*. ACM Press, 2006.

[Her10] Paula Herber. *A Framework for Automated HW/SW Co-Verification of SystemC Designs using Timed Automata*. PhD thesis, Technical University of Berlin, 2010.

[HFG08] Paula Herber, Joachim Fellmuth, and Sabine Glesner. Model Checking SystemC Designs Using Timed Automata. In *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. ACM press, 2008.

[HFG09] Paula Herber, Florian Friedemann, and Sabine Glesner. Combining Model Checking and Testing in a Continuous HW/SW Co-Verification Process. In *Tests and Proofs*, volume 5668 of *LNCS*. Springer, 2009.

[HLM+08] Anders Hessel, Kim G. Larsen, Marius Mikucionis, Brian Nielsen, Paul Pettersson, and Arne Skou. *Formal Methods and Testing*, chapter Testing Real-Time Systems Using UPPAAL. Springer, 2008.

[HMT06] Ali Habibi, Haja Moinudeen, and Sofiene Tahar. Generating Finite State Machines from SystemC. In *Design, Automation and Test in Europe (DATE)*. IEEE Press, 2006.

[HPG10] Paula Herber, Marcel Pockrandt, and Sabine Glesner. Automated Conformance Evaluation of SystemC Designs using Timed Automata. In *IEEE European Test Symposium*, 2010.

[IEE05] IEEE Standards Association. IEEE Std. 1666–2005, Open SystemC Language Reference Manual, 2005.

[KEP06] Daniel Karlsson, Petru Eles, and Zebo Peng. Formal verification of SystemC Designs using a Petri-Net based Representation. In *Design, Automation and Test in Europe (DATE)*. IEEE Press, 2006.

[LMN05] Kim G. Larsen, Marius Mikucionis, and Brian Nielsen. *Formal Approaches to Software Testing*, chapter Online Testing of Real-time Systems Using UPPAAL. Springer, 2005.

[Man05] Ka Lok Man. An Overview of SystemCFL. In *Research in Microelectronics and Electronics*, volume 1, 2005.

[MRR03] Wolfgang Müller, Jürgen Ruf, and Wolfgang Rosenstiel. *SystemC: Methodologies and Applications*, chapter An ASM based SystemC Simulation Semantics. Kluwer Academic Publishers, 2003.

[TCMM07] Claus Traulsen, Jerome Cornet, Matthieu Moy, and Florence Maraninchi:. A SystemC/TLM semantics in Promela and its possible applications. In *Workshop on Model Checking Software (SPIN)*, LNCS 4595. Springer, 2007.

Paula Herber ist seit August 2010 als Stipendiatin des DAAD am International Computer Science Institute in Berkeley, Kalifornien. Ihre Forschungsinteressen gelten der Verifikation und Validierung eingebetteter Systeme. Insbesondere interessiert sie, wie formale und semi-formale Techniken kombiniert werden können um eine umfassende und effiziente Qualitätssicherung von HW/SW-Systemen zu erreichen. Vorher hat sie am Fachgebiet Programmierung eingebetteter Systeme der TU Berlin als wissenschaftliche Mitarbeiterin gearbeitet. Dort hat sie im Februar 2010 ihre Promotion mit dem Titel *A Framework for Automated HW/SW Co-Verification of SystemC Designs using Timed Automata* abgeschlossen. Auch ihr Diplom hat sie im März 2006 von der TU Berlin erhalten, sie hat dort Wirtschaftsingenieurwesen mit der Fachrichtung Informations- und Kommunikationssysteme studiert.