Funktionale Verifikation eingebetteter Systeme: Techniken und Werkzeuge auf Systemebene [1]

Hoang M. Le[2]

Abstract: Aufgrund der rasch zunehmenden Komplexität eingebetteter Systeme ergab sich die Notwendigkeit, die Abstraktionsebene im Systementwurf anzuheben. Es wurde die elektronische Systemebene geschaffen, auf der die Systembeschreibungssprache SystemC und die Konzepte zur Modellierung auf Transaktionsebene (engl. Transaction Level Modeling, TLM) große Bedeutung erlangten. TLM-Modelle, die in SystemC geschrieben sind, ermöglichen den Entwicklern, sehr früh mit der Entwicklung von Software sowie einer Verifikationsumgebung für herkömmliche Hardware-Modelle, die weniger abstrakt und erst viel später im Entwurfsablauf verfügbar sind, zu beginnen. Die resultierende Zeitersparnis und Steigerung der Produktivität hängt jedoch sehr stark von der Korrektheit der TLM-Modelle ab, die als Referenz für die weitere Software- und Hardware-Entwicklung dienen. Aus diesem Grund ist die funktionale Verifikation von TLM-Modellen unerlässlich. Hierfür wurden in der vorliegenden Dissertation zwei wesentliche Ergebnisse erzielt. Erstens wird die Kapazität der formalen Verifikation durch die vorgeschlagenen Techniken in vielen Fällen um mehrere Größenordnungen erhöht. Zweitens ist es mit den entwickelten Verfahren zur Fehlerlokalisierung erstmalig möglich, das Debugging auf der TLM-Abstraktion zu automatisieren. Weitere neuartige Ansätze runden die ganzheitliche Betrachtung des Themas funktionale Verifikation ab.

1 Einführung

Eingebettete Systeme werden heutzutage in unseren Alltag mehr und mehr integriert und übernehmen in vielen Fällen sicherheitskritische Funktionen (z.B. Herzschrittmacher oder Airbag-Steuerung). Das Versagen solcher Systeme kann sehr ernste, wirtschaftliche und menschliche Folgen haben. Daher ist es äußerst wichtig, die funktionale Korrektheit dieser Systeme vor deren Einsatz in der realen Welt zu gewährleisten. Dies ist eine sehr große Herausforderung angesichts der rasch zunehmenden Komplexität und des Time-to-Market-Drucks.

Ausgehend von seiner Spezifikation unterzieht sich jedes eingebettete System einem komplexen Entwurfsablauf, wobei sowohl Hardware- als auch Software-Entwicklung stattfinden. Der herkömmliche Entwurfsablauf beginnt mit der Entwicklung von HW-Modellen auf der *Register-Transfer-Ebene* (engl. Register Transfer Level, RTL) unter Verwendung einer Hardwarebeschreibungssprache (z.B. VHDL [IEE08] oder SystemVerilog [IEE12]).

Nach einem langwierigen und aufwändigen Prozess, wobei der größte Aufwand auf die *funktionale Verifikation* entfällt [WM12], entsteht ein physikalischer HW-Prototyp. Aufgrund der langsamen RTL-Simulation kann erst nach der Verfügbarkeit dieses Prototypen die SW-Entwicklung und Integration beginnen. Diese Abhängigkeit zusammen mit der

[1] Englischer Titel: "Automated Techniques for Functional Verification at the Electronic System Level" [Le15]

[2] Universität Bremen, AG Rechnerarchitektur, Bibliothekstraße 1, 28359 Bremen, hle@cs.uni-bremen.de

funktionalen Verifikation von HW sind zwei Engpässe, die nicht mit dem stetigen Komplexitätswachtum skalieren.

Die Anhebung der Abstraktionsebene ist eine bewährte Methode, um mit Komplexität umzugehen. In den letzten Jahren etablierte sich die so genannte *eletronische Systemebene* (engl. Electronic System Level, ESL) [BMP07]. Die HW-Modellierung auf dieser Ebene ist gängige Praxis in der Industrie geworden [Gh06, BG10]. Dies hat sie zum größten Teil der IEEE-standardisierten Systembeschreibungsprache SystemC [IEE11] und seinem frei verfügbaren Referenzsimulator von *Accellera Systems Initiative* (ASI) zu verdanken. SystemC bietet grundlegende Primitiven wie Prozesse, Module, Schnittstellen und Kanäle zusammen mit einem event-basierten Simulationskernel zur Modellierung auf unterschiedlichen Abstraktionsebenen. Primitiven für die Modellierung auf RTL wie Drähte, Signale, HW-Register, Taktzyklen, usw. sind ebenfalls vorhanden. Allerdings machen ESL HW-Modelle in SystemC von ihnen sehr wenig oder keinen Gebrauch. Die Abstraktion von taktgenauer Kommunikation und Datenpfaden auf Funktionsaufrufe und -parameter ist das zentrale Konzept der ESL-SystemC-Modellierung. Diese Technik ist als *Transaction Level Modeling* (TLM) bekannt und wurde bereits in SystemC standardisiert. Die ESL-Hardware-Modelle in SystemC werden daher auch oft als TLM-Modelle bezeichnet.

Aufgrund der Abstraktion bieten TLM-Modelle die gleichen Funktionalitäten wie RTL-Modelle aus der SW-Sicht. TLM-Modelle sind allerdings viel früher verfügbar und können viel schneller simuliert werden (bis zu Faktor 10.000 [BG10]). Die SW-Entwicklung kann somit viel früher beginnen, d.h. sobald ein *virtueller Prototyp* (VP) zur Verfügung steht, welcher aus TLM-Modellen der realen HW-Bausteine besteht. Darüberhinaus können mit Hilfe von TLM-Modellen eine Verifikationsumgebung entwickelt werden, um damit später RTL-Modelle zu verifizieren. Dabei dienen TLM-Modelle auch als Referenzmodelle.

Offensichtlich sind TLM-Modelle von großer Bedeutung, da mit ihnen die beiden genannten Engpässe in der Entwicklung deutlich gemildert werden. Allerdings haben Fehler in TLM-Modellen zwei wichtige Konsequenzen: SW-Entwicklung findet auf einem fehlerhaften VP statt und RTL-Modelle werden mit inkorrekten Referenzmodellen verifiziert. Somit können sich TLM-Fehler in SW/HW-Teile und möglicherweise auch ins Endprodukt fortpflanzen. Deshalb ist die Absicherung der funktionalen Korrektheit bereits auf der TLM-Ebene sehr wichtig, weil je früher Fehler gefunden werden, desto geringer ist der Aufwand, sie zu beheben. Diese Dissertation [Le15] liefert für die funktionale Verifikation von TLM-Modellen wichtige Beiträge, welche ein breites Spektrum von Grundlagenforschung bis zur direkten industriellen Anwendbarkeit abdecken. Die Beiträge werden in Abschnitt 3 näher erläutert.

2 Stand der Technik

Bevor die wissenschaftlichen Beiträge der Dissertation präsentiert werden, gibt dieser Abschnitt eine kompakte Darstellung vom Stand der Technik. Der Fokus liegt dabei auf den zwei wichtigste Aufgaben im Verifikationsablauf: *Verifkation* und *Debugging*. Funktionale Verifikation für RTL Modelle ist ein sehr ausgereiftes Forschungs- und Entwicklungsfeld mit der zugehörigen etablierten *Electronic Design Automation* (EDA) Industrie. Die

folgende Gegenüberstellung der verfügbaren Techniken auf den beiden Ebenen soll die fehlenden Teile auf Systemebene aufzeigen.

2.1 Simulationsbasierte Verifikation

Simulationsbasierte Verifikation ist und bleibt dank der hohen Benutzbarkeit und Skalierbarkeit die Standard-Technik für funktionale Verifikation in der Industrie. Zur Durchführung wird eine Verifikationsumgebung bzw. *Testbench* benötigt, welche mehrere Komponenten enthält. Diese fallen in eine der beiden folgenden Kategorien:

- Stimuligenerator: Generierung und Weiterleitung von Eingabewerten an das *Design-Under-Verification* (DUV) für die Simulation;
- Monitor und Checker: Überwachung der Ausgaben/Reaktion und internen Variablen vom DUV und Überprüfung ihrer Konformität mit der Spezifikation.

Dabei spielt das Konzept der constraint-basierten Verifikation (engl. Constrained Random Verification, CRV) [YPA06] eine entscheidende Rolle. CRV generiert Stimuli fürs DUV, welche Lösungen von logischen und arithmetischen Constraints darstellen. Die Lösungen werden mit Hilfe eines Constraint-Solvers ermittelt. Constraints werden von Verifikationsingenieuren formuliert, um den gesamten Eingaberaum auf die "interessanten" Werte zu reduzieren. CRV bietet zwei wesentliche Vorteile. Erstens ermöglicht CRV unerwartete Fehler im DUV zu finden, da Szenarien simuliert werden, an denen Verifikationsingenieuren nicht gedacht haben. Zweitens wird durch CRV die Stimuligenerierung vollautomatisiert und daher eine große Reihe von Szenarien überprüft. Für RTL Verifikation ist CRV-Testbench bereits als *Universal Verification Methodology* (UVM) [Acc14] standardisiert und wird von EDA-Firmen angeboten.

Zwei wesentliche CRV-Bausteine bilden ein effizienter Constraint-Solver und eine kompakte und dennoch ausdrucksstarke Constraint-Sprache. Für SystemC ist die einzige CRV-Option die *SystemC Verification* (SCV) Bibliothek [IS03] von ASI. Allerdings weist diese Bibliothek viele Schwächen in Bezug auf die beiden genannten Punkte auf und deshalb nur sehr eingeschränkt einsetzbar zur Verifikation von TLM-Modellen. Ein Beispiel ist die ausschließliche Nutzung von BDDs [Br86] für das Constraint-Solving, die typischerweise große Mengen von Constraints in TLM Testbenches nicht lösen kann. Existierende Verbesserungen von SCV [GED07, Wi09] kompensierten die Schwachpunkte bis zu einem gewissen Grad, allerdings wurden die begrenzte Ausdrucksmächtigkeit und Nutzbarkeit der Spezifikation von Constraints noch nicht behandelt.

2.1.1 Formale Verifikation

Trotz der hohen Skalierbarkeit haben simulationsbasierte Techniken eine fundamentale Schwäche. Fehler werden nur gefunden, wenn entsprechende Stimuli vorliegen. CRV löst diese Problem nur teilweise, weil es praktisch unmöglich ist, alle möglichen Stimuli zu simulieren. Formale Verifikation, hier in Form der *Eigenschaftsprüfung*, hingegen versucht,

die Abwesenheit von Fehlern mathematisch zu beweisen. Im Gegensatz zu Simulation ist ein Testbench nicht erforderlich. Ein Eigenschaftsprüfer liest die RTL-Beschreibung und eine Eigenschaft. Danach wird der Zustandsraum des DUV erschöpfend exploriert. Dabei kann entweder beweisen werden, dass die Eigenschaft für alle gültigen Eingangsstimuli gilt, oder ein Gegenbeispiel (d.h. eine konkrete Folge von Stimuli) gefunden werden, die zu einer Verletzung der Eigenschaft führen. Einst nur vom akademischem Interesse, werden RTL-Eigenschaftsprüfer von EDA-Firmen angeboten. Der erfolgreichste Algorithmus zur Eigenschaftsprüfung ist *Bounded Model Checking* (BMC) [Bi03] unter Verwendung von SAT-Solvern [Mo01, ES03] (d.h. Solver für das Boolesche Erfüllbarkeitsproblem). BMC rollt das RTL-DUV zusammen mit der zu beweisenden Eigenschaft für eine bestimmte Anzahl von Zeitschritten ab und übersetzt diese abgerollte Beschreibung in eine SAT-Instanz, welche nur erfüllbar ist, wenn die Eigenschaft nicht gilt.

Für den Einsatz formaler Verifikation auf der TLM-Abstraktion stellen die Objektorientierung und die event-basierte Simulationssemantik von SystemC eine große Herausforderung dar [Va07]. Zunächst muss ein TLM-Eigenschaftsprüfer offensichtlich alle möglichen Stimuli betrachten. Zweitens besteht das Gesamtverhalten eines typischen TLM-DUV aus mehreren asynchronen Prozessen. Jede mögliche Ausführungsreihenfolge der Prozessen kann potentiell zu einem anderen Ergebnis führen. Deshalb müssen ein TLM-Eigenschaftsprüfer auch alle Ausführungsreihenfolgen berücksichtigen. Drittens findet man in TLM-Modellen neben SystemC-Primitiven auch beliebige C++-Konstrukte. Infolgedessen muss ein TLM-Eigenschaftsprüfer in der Lage sein, ein formales Modell unter Betrachtung der vollen Komplexität von C++ zu extrahieren.

Für die Extrahierung existiert bis heute keine zufriedenstellende Lösung trotz vieler Versuche z.B. [Fe04, FZI, MMMC05]. Es existiert nichtsdestotrotz bereits eine große Anzahl formaler Verifikationsansätze für SystemC TLM, die unterschiedliche kleine Untermenge der Sprachkonstrukte unterstützen. Viele der existierenden Ansätze beschränken sich nur auf das oben erwähnte Problem der vielen möglichen Ausführungsreihenfolgen der asynchronen Prozesse. Diese Lösungen [KGG08, HMM09] versuchen, unter einer vorgegebenen Eingabe die Ausführungsreihenfolgen möglichst effizient zu explorieren. Dabei wird auf Varianten von *Partial Order Reduction* (POR) Techniken [Go96, FG05] zurückgegriffen. Generell versuchen POR-Techniken, äquivalenten Reihenfolgen zu erkennen und wiederholte Explorationen zu vermeiden. Weitere Ansätze [KS05, KEP06, Tr07, HFG08] betrachten dazu auch noch alle möglichen Eingaben und verwenden dabei viele verschiedene Formalismen unter anderem Petri-Netze, Promela oder Timed Automata. Danach wird in den meisten Fällen ein expliziter Modellprüfer zur Verifikation eingesetzt. Explizite Modellprüfung [CGP99] hat bekanntlich Skalierbarkeitsprobleme bei Modellen, die Variablen mit großen Wertebereichen besitzen.

2.1.2 Debugging

Wenn ein Fehler festgestellt wird, sei es durch simulationsbasierte oder formale Verifikation, wird ein Fehlerprotokoll bzw. Gegenbeispiel geliefert. Beim traditionellen Debugging untersuchen Verifikationsingenieure manuell dieses Fehlerprotokoll mit Hilfe eines Debuggers, um die Fehlerursache zu finden. Automatisierte Debugging-Techniken reduzie-

ren diesen manuellen Aufwand, indem sie automatisch eine Liste von Fehlerkandidaten im DUV berechnen und den Ingenieuren präsentieren. In dieser Arbeit werden solche Techniken als *Fehlerlokalisierung* bezeichnet.

Auf der RTL-Abstraktion ist die Fehlerlokalisierung recht gut verstanden. Mehrere skalierbare Ansätze [SVV04, Fe08, SV09] existieren, die die Fehlerlokalisierung als Boolesches Erfüllbarkeitsproblem formulieren und somit die Leistungsfähigkeit moderner SAT-Solver ausnutzen können. Zur Fehlerlokalisierung in TLM-Modellen war vor dieser Dissertation kein Verfahren bekannt.

3 Funktionale Verification von TLM-Modellen

Im Rahmen dieser Dissertation wurden wichtige Beiträge zur Weiterentwicklung der funktionalen Verifikation auf der TLM-Abstraktion geleistet. Die Beiträge adressieren vor allem die im vorherigen Abschnitt identifizierten Schwachpunkte:

1. Ein neuer constraint-basierter Stimuligenerator wurde für die simulationsbasierte TLM-Verifikation entwickelt, welcher im Vergleich zum Stand der Technik deutlich effizienter ist und wesentlich ausdrucksstärkere Constraints unterstützt;
2. Neue Ansätze für TLM-Eigenschaftsprüfung wurden entwickelt, die existierende Lösungen leistungsmäßig weitaus übertreffen;
3. Neue Fehlerlokalisierungsverfahren wurden entwickelt, welche es erstmalig erlauben, für TLM-Modelle Fehlerstellen automatisch und genau zu bestimmen.

Die einzelnen Punkte werden im Folgenden kurz näher beschrieben.

3.1 Constraint-basierte Stimuligenerierung

Wir entwickelten eine Bibliothek namens CRAVE (Constrained RAndom Verification Environment) für SystemC. CRAVE ist komplett quelloffen[3] und bietet viele neue Funktionalitäten, die in SCV nicht ohne weiteres möglich sind:

- Eine neue und vollständig überarbeitete API zur Constraint-Spezifikation;
- Dynamische Constraints, die zur Laufzeit gesteuert werden können;
- Automatische Diagnose von widersprüchlichen Constraints;
- Constraints für dynamische Datenstrukturen (z.B. STL-Vektoren), Soft-Constraints und Verteilungsconstraints;
- Parallelisiertes Constraint-Solving;
- automatische Partitionierung von Constraints in unabhängige Gruppen.

[3] erhältlich unter `http://www.systemc-verification.org/crave`

Die ersten drei Funktionalitäten verbessern die Benutzerfreundlichkeit, die vierte die Ausdrucksmächtigkeit der Spezifikation von Constraints. Die letzten beiden Funktionalitäten beschleunigen den Constraint-Solving-Prozess. Die Konzepte, die Einzelheiten der Umsetzung sowie die experimentellen Ergebnisse wurden in [Ha12, LD14a] veröffentlicht. Darüber hinaus integrierten wir CRAVE in das Framework der *System Verification Methodology* (SVM) [Ol12b] und demonstrierten in einer Fallstudie [Ol12a] die Anwendbarkeit dieser Integration an einem abstrakten Modell eines Zweirad-Elektrofahrzeugs, das von der Infineon Technologies München zur Verfügung gestellt wurde.

3.2 Formale Eigenschaftsprüfung

Wir präsentierten in [GLD10] den ersten Ansatz zur TLM-Eigenschaftsprüfung, der eine vollständige und effiziente Verifikation von "richtigen" TLM Eigenschaften ermöglicht. Neben einfachen Sicherheitseigenschaften kann der Anwender die Wirkungskette von Transaktionen und Events spezifizieren und verifizieren. Der Ansatz basiert auf einer neuartigen und vollständig automatisierten Übersetzung des DUV von TLM nach C. Während der Übersetzung wird ein Monitor aus der zu verifizierenden Eigenschaft erzeugt und als Zusicherungen in das C-Modell eingebettet. Die Verifikation selbst wird durch eine neuartige Variante von BMC und k-Induktion auf der Ebene des C-Codes ausgeführt. In [LGD11] beschrieben wir eine Erweiterung, die TLM-Eigenschaften mit lokalen Variablen zwecks Datenintegritätsprüfung unterstützen.

Um die Effizienz der Eigenschaftsprüfung weiter zu verbessern, schlugen wir in [Le13] einen komplementären Ansatz vor. Die Neuheit an diesem Ansatz zur symbolischen Simulation ist die intelligente Kombination zwei effizienter Verifikationstechniken für SW: POR [Go96, FG05] und symbolische Ausführung [Ki76, CDE08], unter Berücksichtigung der Nebenläufigkeit von SystemC. Während POR redundante Ausführungsreihenfolgen abschneidet, exploriert die symbolische Ausführung alle bedingten Ausführungspfade in Verbindung mit symbolischen Eingabewerten durch. Damit deckt eine symbolische Simulation den gesamten Zustandsraum des DUV ab. Vor kurzem präsentierten wir in [HLD15] eine wichtige Erweiterung dieses Ansatzes, um zyklische Zustandsräume zu unterstützen. Diese Räume entstehen natürlicherweise in vielen TLM-Modellen durch den häufigen Einsatz unbegrenzter Schleifen in SystemC-Prozessen.

Die experimentellen Ergebnisse in [GLD10, Le13, HLD15] zeigten, dass unsere Ansätze existierende Techniken zur TLM-Eigenschaftsprüfung leistungsmäßig weitaus übertreffen, in vielen Fällen um mehrere Größenordnungen.

3.3 Fehlerlokalisierung

In [LGD12] stellten wir den ersten Ansatz zur Fehlerlokalisierung für TLM-Modlle vor. Der Ansatz berechnet anhand eines Fehlerprotokolls automatisch alle Fehlerkandidaten. Jeder Kandidat ist eine Anweisung, die TLM-Primitiven enthält und oft von Entwicklern falsch angewendet wird. Zum Beispiel wird fälschlicherweise eine blockierende Transaktion anstelle einer nicht-blockierenden verwendet oder ein falsches Event wird benachrich-

tigt. Alle Kandidaten besitzen die Eigenschaft, dass eine syntaktische Änderung an ihnen dazu führen kann, dass das Fehlerprotokoll nicht mehr zu reproduzieren ist. Die Berechnung der Fehlerkandidaten ist auf den oben beschrieben formalen Verifikationsansätzen aufgebaut. Die experimentellen Ergebnisse in [LGD12] zeigten, dass dieser Ansatz in den meisten Fällen genau und relativ schnell die Fehlerstelle lokalisieren kann.

Da die komplexesten TLM-Modelle mit ihren riesigen Zustandsräumen nicht komplett mit formalen Methoden behandelbar sind, schlugen wir in [LGD13] ein besonders skalierbares Verfahren zur Fehlerlokalisierung für TLM-Modelle vor. Das neue Verfahren ist durch die Diagnose-Techniken für SW inspiriert, simulationsbasiert und leicht in eine typische Verifikationsumgebung zu integrieren. Wir erweiterten das Konzept der Ausführungsprofile (z.B. in der einfachsten Ausprägung: Anweisungsabdeckung) von SW-Programmen für TLM-Simulationen. Während der Simulation sammelt das Verfahren Ausführungsprofile für jeden einzelnen Eingabe-Berechnung-Ausgabe-Pfad. Dann werden mögliche Fehlerstellen anhand der Unterschiede von Ausführungsprofilen zwischen fehlerfreien und fehlerbehafteten Läufen berechnet. Die experimentellen Ergebnisse in [LGD13] zeigten, dass die Fehlerstellen genau und sehr schnell identifiziert wurden.

3.4 Anwendungen

Über die Kerntechniken hinaus schlugen wir zwei auf TLM-Eigenschaftsprüfung basierte Anwendungen vor, die wiederum die funktionale Verifikation effizienter machen.

Die erste Anwendung [LGD10] ist darauf ausgelegt, Lücke in einem TLM-Eigenschaftssatz zu finden. Dieses Problem ist bereits für RTL-Eigenschaftsprüfung bekannt (“habe ich genug Eigenschaften geschrieben?” [KG99]). In einem TLM-Modell sollte jede mögliche Transaktionsinitierung in mindestens einer Eigenschaft eindeutig beschrieben sein. Eine nicht beschriebene Initierung weist auf Lücke in dem Eigenschaftssatz hin. Unser Hauptbeitrag ist die Formulierung dieses Problems als eine Instanz der TLM-Eigenschaftsprüfung.

Die zweite Anwendung [LD14b] zielt darauf ab, die Eingabe-Ausgabe-Determiniertheit eines TLM-DUV nachzuweisen. Das bedeutet, dass das TLM-DUV für jede mögliche Eingabe die gleiche Ausgabe unter allen Ausführungsreihenfolgen liefert. Determiniertheit ist ein wertvolles Korrektheitskriterium und bietet mehrere Vorteile in Bezug auf sowohl simulationsbasierte als auch formale Verifikation. Wir implementierten einen Prototypen zur Demonstration der Determiniertheitsprüfung für TLM-Modelle. Der Prototyp führt eine Determiniertheitsprüfung darauf zurück, die Äquivalenz der Ausgaben von zwei Versionen des Original-DUV mit einem TLM-Eigenschaftsprüfer zu checken.

4 Zusammenfassung und Ausblick

Die Entstehung und industrieweite Verbreitung der neuen Abstraktionsebene ESL hat zur Produktivitätssteigerung im Entwurfsablauf eingebetteter Systeme geführt. Aber auch die abstrakten TLM-Modelle auf Systemebene bedürfen gründlicher Prüfung ihrer Korrektheit. In dieser Arbeit stellten wir eine Reihe von neuen automatisierten Ansätze zur funktionalen Verifikation von TLM-Modellen vor. Die Ansätze leisteten wichtige Beiträge

dazu, den Verifikationsablauf auf der neuen Abstraktionsebene näher an den Reifegrad der funktionale Verifikation von RTL-Modellen zu bringen. Zum Zeitpunkt der Erstellung dieses Textes ist die Verifikationsbibliothek CRAVE bereits im Integrationsprozess in die standardisierte Verifikationsmethodik UVM-SystemC. Es ist zu erwarten, dass die anderen hier vorgestellten Techniken in den nächsten Jahren auch ihren Weg in den industriellen Einsatz finden.

Literaturverzeichnis

[Acc14] Accellera Systems Initiative. Universal Verification Methodology, 2014.

[BG10] Bailey, B.; Grant, M.: ESL Models and their Applications. Springer, 2010.

[Bi03] Biere, A.; Cimatti, A.; Clarke, E. M.; Strichman, O.; Zhu, Y.: Bounded model checking. Advances in Computers, 58:118–149, 2003.

[BMP07] Bailey, B.; Martin, G.; Piziali, A.: ESL Design and Verification: A Prescription for Electronic System Level Methodology. Morgan Kaufmann/Elsevier, 2007.

[Br86] Bryant, R. E.: Graph-Based Algorithms for Boolean Function Manipulation. IEEE Trans. on Comp., 35(8):677–691, 1986.

[CDE08] Cadar, Cristian; Dunbar, Daniel; Engler, Dawson R.: KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs. In: OSDI. S. 209–224, 2008.

[CGP99] Clarke, E. M.; Grumberg, O.; Peled, D.: Model Checking. MIT Press, 1999.

[ES03] Eén, N.; Sörensson, N.: An Extensible SAT-solver. In: SAT. S. 502–518, 2003.

[Fe04] Fey, G.; Große, D.; Cassens, T.; Genz, C.; Warode, T.; Drechsler, R.: ParSyC: An Efficient SystemC Parser. In: Workshop on Synthesis And System Integration of Mixed Information technologies. S. 148–154, 2004.

[Fe08] Fey, G.; Staber, S.; Bloem, R.; Drechsler, R.: Automatic Fault Localization for Property Checking. IEEE Trans. on CAD, 27(6):1138–1149, 2008.

[FG05] Flanagan, Cormac; Godefroid, Patrice: Dynamic partial-order reduction for model checking software. In: POPL. S. 110–121, 2005.

[FZI] FZI. KaSCPar - Karlsruhe SystemC Parser Suite.

[GED07] Große, D.; Ebendt, R.; Drechsler, R.: Improvements for constraint solving in the SystemC verification library. In: ACM Great Lakes Symposium on VLSI. S. 493–496, 2007.

[Gh06] Ghenassia, F.: Transaction-Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems. Springer, 2006.

[GLD10] Große, D.; Le, H. M.; Drechsler, R.: Proving Transaction and System-level Properties of Untimed SystemC TLM Designs. In: ACM & IEEE International Conference on Formal Methods and Models for Codesign. S. 113–122, 2010.

[Go96] Godefroid, Patrice: Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem. Springer, 1996.

[Ha12] Haedicke, F.; Le, H. M.; Große, D.; Drechsler, R.: CRAVE: An Advanced Constrained RAndom Verification Environment for SystemC. In: Int'l Symposium on System-on-Chip. S. 1–6, 2012.

[HFG08] Herber, P.; Fellmuth, J.; Glesner, S.: Model checking SystemC designs using timed automata. In: Int'l Conference on Hardware/Software Codesign and System Synthesis. S. 131–136, 2008.

[HLD15] Herdt, V.; Le, H. M.; Drechsler, R.: Verifying SystemC using Stateful Symbolic Simulation. In: Design Automation Conf. 2015.

[HMM09] Helmstetter, Claude; Maraninchi, Florence; Maillet-Contoz, Laurent: Full simulation coverage for SystemC transaction-level models of systems-on-a-chip. Formal Methods in System Design: An International Journal, 35(2):152–189, 2009.

[IEE08] IEEE Std. 1076. IEEE Standard VHDL LRM, 2008.

[IEE11] IEEE Std. 1666. IEEE Standard SystemC LRM, 2011.

[IEE12] IEEE Std. 1800. IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language, 2012.

[IS03] Ip, C. Norris; Swan, S.: A Tutorial Introduction on the New SystemC Verification Standard. White paper, www.systemc.org, 2003.

[KEP06] Karlsson, D.; Eles, P.; Peng, Z.: Formal verification of SystemC designs using a petri-net based representation. In: Design, Automation and Test in Europe. S. 1228–1233, 2006.

[KG99] Katz, S.; Grumberg, O.: Have I written enough Properties - A Method of Comparison between Specification and Implementation. In: Correct Hardware Design and Verification Methods. S. 280–297, 1999.

[KGG08] Kundu, S.; Ganai, M.; Gupta, R.: Partial order reduction for scalable testing of SystemC TLM designs. In: Design Automation Conf. S. 936–941, 2008.

[Ki76] King, James C.: Symbolic Execution and Program Testing. Commun. ACM, 19(7):385–394, Juli 1976.

[KS05] Kroening, D.; Sharygina, N.: Formal verification of SystemC by automatic hardware/-software partitioning. In: ACM & IEEE International Conference on Formal Methods and Models for Codesign. S. 101–110, 2005.

[LD14a] Le, H. M.; Drechsler, R.: CRAVE 2.0: The Next Generation Constrained Random Stimuli Generator for SystemC. In: Design and Verification Conference and Exhibition Europe (DVCon Europe). 2014.

[LD14b] Le, H. M.; Drechsler, R.: Towards Verifying Determinism of SystemC Designs. In: Design, Automation and Test in Europe. S. 153:1–153:4, 2014.

[Le13] Le, H. M.; Große, D.; Herdt, V.; Drechsler, R.: Verifying SystemC using an Intermediate Verification Language and Symbolic Simulation. In: Design Automation Conf. S. 116:1–116:6, 2013.

[Le15] Le, Hoang M.: Automated Techniques for Functional Verification at the Electronic System Level. Dissertation, Universitity of Bremen, Germany, 2015.

[LGD10] Le, H. M.; Große, D.; Drechsler, R.: Towards Analyzing Functional Coverage in SystemC TLM Property Checking. In: IEEE International High Level Design Validation and Test Workshop. S. 67–74, 2010.

[LGD11] Le, H. M.; Große, D.; Drechsler, R.: Towards Proving TLM Properties with Local Variables. In: Int'l Workshop on Constraints in Formal Verification. 2011.

[LGD12] Le, H. M.; Große, D.; Drechsler, R.: Automatic TLM Fault Localization for SystemC. IEEE Trans. on CAD, 31(8):1249–1262, August 2012.

[LGD13] Le, H. M.; Große, D.; Drechsler, R.: Scalable Fault Localization for SystemC TLM Designs. In: Design, Automation and Test in Europe. S. 35–38, 2013.

[MMMC05] Moy, Matthieu; Maraninchi, Florence; Maillet-Contoz, Laurent: Pinapa: An Extraction Tool for SystemC Descriptions of Systems-on-a-chip. In: ACM Iternational Conference on Embedded Software. S. 317–324, 2005.

[Mo01] Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; Zhang, L.; Malik, S.: Chaff: Engineering an Efficient SAT Solver. In: Design Automation Conf. S. 530–535, 2001.

[Ol12a] Oliveira, M. F. S.; Kuznik, C.; Le, H. M.; Große, D.; Haedicke, F.; Mueller, W.; Drechsler, R.; Ecker, W.; Esen, V.: The System Verification Methodology for Advanced TLM Verification. In: Int'l Conference on Hardware/Software Codesign and System Synthesis. S. 313–322, 2012.

[Ol12b] Oliveira, M. F. S.; Kuznik, C.; Mueller, W.; Ecker, W.; Esen, V.: A SystemC Library for Advanced TLM Verification. In: Design and Verification Conference and Exhibition (DVCon). 2012.

[SV09] Safarpour, Sean; Veneris, Andreas: Automated design debugging with abstraction and refinement. IEEE Trans. on CAD, 28:1597–1608, Oct. 2009.

[SVV04] Smith, A.; Veneris, A.; Viglas, A.: Design Diagnosis Using Boolean Satisfiability. In: ASP Design Automation Conf. S. 218–223, 2004.

[Tr07] Traulsen, C.; Cornet, J.; Moy, M.; Maraninchi, F.: A SystemC/TLM Semantics in Promela and Its Possible Applications. In: SPIN. S. 204–222, 2007.

[Va07] Vardi, M. Y.: Formal techniques for SystemC verification. In: Design Automation Conf. S. 188–192, 2007.

[Wi09] Wille, R.; Große, D.; Haedicke, F.; Drechsler, R.: SMT-based Stimuli Generation in the SystemC Verification Library. In: Forum on specification and Design Languages. S. 1–6, 2009.

[WM12] The 2012 Wilson Research Group Functional Verification Study.

[YPA06] Yuan, J.; Pixley, C.; Aziz, A.: Constraint-based Verification. Springer, 2006.

Hoang M. Le erhielt 2009 sein Diplom mit Auszeichnung in Informatik von der Universität Bremen. Seit November 2009 ist er als wissenschaftlicher Mitarbeiter in der Arbeitsgruppe Rechnerarchitektur an der Universität Bremen tätig. Im Juli 2016 schloss er seine Promotion erfolgreich mit dem Prädikat *summa cum laude* ab. Für sein Dissertationsprojekt erhielt er bereits zwei Auszeichnungen auf dem PhD-Forum von zwei der bedeutendsten Konferenzen auf seinem Forschungsgebiet: der Asia and South Pacific Design Automation Conference (ASP-DAC) in 2012 und der Design, Automation and Test in Europe (DATE) in 2013.