TECHNISCHE UNIVERSITÄT
CHEMNITZ

Fakultät für Elektro- und Informationstechnik
Institut für Informationstechnik
Professur Schaltkreis- und Systementwurf

MBMV 2015 — Tagungsband

Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen

Chemnitz, 03. – 04. März 2015

Editoren

Ulrich Heinkel
Daniel Kriesten
Marko Rößler

Steinbeis-Forschungszentrum Systementwurf und Test

Impressum

Kontaktadressen

Technische Universität Chemnitz
Professur Schaltkreis- und Systementwurf
D-09107 Chemnitz

Steinbeis-Stiftung für Wirtschaftsförderung (StW)
– Steinbeis-Forschungszentrum Systementwurf und Test –
Haus der Wirtschaft
Willi-Bleicher-Str. 19
D-70174 Stuttgart

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Angaben sind im Internet über `http://dnb.d-nb.de` abrufbar.

ISBN 978-3-944640-34-1

Vorwort

Der Workshop *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen* (MBMV 2015) findet nun schon zum 18. mal statt. Ausrichter sind in diesem Jahr die Professur Schaltkreis- und Systementwurf der Technischen Universität Chemnitz und das Steinbeis-Forschungszentrum Systementwurf und Test.

Der Workshop hat es sich zum Ziel gesetzt, neueste Trends, Ergebnisse und aktuelle Probleme auf dem Gebiet der Methoden zur Modellierung und Verifikation sowie der Beschreibungssprachen digitaler, analoger und Mixed-Signal-Schaltungen zu diskutieren. Er soll somit ein Forum zum Ideenaustausch sein.

Weiterhin bietet der Workshop eine Plattform für den Austausch zwischen Forschung und Industrie sowie zur Pflege bestehender und zur Knüpfung neuer Kontakte. Jungen Wissenschaftlern erlaubt er, ihre Ideen und Ansätze einem breiten Publikum aus Wissenschaft und Wirtschaft zu präsentieren und im Rahmen der Veranstaltung auch fundiert zu diskutieren. Sein langjähriges Bestehen hat ihn zu einer festen Größe in vielen Veranstaltungskalendern gemacht. Traditionell sind auch die Treffen der ITG Fachgruppen an den Workshop angegliedert.

In diesem Jahr nutzen zwei im Rahmen der InnoProfile-Transfer-Initiative durch das Bundesministerium für Bildung und Forschung geförderte Projekte den Workshop, um in zwei eigenen Tracks ihre Forschungsergebnisse einem breiten Publikum zu präsentieren. Vertreter der Projekte *Generische Plattform für Systemzuverlässigkeit und Verifikation* (GPZV) und *GINKO — Generische Infrastruktur zur nahtlosen energetischen Kopplung von Elektrofahrzeugen* stellen Teile ihrer gegenwärtigen Arbeiten vor. Dies bereichert den Workshop durch zusätzliche Themenschwerpunkte und bietet eine wertvolle Ergänzung zu den Beiträgen der Autoren.

Unser Dank gilt allen den Autoren, die Beiträge eingereicht haben. Wir freuen uns, dass die Kritik der Gutachter aufgenommen und in den finalen Versionen umgesetzt wurde. Auch hierfür sprechen wir den Autoren der akzeptierten Beiträge unseren Dank aus. Ebenso bedanken wir uns bei den Gutachtern für die sehr konstruktive Zusammenarbeit und die zielführenden, kritischen und fundierten Gutachten, die den Autoren als eine solide Grundlage für die Überarbeitung ihrer Beiträge diente. Wir freuen uns, dass wir mit Herrn Uwe Grüner und Herrn Markus Goertz zwei erfahrene Ingenieure für die Keynotes gewinnen konnten, die einen Einblick in die industrielle Praxis geben. Beiden Sprechern danken wir herzlich für ihre Beiträge. Wir bedanken uns bei unseren Kollegen Prof. Göran Herrmann, Dr. Erik Markert und Dr. Marco Dienel für die geleistete Zuarbeit und die guten Hinweise, die die Vorbereitungen an vielen Stellen erleichtert haben. Ein besonderer Dank geht an Frau Silvia Eppendorfer für ihre tatkräftige Unterstützung bei der Planung und Umsetzung des Workshops.

Wir hoffen, den Autoren und Zuhörern auch in diesem Jahr eine rundum gelungene Veranstaltung mit spannenden Beiträgen und Diskussionen bieten zu können.

Ulrich Heinkel, Daniel Kriesten, Marko Rößler

Chemnitz, März 2015

Inhaltsverzeichnis

ISBN 978-3-00-048889-4

Verfahren zur Assertion basierten Verifikation bei der High-Level-Synthese

Christian Schott und Marko Rößler und Ulrich Heinkel
Professur für Schaltkreis- und Systementwurf
Fakultät für Elektrotechnik und Informationstechnik, Technische Universität Chemnitz
09107 Chemnitz
{christian.schott,marko.roessler,ulrich.heinkel}@etit.tu-chemnitz.de

Zusammenfassung

Den Herausforderungen durch immer komplexere Systeme und Schaltkreise wird durch Entwurfsautomatisierung begegnet. Die Erhöhung der Abstraktionsebene beim Entwurf und die automatisierte Verifikation sind dabei wesentliche Eckpfeiler. Aktuelle Werkzeuge zur High-Level-Synthese (HLS) unterstützen Assertions nicht. Im Beitrag werden Verfahren zur Umsetzung von High-Level-Assertions diskutiert und ein Verfahren anhand von Beispielen mit mehreren HLS-Werkzeugen umgesetzt. Die Ergebnisse zeigen, dass aus den Assertions Monitore für die Schaltkreisimplementierung generiert werden können und die zusätzlichen Aufwände hinsichtlich Ressourcen und Zeitverzögerung begrenzt sind.

1. Einführung

Die stetige Verkleinerung der Strukturbreiten führt zu größeren und komplexeren Schaltungen in den Schaltkreisen der Halbleiterindustrie. Diese bilden die Grundlage für die effiziente Lösung aktueller Berechnungs- und Kommunikationsaufgaben. In diesem Zusammenhang steigt auch die Komplexität des Entwurfes und der Verifikation. Nimmt man die exponentiell steigenden NRE-Kosten (engl. Non-Recurring Engineering costs / Einmalkosten) für den Lauf eines Schaltkreises durch eine Fabriklinie hinzu, erklärt sich, dass in aktuellen anwendungsbezogenen Schaltkreisprojekten bis zu 70 Prozent der Aufwände im Bereich Verifikation und Test anfallen [Ber06]. Im Entwurfsprozess wird dem durch Anhebung der Abstraktionsebene, beispielsweise durch die Synthese von Verhaltensbeschreibungen, begegnet. Die Verifikation wird durch Formalisierung und Automatisierung effektiviert. Mit Assertion basierter Verifkation (ABV, [FKL04]) existiert ein Verfahren, bei dem kritische Rahmenbedingungen während des Entwurfsprozesses formuliert werden, die später durch Simulation, formale Prüfung oder Emulation automatisiert geprüft werden.

Eine Alternative zum ASIC bilden FPGA, die heute hinsichtlich Leistungsaufnahme und Performanz mit Mikrocontrollern konkurrieren. Dank der Entwicklungen auf dem Gebiet der High-Level-Synthese (HLS) ist der Anwendungsentwurf im Bereich der digitalen Hardware in Hochsprachen wie C, C++, Java oder Matlab/Simulink inzwischen auch für Softwareingenieure beherrschbar und gewinnt ein breites Einsatzfeld. Im Vergleich zur Softwareentwicklung besteht jedoch noch

ISBN 978-3-00-048889-4

immer eingeschränkte Unterstützung bei Verifikation und Debugging, insbesondere im Hinblick auf Assertions.

Behauptungen (engl. Assertions) können als formale Übersetzung spezifizierter Anforderungen (engl. Requirements) angesehen werden, die einen Nachweis der Übereinstimmung einer Implementierung mit der Spezifikation auf verschiedenen Abstraktionsebenen erlauben. Während der Entwicklung erleichtern Assertions die Fehlersuche und erhöhen damit die Entwurfsproduktivität, weil Simulatoren und formale Werkzeuge den Fehler an einer konkreten Stelle der Entwurfsbeschreibung lokalisieren können. Ein weiterer Vorteil ergibt sich bei der Wiederverwendung von Code-Teilen beziehungsweise Designblöcken, indem die anforderungskonforme Nutzung der Blockschnittstellen durch Anwendung der Assertions sichergestellt werden kann.

HLS-Werkzeuge für den Entwurf auf hohen Abstraktionsebenen unterstützen heute nur bedingt den Ansatz der ABV. In diesem Beitrag werden Methoden analysiert, mit denen Assertions beim HLS-Prozess berücksichtigt werden können. Weiterhin wird die Frage untersucht, inwiefern eine von den HLS-Werkzeugen unabhängige Möglichkeit zur Realisierung von ABV besteht. Hintergrund ist zum Einen, dass die HLS-Werkzeuge häufig für spezifische Technologien ausgelegt sindund der Ausgangspunkt von Schaltungsentwicklungen zum Anderen oft in einem generischen Hochsprachenmodell liegt. Während *Vivado-HLS* ausschließlich auf FPGA von Xilinx abzielt, ist *Calypto CatapultC* auf den ASIC Markt gerichtet. Der *CoDeveloper* von *Impulse Accelerated Technologies* ist auf mikrocontrollerintegrierte HW/SW-Systeme mehrerer FPGA-Hersteller fokusiert. Ziel der Arbeiten ist es, aus Assertion-Statements einer Hochsprache auf Verhaltensebene, Monitore auf RTL-Ebene zu erzeugen, die dann sowohl im Simulator als auch in der resultierenden Schaltungslogik integriert werden können.

Ausgehend von einer Analyse über mögliche Verfahren zur Integration wird in diesem Beitrag ein erster Schritt zur werkzeugübergreifenden Realisierung vorgestellt. Im Fokus liegen dabei zunächst die kommerziellen HLS-Werkzeuge CatapultC, ImpulseC und Vivado. Das gewählte Verfahren wandelt die Assertion-Statements in synthetisierbare Ausdrucke, die dann als Monitore durch die HLS-Synthese in VHDL-Ausdrücke überführt werden. Im Rahmen des Beitrages werden schließlich die Kosten hinsichtlich zusätzlich notwendiger HW-Ressourcen sowie der Einfluss auf das Timing und die Performance der Implementierung anhand von drei Beispielen bestimmt.

2. Stand der Technik

Beim Entwurf von digitaler Hardware finden Assertions im Zusammenhang mit der Methodik „Assertion based Verification“ bereits breite Anwendung. Auf Registertransfer-Ebene lassen sich Prüfausdrücke sowohl in den üblichen Hardwarebeschreibungssprachen (VHDL, Verilog) als auch in den Systemmodellierungs- und Verifikationsumgebungen wie Systemverilog (SVA), Open Verification Library (OVL) oder der Property Specification Language (PSL) formulieren, die bei der Simulation dann geprüft werden. Verschiedene Gruppen zeigen die Umsetzung der Prüfausdrücke in Monitore, die über die Simulation hinaus in die Zielimplementierung integriert werden und dort auch zur Laufzeit im Feld eine Prüfung der Ausdrücke ermöglichen. Zu nennen sind für die InCiruit-Monitore im Bereich des ASIC-Entwurfs die Arbeiten [PLBN05] für SVA, [KRM08] für OVL und [BCZ07] für PSL.

ISBN 978-3-00-048889-4

Auch für den Bereich der High-Level-Synthese gibt es Arbeiten zur Verbesserung der Debug- und Verifikationsmöglichkeiten. Goeders et. al. zeigen in [GW14] eine erweiterte Analysemöglichkeit, die es erlaubt die synthetisierte Hardwareimplementierung im Single-Step-Mode zu durchlaufen und dabei den Bezug zur Hochsprachenbeschreibung herzustellen. In [RLGJD11] zeigen Ribon et. al. die Integration und Propagierung von PSL-Ausdrücken von der Hochsprache bis in die RT-Simulation. Die Untersuchung von Curreri et. el. [CSG10] beschäftigen sich bereits mit der Behandlung von C-Assertions für das HLS-Werkzeug CoDeveloper.

Nach bestem Wissen und Gewissen gibt es aktuell keine Arbeit die Assertions aus Hochsprache mittels HLS als Monitore in der finalen Implementierung integriert und den damit einhergehenden Overhead übergreifend für mehrere HLS-Werkzeuge untersucht.

3. Methoden zur Integration von Assertions in die Verhaltenssynthese

Im Folgenden gehen wir von der Notation einer Assertion in folgender Form:

```
void assert(int expression);
```

aus. Das entspricht der Deklaration aus der C-Standard-Bibliothek im Header-File assert.h . Der Ausdruck innerhalb des Aufrufes wird geprüft. Der Erwartungswert ist dabei *Wahr* (entspricht einem Wert von ungleich 0). Ist die Erwartung nicht erfüllt, wird der Ablauf gestoppt und ein Fehler nach Listing 1 gemeldet, der die entsprechende Zeilennummer im Quellcode beinhaltet.

```
Assertion violation: file tripe.c, line 34
```

Listing 1: Meldung einer fehlgeschlagenen Assertion

Dieses Verhalten im Rahmen der High-Level-Synthese umzusetzen, ist auf die in Abbildung 1 dargestellten Verfahren 1-3 möglich. Daraus resultieren unterschiedliche Auswirkungen hinsichtlich der resultierenden Implementierung als InCircuit-Monitor und in Bezug auf die werkzeugübergreifende Nutzbarkeit des Ansatzes.

Bei Verfahren 1 ersetzt ein Präcompiler den entsprechenden Ausdruck *statisch* durch ein Makro nach Listing 2. An Ort und Stelle im Quellcode erfolgt die Ersetzung durch eine IF-Anweisung. Deren Bedingung bildet der negierte Ausdruck der Assertion. Der IF-Zweig generiert eine entsprechende Fehlermeldung. Dies ist der einfachste und direkte Weg, die HLS wird den Ausdruck ohne weitere Änderungen in die resultierende Schaltung integrieren. Dabei wird der Kontrollfluss zur ursprünglichen Anwendung verändert, was in Abhängigkeit zur Komplexität der Assertion zu langen Verzögerungen im Laufzeitverhalten führen kann. Die Makro-Ersetzung ist als Verfahren im C-Standard festgelegt und wird von allen konformen Compilern unterstützt. Es ist daher werkzeugübergreifend einsetzbar.

```
#define assertion(expr) if(!(expr)) {* assertion_trigger = true;\
                                     * assertion_line = __LINE__;}
```

Listing 2: Makrodefinition einer Assertion

ISBN 978-3-00-048889-4

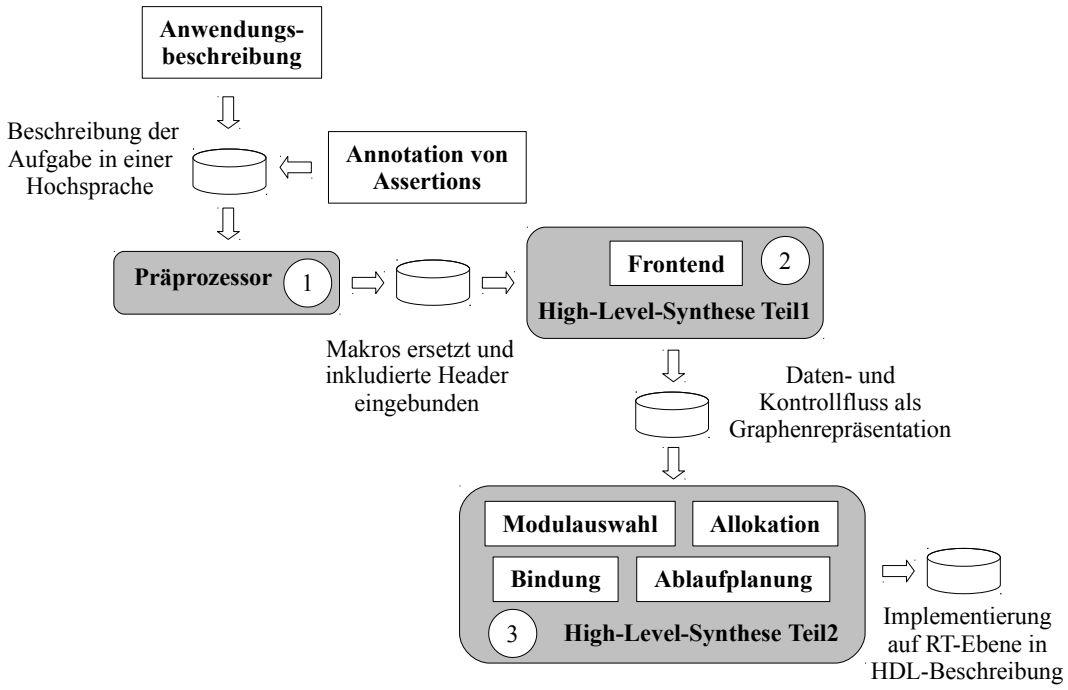


Abbildung 1: Integrationsmöglichkeiten für Assertions bei der Verhaltenssynthese

Verfahren 2 analysiert und erzeugt aus der Eingangsbeschreibung im Frontend zunächst einen abstrakten Syntaxbaum, in dem der *assert()*-Aufruf als Unterprogrammruf verstanden wird. Über diesen Syntaxbaum können dann Transformationen erfolgen und schließlich ein valider C-Code re-exportiert werden. Dieses Verfahren kann auch als „intelligenter Präcompiler“ verstanden werden, mit dem der Kontext ein oder mehrerer Assertions erfasst und analysiert werden kann. Die Prüfausdrücke der Assertions lassen sich dekomponieren und dabei günstigenfalls in ihrer Komplexität reduzieren. Weiterhin können mehrere Assertions innerhalb eines Code-Blocks zusammengefasst oder redundante Prüfanteile erkannt und reduziert werden. Begrenzt ist dieses Verfahren auf Transformationen, deren Ergebnis in der Eingangssprache der HLS (C/C++) ausgedrückt werden kann. Dieses Verfahren ist für alle HLS-Werkzeuge nutzbar.

Verfahren 3 erweitert den Kern der HLS und führt spezifische Optimierungen innerhalb des HLS-Compilers aus. Basis sind die Kontroll-Datenflussgraphen, welche sämtliche Daten- und Kontrollabhängigkeiten der Anwendung repräsentieren. Dies eröffnet weitreichende Möglichkeiten, die eine Code-Block-übergreifende Verschiebung von Prüfausdrücken oder Prüfausdrucksteilen erlauben. Ein Beispiel hierfür ist, dass erkannt wird, wenn eine nicht schreibend zugegriffene Variable innerhalb einer Schleife durch eine Assertion abgesichert wird. Verfahren 1 und 2 würden hier die Prüflogik in die Pipeline-Stufen der Schleife einbauen und dadurch unnötige Kontrolllogik und Schaltaktivität in der resultierenden Implementierung erzeugen. Die dedizierte Behandlung der Assertions innerhalb der HLS erlaubt es auch, die Prüf- und Signalisierungslogik optimiert neben dem Kontroll- und Datenfluss der Hauptanwendung als Assertion-Checker-Automat zu generieren. Die Umsetzung dieses Verfahrens ist spezifisch auf das anzupassende HLS-Werkzeug zugeschnitten.

Zusammenfassen lässt sich, dass die Verfahren 1 und 2 eine Instrumentierung des Eingangs-Codes der HLS verfolgen, wohingegen das Verfahren 3 auf eine Instrumentierung des resultierenden HDL-

Codes abzielt. Im weiteren Verlauf des Artikels wird die Umsetzung von Verfahren 1 bis hin zur Umsetzung im Schaltkreis behandelt. Verfahren 2 und 3 werden als Inhalt künftiger Untersuchungen zunächst nicht weiter betrachtet.

4. Umsetzung

Für die Nutzung der C-Assertions wird von einer Systemarchitektur ausgegangen, die mindestens einen Mikrocontroller sowie einen FPGA-Bereich enthält. Die Untersuchungen zielen auf Anwendungen, die verteilt über diese Rechenressourcen in den Domänen Hardware und Software ablaufen. Ein zentraler Steuertask auf dem Mikrocontroller der Anwendung integriert u.a. die Funktionalität zur Detektion fehlgeschlagener Assertions in den Teilanwendungen und die Ausgabe der entsprechenden Debug-Information an den Nutzer über die Standard-Fehlerausgabe (stderr).

```
int bubblesort(int output[max_size_array], bool * assertion_trigger ,
hls :: stream<int16> & assertion_line , int input[max_size_array], unsigned int size){

 assertion (size <= max_size_array);              // precondition assertion
 ...

for(int i = size; i > 0; i--){
  for(int j = 0; j < i-1; ++j){

    assertion (j <= max_size_array);             // internal assertion
      ...
  }
}

for( int n = 0; n<size;n++){
   assert (min < A[n]);                          // postcondition assertion
  min = A[n];
   ...
}
}
```

Listing 3: Funktionskopf und „Design by Contract Assertions“ für das Bubblesort Beispiel

Ausgehend vom „Design by Contract“-Paradigma nach [Mey92] aus dem Bereich der Softwareentwicklung werden die Hochsprachen-Assertions in drei verschiedene Klassen eingeteilt. Die *Precondition*-Prüfungen stellen sicher, dass die eingehenden Daten von einem übergeordneten System konform zur Spezifikation der generierten HW-Komponente sind. *Postcondition* stellen die valide Ausgabe der Komponente sicher. Diese Klasse wirkt nach *außen* und erleichtern die Integration, da alle inkorrekten Daten zu und von der Komponente abgefangen und signalisiert werden. Die Klasse der *Invariants* prüft den sicheren Ablauf innerhalb der Komponente, beispielsweise den Überlauf von Wertebereichen oder den Eintritt in nicht erlaubte Zustände in der Kontrolllogik. Die

von Curreri et.al. [CSG10] gezeigten zeitbezogenen Assertions zur Hang-Detektion gehören auch zu dieser Klasse. Listing 3 zeigt den Einsatz von Assertions beispielhaft für Bubblesort.

Für die Prüfung der generierten In-Circuit Monitore, beziehungsweise zum Erkennen einer negativen Assertion, ist die Erweiterung der Komponentenschnittstelle als Rückführung nach außen notwendig. Die dafür erforderlichen Ausgabe-Ports werden, entsprechend dem HLS-Werkzeug, als Output-Port dem Design hinzugefügt. Den Entwürfen wurden dazu zwei Ausgänge *assertion_trigger* und *assertion_line* ergänzt. Die Rückgabe der Zeilennummer ist als gepuffertes *Streaming*-Interface realisiert, in dem die Sequenz der fehlgeschlagenen Bedingungen aufgenommen wird. Das Indikationssignal *assertion_trigger* vom Typ *boolean* besitzt einen *Register*-Ausgang zur Signalisierung mindestens einer fehlgeschlagenen Assertion.

Die integrierten Assertions folgen den Empfehlungen für minimal erzeugten Ressourcenoverhead nach Currerie et.al. [CSG10]. In den verwendeten Prüfregeln wurden ausschließlich bereits berechnete Werte behandelt, sodass die Notwendigkeit für zusätzliche Arithmetik minimiert wurde. Weiterhin wurde der Zugriff auf Werte in Arrays vermieden und auf komplex verschachtelte logische Ausdrücke, die in großen Automaten zu entsprechenden Verzögerungen führen, verzichtet. Die Integration von prüfenden Ausdrücken im Kern der Algorithmen, häufig innerhalb von Schleifen, erfolgte hingegen zielgerichtet unter Beachtung der zuvor genannten Regeln. Diese Assertions sind notwendig um beispielsweise Überläufe von Wertebereichen zu überwachen.

Die Realisierung erfolgte anhand von drei Beispielen (BubbleSort, FIR-Filter und DCT) mit den kommerziellen HLS-Werkzeugen *Vivado High Level Synthesis* der Version 2013.2 von *Xilinx*, *Catapult C Synthesis* der Version 8.0 von *Calypto*, sowie *CoDeveloper* in der Version 3 von *Impulse Accelerated Technologies*. Für die verschiedenen HLS-Werkzeuge variiert die Anzahl der automatisch generierten Handshake-Ports zur Steuerung der erzeugten Hardware-Komponente. Die Komponentensteuerung besitzt, bis auf das Rücksetzen der Signalisierungs-Ports, keinen weiteren Einfluss auf die Assertions. Als Ziel-Hardware wird ein *Xilinx Virtex 7*-FPGA *XC7VX330* genutzt. Die RTL-Synthese wurde mit dem Werkzeug *Precision* in der Version 2013b durchgeführt. Für alle Beispiele wurde eine Komponente für die Xilinx-spezifischen Plattformentwicklungswerkzeuge erzeugt. Dieser sogenannte *PCORE* stellt eine Art *Black Box* für das entsprechende Design dar, welche schließlich in den FPGA instantiiert wurde. Die Auswertung der in der Zielplattform integrierten Assertions erfolgt über eine Anwendung auf dem Mikrocontroller.

5. Ergebnisse

In Tabelle 1 und 2 sind die charakteristischen Werte aus der Synthese der Implementierung dargestellt. Die Spalten *Original* und *Assert* zeigen die Ressourcenaufwände jeweils für die Umsetzung mit und ohne Assertions. Die Spalte *Diff* weist den absoluten und den relativen Mehraufwand durch die generierten Monitore aus. Um die Vergleichbarkeit zu erhalten, wurden die Constraints für alle HLS-Werkzeuge identisch auf minimale Fläche bei einer Zielfrequenz von 100 MHz gesetzt.

Die resultierenden Zusatzaufwände liegen bei allen Werkzeugen im ähnlichen Bereich. Da der CoDeveloper jedoch die Schnittstellen als Stream umsetzt und den Zugriff dieser mittels doppeltem Handshake-Protokoll, sowie für den Stream spezifische Schreib- und Lesefunktionen umsetzt, zeigt

Tabelle 1: Ressourcenaufwände nach Synthese für den Xilinx XC7VX330

Bubble Sort

	Catapult			Vivado			CoDeveloper		
XC7VX330	Orig	Assert	Diff	Orig	Assert	Diff	Orig	Assert	Diff
CLB Slices	126	158	+32 (+25,4%)	101	123	+22 (+21,78%)	142	164	+22 (+15,49%)
LUT	502	629	+127 (+25,3%)	403	492	+89 (+22,08%)	566	656	+90 (+15,9%)
DFFs	410	490	+80 (+19,51%)	246	284	+38 (+15,45%)	587	635	+48 (+8,18%)

FIR-Filter

	Catapult			Vivado			CoDeveloper		
XC7VX330	Orig	Assert	Diff	Orig	Assert	Diff	Orig	Assert	Diff
CLB Slices	238	261	+23 (+9,66%)	49	58	+9 (+18,37%)	274	293	+19 (+6,93%)
LUT	951	1044	+93 (+9,78%)	195	230	+35 (+17,95%)	1095	1172	+77 (+7,03%)
DFFs	1123	1169	+46 (+4,1%)	170	171	+1 (+0,59%)	594	644	+50 (+8,41%)
DSP48	6	6	(+/-0%)	6	6	(+/-0%)	11	11	(+/-0%)

DCT

	Catapult			Vivado			CoDeveloper		
XC7VX330	Orig	Assert	Diff	Orig	Assert	Diff	Orig	Assert	Diff
CLB Slices	378	402	24 (+6,35%)	364	371	7 (+1,92%)	nicht implementiert		
LUT	1510	1605	95 (+6,29%)	1454	1483	29 (+1,99%)			
DFFs	693	703	10 (+1,44%)	963	934	-29 (-3,01%)			

Tabelle 2: Maximale Frequenz in MHz für den Xilinx XC7VX330 FPGA

	Catapult			Vivado			CoDeveloper		
	Original	Assert	Diff	Original	Assert	Diff	Original	Assert	Diff
Bubblesort	131	124	-7 (-5,34%)	126	140	+14 (+11,11%)	206	197	-9 (-4,36%)
FIR	101	101	+/-0	203	203	+/-0	91	91	+/-0
DCT	38	38	+/-0	125	125	+/-0	-	-	-

ISBN 978-3-00-048889-4

sich beim CoDeveloper ein latent höherer Mehraufwand im Zusammenhang mit der Assertion-Signalisierung. Die im Vergleich zum Teil recht starke Diskrepanz am generellen Ressourcenverbrauch lässt sich mit einer unterschiedlichen Integration der Technologie-Daten der Ziel-Hardware in den Tools und der daraus resultierenden Optimierungsergebnisse erklären. Die Ergebnisse des Vivado-HLS-Tools bestätigen dies. Beim Bubblesort liegen die Mehraufwände zwischen einem Fünftel und einem Viertel, wobei der CoDeveloper noch etwas weniger Overhead produziert. Die Ergebnisse sind, unter Berücksichtigung der tool-spezifischen Eigenheiten, für dieses Beispiel gut nachvollziehbar.

Beim FIR-Filter wurden keine Assertions für die Klasse der Invarianten integriert. Vielmehr entstehen die umfassenden Zusatzaufwände durch die Absicherung der Modulschnittstelle, bei der sowohl die Dateneingangs- und Datenausgangswerte als auch die Koeffizientenmatrix auf obere und untere Grenzen geprüft werden. Der Ressourcenbedarf des Catapults und des CoDevelopers sind in etwa gleich, wobei die generelle Anzahl der genutzten DSP-Slices bei letzterem um das Doppelte größer und dafür die Menge an FlipFlops um die Hälfte reduziert ist. Der viel größere Ressourcenbedarf der Codeveloper- und Catapult-Implementierung des FIR ist auf eine unterschiedliche Implementierung zur Unterstützung kontinuierlicher Werteverarbeitung im Vergleich zur Vivado-Variante zurückzuführen. Der direkte Vergleich zwischen CatapultC und CoDeveloper zeigt für letzteren einen höheren DSP-Verbrauch zugunsten einer geringeren Nutzung der D-FlipFlops.

In der DCT-Implementierung wurden die Assertions ähnlich zum FIR-Filter integriert, wobei zusätzlich eine Prüfung der Invariants - hier bezüglich des korrekten Wertebereiches des Kosinus - erfolgte. Die Mehraufwände liegen im einstelligen Prozentbereich, wobei eine leichte Reduzierung des FlipFlop-Verbrauches beim Vivado-Tool bei der Integrierung von Assertions auftritt. Die DCT konnte im CoDeveloper aufgrund einer fehlenden bzw. zu den beiden anderen Tools vergleichbaren Implementierung einer Kosinus-Funktion mittels CORDIC-Algorithmus nicht realisiert werden.

Hinsichtlich der maximalen Frequenz zeigt sich, dass das Constraining mit und ohne Assertions weitgehend eingehalten werden kann. Die Werte liegen beim Bubblesort aufgrund der invarianten Prüfung im Kern des Sortieralgorithmus um bis zu 5 Prozent unterhalb des Originals. Gleichwohl ist das Vivado-Resultat hier mit den Assertions sogar schneller als ohne. Die Wertebereichsprüfungen an der Komponentenschnittstelle haben kaum Einfluss auf die Performanz. Dies ist auch darin begründet, dass auf komplexe Ausdrücke in den Assertions verzichtet wurde und lediglich Vergleiche mit festen Werten stattfinden. Die Constraints für den FIR-Filter im CoDeveloper konnten sowohl mit, als auch ohne Assertions nicht eingehalten werden. Die Ursache dafür ist u.a. im Streaming-Ansatz des CoDevelopers bzw. die darauf ausgerichtete Implementierung zu suchen, worin zunächst alle für den FIR erforderlichen Operanden in einen internen Puffer geladen werden. Gleichwohl lässt sich, ähnlich den anderen HLS-Tools, eine Änderung der maximalen Frequenz beim FIR-Filter durch Nutzung von Assertions nicht feststellen.

6. Zusammenfassung

Zusammenfassend kann für die Untersuchungen festgestellt werden, dass mit allen HLS-Werkzeugen die Assertions als Monitore in die Hardware integriert werden konnten. Eine ANSI-C basierte Verifikation in Simulation und auf der Zielplattform ist mittels einfacher Makrodefinition realisierbar.

ISBN 978-3-00-048889-4

Die Ressourcenaufwände und die Einflüsse auf die Performanz sind spezifisch für das genutzte HLS-Werkzeug. Für weitergehende Untersuchungen ist es geplant, die Assertionintegration mittels eines intelligenten Präprozessors und durch spezifische Anpassung eines HLS-Compilers zu realisieren.

Literatur

[BCZ07] Boule, M., J. S. Chenard und Z. Zilic: *Assertion Checkers in Verification, Silicon Debug and In-Field Diagnosis*. In: *Quality Electronic Design, 2007. ISQED '07. 8th International Symposium on*, Seiten 613–620, March 2007.

[Ber06] Bertacco, V.: *Scalable Hardware Verification with Symbolic Simulation*. Springer Science+Business Media, Inc, 2006.

[CSG10] Curreri, J., G. Stitt und A.D. George: *High-level synthesis techniques for in-circuit assertion-based verification*. In: *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, Seiten 1–8, April 2010.

[FKL04] Foster, H., A. Krolnik und D. Lacey: *Assertion-Based Design 2nd Edition*. Kluwer Academic Publishers, 2004.

[GW14] Goeders, Jeffrey und Steven J.E. Wilton: *Effective FPGA debug for high-level synthesis generated circuits*. In: *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*, Seiten 1–8, Sept 2014.

[KRM08] Kakoee, M.R., M. Riazati und S. Mohammadi: *Enhancing the Testability of RTL Designs Using Efficiently Synthesized Assertions*. In: *Quality Electronic Design, 2008. ISQED 2008. 9th International Symposium on*, Seiten 230–235, March 2008.

[Mey92] Meyer, B.: *Applying 'design by contract'*. Computer, 25(10):40–51, Oct 1992, ISSN 0018-9162.

[PLBN05] Pellauer, M., M. Lis, D. Baltus und R. Nikhil: *Synthesis of Synchronous Assertions with Guarded Atomic Actions*. In: *Proceedings of the 2Nd ACM/IEEE International Conference on Formal Methods and Models for Co-Design*, MEMOCODE '05, Seiten 15–24, Washington, DC, USA, 2005. IEEE Computer Society, ISBN 0-7803-9227-2.

[RLGJD11] Ribon, A., B. Le Gal, C. Jego und D. Dallet: *Assertion support in high-level synthesis design flow*. In: *Specification and Design Languages (FDL), 2011 Forum on*, Seiten 1–8, Sept 2011.

Modulare Verifikation von Non-Mainline Chip-Level Funktionen

Matteo Michel, Johannes Koesters, Benedikt Geukes

IBM Deutschland R&D GmbH, Schoenaicher Straße 220, 71032 Boeblingen

Matteo.Michel@de.ibm.com

Abstract

In modernen Prozessorentwicklungsprojekten kann die Verifikation von non-mainline Funktionen bis zu einem Drittel des Gesamtaufwandes für Verifikationstätigkeiten annehmen. Non-mainline Features beinhalten Initialisierungs- und Resetsequenzen, Zuverlässigkeits-, Zugriffs- und Debugschnittstellen, Sensoren, Bist-Mechanismen oder auch Debug- und Trace-Funktionen. Im Gegensatz zur bereits stark fortschrittlichen funktionalen Verifikation gestaltet sich die Verifikation von non-mainline-Funktionen noch immer stark manuell - moderne Methoden und Werkzeuge halten nur langsam Einzug. In dieser Arbeit möchten wir die neusten praxisbezogenen Methoden in der Chip-Level-Verifikation vorstellen und welche Veränderungen das jeweils auf Seiten des Designs und der Verifikationsmethodiken hat.

Einleitung

Während der Durchführung von früheren Prozessorprojekten haben wir ein stetiges ansteigen an Turnaround-Zeiten und Fixraten gesehen. Dies ist einerseits der Zunahme an Komplexität auf dem Gesamtchip geschuldet, auf der anderen Seite jedoch wurden die Verifikationsmethoden nicht entsprechend angepasst. Die Verifikation von non-mainline-Funktionen ist entsprechend spät im Design-Zyklus angesiedelt, da das Design auf dem Chip verteilt liegt und entsprechend in verschiedene Hierarchieebenen integriert werden muss, bevor eine Verifikation dieser Funktionalität überhaupt beginnen kann. Des Weiteren bestehen Abhängigkeiten anderer Teams an den Erfolg dieser Verifikationsarbeiten: DFT oder Systemsimulation können ihre Arbeiten erst starten, wenn eine gewisse Qualität erreicht wurde. Weiterhin wird das zeitliche Fenster vom Tape-Out eingegrenzt.
Die größten Schwachstellen im Design- und Verifikationsprozess sind die Verbindungen auf der Chip-Ebene zwischen einzelnen Funktionsblöcken. Erschwerend kommt hinzu, dass diese Chip-Level RTL bereits die physikalische Struktur besitzt und somit jegliche logische Änderung in einem größeren Änderungsaufwand resultiert.
Das beschriebene Verfahren basiert auf 2 Neuerungen, die im Entwicklungsprozess eingeführt wurden. Einerseits entsteht eine Trennung von logischer und physikalischer Sicht(RTL) im Design. Die RTL wird pro Funktionseinheit entwickelt. Die spätere physikalische Gestalt wird dagegen nur noch durch Werkzeuge bestimmt und nicht mehr manuell modifiziert. Dabei werden einzelne Blöcke aus den jeweiligen Funktionseinheiten automatisch auf dem Chip platziert, ohne jedoch ihre entsprechenden Signalabhängigkeiten zu verändern. Mögliche Änderungen sind durch Backannotierung in der logischen RTL sichtbar. Der abschließende Beweis der logischen Gleichheit zwischen logischer und physikalischer RTL basiert auf formalen Methoden und wird mit Equivalenzprüfungen durchgeführt.

ISBN 978-3-00-048889-4

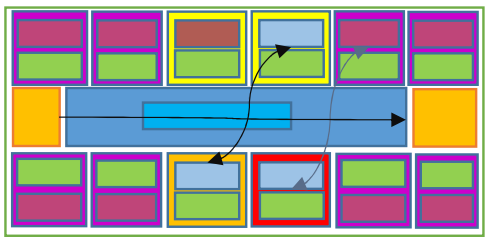


Abbildung 1: Herkömmliche physikalische Sicht auf Chip-Level-RTL

Der zweite Schritt besteht in der Anpassung des Verifikationsprozesses, der entsprechende generische Schnittstellen für die einzelnen Funktionseinheiten benötigt, als auch eine im danach hohen Maße Chip-Level-unabhängige Simulation ermöglicht und diese Arbeitsweise auch in DFT- und System-Umgebungen ermöglicht.
Das dadurch entstehende Verifikations-Framework möchten wir hier ansatzweise darstellen, sowie dessen Vor- und Nachteile und Bezug auf Performanz und Datenstrukturen.

Details

Um die in der Einleitung beschriebenen Neuerungen umsetzen zu können, wurden die im Folgenden beschriebenen 4 Schritte durchgeführt, um den Entwicklungsprozess anzupassen.

1. Die Entkopplung der Chip-Level RTL von Flurplan-Abhängigkeiten

   Die RTL-Beschreibung ist voll auf die logische Funktion abgestellt und beachtet keine physikalischen Abhängigkeiten mehr. Bestimmte Non-Mainline Funktionen wie Clock und Scan werden zudem durch automatisierte Prozesse in die logischen Funktionsblöcke eingebaut. Damit sind dem Design-Ingenieur alle Möglichkeiten gegeben, effizient und übersichtlich die eigentliche Funktion zu beschreiben, ohne auf externe Gegebenheiten Rücksicht nehmen zu müssen.

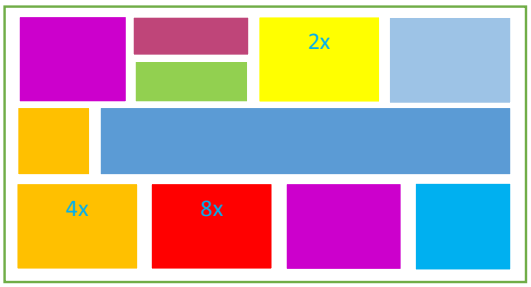


Abbildung 2: Neue logische Sicht auf Chip-Level-RTL

Physikalische Informationen und Abhängigkeiten können durch den Designingenieur in sogenannten Sidefiles spezifiziert werden und haben später einen Einfluss im weiteren Prozessablauf. Die logische Simulation findet nur auf der logischen RTL statt. Das Verdrahten der logischen Blöcke wird automatisiert in RTL durchgeführt.

Um nun zur physikalischen Sicht zu gelangen, ist eine Transformation notwendig. Diese liest die bestehenden RTL-Designdaten ein und nimmt zusätzlich in die Analyse die physikalisch festgelegten Informationen in den Sidefiles auf. Bei einer Transformation wird die jeweilige Lage und Hierarchie einzelner Funktionsblöcke geändert, ohne dass die Signalverbindungen verändert werden dürfen. Dies erlaubt nun, die Position auf Chip-Level nach physikalischen Gesichtspunkten zu verändern und zu optimieren.

2. Einführung einer einheitlichen Schnittstelle für non-mainline Funktionen

   In den bisherigen Projekten wurden die non-mainline-Anschlüsse an die funktionalen Blöcke stark an die jeweiligen Bedürfnisse angepasst. Somit konnten keine Treiber und Monitore jeweils wiederverwendet werden und man hat zur einfachen Handhabung das Chip-Debug-Interface(zum Beispiel JTAG) zum Treiben genutzt, um eine einheitliche Basis für sämtliche Blöcke zu haben.

   In der jetzigen Fassung wurde eine einheitliche Schnittstelle für alle non-mainline Funktionen definiert. Diese ist von jedem Funktionsblock so umzusetzen. Sie beinhaltet alle für Non-Mainline Funktionen notwendige Signale, egal ob sie innerhalb des Blockes benutzt werden.

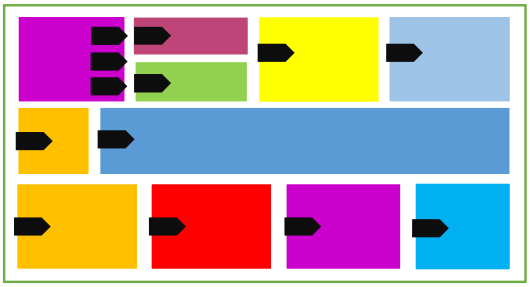


Abbildung 3: Generisches Interface für jeden Funktionsblock

3. Implementierung eines generischen Schnittstellentreibers

   Als logische Folge zur Einführung der einheitlichen Schnittstelle wurden entsprechende Treiber und Monitore implementiert, die genau auf dieser Schnittstelle aufsetzen und eine Überwachung an dieser Grenze und innerhalb gewährleisten. Besonderes Augenmerk gilt an dieser Stelle der Ordnung nach Clock- und Spannungsregionen, sowie notwendigen Chip- und Sub-Chip-Hierarchien. Diese sind notwendig, um bei entsprechender Adressierung die richtigen Treiber in einem Mehrfach-Treiber-Umfeld auswählen zu können.

Abbildung 4: Simulation losgelöst von Chip-Level-Abhängigkeiten

ISBN 978-3-00-048889-4

Wie in Abbildung 4 ersichtlich ist es nun möglich, die Verifikation losgelöst von jeglicher Chip-Level-RTL-Verfügbarkeit zu beginnen, sobald ein Design-Team Logik fertiggestellt hat. Auch sind sonstige sequenzielle Abhängigkeiten nicht mehr vorhanden. Dies unterstützt somit eine zunehmend Möglichkeit einer agilen Verifikationstätigkeit.

4. Neustrukturierung der Simulationsumgebung

Da die wesentlichen Non-Mainline-Funktionen innerhalb dieser neu entstandenen Funktionsblöcke gekapselt sind, können sämtliche Testszenarien zu 95% auf dieser Stufe ausgeführt werden. Es bleibt nur eine abschließende Integrationsumgebung auf Chip- und Systemlevel, die wesentliche Chipverbindungen vor dem Tape-Out abdeckt.

Abbildung 5: Abschließende Verifikation auf einem Integrationsmodell

Die wesentliche Arbeit an den Umgebungen entfiel auf die Integration des neuen Treibers und der Anpassung der Checking-Mechanismen auf die neue Struktur. Dabei wurde darauf geachtet, dass sämtliche Konfigurationsmöglichkeiten soweit wie möglich generisch gestaltet wurden, um eine einfache und schnelle Portierung der Umgebungen zwischen den unterschiedlichen Simulationsmodellen zu ermöglichen.

Als weitere Verbesserung ist es nun zum Beispiel auch möglich, spezifische Systemfunktionen auf kleineren Modellen zu verifizieren, so zum Beispiel IO-Initialisierungssequenzen.

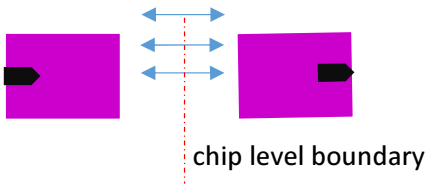


Abbildung 6: IO-Verifikation auf optimiertem Modell

Dabei werden zwei oder mehrere dieser neuen Funktionsblöcke im Sinne einer Systemarchitektur zusammengeschaltet und Systemtestszenarien ausgeführt. Der Treiber ist dabei intelligent genug und führt nur die Teile des Szenarios aus, welche auch wirklich in jenem Funktionsblock benötigt werden. Das gleiche Testszenario kann dann auf einem richtig voll-populierten Systemmodell simuliert werden als auch auf dem hergestellten Produkt final verifiziert werden.

ISBN 978-3-00-048889-4

Ergebnisse

Mit der vorgestellten Methodik sind wir nun in der Lage, sehr flexibel und agil auf verfügbare RTL-Beschreibungen zu reagieren. Erste Ergebnisse aus einem aktuellen Projekt geben eine stark positive Richtung hin zu Effizienzsteigerungen.

1. Dauer der Erstellung von Simulationsmodellen haben sich bis zu 80% verkürzt

   Durch die starke Verkleinerung der Modelle ist alleine die Zeit zum erstellen eines Simulationsmodells erheblich gesunken. In früheren Projekten konnte maximal 1 neues Modell pro Tag erstellt werden, was zu starken Verzögerungen im gesamten Projektablauf geführt hat. Mittlerweile sind einige dieser Modifikationen an einem Tag möglich und somit können Probleme ganzheitlich erfasst und gelöst werden.

   Nachteilig ist die gestiegene Anzahl der Modelle, welche besondere Anforderungen an Datenstrukturen stellt. Da noch kein kompletter Projektzyklus damit erprobt wurde, werden an dieser Stelle aktuell noch Erfahrungen gesammelt.

2. Simulationszeiten haben sich bis zu 50% verkürzt

   Im Vergleich zu Testszenarien in früheren Projekten ist deren Laufzeit erheblich verkürzt worden, was zu großen Teilen auf die kleine Modellgröße und neue Treiberpunkte zurückzuführen ist.

3. Turnaround von 4 Wochen auf wenige Stunden

   Mit der früheren Abhängigkeit an Chip-Level-Verdrahtung ist man auch an die Bibliothek-Release-Zyklen gebunden gewesen. Mit dieser neuartigen Modularisierung ist eine Unabhängigkeit von diesen Mechanismen gegeben, sodass mittlerweile Fixes innerhalb von wenigen Stunden probiert werden können und keine bis zu mehreren Wochen dauernden Wartezeiten entstehen.

4. Konstante Teamgröße

   Mit dieser Methode sind wir im Augenblick in der Lage, wesentlich komplexere Designs mit der gleichen Anzahl an Teammitgliedern zu bearbeiten, was natürlich auch ein Ausdruck der gestiegenen Produktivität ist. So zum Beispiel können wir einen Anstieg von mehr als 40% mehr Arrays kompensieren.

ISBN 978-3-00-048889-4

Zusammenfassung

Die Komplexität von Prozessorsystemen nimmt weiterhin stark zu. In dieser Arbeit konnten wir eine Möglichkeit vorstellen, wie mit Hilfe von Modularisierung selbst auf Chip-Level spezialisierte Funktionen besser verifiziert werden können. Dabei konnten wir Verbesserungen in unterschiedlichsten Aspekten beobachten:

Die ***Produktivität*** konnte spürbar verbessert werden, in dem von einem starr sequenziellen Ansatz auf einen hoch-parallele Methode umgestellt wurde. Dies ermöglicht es mittlerweile selbst für andere Teams wie DFT oder Systemsimulation die Arbeit zu beginnen, ohne eine komplett verifizierte Chip-Level-RTL-Beschreibung zur Verfügung zu haben.

Mit der dadurch entstandenen ***Flexibilität*** verhindern wir Stillstandsphasen und haben einen schnelleren Verifikationszyklus geschaffen, der auch dem Design-Team zeitnah und effektiv eine Rückmeldung ermöglicht. Entsprechende Abhängigkeiten auf Chip-Level wurden gelöst und verzögern nun nicht mehr den Projektfortschritt.

Auch im Arbeitsumfeld sind Verbesserungen entstanden. Während in früheren Projekten sich die Teammitglieder vorwiegend auf einen Aspekt konzentrieren mussten, erlaubt der neue Ansatz eine agile Arbeitsweise. Diese hat zur Folge, dass nun bei Konzentration auf einen bestimmten Funktionsblock durch eine Person weitaus unterschiedliche Aspekte bearbeitet werden können. Dies verbessert einerseits die Fähigkeiten eines jeden Teammitglieds und damit auch die Einsetzbarkeit, als andererseits auch die ***Kommunikation*** innerhalb des Teams.

ISBN 978-3-00-048889-4

Formale Verifikation von eingebetteter Software für das Betriebssystem Contiki unter Berücksichtigung von Interrupts

Thilo Vörtler[1], Benny Höckner[2], Petra Hofstedt[2] und Thomas Klotz[3]
[1]Fraunhofer-Institut für Integrierte Schaltungen IIS
Institutsteil Entwurfsautomatisierung EAS, Dresden
thilo.voertler@eas.iis.fraunhofer.de

[2]Brandenburgische Technische Universität Cottbus-Senftenberg
{benny.hoeckner|petra.hofstedt}@tu-cottbus.de

[3]Bosch Sensortec GmbH
thomas.klotz@bosch-sensortec.com

Zusammenfassung

Diese Arbeit stellt einen Ansatz für die formale Verifikation von Anwendungen für das Betriebssystem Contiki basierend auf Software-Model-Checking vor. Insbesondere die Verwendung von Interrupts muss bei der Modellierung des Systemverhaltens beachtet werden. Nach einer Beschreibung des Contiki-Systems, werden verschiedene Ansätze zur Modellierung von Interrupts diskutiert und ein Vorgehen abgleitet, wie Anwendungen für Contiki mit Hilfe des Model-Checking-Tools CBMC überprüft werden können.

1. Einleitung

Die Anwendungsbereiche zur Nutzung von vernetzten eingebetteten Systemen, z.B. durch die Verwendung in Wireless Sensor Nodes, wachsen stetig. Zur Programmierung solcher eingebetteter Systeme werden zunehmend auch ereignisbasierte Betriebssysteme wie Contiki [Con14, Wir14] verwendet. Contiki ermöglicht es Anwendungen zu entwickeln, welche auf verschiedenen Plattformen von eingebetteten Systemen laufen und mit geringem Aufwand zwischen diesen Plattformen portiert werden können. Im Gegensatz zu Betriebssystemen wie Linux ist Contiki dabei auf Mikrocontroller mit sehr geringem Energiebedarf und geringer Speicherausstattung (z. B. TI-MSP430 [MSP14], Atmel AVR [AVR14]) optimiert und benötigt nur wenige Kilobyte Arbeitsspeicher.
Eingebettete Systeme werden auch für sicherheitskritische Anwendungen z. B. in der Automobilindustrie eingesetzt, wobei die Systeme fehlerfrei arbeiten müssen. Formale Methoden wie Software-Model-Checking (SMC) erlauben es, die Korrektheit von Systemen gegenüber einer Spezifikation zu zeigen. Eine Besonderheit von Software (SW) für eingebettete Systeme, ist die enge Interaktion mit der Hardware (HW) des Systems. Diese muss bei der Modellierung für Model-Checking (MC) berücksichtigt werden. Der Beitrag dieser Arbeit setzt sich wie folgt zusammen:

- Das Betriebssystem Contiki und eine Formalisierung seines Kernels werden vorgestellt.
- Modellierungstechniken für Interrupts werden verglichen. Aufbauend wird eine neue Modellierungstechnik gezeigt, die es erlaubt Eigenschaften, welche sich auf die Häufigkeit des Auftretens von Interrupts beziehen, zu verifizieren.
- Es wird ein Framework vorgestellt, welches die gezeigten Techniken implementiert.

ISBN 978-3-00-048889-4

Die Arbeit ist folgendermaßen gegliedert: In Abschnitt 2 wird der aktuelle Stand der Forschung im Gebiet der Verifikation von SW für eingebettete Systeme erläutert. Abschnitt 3 stellt das Betriebssystem Contiki und dessen ereignisbasierten Kernel anhand einer Beispielanwendung vor. Anschließend wird in Abschnitt 4 beschrieben, wie auf Contiki basierende Systeme für die formale Verifikation modelliert werden müssen. Abschnitt 5 fasst die Arbeit zusammen.

2. Verifikation von Software für eingebettete Systeme

In [DKW08] werden verschiedene Techniken zur Verifikation von SW vorgestellt. Insbesondere auf MC [CGP00] basierende Techniken erlauben es, SW automatisch zu überprüfen. Für eingebettete Systeme ist es dabei besonders wichtig die Umgebung des Systems korrekt zu modellieren. [LFCJ09] beschreibt solch ein Vorgehen, in dem der Code einer SW für eingebettete Systeme in einen HW-abhängigen und HW-unabhängigen Teil aufgeteilt wird. Es werden verschiedene MC-Tools, darunter auch CBMC [CKL04] verglichen. [CRJ13] stellt einen allgemeinen Ansatz zur Verifikation von eingebetteten Systemen basierend auf CBMC für medizinische Anwendungen vor. Zu bestimmten Zeitpunkten wird ein Interrupt-Scheduler aufgerufen, welcher das Verhalten von Interrupts modelliert. Es wird jedoch ebenfalls kein Betriebssystem verwendet, die Modellierung von HW und externen Geräten erfolgt somit in beiden Arbeiten auf der Ebene von HW-Registern.
Bucur und Kwiatkowska stellen in [BK11] ein Framework für die Verifikation *TinyOS* und den MSP430 Mikrocontroller vor. Weil TinyOS-Anwendung in nesC geschrieben werden, müssen diese zuerst in **C** übersetzt werden, bevor sie mit Hilfe von CBMC verifiziert werden können. In dieser Arbeit wird unter anderen Partial Order Reduction (POR) [CGP00] im Zusammenhang mit Interrupts für **C**-Code vorgestellt. Eine Weiterentwicklung des Ansatzes für SW, welche mehrere Interrupts verwendet, wird in [KLM$^+$15] vorgestellt. Mit Hilfe von *partial-order encoding* werden die Abhängigkeiten zwischen Interrupts verschiedener Priorität untersucht und unmögliche Ausführungen verworfen. Der Ansatz benutzt jedoch eine Über-Abstraktion (siehe nachfolgend in Abschnitt 4.2) und unterliegt den gleichen Einschränkungen.
In [MVÖ$^+$10] wird das MC-Tool Anquiro vorgestellt. Anquiro erlaubt es Contiki-Anwendungen zu verifizieren, jedoch ist der Fokus von Anquiro die Verifikation von Netzwerkanwendungen, welche die von Contiki zur Verfügung gestellten Netzwerk-Stacks verwenden. Deshalb wird nicht darauf eingegangen, wie die HW des Systems modelliert wird. Da Anquiro auf dem Bogor-Model-Checker basiert, müssen Contiki-Anwendungen vor der Verifikation transformiert werden.

3. Betriebssystem Contiki

3.1. Einführung und Beispielanwendung

Contiki ist ein quelloffenes Betriebssystem, dessen Haupteinsatzgebiet eingebettete Systeme mit sehr niedrigem Energieverbrauch sind und welches deshalb einen ereignisbasierten Kernel verwendet. Da das Betriebssystem, sowie die darauf laufenden Anwendungen in **C** geschrieben werden, kann es einfach auf verschiedene HW-Plattformen portiert werden. Anwendungen sowie Systemfunktionen werden in Contiki als Prozesse geschrieben. Um die Programmierung dieser Prozesse zu erleichtern, verwendet Contiki das Protothread-Programmiermodell [DSVA06], welches aus einer Sammlung von **C**-Makros besteht.
Ein Beispiel für die Programmierung einer Contiki-Anwendung wird in Abbildung 1 gezeigt. Diese Anwendung schaltet im Abstand von einer Sekunde die LEDs eines eingebetteten Systems ein und wieder aus (`while`-Schleife Zeile 6-13). Das Warten von einer Sekunde, wird mit dem *Event-Timer*-System von

```
1  PROCESS(blink_process, "Blink");
2  AUTOSTART_PROCESSES(&blink_process);
3  PROCESS_THREAD(blink_process, ev, data) {
4    PROCESS_BEGIN();
5    static struct etimer et;
6    while(1) {
7      etimer_set(&et, CLOCK_SECOND);
8      PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
9      leds_on(LEDS_ALL);
10     etimer_set(&et, CLOCK_SECOND);
11     PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
12     leds_off(LEDS_ALL);
13   }
14   PROCESS_END();
15 }
```

Abbildung 1: Contiki *LED-Blink* Anwendung

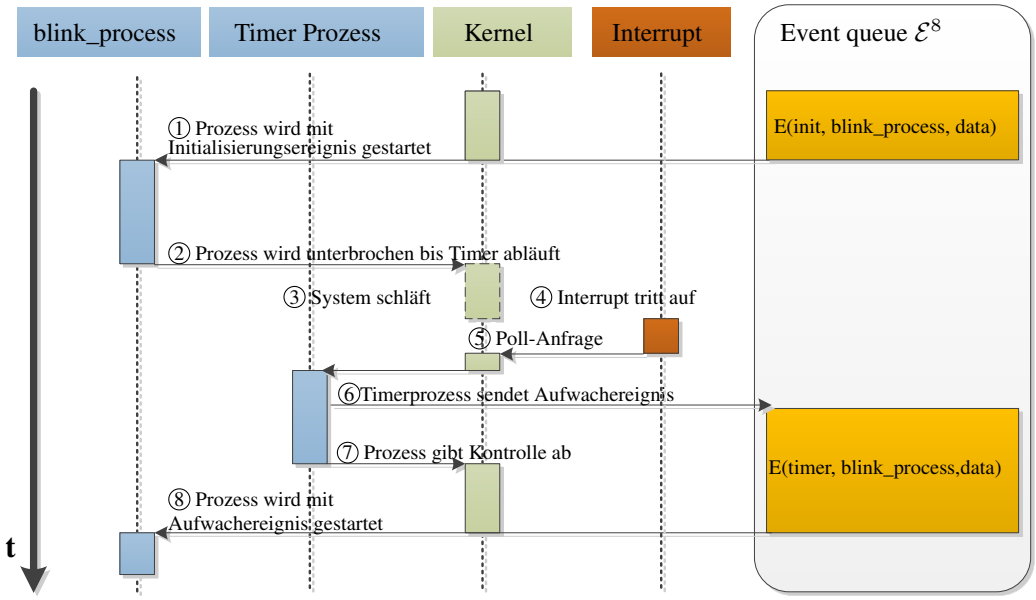


Abbildung 2: Contiki Prozess-Kommunikation anhand der *LED-Blink* Anwendung

Contiki realisiert, welches die Systemzeit mit Hilfe von Interrupts zählt. In Zeile 7 wird ein *Event-Timer* auf eine Sekunde gesetzt und im System registriert. In Zeile 8 wird anschließend der Anwendungsprozess unterbrochen bis der Timer abgelaufen ist. Ist dies der Fall, werden in Zeile 9 die LEDs des Systems eingeschaltet. Dieser Vorgang wird in den Zeilen 10-12 zum Abschalten der LEDs wiederholt. Die zum Ansteuern der LEDs verwendete API und das Makro `CLOCK_SECOND` werden von Contiki zur Verfügung gestellt und müssen für jede HW-Plattform konfiguriert werden. Obwohl die Beispielanwendung in einer Endlosschleife läuft, blockiert sie nicht die Nutzung des Systems, für andere Prozesse, da sie in jedem Schleifendurchlauf die Kontrolle an das Betriebssystem abgibt (Zeile 8 und 11). Weiterhin verdeutlicht die Anwendung die Portabilität von Contiki-Anwendungen, da sie nur APIs, wie das *Event-Timer*-System oder die LED-API, benutzt.

3.2. Formalisierung des Contiki-Kernels

Der Kernel des Contiki-Betriebssystems arbeitet mit Ereignissen und Prozessen. Im System gibt es Listen aller abzuarbeitenden Ereignisse, wobei ein Prozess immer mit Hilfe eines Ereignisses aufgerufen wird. Die im System laufenden Prozesse werden in einer Liste gespeichert und enthalten Zusatzinformation über den Status des Prozesses sowie den Rücksprungpunkt bei Wiederaufnahme des Prozesses. Contiki benutzt ein nicht-preemptives Multitasking, das heißt Prozesse müssen selbst die Kontrolle an den Kernel des Betriebssystems über bestimmte Befehle abgeben. Weiterhin können Prozesse Ereignisse verschicken, entweder an einen bestimmten Prozess oder an alle Prozesse ($ProcessBroadcast$). Die Hauptaufgabe des Kernels besteht darin, Contiki-Prozesse mit Hilfe von Ereignissen aus der Ereignisliste zu starten.
Zusätzlich können Prozesse über sogenannte Poll-Anfragen aufgerufen werden. Eine Poll-Anfrage kann von einem Interrupt gesetzt werden und wird mit höherer Priorität als Ereignisse abgearbeitet. Um den ereignisbasierten Kernel formal zu beschreiben, werden folgende Notationen eingeführt. Da es keine formalisierte Beschreibung des Contiki Kernel gibt, ist diese Beschreibung aus dem Quellcode des Betriebssystems sowie aus [DGV04] und [DSVA06] abgeleitet.

Definition 1. *Ein* **Ereignis** *ist ein Tupel* $E = \langle e_{type}, e_{process}, e_{data} \rangle$ *wobei*

- e_{type} *ein String ist, welcher den Typ eines Ereignis angibt.*
- $e_{process}$ *ein Zeiger auf eine Prozessstruktur* P *ist, welche mit dem Ereignis aufgerufen werden soll. Sollen alle Prozesse aufgerufen werden, ist der Zeiger null (*$ProcessBroadcast$*).*
- e_{data} *ein Zeiger auf ein Datenelement ist.*

Contiki-Anwendungen ist es erlaubt, eigene Ereignistypen anzulegen und auf diese zu reagieren.

Definition 2. *Sei* $\mathcal{E}^{MaxEvents}$ *eine Warteschlange von Ereignissen* $\mathcal{E}^{MaxEvents} = (e_0, .., e_n)$ *mit* $n < MaxEvents$ *und* $n \in \mathbb{N}$*, sowie* e *ein beliebiges Ereignis. Solch eine Warteschlange wird* **Ereignisliste** *genannt.*

Durch den Parameter $MaxEvents$ kann der benötigte Speicher der Ereignisliste festgelegt werden.
Für die Handhabung von Prozessen wird eine eigene Datenstruktur innerhalb des Kernels verwendet, welche alle Informationen die zu einem Prozess gehören speichert.

Definition 3. *Eine* **Prozessstruktur** *ist ein Tupel* $P = \langle p_{func}, p_{cont}, p_{polled}, p_{state} \rangle$ *mit*

- p_{func} *ist ein Zeiger auf die Funktion, welche das Verhalten eines Prozesses implementiert. Diese Funktion hat die Parameter* p_{cont}*,* e_{type}*, und* e_{data}*.*
- p_{cont} *wird benutzt um den Rücksprungpunkt innerhalb eines Protothread zu speichern.*
- $p_{polled} \in \{true, false\}$ *gibt an, ob für diesen Prozess eine Poll-Anfrage vorliegt.*
- $p_{state} \in \{None, Running, Called\}$ *gibt den Zustand eines Prozesses an.*

Definition 4. *Sei* $\mathcal{P}$ *eine Liste von Prozessstrukturen die auf dem System laufen. Eine solche Liste wird* **Prozessliste** *genannt.*

In Algorithmus 1 wird der Ablauf des Contiki-Kernels dargestellt. In den Zeilen 1 bis 4 werden die Variablen des Algorithmus initialisiert. Die Variable E wird benutzt um das aktuelle Ereignis, welches bearbeitet wird zu speichern. Die Variable *PollRequested* wird benutzt um zu Signalisieren, dass eine Poll-Anfrage, welche durch einen Interrupt ausgelöst wird, für einen Prozess anliegt. Beim Start wird zunächst das Contiki-System und die HW-Funktionen des Systems initialisiert (Zeile 5), und Anwendungsprozesse gestartet, die zum Systemstart laufen werden sollen.

Algorithmus 1 : Initialisierung des Contiki-Kernels und Hauptschleife

```
   Variablendeklaration :
1  P ← ∅;                                                          // Prozessliste
2  E^MaxEvents ← ∅;                                                // Ereignissliste
3  PollRequested ← false;   // Signalisiert eine Poll-Anfrage, wird auf true gesetzt durch Interrupts
4  E ← ∅ ;                                                         // Hilfsvariable zum speichern eines Events
5  Systemstart;
6  while (true) do
7      DoPoll();
8      if E^MaxEvents.size > 0 then
9          E ← E^MaxEvents.dequeue;                                // Entfernt erstes Ereignis aus Ereignisliste
10         if E.e_process = ProcessBroadcast then                  // Ereignis wird an alle Prozesse gesendet
11             foreach P in P do
12                 DoPoll();   // Ausführen einer Poll-Anfrage zwischen dem Starten von Prozessen
13                 CallProcess(E.e_type,P,E.e_data);
14         else                                                    // Prozess wird mit Ereignis gestartet
15             if E.e_type = ProcessInit then
16                 E.e_process ↦ p_state ← Running;
17             CallProcess(E.e_type,E.e_process, E.e_data);
18     Optionale Schlafphase um Energie einzusparen;
```

Die Hauptschleife des Betriebssystems ist in Zeile 6-18 des Algorithmus gezeigt. In Zeile 7 wird mit der Funktion `DoPoll` geprüft ob eine Poll-Anfrage vorliegt. Dazu werden in der Liste alle Prozesse geprüft, ob für einen konkreten Prozess eine Poll-Anfrage vorliegt. Ist dies der Fall, wird der Prozess aufgerufen. Poll-Anfragen werden immer vor der Abarbeitung von Ereignissen bearbeitet und haben somit eine höhere Priorität. Anschließend wird geprüft, ob ein Ereignis zum Abarbeiten in der Ereignisliste vorliegt (Zeile 8). Ist dies der Fall, wird es aus der Liste entfernt und in die Variable E (Zeile 9) gespeichert. Abhängig vom Typ des Ereignisses wird das Ereignis entweder an alle Prozesse (Ereignistyp *Process-Broadcast*, Zeile 10-13) oder an einen spezifischen Prozess gesendet (Zeile 14-17). Bei einem *Process-Broadcast* wird zwischen dem Aufrufen der Prozesse geprüft, ob ein Interrupt aufgetreten ist, welcher bearbeitet werden muss (`DoPoll`, Zeile 12). Dies stellt sicher, dass Interrupts mit höchster Priorität abgearbeitet werden. Wenn das Ereignis zu einem spezifischen Prozess gesendet wird, wird noch zusätzlich geprüft, ob das Ereignis dazu dient den Prozess zu starten und dementsprechend der Status des Prozesses geändert (Zeile 15-16). Der Aufruf eines Prozesses erfolgt mit der Funktion `CallProcess` (Zeile 13 und 17), welche den Prozess mit den zu dem Ereignis gehörenden Daten aufruft. Der Prozess wird an der Stelle an welcher er unterbrochen wurde, wiederaufgenommen, bis er die Kontrolle an den Kernel zurückgibt. Wenn die Ereignisliste leer ist, kann der Prozessor in einen Schlafmodus versetzt werden (Zeile 18), um Energie einzusparen. Dieser Modus wird verlassen, wenn ein Interrupt auftritt. Durch den Poll-Mechanismus werden anschließend wieder Prozesse angestoßen.
In Abbildung 2 wird noch einmal der Ablauf der involvierten Prozesse und Interrupts aus dem Beispiel in Abschnitt 3.1 gezeigt. Die Abbildung entspricht den Zeilen 1-9 der Beispiel-Anwendung.

```
1  int current_time;
2  int delay_time;
3  ...//Initialisierung
4  while (1) {
5     current_time = clock_time();
6     statement_1;
7     ...
8     statement_n;
9     assert(clock_time() < (current_time + delay_time));
10 }
```

(a) Programm, kann durch Interrupt unterbrochen werden

```
1  void timer_interrupt(void) {
2     count++;
3     if ((count % CLOCK_CONF_SECOND) == 0)
4        seconds++;
5     if(etimer_pending() &&
6        (etimer_next_expiration_time() − count − 1)
7        > MAX_TICKS) {
8        etimer_request_poll();
9     }
10 }
```

(b) Modell eines Interrupt-Treibers für Contiki

Abbildung 3: Verallgemeinerte Beispielanwendung mit Interrupts

4. Verifikation und Modellierung für Software-Model-Checking

4.1. Verifikationsansatz

Zur formalen Verifikation des eingebetteten Systems soll ein SMC-Tool verwendet werden. SMC ermöglicht es den Quellcode einer Anwendung zu verifizieren. Für unsere Untersuchungen wird CBMC verwendet, welches die für die Verifikation von Contiki notwendigen **C** Sprachfeatures wie Zeiger, Funktionszeiger, sowie Makros unterstützt. CBMC erlaubt es Sicherheitseigenschaften mit Hilfe von `assert`-Statements, direkt in den Quellcode des zu untersuchenden Programmes zu schreiben. Hierbei prüft z.B. `assert(x==0)`, ob für alle möglichen Programmabläufe an der Stelle des Aufrufs, `x` den Wert `0` besitzt. Zur Beschreibung von Eingaben an die SW z. B. durch Nutzereingaben oder HW-Kommunikation werden nicht-determinierte Variablen unterstützt, welche in ihrem Wertebereich eingeschränkt werden können. Der Ausdruck

```
int x = nondet_int(); __CPROVER_assume(x <=10);
```

beschreibt, dass `x` einen beliebigen Integer-Wert kleiner zehn besitzt. Um SMC für eingebettete Systeme durchführen zu können, müssen alle Eingaben aus der Umgebung des Systems modelliert werden. Für HW-Treiber innerhalb von Contiki wurde eine solche Modellierung in [VRH12] gezeigt.

4.2. Modellierung von Interrupts

Eine besondere Herausforderung bei der Verifikation von SW ist die Modellierung von Interrupts. Beim Auftreten eines Interrupts, wird der aktuelle Zustand des Systems gespeichert und eine dem Interrupt zugeordnete Interrupt-Service-Routine (ISR) ausgeführt. Nach dem Abarbeiten dieser Routine, wird die Ausführung des Programms an der ursprünglichen Stelle wieder aufgenommen. Da Interrupts asynchron zum normalen Programmablauf auftreten, kann die Programmierung fehleranfällig sein, insbesondere da Registerinhalte durch Interrupts modifiziert werden. In Contiki werden ISR verwendet um Prozesse über Poll-Anfragen anzustoßen. Die Modellierung des Interrupt-Verhalten ist deshalb essentiell um das Systemverhalten, insbesondere um die Reihenfolge von Prozessen abzubilden. Weiterhin gibt es bei einem Betriebssystem keine Einschränkung hinsichtlich der Zahl an auftretenden Interrupts, da ein Betriebssystem normalerweise nicht terminiert. Die folgenden Betrachtungen beziehen sich auf Systeme, welche einen periodisch auftretenden Interrupt verwenden. Die Herausforderungen bei der Modellierung von Interrupts, werden im Beispiel in Abbildung 3a und der ISR in Abbildung 3b gezeigt.
In dieser Beispielanwendung speichern die globalen Variablen `count` und `seconds` die aktuelle Systemzeit. Diese Variablen werden ausschließlich inkrementiert, wenn der Interrupt aus Abbildung 3b

```
1  int current_time;
2  int delay_time;
3  ..//Initialisierung
4  while (1) {
5    if (!nondet_0())// nicht−determinierter Aufruf
6      timer_interrupt();
7    current_time = clock_time();
8    if (!nondet_0())
9      timer_interrupt();
10   statement_1;
11   ...
12   if (!nondet_0())
13     timer_interrupt();
14   statement_n;
15   if (!nondet_0())
16     timer_interrupt();
17   assert(clock_time() < (current_time + delay_time));
18 }
```

(a) Ohne Reduzierung der Interrupt-Aufrufe

```
1  int current_time;
2  int delay_time = 5;
3  ..//Initialisierung
4  while (1) {
5    if (!nondet_0()) // nicht−determinierter Aufruf
6      interrupt_wrapper();
7    current_time = clock_time();
8    statement_1;
9    ...
10   statement_n;
11   if (!nondet_0())
12     interrupt_wrapper();
13   assert(clock_time() < (current_time + delay_time));
14 }
15
16
17
18
```

(b) Reduzierung der Interrupt Aufrufe mit POR

Abbildung 4: Anwendung nach Instrumentierung mit Interrupts

ausgelöst wird. In der Beispielanwendung wird zunächst in Zeile 5 mit Hilfe der Contiki-API Funktion `clock_time`, der aktuelle Wert von `count` in `current_time` gespeichert. Anschließend werden n beliebige Programmschritte ausgeführt (Zeile 6-8), in denen nicht auf Variablen der ISR und `current_time` zugegriffen wird. Die Eigenschaft in Zeile 9 prüft, dass die vergangene Zeit sich nur um den Wert der Variable `delay_time` gegenüber dem in `current_time` gespeicherten Wert ändern darf. In Abbildung 3b wird die zum System zugehörige ISR für das Contiki *Event-Timer*-System gezeigt. Immer wenn der Interrupt aufgerufen wird, wird die Variable `count` inkrementiert (Zeile 2). Die Variable `seconds` wird inkrementiert, wenn eine Sekunde Realzeit auf der HW-Plattform vergangen ist (Zeile 3-4). Ist ein *Event-Timer* im System registriert, und dessen gesetzte Zeit erreicht (Zeile 5-7[1]), so wird über den Poll-Mechanismus (Zeile 8) der *Event-Timer* Prozess informiert (vgl. Abbildung 2).
Um Interrupts auf der Ebene eines **C**-Programms zu modellieren gibt es verschiedene Ansätze. Eine Möglichkeit besteht darin, ISR-Aufrufe zu sequentialisieren und an jeder Anweisung im Quellcode einen Interrupt-Aufruf einzufügen [BK11]. Die so transformierte Beispielanwendung wird in Abbildung 4a gezeigt. Zwischen jeder Programmanweisung des ursprünglichen Programms erfolgt jetzt ein Aufruf der ISR `timer_interrupt`. Der Aufruf der ISR erfolgt nicht-determiniert mit Hilfe der Funktion `nondet_0`, welche CBMC zur Verfügung stellt. Um die Größe des resultierenden Programms zu reduzieren, können die Aufrufe der ISR mit *Partial Order Reduction* (POR) reduziert werden. Diese Technik ist u.a. von [SNBB11] und [BK11] implementiert wurden. POR reduziert die Aufrufe der ISR auf Anweisungen, an denen ein Interrupt den tatsächlichen Ablauf des Programms beeinflusst. Die so transformierte Beispielanwendung wird in Abbildung 4b gezeigt. Durch die Verwendung von POR entspricht jedoch die Anzahl der ISR-Aufrufe nicht mehr der, welche im ursprünglichen Programm möglich war (die Anzahl ist unabhängig von der Anzahl der Programmschritte n). Eine korrekte Über-Abstraktion des Systemverhaltens kann dadurch erreicht werden, dass der Aufruf des Interrupts mit Hilfe der Wrapper-Funktion in Abbildung 5a gekapselt wird. Dieser Wrapper erlaubt es den ursprünglichen `timer_interrupt` beliebig oft aufzurufen. Da diese Modellierung alle möglichen Programmabläufe berücksichtigt, ist das Programm hinsichtlich der untersuchten Eigenschaften vollständig verifiziert.

[1]Der Vergleich beruht darauf, dass -1 in einer vorzeichenlosen Zahlendarstellung dem größtmöglichen Integer-Wert entspricht.

```
1  void interrupt_wrapper(void) {
2    while (1) {
3      if (!nondet_0())
4        timer_interrupt();
5      else
6        return;
7    }
8  }
9
10
11
12
```

(a) Interrupt-Wrapper-Funktion für POR

```
1  static unsigned int statement_counter;
2  unsigned int interrupt_period = 10;
3  void initialize_interrupt(void) {
4    statement_counter = nondet_int();
5    __CPROVER_assume(statement_counter < interrupt_period);
6  }
7  void periodic_interrupt(void) {
8    statement_counter++;
9    if (statement_counter == interrupt_period) {
10     timer_interrupt();
11     statement_counter = 0;}
12 }
```

(b) Funktionen für *Periodic interrupt modeling*

Abbildung 5: Wrapper-Funktionen für Modellierte Interrupt Funktionen

Bewertung der Modellierungsverfahren Die beschriebenen Modellierungen von Interrupts beruhen auf der Annahme, dass nach jeder Anweisung ein ISR-Aufruf auftreten kann. So modellierte Interrupts erlauben es jedoch nicht Eigenschaften zu verifizieren, die sich direkt oder indirekt auf die Häufigkeit von Interrupt-Aufrufen beziehen, wie die Eigenschaft aus der Beispiel-Anwendung. Ursache dafür ist, dass die Modellierung nicht das periodische Auftreten von Interrupts berücksichtigt. In Abbildung 4a bestimmt die Anzahl der Programmschritte n zwischen Speichern der Systemzeit in Zeile 7 und dem Überprüfen in Zeile 17 die Anzahl der ISR-Aufrufe, und somit ob die Eigenschaft erfolgreich verifiziert werden kann. Überprüft man das Programm aus Abbildung 4b schlägt die Überprüfung fehl, da die Variable `count` unbegrenzt oft inkrementiert werden kann. Mit den beschriebenen Modellierungsverfahren ist es weiterhin nicht möglich zu zeigen, ob die Größe der Contiki Ereignisliste ausreichend ist, also ob immer gilt $|\mathcal{E}^{MaxEvents}| \leq MaxEvents$. Um Eigenschaften, die sich auf die Häufigkeit von Interrupts beziehen, ebenfalls überprüfen zu können, wird das Modellierungsverfahren (*Periodic Interrupt Modelling*) für Interrupts vorgestellt, welches die Häufigkeit von Interrupt-Aufrufen berücksichtigt.

Periodic Interrupt Modelling Die Modellierung beruht darauf, dass für einen periodisch auftretenden Interrupt der Abstand der ISR-Aufrufe eines Systems aus der Taktfrequenz des Prozessors und der Dauer eines Befehls abgeschätzt werden kann[2]. Dieser Wert wird anschließend bei der Modellierung der ISR berücksichtigt. In Abbildung 5b ist eine solche Modellierung, bestehend aus zwei Funktionen, gezeigt. In `initialize_interrupt` werden die benötigten Variablen initialisiert, `periodic_interrupt` ist eine Wrapper-Funktion für den eigentlichen ISR-Aufruf. Die Variable `statement_counter` wird verwendet um zu Zählen, wie viele Anweisungen zwischen dem letzten Aufruf der ISR vergangen sind. Um den Abstand zwischen den Aufrufen der ISR festzulegen, wird die (auch zur Laufzeit veränderbare) Variable `interrupt_period` verwendet. Die Funktion `initialize_interrupt` muss als erste Anweisung des Programms (innerhalb der **C**-`main`-Funktion) aufgerufen werden. Innerhalb der Funktion wird `statement_counter` auf einen beliebigen Wert im Bereich $0 \leq$ `statement_counter` $<$ `interrupt_period` gesetzt, um das Verhalten zu beschreiben, dass zum Start des Systems nicht bekannt ist, wie viel Zeit bis zum nächsten Interrupt-Aufruf benötigt wird. Durch die Verwendung von nicht-determinierten Variablen wird der Zustandsraum für alle möglichen Varianten durchsucht.
Die eigentliche Überprüfung, ob die ISR aufgerufen werden muss, erfolgt mit der Funktion `periodic_interrupt`. Diese Funktion wird, wie in Abbildung 4a gezeigt, und nach jeder Anweisung des ur-

[2] Eine genauere Abschätzung der Dauer eines Befehls, benötigt Informationen über Zielarchitektur und verwendete Compiler und wird in dieser Arbeit nicht berücksichtigt.

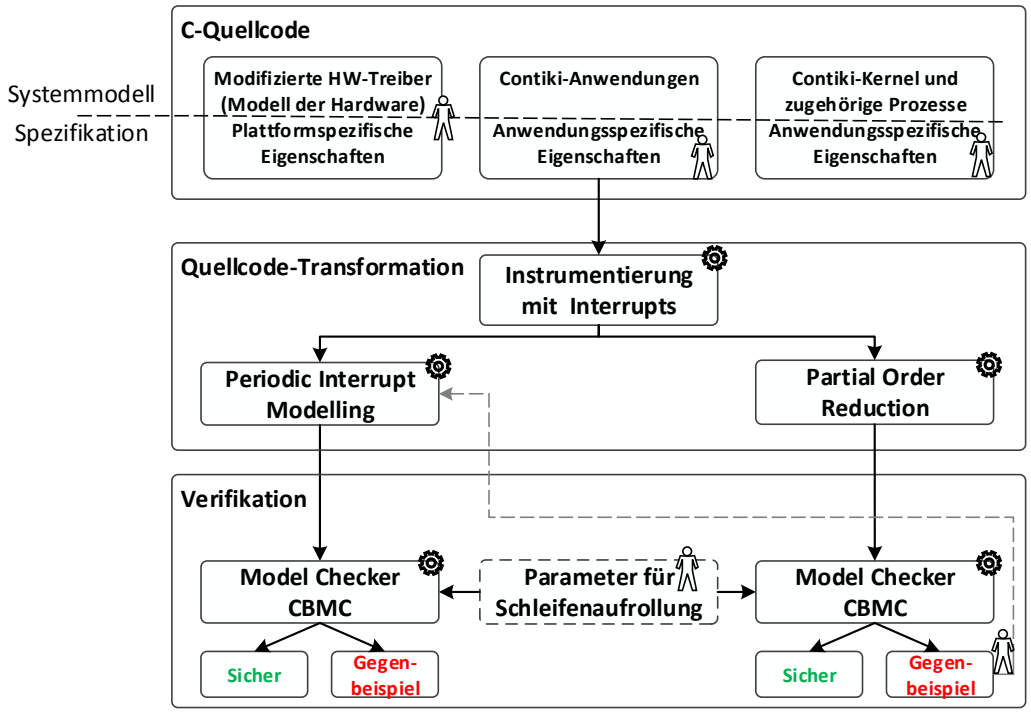


Abbildung 6: Verifikationsframework für Contiki-Anwendungen

sprünglichen Programms aufgerufen. Beim Aufruf wird die Variable `statement_counter` inkrementiert (Zeile 8 in Abbildung 5b). Wenn der Wert von `interrupt_period` erreicht wird, wird die ursprüngliche ISR aus Abbildung 3b aufgerufen und der Zähler zurückgesetzt (Zeile 9-11).

4.3. Das Verifikationsframework

Die vorgestellten Ansätze zur Modellierung von Anwendungen für das Contiki-Betriebssystem wurden in einem Framework umgesetzt, welches in Abbildung 6 gezeigt wird. Zuerst wird der gesamte Quellcode bestehend aus Modellen für HW-Treiber, Betriebssystem-Kernel und Anwendungen zusammengefasst. Das resultierende Programm kann anschließend mit Interrupts instrumentiert werden. Danach wird entschieden, ob die Interruptaufrufe mit POR reduziert werden oder *Periodic Interrupt Modelling* verwendet wird. Diese Quellcode-Transformation ist dabei automatisch ausführbar. Für die Verifikation mit CBMC müssen alle unbeschränkten Schleifen eines Systems begrenzt werden. Dies sind die in Algorithmus 1 gezeigte Hauptschleife des Contiki Systems, sowie die Schleife des POR-Interrupts-Wrappers in Abbildung 5a. Alle anderen Schleifen in Contiki sind begrenzt und ihre Aufrolltiefen können automatisch ermittelt bzw. geschätzt werden, da CBMC prüft, ob die gewählte Aufrolltiefe für Schleifen ausreichend ist.
Verifikationsergebnisse zeigen, dass aufgrund der zusätzlichen Aufrufe der Interruptfunktion *Periodic Interrupt Modelling* insbesondere bei dem Erstellen der SAT-Formel für Bounded-Model-Checking deutlich langsamer ist. Im Gegensatz zu POR ist die Verifikationszeit jedoch praktisch unabhängig von der Anzahl de untersuchten ISR-Aufrufe. Bei POR steigt mit der Anzahl der möglichen untersuchten ISR-Aufrufe auch die Größe des Verifikationsproblems. Es ist trotzdem empfehlenswert, erst die Verifikation mit POR durchzuführen. Ist diese erfolgreich, ist aufgrund der Über-Approximation des Systemverhaltens die Eigenschaft verifiziert. Kommt es zu einem Gegenbeispiel, muss geprüft werden, ob dieses aufgrund unpräziser Interruptmodellierung entsteht. In diesem Fall können jedoch die ermittelten Werte für die Tiefe von Schleifen für die Verifikation mit *Periodic Interrupt Modelling* übernommen werden.

5. Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Ansatz zur formalen Verifikation mit Hilfe von SMC für eingebettete Systeme, welche auf dem Betriebssystem Contiki basieren, vorgestellt. Die Modellierung von Interrupts spielt dabei eine entscheidende Rolle. Mit *Periodic Interrupt Modelling* wurde eine neue Modellierungstechnik vorgestellt, welche es ermöglicht Eigenschaften zu überprüfen, die sich auf die Häufigkeit des Auftretens von periodischen Interrupts beziehen. In zukünftigen Arbeiten soll untersucht werden wie der Ansatz bei der Modellierung mehrere Interrupts und unterschiedlicher Häufigkeit von Interrupt-Aufrufen skaliert.

Literatur

[AVR14] *Atmel AVR 8- and 32-bit Microcontrollers*. `http://www.atmel.com/products/avr/`, Juli 2014.

[BK11] Bucur, D. und M. Kwiatkowska: *On software verification for sensor nodes*. Journal of Systems and Software, 84(10):1693–1707, 2011, ISSN 0164-1212.

[CGP00] Clarke, E. M., O Grumberg und D. A. Peled: *Model checking*. MIT, Cambridge and Mass., 2. print. Auflage, 2000, ISBN 0262032708.

[CKL04] Clarke, E., D.l Kroening und F. Lerda: *A Tool for Checking ANSI-C Programs*. In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2004)*, Band 2988 der Reihe *Lecture Notes in Computer Science*, Seiten 168–176. Springer, 2004, ISBN 3-540-21299-X.

[Con14] *Contiki OS*. `http://www.contiki-os.org/`, Juli 2014.

[CRJ13] Chunxiao, Li, A. Raghunathan und N. K. Jha: *Improving the Trustworthiness of Medical Device Software with Formal Verification Methods*. Embedded Systems Letters, IEEE, 5(3):50–53, 2013.

[DGV04] Dunkels, A., B. Gronvall und T. Voigt: *Contiki - a lightweight and flexible operating system for tiny networked sensors*. 2004. 29th Annual IEEE International Conference on Local Computer Networks, Seiten 455–462, 2004, ISSN 0-7695-2260-2.

[DKW08] D'Silva, V., D. Kroening und G. Weissenbacher: *A Survey of Automated Techniques for Formal Software Verification*. IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, 27(7):1165–1178, 2008, ISSN 0278-0070.

[DSVA06] Dunkels, A., O. Schmidt, T. Voigt und M. Ali: *Protothreads: Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems*. In: *Proceedings of the Fourth ACM Conference on Embedded Networked Sensor Systems (SenSys 2006)*, Boulder and Colorado and USA, 2006.

[KLM+15] Kroening, D., L. Liang, T. Melham, P. Schrammel und M. Tautschnig: *Effective Verification of Low-Level Software with Nested Interrupts*. In: *Design, Automation and Test in Europe (DATE)*. IEEE, 2015.

[LFCJ09] L. Cordeiro, B. Fischer, H. Chen und J. Marques-Silva: *Semiformal Verification of Embedded Software in Medical Devices Considering Stringent Hardware Constraints*. Embedded Software and Systems, Second International Conference on, Seiten 396–403, 2009.

[MSP14] *MSP430 Low Power Microcontroller*. `http://www.ti.com/430brochure`, Juli 2014.

[MVÖ+10] Mottola, Luca, Thiemo Voigt, Fredrik Österlind, Joakim Eriksson, Luciano Baresi und Carlo Ghezzi: *Anquiro: enabling efficient static verification of sensor network software*. In: *Proceedings of the 2010 ICSE Workshop on Software Engineering for Sensor Network Applications*, Seiten 32–37. ACM, 2010.

[SNBB11] Schlich, B., T. Noll, J. Brauer und L. Brutschy: *Reduction of Interrupt Handler Executions for Model Checking Embedded Software*. In: *Hardware and Software: Verification and Testing*, Band 6405 der Reihe *Lecture Notes in Computer Science*, Seiten 5–20. Springer Berlin / Heidelberg, 2011.

[VRH12] Vörtler, T., S. Rülke und P. Hofstedt: *Bounded model checking of Contiki applications*. In: *Design and Diagnostics of Electronic Circuits Systems (DDECS), 2012 IEEE 15th International Symposium on*, Seiten 258–261, 2012.

[Wir14] *Out in the Open: The Little-Known Open Source OS That Rules the Internet of Things*. `http://www.wired.com/2014/06/contiki/`, Juni 2014.

Towards Verification of Artificial Neural Networks *

Karsten Scheibler, Leonore Winterer, Ralf Wimmer, and Bernd Becker
Chair of Computer Architecture
Albert-Ludwigs-Universität Freiburg im Breisgau, Germany
{scheibler,winterel,wimmer,becker}@informatik.uni-freiburg.de

Abstract

We consider the safety verification of controllers obtained via machine learning. This is an important problem as the employed machine learning techniques work well in practice, but cannot guarantee safety of the produced controller, which is typically represented as an artificial neural network. Nevertheless, such methods are used in safety-critical environments.

In this paper we take a typical control problem, namely the Cart Pole System (a. k. a. inverted pendulum), and a model of its physical environment and study safety verification of this system. To do so, we use bounded model checking (BMC). The created formulas are solved with the SMT-solver iSAT3. We examine the problems that occur during solving these formulas and show that extending the solver by special deduction routines can reduce both memory consumption and computation time on such instances significantly. This constitutes a first step towards verification of machine-learned controllers, but a lot of challenges remain.

1. Introduction

Machine learning [MRT12] is a powerful method to derive a controller automatically for systems which are hard to handle otherwise: There may either be no closed-form description of the environment, or the optimal behavior cannot be determined, or the system has to adapt itself to changing environments. The user specifies which variables describe the state space of the system and the possible actions that can be used to control it. For so-called reinforcement learning [SB98], additionally a reward (or, equivalently, cost) function is used which measures how desirable a certain state is. Machine learning methods then explore the state space using the available actions. The obtained information about the system is used to derive a controller which maximizes the reward. Since exploring the whole state space is typically infeasible, the learning algorithms generalize the information obtained by exploration to unexplored parts. When the learning phase is completed, the result is a controller that is typically represented as an artificial neural network.

Such learned controllers are used successfully in many scenarios. Many of them are safety-critical; examples are controlling autonomous driving [For02], deep-brain stimulation against epileptic

*This work has been partially supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (DFG, SFB/TR 14 AVACS, http://www.avacs.org/) and by the Cluster of Excellence BrainLinks-BrainTools (DFG, grant number EXC 1086, http://www.brainlinks-braintools.uni-freiburg.de/)

seizures [KWB+14, BCP08, PMG+14], and robots [KCC13]. However, the learning algorithms in general cannot provide any guarantees regarding the safety of the controller. Therefore, after learning, a verification step should be included, which checks that the controller actually prevents the system from running into an unsafe state. Nevertheless, only few works have addressed this issue so far [Bor14, Gal14]. In this paper we present our first attempts to improve this situation.

We apply bounded model checking [BCC+03] to a typical control problem, namely the Cart Pole System, also known as inverted pendulum [BSA83]: a cart moves left and right on a rail and has to balance a loosely attached pole on top of it. Since both the neural network representing the controller and the description of the physical environment are non-linear (they contain transcendental functions like exp and cos), an SMT solver is needed which can handle such constraints. The state-of-the-art SMT solver iSAT3 [SKB13] is one of the few solvers which can cope with transcendental functions. It is based on interval deductions to exclude variable assignments which cannot satisfy the formula, and is tightly integrated with a modern SAT-solver to handle the Boolean structure of the formula.

We first present the translation of the closed-loop system into an SMT formula by discretizing the differential equation that describes the behavior of the cart pole using a standard Runge-Kutta method. As experiments confirm, the problem of this approach is that for formula generation the loops typically executed during the discretization have to be unrolled, leading to a large number of auxiliary variables and constraints and consequently to high memory requirements and long computation times. We solve this problem by extending the solver core with problem-specific operations. Experiments confirm that this reduces the memory consumption significantly as well as the computation times. However, this constitutes only the first step towards an efficient verification of such controllers as in spite of our improvements the analysis is restricted to rather short time horizons.

Structure of the paper In the following section, we introduce the Cart Pole System, define artificial neural networks, and sketch how iSAT3 works. In Section 3 we describe the translation of the controlled Cart Pole System into the iSAT3 formalism and analyze the problems with this approach. In Section 4 we propose the introduction of user-defined operations to speed up the solution of the BMC formulas. Section 5 presents our experimental results, and the paper finishes in Section 6 with a conclusion and pointers to future work.

2. Preliminaries

In this section we first describe the system we want to analyze, namely a Cart Pole System, which is a typical example from control theory and machine learning. We then define multi-layer perceptrons; they are the variant of artificial neural networks that we use to represent controllers. Finally, we describe the core mechanisms underlying the SMT-solver iSAT3 used in our studies for bounded model checking.

2.1. Cart Pole System

A standard benchmark for machine learning is the Cart Pole system (or inverted pendulum) – a cart that is attached to a rail moves left and right, trying to balance a loosely attached pole at right angle on top of it. Several versions of the model exist, with more complex ones having the start position with the pole hanging upside down, requiring the algorithm to first perform a ‘swing up’ move

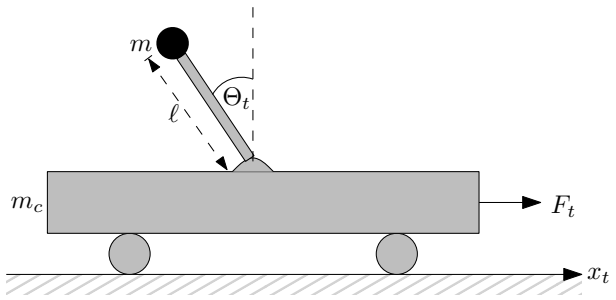


Figure 1: Visualization of the Cart Pole Model

before starting the balancing phase. The model used here has the pole starting in an almost-upright position and requires the cart to move to a certain area of the rail without losing balance of the pole.

The exact movements of the cart and pole are described by the following ordinary differential equations, taken from [BSA83]. They describe the evolution of the current position x_t of the cart as well as the angle Θ_t of the pole, depending on the current time $t \geq 0$ (cf. Figure 1):

$$\ddot{\Theta}_t = \frac{g \cdot \sin\Theta_t + \cos\Theta_t \cdot \left[\frac{-F_t - m \cdot \frac{\ell}{2} \cdot \dot{\Theta}_t^2 \cdot \sin\Theta_t + \mu_c \cdot \mathrm{sgn}(\dot{x}_t)}{m_c + m}\right] - \frac{\mu_p \cdot \dot{\Theta}_t}{m \cdot \frac{\ell}{2}}}{\frac{\ell}{2}\left[\frac{4}{3} - \frac{m \cdot \cos^2(\Theta_t)}{m_c + m}\right]}, \tag{1a}$$

$$\ddot{x}_t = \frac{F_t + m \cdot \frac{\ell}{2} \cdot \left[\dot{\Theta}_t^2 \cdot \sin(\Theta_t) - \ddot{\Theta}_t \cdot \cos(\Theta_t) - \mu_c \cdot \mathrm{sgn}(\dot{x}_t)\right]}{m_c + m}, \tag{1b}$$

The constant $g \approx 9.81 \frac{\mathrm{m}}{\mathrm{s}^2}$ is the gravity acceleration of the earth, m_c and m are the masses of the cart and pole, resp., ℓ is the length of the pole, μ_c and μ_p are the coefficients of friction for the cart on the rails and the pole on the cart, and F_t is the force applied to the cart's center at time t. The value of the external force F_t is the parameter which can be used to control the system. The current state of the system is given by the vector $(\Theta_t, \dot{\Theta}_t, x_t, \dot{x}_t)$.

2.2. Artificial Neural Networks

The external force F_t in our cart pole example is controlled by an artificial neural network. It obtains the current state of the system and returns the value of the force to be applied. The particular type of neural network we use in this paper is a multi-layer perceptron:

Definition 1 (Multi-layer perceptron) *A* multi-layer perceptron *is a weighted directed acyclic graph $G = (V, E, w)$ with V being a finite set of nodes, $E \subseteq V \times V$ the set of edges, and $w : E \to \mathbb{R}^{\geq 0}$ the edge-weight function. Nodes $I \subseteq V$ without incoming edges are called* input nodes*, nodes $O \subseteq V$ without outgoing edges* output nodes*.*

For a multi-layer perceptron $G = (V, E, w)$ and a node $v \in V$ we denote the set of predecessor nodes of v by $\mathrm{pre}(v)$ and the set of successor nodes by $\mathrm{succ}(v)$. Given an input assignment $\mathrm{val}(v)$ for the input nodes $v \in I$, we can assign a value to each node of the network by traversing it in topological order, using the activation function of the nodes:

$$\mathrm{val}(v) = \frac{1}{1 + \exp\Big(\sum_{u \in \mathrm{pre}(v)} w(u, v) \cdot \mathrm{val}(u)\Big)} \quad \text{for } v \in V \setminus I. \tag{2}$$

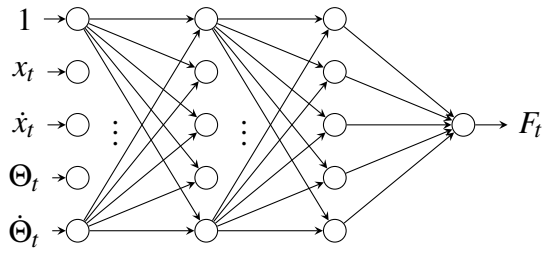


Figure 2: Visualization of a simple neural network with two hidden layers as used in this work (the edge weights have been omitted for the sake of readability).

Other activation functions are possible. The output of the network is then given by the values assigned to the output nodes O. Figure 2 shows an illustration of the neural network used to control the Cart Poly System in our experiments.

Machine learning trains a multi-layer perceptron by adapting the weight function w in order to minimize the difference between the actual output of the perceptron network and its expected output. Standard algorithms to do so are the back-propagation algorithm [RHW86] and reinforcement learning [SB98]. We omit the details about the learning, because in our scenario we are given a readily trained multi-layer perceptron which computes the external force applied to the cart (normalized to the interval $[0,1]$). For more details we refer the reader to the machine learning literature, e. g., [Bis96].

2.3. SAT and SMT

The *propositional satisfiability problem* (SAT) addresses the question whether there exists an assignment for the variables of a propositional formula such that the formula is satisfied (evaluates to true). Programs handling this kind of problems are called SAT-solvers. Usually, SAT-solvers expect formulas to be given in *conjunctive normal form* (CNF). A CNF consists of a conjunction of clauses. Each clause is a disjunction of literals and each literal is a Boolean variable or its negation. The Tseitin-transformation [Tse68] allows the conversion of arbitrary propositional formulas into CNF.

In [DLL62] the DPLL method was proposed. In DPLL the search for a satisfying assignment exploits that a CNF is satisfied if and only if all clauses are satisfied. If during the search process a (partial) assignment of the variables results in an unsatisfied clause (a clause with all literals being false), backtracking is performed in order to enter a different part of the search space. Additionally, *Boolean constraint propagation* (BCP) is used to detect implied assignments. Everytime a clause with n literals contains $n-1$ literals being false, the remaining literal has to be true in order to retain a chance to satisfy the formula. Moreover, today's SAT-solvers do non-chronological backtracking and add conflict clauses to the formula to prune the search space even further – so-called *conflict-driven clause learning* (CDCL) [SS96].

SAT Modulo Theories (SMT) allows Boolean combinations of so-called theory atoms. A theory atom is an arithmetic expression which can be evaluated to true or false, e. g., $(x+y<10)$. Depending on the underlying theory such an expression may contain linear and/or non-linear arithmetic. When doing *eager SMT* a given Boolean combination of theory atoms is completely

translated into an equi-satisfiable propositional formula. This formula is then passed to a SAT-solver. In many cases such an approach is either infeasible (the CNF would become to large or the translation itself would take to long) or not possible at all due to properties of the underlying theory. Therefore, *lazy SMT* is mostly used. In lazy SMT, each theory atom is abstracted by a literal, resulting in a Boolean abstraction of the original SMT formula. This abstraction is given to a SAT-solver. If the abstraction is unsatisfiable, then the original SMT formula is also unsatisfiable. If the SAT-solver returns a satisfying assignment, we employ a theory solver to check if the selected truth values for the theory atoms are indeed a solution. If this is not the case, then a conflict clause is added to the Boolean abstraction which contains the literals of the conflicting theory atoms. Thus, the Boolean abstraction is lazily refined with additional knowledge. This scheme is also abbreviated as DPLL(T) or CDCL(T) – with T being the theory used.

2.4. The iSAT algorithm, iSAT3

The iSAT algorithm [FHT+07, Her11] uses *interval constraint propagation* (ICP, see, e. g., [BG06]) to check the consistency of theory atoms. But in contrast to classical lazy SMT there is no clear separation between the theory and SAT-solver part; instead ICP is tightly integrated into the CDCL framework. This deep integration allows to share the common core of the search algorithms between the propositional and the theory-related part of the solver. Therefore the iSAT algorithm goes beyond CDCL(ICP).

The iSAT algorithm allows theory atoms to contain linear and non-linear arithmetic as well as transcendental functions, e. g., $(x+y^2=z^2)$ or $(\sqrt{x}+\cos z < e^y)$. Furthermore, three variable types are supported: Boolean, integer-, and real-valued variables. The latter two types have to be declared with a bounded initial interval. Before solving a given formula, each theory atom is decomposed into *simple bounds* and *primitive constraints* by introducing fresh auxiliary variables similar to the Tseitin-transformation. A simple bound imposes a lower or upper bound to a variable – either strict, e. g., $(x<1)$ or non-strict $(x\leq 1)$. Since we store these bounds as floating-point numbers, outward rounding is used to get a safe interval enclosure. Furthermore, this decomposition enables us to rely on a fixed set of primitive constraints which are used to apply ICP later on. A primitive constraint consists of a unary or binary operator and the associated variables, e. g., $(h=\sin x)$ or $(h=x+y)$. Applying ICP means, a so-called contractor is used to narrow the intervals of the variables occurring in each primitive constraint in order to exclude definitive non-solutions[1].

The CDCL framework consists of three building blocks: (1) decide, (2) propagate and (3) resolve conflicts. The iSAT-algorithm builds on CDCL and extends it for the operation on integer- and real-valued intervals in addition to Boolean variables. Firstly, deciding an integer- or real-valued variable corresponds to splitting the interval of the variable and selecting the lower or upper part. Secondly, in addition to BCP for Boolean variables, ICP is used to narrow the intervals of the variables occurring in primitive constraints. Similar to BCP we use an implication queue for ICP to keep track of all changed variables – primitive constraints containing these variables will be processed by ICP. The propagation phase is finished if the BCP and ICP implication queues are empty. And finally, conflict resolution has additionally to cope with conflicts occurring in

[1] For instance, if we have the primitive constraint $(h=x+y)$ and $h\in[1,9]$, $x\in[1,3]$ and $z\in[4,10]$ then it is possible to contract the interval of h. This is done by calculating $x+y$ with interval arithmetic and intersecting the result with the current interval of h. In this example $x+y$ would yield $[5,13]$ resulting in $[5,9]$ as the new interval for h. In order to contract the intervals for x and y the primitive constraint is redirected accordingly and a similar calculation is done. In this example x stays unchanged and y could be shrunken to $[4,8]$.

inconsistent primitive constraints. A primitive constraint is consistent if the interval on the left-hand side has a non-empty intersection with the resulting interval of the right-hand side – otherwise the primitive constraint is inconsistent. For instance, $(h = x + y)$ with $h \in [1,1]$ and $x, y \in [0,1]$ is consistent while it would be inconsistent with $h \in [3,3]$, $x \in [0,1]$ and $y \in [-1,1]$.

In order to prevent infinite propagation sequences, we check for every newly deduced bound for a variable if it has reasonable progress compared to the current bound of that variable (so-called *minimum progress*) – if the progress is too small the new bound will be discarded. Furthermore, interval splits are only performed if the considered interval is larger than the so-called *minimal splitting width*. Due to the undecidability of the supported logic, the iSAT-algorithm may terminate with an inconclusive answer. That means an interval box (so-called *candidate solution*) will be returned which may contain a solution. To be precise, all primitive constraints have solutions in this interval box (i. e., all primitive constraints are consistent), but it is not guaranteed that all primitive constraints have a common solution point in that interval box.

In this paper we focus on iSAT3 [SKB13] – the third implementation of the iSAT-algorithm. iSAT3 and its predecessors HySAT [HEFT08] and iSAT [EKKT08] share the same major core principles of tightly integrating ICP into the CDCL framework – but they differ on a more technical level in the handling of the simple bounds. While the core solvers of HySAT and iSAT operate on simple bounds, the core of iSAT3 uses literals. This results in a different way of doing decisions: HySAT and iSAT always perform interval splits when doing a decision – in contrast iSAT3 first decides existing literals, before splitting intervals.

3. Translation into the iSAT3 input language

In a first attempt to verify properties of the Cart Pole System and its neural network controller, we translated both components into the input language of iSAT3. The translation of the neural network was directly possible, because all needed operations are natively supported by iSAT3.

For the translation of the Cart Pole System, we had to abstract the differential equations as iSAT3 does not support these. However, we had access to a C++-implementation which was also used to train the neural networks that were provided to us. In this implementation, the differential equations (1a) and (1b) have been iteratively approximated using the Runge-Kutta-methods [But87] – resulting in two nested loops (as seen in Algorithm 1 and Algorithm 2) computing the next-step values of the parameters, given a fixed step size δ_t (20 ms in all examples used).
Since iSAT3 only supports total operations, we had to rearrange some constraints, e. g.,

$$\ddot{x}_{new} = \frac{(h - m \cdot l \cdot \ddot{\Theta} \cdot \cos(\Theta)) - fric_{cart}}{m_c + m} \tag{3}$$

is translated into iSAT3 syntax as follows:

```
x_pp' * (m_c + m) = h - m * l * Theta_pp' * cos(Theta) - fric_cart;
```

Primed variables (e.g. `x_pp'`) indicate the values computed during the current step and unprimed variables refer to the values computed in the last step. Furthermore, since iSAT3 does not support any loops, we had to unroll the ten iterations of the for-loop which lead to a large instance with over 400 constraints.

ISBN 978-3-00-048889-4

Algorithm 1 Auxiliary function, approximating the second-order derivatives in a given state.

2nd_deriv(x, $\dot{x}$, Θ, $\dot{\Theta}$, F^t)
begin

$$h = F^t + m \cdot \ell \cdot \dot{\Theta}^2 \cdot \sin(\Theta) \tag{1}$$

$$fric_{pole} = \frac{\mu_p \cdot \dot{\Theta}}{m \cdot \ell} \tag{2}$$

$$\ddot{\Theta}_{new} = \frac{((m+m_c) \cdot g \cdot \sin(\Theta) - \cos(\Theta) \cdot h + \cos(\Theta) \cdot \mu_c \cdot \mathrm{sgn}(\dot{x})) - fric_{pole}}{\frac{4}{3}(m+m_c) \cdot \ell \cdot \cos^2(\Theta)} \tag{3}$$

$$fric_{cart} = \mu_c \cdot \mathrm{sgn}(\dot{x}) \tag{4}$$

$$\ddot{x}_{new} = \frac{(h - m \cdot \ell \cdot \ddot{\Theta} \cdot \cos(\Theta)) - fric_{cart}}{m_c + m} \tag{5}$$

return $(\ddot{x}_{new}, \ddot{\Theta}_{new})$ (6)

end

4. Special Deduction Functions

While the translation of the Cart Pole System and the neural network to the input language of iSAT3 was easily doable, we observed a poor performance of the solver on these translated instances. This is mainly due to two problems:

1. As mentioned in Section 2, newly deduced bounds need to have a reasonable progress compared to the current bounds (minimum progress) in order to prevent infinite propagation sequences. Of course, every discarded bound leads to a coarser interval. This could result in additional conflicts occurring later on, forcing the solver to do further decisions and slowing down the overall performance. To mitigate this problem, one could decrease the minimum progress in order to prevent the solver from discarding bounds. But this comes at the cost of millions of new literals (each representing a bound) to be present in the solver and blowing up the implication graph. This could be tackled with implication graph compression as proposed in [SB14] – nonetheless, the following problem would be still present.

2. An implication queue (IQ) is used to keep track of all variables changed. For every variable in the queue we have to apply ICP to all primitive constraints containing this variable. This may result in further changed variables. Depending on the structure of the original constraints, variables could be enqueued in a bad order – i. e., an order where a variable is deduced multiple times whereas a different order would require only one deduction per variable. For example if we have the primitive constraints $C_1 : (h_1 = x + h_2)$, $C_2 : (h_2 = x^2)$. Now assume x is changed and enqueued to the IQ. When x is dequeued from the IQ, we examine the constraint C_1. This results in new bounds for h_1 – hence h_1 is enqueued to the IQ. When examining C_2 we get stronger bounds for h_2 as well – therefore h_2 is enqueued too. Now we would dequeue h_1 from the IQ and proceed with all primitive constraints containing this variable. If we now dequeue h_2 from the IQ we have to examine C_1 once more – yielding again new bounds for h_1. With a different ordering (h_2 before h_1) we would consider h_1 only once. This example illustrates that the order of the IQ impacts how many deductions are made. Since the primitive constraints are deduced in all directions, it is not sufficient to just apply a topological sorting. Unfortunately, the translated instances contain hundreds of constraints with a structure as shown in the example. That means the overall deduction process has to deal with a huge overhead.

Algorithm 2 Given the current state, `NextState` computes the successor state after δ_t time steps.

NextState(x^0, $\dot{x}^0$, Θ^0, $\dot{\Theta}^0$, F, $\varepsilon = \frac{\delta_t}{10}$)
begin

for $i = 1$ **to** 10 **do** (1)

$(\ddot{x}_1^i, \ddot{\Theta}_1^i) = 2nd\_deriv(x^{i-1}, \dot{x}^{i-1}, \Theta^{i-1}, \dot{\Theta}^{i-1}, F^t)$ (2)

$x_1^i = x^{i-1} + \frac{\varepsilon}{2} \cdot \dot{x}^{i-1}$ (3)

$\dot{x}_1^i = \dot{x}^{i-1} + \frac{\varepsilon}{2} \cdot \ddot{x}_1^i$ (4)

$\Theta_1^i = \Theta^{i-1} + \frac{\varepsilon}{2} \cdot \dot{\Theta}^{i-1}$ (5)

$\dot{\Theta}_1^i = \dot{\Theta}^{i-1} + \frac{\varepsilon}{2} \cdot \ddot{\Theta}_1^i$ (6)

(7)

$(\ddot{x}_2^i, \ddot{\Theta}_2^i) = 2nd\_deriv(x_1^i, \dot{x}_1^i, \Theta_1^i, \dot{\Theta}_1^i, F^t)$ (8)

$x_2^i = x^{i-1} + \frac{\varepsilon}{2} \cdot \dot{x}_1^i$ (9)

$\dot{x}_2^i = \dot{x}^{i-1} + \frac{\varepsilon}{2} \cdot \ddot{x}_2^i$ (10)

$\Theta_2^i = \Theta^{i-1} + \frac{\varepsilon}{2} \cdot \dot{\Theta}_1^i$ (11)

$\dot{\Theta}_2^i = \dot{\Theta}^{i-1} + \frac{\varepsilon}{2} \cdot \ddot{\Theta}_2^i$ (12)

(13)

$(\ddot{x}_3^i, \ddot{\Theta}_3^i) = 2nd\_deriv(x_2^i, \dot{x}_2^i, \Theta_2^i, \dot{\Theta}_2^i, F^t)$ (14)

$x_3^i = x^{i-1} + \varepsilon \cdot \dot{x}_2^i$ (15)

$\dot{x}_3^i = \dot{x}^{i-1} + \varepsilon \cdot \ddot{x}_3^i$ (16)

$\Theta_3^i = \Theta^{i-1} + \varepsilon \cdot \dot{\Theta}_2^i$ (17)

$\dot{\Theta}_3^i = \dot{\Theta}^{i-1} + \varepsilon \cdot \ddot{\Theta}_3^i$ (18)

(19)

$(\ddot{x}^i, \ddot{\Theta}^i) = 2nd\_deriv(x_3^i, \dot{x}_3^i, \Theta_3^i, \dot{\Theta}_3^i, F^t)$ (20)

$x^i = x^{i-1} + \frac{\varepsilon}{6} \cdot (\dot{x}^{i-1} + 2 \cdot (\dot{x}_1^i + \dot{x}_2^i + \dot{x}_3^i))$ (21)

$\dot{x}^i = \dot{x}^{i-1} + \frac{\varepsilon}{6} \cdot (\ddot{x}_1^i + 2 \cdot (\ddot{x}_2^i + \ddot{x}_3^i + \ddot{x}^i))$ (22)

$\Theta^i = \Theta^{i-1} + \frac{\varepsilon}{6} \cdot (\dot{\Theta}^{i-1} + 2 \cdot (\dot{\Theta}_1^i + \dot{\Theta}_2^i + \dot{\Theta}_3^i))$ (23)

$\dot{\Theta}^i = \dot{\Theta}^{i-1} + \frac{\varepsilon}{6} \cdot (\ddot{\Theta}_1^i + 2 \cdot (\ddot{\Theta}_2^i + \ddot{\Theta}_3^i + \ddot{\Theta}^i))$ (24)

end for (25)

return $(x^{10}, \dot{x}^{10}, \Theta^{10}, \dot{\Theta}^{10})$ (26)

end

Furthermore, if we look closer to the problem, then it is not needed to expose the whole internal structure of the calculations of the Cart Pole System and the neural network as constraints to the solver. Instead the solver should concentrate on these five important variables per unrolling depth: the position x_t and velocity $\dot{x}_t$ of the cart, the angle Θ_t and the velocity $\dot{\Theta}_t$ of the pole and the action F_t determined by the neural network. Everything else is an intermediate result – used to calculate one of these five values. Therefore, we decided to implement dedicated deduction functions for the neural network and the Cart Pole System. This also eliminates the problem of discarded bounds and a badly ordered implication queue.

We have added five new functions to the iSAT3 input language: `nncart_pole_angle`, `nncart_pole_v`, `nncart_cart_pos`, `nncart_cart_v`, and `nncart_action`. The first four provide the computations needed for the next state of the Cart Pole System while the last function returns the action computed by the neural network.

We expect that many other applications can benefit from such solver extensions: They can be applied whenever a complicated function has to be expressed as a (large) set of simpler constraints and auxiliary variables which are not referenced outside the function computation.

5. Experimental Results

We have run our experiments on a computer with 4 Quad-Core AMD Opteron^{TM} 8356 processors, running at 2.3 GHz with 64 GB DDR2 RAM. We have set a time limit of 900 seconds and a memory limit of 8000 MB. Instances of the *simple_property*-series represent verification of a simple property, which asks for the states reachable from a given input range and mainly involves deductions from the input nodes to the output nodes, whereas instances of the *complex_property*-series verify a more advanced property of the system at hand, which checks reachability of a certain set of states and requires many deductions from output values to input values. The number appended represents to which depth the solver has to advance in order to verify the given property. We encoded the instances in two ways: (1) as described in Section 3 as a large number of constraints and (2) exploiting the special deduction routines presented in Section 4.

Table 1 illustrates that the special deduction routines boost the performance of the solver on this kind of instances by several orders of magnitude – while the plain translation can only handle up to 3 unrollings, the special deduction routines perform well even for hundreds of steps. Furthermore, when the special deduction functions are used, the memory footprint is reduced significantly – the number of variables indicates this. Every simple bound is represented as a literal in the solver and therefore increases the number of variables. The more real-valued variables are declared in the instance, the more simple bounds will be created during solving. Since the translated instances use hundreds of constraints and declare just as many additional real-valued variables, it is not surprising that they exceed the memory limit.

As mentioned in Section 2, we do outward rounding for every interval calculation to get a safe enclosure of all possible solutions. Even if intervals were tight at the beginning, they could "blow up" if a large number of interval operations is applied. This is even more the case with large unrolling depths – in particular when the exponential function is used as it is done for the neural network. Thus, the precision of a standard 64 bit double could be not enough to get reasonable results. Therefore, we exploit the ability of iSAT3 to use MPFR arithmetic with a user-specified mantissa width instead. The last column of Table 1 indicates which bitwidth was used to solve the instance (a hyphen stands for standard double precision).

6. Conclusion

We have considered the problem of safety verification for controllers obtained by machine learning techniques. Such controllers are typically represented using an artificial neural network. To ensure its safety, we applied bounded model checking using the solver iSAT3, which supports the required non-linear arithmetic.

We observed that even the smallest BMC instances could hardly be solved. One reason is the strong non-linearity and non-invertability of the neural network: It contains nested exponential functions, which makes in particular deducing input values from given output values very hard.

A further reason is the discretization of the environment and the decomposition of the constraints into triplets which introduces a large number of auxiliary variables and constraints, which prevent

ISBN 978-3-00-048889-4

Table 1: Comparing the approach presented in Section 3 with the approach from Section 4

Instance	Approach Section 3		Approach Section 4		Bit-Width
	Time	# Variables	Time	# Variables	
simple_property_01.hys	2.07	46907	0.34	145	–
simple_property_02.hys	95.10	116497	0.36	253	–
simple_property_03.hys	660.12	187804	0.40	361	–
simple_property_04.hys	timeout	258198	0.43	469	–
simple_property_05.hys	timeout	311794	0.47	577	–
simple_property_10.hys	memout (136.18)	16739789	0.70	1417	64
simple_property_50.hys	memout (129.33)	16776270	3.01	6937	128
simple_property_100.hys	memout (130.68)	16776270	5.94	13837	128
simple_property_500.hys	memout (291.31)	16725363	75.32	69037	1024
complex_property_01.hys	6.06	49189	0.41	192	–
complex_property_02.hys	78.04	107691	0.48	330	–
complex_property_03.hys	timeout	193361	0.98	477	–
complex_property_04.hys	timeout	258535	2.88	637	–
complex_property_05.hys	timeout	341588	3.80	775	–
complex_property_10.hys	timeout	725342	1.74	1434	–

an efficient deduction. To overcome this latter problem we have extended the solver with problem-specific deduction operators, allowing the user to directly describe the behavior of the neurons and the environment without encoding them using primitive constraints. Experiments show that this can give a speed-up of several orders of magnitude.

However, this still does not solve the problem of safety verification. In spite of the obtained speed-up only rather small unrolling depths and very basic safety properties can be handled. Therefore, we are investigating further optimizations like improved decision heuristics that are adapted to the verification task as well as alternative methods based on Lyapunov's stability theory.

References

[BCC+03] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Y. Zhu. Bounded model checking. *Advances in Computers*, 58, 2003.

[BCP08] Douglas J. Bakkum, Zenas C. Chao, and Steve M. Potter. Spatio-temporal electrical stimuli shape behavior of an embodied cortical network in a goal-directed learning task. *J. Neural Eng.*, 5:310–323, 2008.

[BG06] F. Benhamou and L. Granvilliers. Continuous and interval constraints. In *Handbook of Constraint Programming*, Foundations of Artificial Intelligence, pages 571–603. 2006.

[Bis96] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Advanced Texts in Econometrics. Oxford University Press, 1996.

[Bor14] Luca Bortolussi. Machine learning meets stochastic model checking. In *Proc. of EPEW*, September 2014. (invited talk).

[BSA83] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. on Systems, Man, and Cybernetics*, 13(5):835–846, September/October 1983.

[But87] John C. Butcher. *The Numerical Analysis of Ordinary Differential Equations. Runge-Kutta and General Linear Methods*. (A Wiley-Interscience publication). Wiley, 1987.

[DLL62] Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, 1962.

[EKKT08] Andreas Eggers, Natalia Kalinnik, Stefan Kupferschmid, and Tino Teige. Challenges in constraint-based analysis of hybrid systems. In *Proc. of CSCLP*, volume 5655 of *LNCS*, pages 51–65. Springer, 2008.

[FHT+07] Martin Fränzle, Christian Herde, Tino Teige, S. Ratschan, and Tobias Schubert. Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. *Journal on Satisfiability, Boolean Modeling, and Computation*, 1(3-4):209–236, 2007.

[For02] Jeffrey Roderick Norman Forbes. *Reinforcement Learning for Autonomous Vehicles*. PhD thesis, University of California at Berkeley, Berkeley, CA, USA, 2002.

[Gal14] Galois wins NASA award for formal methods in machine learning. `http://galois.com/blog/2010/12/galois-wins-nasa-award-for-formal-methods-in-machine-learning/`, November 2014.

[HEFT08] Christian Herde, Andreas Eggers, Martin Fränzle, and Tino Teige. Analysis of hybrid systems using HySAT. In *Proc. of ICONS*, pages 196–201. IEEE CS, 2008.

[Her11] Christian Herde. *Efficient solving of large arithmetic constraint systems with complex Boolean structure: proof engines for the analysis of hybrid discrete-continuous systems*. PhD thesis, Carl-von-Ossietzky-Universität Oldenburg, 2011.

[KCC13] Petar Kormushev, Sylvain Calinon, and Darwin G. Caldwell. Reinforcement learning in robotics: Applications and real-world challenges. *Robotics*, 2(3):122–148, 2013.

[KWB+14] Sreedhar Saseendran Kumar, Jan Wülfing, Joschka Boedecker, Ralf Wimmer, Martin Riedmiller, Bernd Becker, and Ulrich Egert. Autonomous control of network activity. In *Proc. of the 9th Int'l Meeting on Substrate-Integrated Microelectrode Arrays (MEA)*, July 2014.

[MRT12] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. MIT Press, 2012.

[PMG+14] Eric A. Pohlmeyer, Babak Mahmoudi, Shijia Geng, Noeline W. Prins, and Justin C. Sanchez. Using reinforcement learning to provide stable brain-machine interface control despite neural input reorganization. *PLoS ONE*, 9(1), January 2014.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(9):533–536, October 1986.

[SB98] Richard Sutton and Andrew Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[SB14] Karsten Scheibler and Bernd Becker. Implication graph compression inside the SMT solver iSAT3. In *Proc. of MBMV*, pages 25–36. Cuvillier, 2014.

[SKB13] Karsten Scheibler, Stefan Kupferschmid, and Bernd Becker. Recent improvements in the SMT solver iSAT. In *Proc. of MBMV*, pages 231–241. Institut für Angewandte Mikroelektronik und Datentechnik, Fakultät für Informatik und Elektrotechnik, Universität Rostock, 2013.

[SS96] João P. Marques Silva and Karem A. Sakallah. GRASP – a new search algorithm for satisfiability. In *Proc. of ICCAD*, pages 220–227, 1996.

[Tse68] Grigori S. Tseitin. On the complexity of derivations in propositional calculus. In A. Slisenko, editor, *Studies in Constructive Mathematics and Mathematical Logics*. 1968.

SpecScribe – ein pragmatisch einsetzbares Werkzeug zum Anforderungsmanagement

Chris Drechsler, Matthias Sauppe, Christian Pätz und Ulrich Heinkel
Fakultät für Elektrotechnik und Informationstechnik, Technische Universität Chemnitz
Chemnitz, Deutschland
{chd,saum,cpae,uheinkel}@etit.tu-chemnitz.de

Zusammenfassung

Im Bereich des Anforderungsmanagements konzentrieren sich die klassischen Werkzeuge heutzutage auf eine strukturierte Aufzeichnung und Verwaltung von meist funktionalen Anforderungen in textueller Form. Mit SpecScribe wurde in der Vergangenheit bereits ein Werkzeug vorgestellt, mit dem sich die informellen Anforderungen noch während der Erfassungsphase in eine formalere Notation bis hin zur maschinenlesbaren Spezifikation teilweise automatisiert überführen lassen. Das vorliegende Papier beschreibt die Erweiterung des Werkzeugs für ein Verfassen und Verwalten der Anforderungen im Team. Darüber hinaus werden der Entwurfsprozess und die zu erstellende Anforderungsspezifikation mit graphischen Darstellungen visualisiert, sodass sich systematische Fehler auch ohne konkrete Kenntnisse des einzelnen Anforderungselementes leichter erkennen lassen. Im Rahmen der Evaluation wird anhand einer realen Industriespezifikation Sinn und Wert des Ansatzes demonstriert.

1. Einleitung

Die Erfassung und Verwaltung von Anforderungen ist der erste Schritt eines Entwurfsprozesses entsprechend des V-Modells. Im Vergleich zu anderen Entwurfsschritten ist das Anforderungsmanagement vergleichsweise wenig formalisiert und die durchaus vorhandenen elektronischen Werkzeuge weniger in die für andere Entwurfsschritte vorhandenen Werkzeugketten integriert.

Einer der Gründe dafür ist die Notwendigkeit, Anforderungen in einem Team mit sehr heterogenen Rollen zu erfassen und zu verwalten. Diese Heterogenität, die von System-Ingenieuren über Marketing- und Vertriebsmitarbeitern bis hin zu Juristen reicht, führt zu schwierigen Abstimmungsprozessen innerhalb der Teams sowie vielen Irritationen und Missverständnissen, die letztlich zu einer Vielzahl von Fehlern und Widersprüchen in der zu erstellenden Anforderungsdokumentation führt. Die wenig bis gar nicht formalisierten Anforderungen müssen danach in der Regel manuell in andere Werkzeuge des dann deutlich formaleren und damit automatisierbaren Prozesses überführt werden. Auch der Rückweg, das Zurückschreiben von Test- und Validierungsergebnissen, birgt aufgrund der Inkompatibilität der verwendeten Werkzeugwelten ein großes Fehlerpotential.

Heutzutage konzentrieren sich die klassischen Werkzeuge im Bereich des Anforderungsmanagements auf eine strukturierte Aufzeichnung und Verwaltung von meist funktionalen Anforderungen

ISBN 978-3-00-048889-4

in textueller Form. Weitere wesentliche Vorteile von diesen liegen sowohl in der Verknüpfung von Anforderungen, um die Rückverfolgbarkeit (eng. Traceability) zu gewährleisten, als auch in der Versionierung der Anforderungen, sodass Änderungen nachvollziehbar sind. Im Gegensatz dazu werden mithilfe der Werkzeuge für den Systementwurf sowie der Werkzeuge für die Implementierung dessen die o. g. Anforderungen meist manuell in eine formalisierte Form gebracht.

Eine Verknüpfung der verschiedenen Werkzeuge miteinander und damit eine Abbildung der Entwurfsphasen (beispielsweise des V-Modells) innerhalb eines Werkzeuges ist momentan nicht Stand der Technik. Die Weiterentwicklung in diese Richtung ist lohnenswert und bietet mehrere Vorteile: Einerseits ließen sich damit Inkonsistenzen zwischen benachbarten Entwurfsphasen als auch über mehrere Entwurfsphasen hinweg leichter entdecken. Es könnte etwa nachvollzogen werden, ob eine Anforderung auch tatsächlich im Systementwurf berücksichtigt, in der Implementierungsphase umgesetzt und in den anschließenden Testphasen daraufhin auch getestet wurde. Andererseits kann damit eine weitere Unterstützung für den Lebenszyklus des Systems insbesondere hinsichtlich des Änderungsmanagements wie folgt geleistet werden: Üblicherweise zieht die Änderung einer Anforderung auch Änderungen an davon abhängigen Anforderungen nach sich. Wurden alle Abhängigkeiten mit den klassischen Werkzeugen des Anforderungsmanagements sorgfältig definiert, so sind von der Änderung betroffene Anforderungen unmittelbar identifizierbar. Findet darüber hinaus eine Verknüpfung der Anforderungen mit den entsprechenden Teilen in den Phasen von Systementwurf, Implementierung und Test statt, so lassen sich ebenfalls auch dort genau jene Bereiche identifizieren, an denen sich die o.g. Änderungen auswirken und entsprechende Modifikationen (mitunter automatisiert) durchzuführen sind.

Die Forschungs- und Entwicklungsarbeiten am Lehrstuhl für Schaltkreis- und Systementwurf der TU Chemnitz mit dem Werkzeug SpecScribe konzentrierten sich in den vergangenen Jahren auf Verfahren, die informellen Anforderungen aus der Erfassungsphase noch während dieser Erfassungsphase – sprich, noch innerhalb des dafür verwendeten Werkzeuges – in eine formalere Notation zu überführen, und zwar genau nur dort, wo es aufgrund der Natur der Anforderungen und der Notwendigkeit und der Art des Tests auch sinnvoll und zweckdienlich ist. Das Ergebnis dieser Tools ist eine automatisierte softwaregestützte Weiterverarbeitung der Anforderungen in Testspezifikationen und/oder in formaleren Beschreibungssprachen wie SystemC oder Verilog.

Das vorliegende Papier beschreibt die nunmehr notwendigen Schritte, diese Prozesse und Werkzeuge zur Teilautomatisierung mit den Notwendigkeiten an eine Teamarbeit bei der Erfassung der Anforderungen zu erweitern. Zum einen wurden dazu Rollenmodelle von Review-Unterstützungen in SpecScribe integriert. Zum anderen werden graphische Darstellungen angeboten, um den Entwurfsprozess in seiner Dynamik und das Ergebnis – die Anforderungsspezifikation – in seiner Komplexität anschaulich zu machen und systematische Fehler auch ohne konkrete Kenntnisse des einzelnen Anforderungselementes zu erkennen.

Im Rahmen der Evaluation wird anhand einer konkreten Spezifikation, die von einem Industriepartner ohne entsprechende Werkzeugunterstützung erstellt wurde, Sinn und Wert des Ansatzes demonstriert. Die entsprechende Spezifikation wurde innerhalb des Werkzeuges SpecScribe erneut erfasst und entsprechend annotiert. Dadurch wurden nur durch den Werkzeugeinsatz und ohne konkretes Fachwissen in Bezug auf das spezifizierte Objekt Inkonsistenzen und fehlende Angaben erkannt und konnten behoben werden.

Insgesamt kann die Qualität von Spezifikation und Anforderungserfassung durch die in diesem Papier vorgestellte Konsolidierung von Anforderungsmanagement und formaler Systemspezifikation in einem Tool deutlich verbessert werden. Aufgrund rollenübergreifender Traceability können

ISBN 978-3-00-048889-4

Inkonsistenzen und Fehler nicht nur früher erkannt, sondern ebenso entlang des V-Modells zur nächsthöheren oder -niedrigeren Abstraktionsebene verfolgt werden, was letztendlich in geringeren Entwicklungskosten und einem effizienteren Workflow resultiert.

Das vorliegende Papier ist wie folgt gegliedert: In Kapitel 2 wird ein Überblick über die in diesem Bereich vorherrschende Literatur erarbeitet. Darüber hinaus wird der aktuelle Stand des bisher entwickelten Werkzeuges SpecScribe dargestellt. Kapitel 3 gibt Aufschluss über die neu entwickelten Fähigkeiten in SpecScribe. Die Evaluierung dieser erfolgt in Kapitel 4 an einer konkreten Spezifikation, welche von einem Industriepartner ohne Werkzeugunterstützung erstellt wurde. Kapitel 5 fasst die Ergebnisse aus diesem Papier zusammen und gibt einen Ausblick auf weiterführende Forschungsarbeiten.

2. Stand der Technik

Im Umfeld des Fachgebietes *Requirement Engineering / Requirements Management* (RE/RM) existieren eine Vielzahl von Werkzeugen, die mit ihren teils unterschiedlichen Fähigkeiten den Prozess der Systemanforderungsanalyse unterstützen. Neben der essentiellen Erhebung von Anforderungen identifizieren Carrillo de Gea et al. in [CNA+11] (u. a. auf Basis von ISO/IEC TR 24766) weitere Kategorien von Fähigkeiten, welche von den jeweiligen Tools bereitgestellt werden. Diese betreffen etwa die Analyse der Anforderungen, die Unterstützung bei der Anfertigung von Spezifikationen, die Erstellung von Anforderungsmodellen, die Verifikation und Validierung von Anforderungen, die Verwaltung von Anforderungen sowie die Rückverfolgbarkeit von Anforderungen. In [CNA+11] und [CdGNFA+12] zeigen Carrillo de Gea et al. an die 100 verschiedene Werkzeuge auf, die im Umfeld des Fachgebietes *Requirements Engineering* existieren und Unterstützung bieten.

Bei dem Großteil der im Einsatz befindlichen Werkzeuge werden die Anforderungen in textueller Form erfasst und weiterverarbeitet. Meist werden die Anforderungen dabei von einem Team gesammelt, deren heterogene Rollen von System-Ingenieuren über Marketing- und Vertriebsmitarbeiter bis hin zu Juristen reichen. Am Ende der Systemanforderungsanalyse wird ein Dokument (üblicherweise ein Pflichtenheft) in natürlicher Sprache erstellt, welches für alle beteiligten Vertragspartner als rechtsverbindliche Grundlage dient und für alle verständlich bzw. leicht prüfbar sein muss.

Textuell verfasste Anforderungen weisen jedoch einige Nachteile auf. Die verwendete natürliche Sprache kann zu Mehrdeutigkeiten führen bzw. von den Beteiligten unterschiedlich interpretiert werden. Da natürliche Sprache nicht formal ist, müssen die auf diese Weise erstellten Anforderungen in der Regel manuell in andere Werkzeuge des dann deutlich formaleren und damit automatisierbaren Prozesses überführt werden. Das kostet Zeit und ist fehleranfällig.

Für die Spezifikation, welche bereits von Beginn der Systemanforderungsanalyse-Phase die Anforderungen formal (im Gegensatz zu natürlicher und freier Sprache) erfasst, existieren in der Literatur verschiedene Klassen von Ansätzen. Hummel et al. geben in [HT09] bereits einen guten Überblick diesbezüglich, der nachfolgend skizziert werden soll.

Linguistische Ansätze Eine Klasse bilden die linguistischen Verfahren, bei denen die Anforderungen in einer semi-formalen Weise weiterhin textuell erfasst werden. Dabei geben sie Regeln bzgl. der Wortreihenfolge und -wahl sowie des Aufbaus der formulierten Sätze vor. Holtmann et al.

ISBN 978-3-00-048889-4

beschreiben in [Hol10] und [HMvD11] ein derartiges Verfahren, bei dem sie die natürliche Sprache beim Erfassen der Anforderungen mithilfe von vorgegebenen Satzmustern einschränken und nur noch bestimmte Formulierungen zulassen. Als Basis für die etwas mehr als 100 vorgegebenen Satzmuster in [Hol10] nutzen Holtmann et al. eine Analyse von deutschsprachigen Pflichtenheften von eingebetteten Systemen eines Zulieferers der Automobilbranche. Mit dieser Methodik verfasste Anforderungen weisen laut Holtmann et al. keine sprachlichen Mehrdeutigkeiten mehr auf und lassen sich anschließend automatisiert weiterverarbeiten. Holtmann et al. sehen die Vorteile in automatisierten Qualitätsanalysen sowie einer schnelleren und zuverlässigen Anbindung an die modellbasierte Entwicklung. Aufgrund dem Ursprung der Satzmuster ist dieser spezielle Ansatz derzeit jedoch auf die Anwendung im Automobilbereich zur Anforderungsanalyse von eingebetteten Systemen beschränkt. Allgemein spielen die linguistischen Ansätze für die formale Spezifikation und die Anwendung von formalen Methoden eine untergeordnete Rolle.

Spezifikationstechniken Insbesondere für reaktive Systeme, welche mit ihrer Umwelt über Eingabesignale (Sensoren) und Ausgabesignale (Aktuatoren) interagieren, bieten sich Spezifikationstechniken an. Diese zielen darauf ab, die Spezifikation leichter verständlich bzw. überschaubarer zu gestalten und einfacher zu erstellen, indem bereits Fachexperten die Anforderungen in einer sehr technischen und an der Umsetzung orientierten Form verfassen. Es existieren zwei Arten von Spezifikationstechniken: a) Auf der einen Seite diejenigen Techniken, welche mithilfe von Zuständen das Systemverhalten beschreiben. Dabei kommen meist Automaten zum Einsatz, bei denen die Ausgabesignale von den Eingabesignalen sowie dem aktuellen Zustand abhängen (vgl. [Har87], [VdB94]). Hierbei wird versucht, den inneren Aufbau eines Systems ähnlich einer *white box* zu beschreiben. b) Im Gegensatz dazu kann das Systemverhalten auch von außen betrachtet als *black box* beschrieben werden. Hierbei dienen die zeitlichen Abfolgen bzw. Sequenzen (eng. *streams*) von Ein- und Ausgabesignalen als Grundlage für die Systembeschreibung (vgl. [BS01]), welche bspw. mithilfe von mathematischen Logikgleichungen erfasst werden können.

Einordnung SpecScribe und bisheriger Stand Der Bedarf an Werkzeugen, welche Anforderungsmanagement und formale sowie nichtformale Systemspezifikation vereinen, ist seit längerer Zeit vorhanden. Beispielsweise wurde die Thematik ausführbarer, formaler Systemspezifikationen bereits 2007 als Handlungsfeld auf der International Technology Roadmap for Semiconductors (ITRS) definiert. Insbesondere wurde erkannt, dass Systemspezifikationen hinsichtlich Konsistenz und Vollständigkeit prüfbar sein müssen, was bei wachsender Spezifikationsgröße manuell kaum beherrschbar ist. Um den Entwicklungsprozess nachverfolgen zu können, wurde für die Verifikationsdomäne (Simulation, formale Verifikation) außerdem gezeigt, dass Möglichkeiten zur Rückführung von Verifikationsergebnissen in Spezifikationswerkzeuge geschaffen werden sollten.

Das an der TU Chemnitz entworfene Werkzeug SpecScribe wurde erstmalig 2008 vorgestellt [PML+08] und bietet die Möglichkeit, Spezifikation und konkrete Implementierung zu koppeln. Die Spezifikation ist dabei auf mehreren Abstraktionsebenen möglich. Zunächst kann eine Spezifikation, analog zu RM-Tools, hierarchisch auf rein textueller Ebene erfasst werden. Die so modellierten Anforderungen enthalten verschiedene Attribute, die für die Anforderungsverwaltung nützlich sind, etwa Autor, Zeitpunkt der letzten Änderung und Fortschritt von Implementierung und Verifikation. Außerdem können neben der reinen Anforderungshierarchie sog. *Traces* definiert werden, welche Anforderungen in Beziehungen setzen.

ISBN 978-3-00-048889-4

Neben der rein textuellen Anforderungserfassung ist es ebenso möglich, für bestimmte Anforderungsdomänen formale Anforderungen zu erfassen. Das kann in der Verifikationsdomäne beispielsweise die formale Definition von Bedingungen sein, die von einem analogen Signal erfüllt werden müssen (z. B. in Form von Differentialgleichungen). Ebenso ist es möglich, eine formal spezifizierte Komponente bzw. Implementierung mit einer Anforderung zu verbinden, beispielsweise um die formale Definition eines hybriden Zustandsautomaten an eine Anforderung zu binden.

Für jede Anforderung, die entweder selbst formal ist oder für die mindestens eine formale Implementierung vorliegt, ist die maschinelle Weiterverarbeitung gewährleistet. In SpecScribe ist beispielsweise der Export von Struktur- und verschiedenartigen Verhaltensbeschreibungen nach VHDL und SystemC möglich. Formale Verifikationsanforderungen können nach SAL exportiert werden. Die Verbindung von formalen/nichtformalen Anforderungen mit konkreten Implementierungen ist zentraler Bestandteil von SpecScribe.

Allerdings sind einige wünschenswerte Eigenschaften von RM-Tools in SpecScribe nicht umgesetzt, so fehlt etwa eine grundlegende Unterstützung der Zusammenarbeit im Team (etwa E-Mail-Benachrichtigungen), auch wird kein geeignetes Datenformat unterstützt, welches die Wiederverwendung von Anforderungen oder Komponenten aus anderen Spezifikationen unterstützt. Aus diesen Gründen wurde ein Redesign des Werkzeugs mit verändertem Fokus vorgenommen.

3. SpecScribe

Aufgrund der Notwendigkeit, Anforderungen in einem Team zu erfassen und zu verwalten, sowie für die Evaluierung neuer Ansätze wurde SpecScribe überarbeitet und als webbasierte Anwendung mit Client-Server-Struktur neu implementiert. Weitere Vorteile liegen hierbei in der Plattformunabhängigkeit sowie in einer zentralen Datenbasis. In diesem Zusammenhang erfolgte die Erweiterung des Werkzeuges für ein Verfassen und Verwalten der Anforderungen im Team. Dazu wurde SpecScribe um die Fähigkeiten einer Nutzer- und Rollenverwaltung ergänzt.

Nach der Anmeldung des Nutzers werden neue Anforderungen nunmehr automatisch mit seinem Namen als Eigentümer (eng. owner) angelegt. Dieser kann bei Bedarf einen weiteren Nutzer als Editor festlegen, der die Anforderungen bearbeiten kann. Für alle anderen Nutzer verbleibt sie nur lesbar. Um in der Vielzahl von Anforderungen eine verbesserte Suche zu erreichen, besteht die Möglichkeit, Anforderungen mit Tags zu versehen.

Insbesondere textuelle Anforderungen werden innerhalb eines sog. *rich text editors* direkt im Browser editiert, der ähnlich eines modernen Textverarbeitungsprogramms umfangreiche Textformatierungen erlaubt. Sobald die Anforderung aus Sicht des Eigentümers vollständig erfasst wurde, kann er den Status dieser von *Edit* auf *Commit* ändern. Die verschiedenen Status-Codes zeigen den aktuellen Bearbeitungsstand bzw. den als nächstes erforderlichen Bearbeitungsschritt an. Neben den beiden genannten Status existiert mit *Need Review* ein weiterer, welcher eine erneute Überprüfung der Anforderung anzeigt.

SpecScribe ermöglicht es, Anforderungen über sog. Assoziationen miteinander zu verknüpfen, um Abhängigkeiten unter den Anforderungen zu markieren. Während in der Literatur dafür häufig die Begriffe *Abhängigkeiten* oder *Links* verwendet werden, sehen wir diesen Begriff etwas weiter gefasst. Mithilfe von Assoziationen lassen sich in unserem Modell nicht nur die Abhängigkeiten zwischen den Anforderungen markieren, sondern ebenfalls auch detailliert (bspw. mithilfe von Berechnungs-/Assoziationsvorschriften) beschreiben. Das hilft einerseits, die konkrete Abhängigkeit inhaltlich besser zu verstehen bzw. (von Außenstehenden) nachzuvollziehen, und andererseits,

ISBN 978-3-00-048889-4

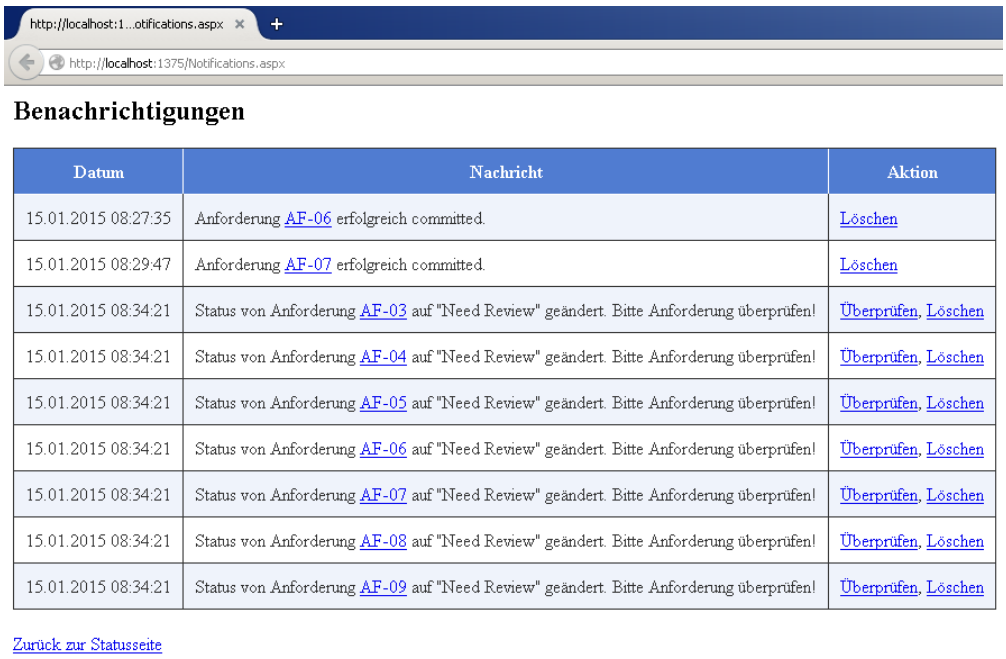


Abbildung 1: Eintreffende Benachrichtigungen bzgl. der Anforderungen eines Eigentümers

bei Änderungen von Anforderungen auch abhängige Anforderungen entsprechend der Abhängigkeitsbeschreibung (mitunter automatisiert) zu modifizieren.

Diese Assoziationen können sowohl unidirektional (Anforderung A beeinflusst Anforderung B) als auch bidirektional (Anforderungen A und B beeinflussen sich gegenseitig) gesetzt werden. Diese Fähigkeit wird innerhalb von SpecScribe insbesondere für den Review-Prozess bzw. für das Änderungsmanagement verwendet. Ändert sich bspw. der Status einer Anforderung A auf *Need Review* (siehe oben), so ändert sich ebenfalls bei allen von Anforderung A assoziierten Anforderungen der Status auf *Need Review*. Dabei werden die jeweiligen Eigentümer der betreffenden Anforderungen über entsprechende Mitteilungen (sog. *Notifications*) benachrichtigt, sodass sie ihrerseits die Auswirkungen auf die von ihnen verfassten Anforderungen identifizieren und bei Bedarf Modifikationen durchführen können (vgl. Abbildung 1).

Darüber hinaus lassen sich in SpecScribe die o. g. Assoziationen aller Anforderungen visualisieren, um einerseits den Entwurfsprozess in seiner Dynamik und das Ergebnis – die Anforderungsspezifikation – in seiner Komplexität anschaulich zu machen und andererseits, um systematische Fehler auch ohne konkrete Kenntnisse des einzelnen Anforderungselementes zu erkennen. SpecScribe nutzt dazu intern das Tool Graphviz (vgl. [Gra]) für die Darstellung der Anforderungen und ihrer Assoziationen untereinander. Abbildung 2 veranschaulicht eine beispielhafte Spezifikation mit verschiedenen Anforderungen und deren Assoziationen.

4. Evaluation

Die Evaluierung der Neuimplementierung von SpecScribe und der in diesem Papier neu vorgestellten Fähigkeiten erfolgte post mortem an einer bestehenden Spezifikation, welche von einem Industriepartner ohne entsprechende Werkzeugunterstützung erstellt wurde. Die Spezifikation ist im Bereich von Testhardware für elektronische Schaltungen einzuordnen und umfasst ca. 100 An-

ISBN 978-3-00-048889-4

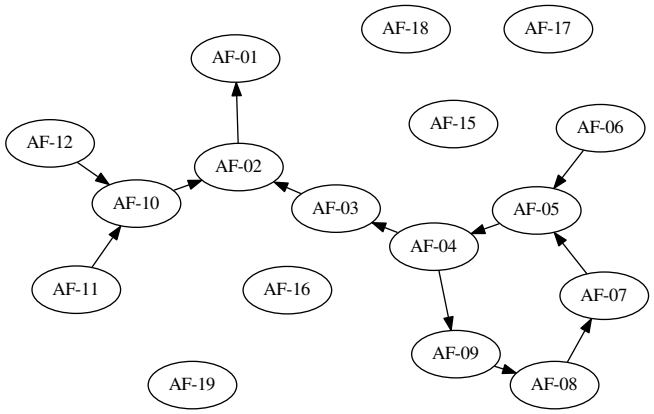


Abbildung 2: Graphische Darstellung der Assoziationen zwischen den Anforderungen

forderungen. Diese beziehen sich auf elektrische Aspekte (Strom, Spannung, Energieverbrauch usw.), thermische Vorgaben, Gesichtspunkte hinsichtlich der digitalen Signalverarbeitung sowie Forderungen bzgl. der verwendeten Betriebssystemsoftware.

Jede einzelne Anforderung aus der Spezifikation wurde zunächst manuell in SpecScribe neu angelegt und inhaltlich übertragen. Anschließend erfolgte eine Umgruppierung der Anforderungen nach thematischen Gesichtspunkten. In einem ersten Schritt geschah dies zunächst in einer grobgranularen Form, sodass eine Trennung in die bereits o. g. Bereiche für elektrische Aspekte, thermische Vorgaben, Gesichtspunkte hinsichtlich der digitalen Signalverarbeitung sowie Forderungen bzgl. der verwendeten Betriebssystemsoftware erfolgte. In einem weiteren Schritt wurden in diesen groben Bereichen feinere Unterbereiche gebildet, welche die Anforderungen bzgl. eines konkreten Spezifikationsobjektes inhaltlich zusammenfassen. Zum Schluss erfolgte das Setzen aller offensichtlichen Assoziationen zwischen den jeweiligen Anforderungen.

Insbesondere die Verfeinerung der Hierarchie im Zusammenhang mit einer übersichtlichen Darstellung durch SpecScribe hat sich als sehr positiv gestaltet, sodass bereits dadurch erste Fehler bzw. Inkonsistenzen erkennbar waren: Konkret wurden aus der Spezifikation u. a. verschiedene Spannungsdomänen und die an sie gestellten Anforderungen zu einzelnen Gruppen zusammengefasst. Auszugsweise umfassen diese Forderungen zu dem Spannungsbereich der Domäne, der einstellbaren Auflösung der Spannung, der Genauigkeit der tatsächlich eingestellten Spannung, dem Maximalstrom, dem Verhalten des Spannungspegels nach Lastspitzen und viele weitere.

Bei der Gegenüberstellung dieser Gruppen von sehr ähnlichen Spezifikationsobjekten wurden sofort Unterschiede in der Anzahl und Art der beteiligten Anforderungen sichtbar. Während einige dieser doch höchst ähnlichen Spezifikationsobjekte von Spannungsdomänen außerordentlich detailliert (mithilfe der entsprechenden Anforderungen) beschrieben waren, fehlten bei anderen Spannungsdomänen einzelne oder sogar mehrere Anforderungen. Hieran ist erkennbar, dass die Schablone von Anforderungen zur Beschreibung einer Spannungsdomäne nicht konsequent für alle in der Spezifikation enthaltenen Spannungsdomänen angewandt wurde. Allein aus dem Vergleich über die Anzahl (und im Weiteren über die Art) der enthaltenen Anforderungen kann geschlussfolgert werden, dass einige dieser sehr ähnlichen Spezifikationsobjekte unvollständig erfasst worden sind.

ISBN 978-3-00-048889-4

Eine weitere Hilfestellung lieferte SpecScribe beim Setzen, Visualisieren und Analysieren der zwischen den Anforderungen bestehenden Assoziationen. Allein der Aspekt, Assoziationen in einem initialen Schritt zu identifizieren und zu beschreiben, fördert eine vertiefte Auseinandersetzung über die jeweiligen Anforderungen und deren Beziehungen zueinander. Nachdem diese gesetzt sind, stellt SpecScribe für jede Anforderung alle direkt assoziierten Anforderungen dar. Fokussiert man ausschließlich auf genau jene Anforderungen (indem man sich bspw. die Inhalte der einzelnen Anforderungen der Reihe nach aufmerksam durchliest), lassen sich im Zusammenhang betrachtet weitere Fehler als auch Inkonsistenzen erkennen. Bezogen auf die o.g. Industriespezifikation standen bspw. einige der assoziierten Anforderungen im Widerspruch: Während eine Anforderung die Maximalstromstärke für eine Spannungsdomäne bei einer fixen Spannung A begrenzte, war in einer anderen der variabel einstellbare Spannungsbereich dieser Domäne mit einer Maximalspannung größer A definiert. Damit blieb unklar, welche Maximalstromstärke tatsächlich über den gesamten Spannungsbereich zulässig ist.

In einer weiterführenden Analyse aller betrachteten Anforderungen und deren Assoziationen mithilfe der graphischen Darstellung (vgl. Abbildung 2) von SpecScribe ließen sich diejenigen kritischen Anforderungen bestimmen, von denen sehr viele Assoziationen ausgehen. Treten bei einer/einigen davon im Nachhinein inhaltliche Änderungen (z. B. im Review-Prozess) auf, resultiert aus der Vielzahl der assoziierten Anforderungen ein entsprechend großer Änderungsaufwand. Folglich lohnt es sich genau bei diesen kritischen Anforderungen, vor einem Commit-Vorgang auf einen vollständigen und konsistenten Inhalt zu achten.

In der o.g. Industriespezifikation ließen sich auf diese Weise einige wenige kritische Anforderungen identifizieren. Die längste Assoziationskette hatte eine Tiefe von zwei, d.h. von einer Anforderung ist eine Ebene von Anforderungen direkt abhängig und von dieser Ebene eine weitere. Leider konnten wir bei der Übertragung der Industriespezifikation in SpecScribe nur die offensichtlichen Assoziationen setzen, da wir einerseits nicht über ein dafür erforderliches Fachwissen verfügen und andererseits sehr viel verdecktes Wissen vorausgesetzt wird. Mithilfe der bei der Erstellung beteiligten Personen der Spezifikation wären mitunter weiterführende Erkenntnisse aus der Analyse der Assoziationen möglich gewesen.

Aufgrund der Post-mortem-Analyse der hier verwendeten Industriespezifikation, ließen sich die neu implementierten Fähigkeiten von SpecScribe für die Verfassung und Verwaltung von Anforderungen im Team leider noch nicht evaluieren.

5. Zusammenfassung

Dieser Beitrag stellte die Überarbeitung des Werkzeugs SpecScribe vor, welches für die formale Spezifikation und die Erforschung neuer Ansätze auf diesem Gebiet entworfen wurde. Konkret erfolgte die Neuimplementierung des Werkzeugs als webbasierte Anwendung mit Client-Server-Struktur und die Ergänzung um Fähigkeiten für das Verfassen und Verwalten von Anforderungen im Team. Darüber hinaus lassen sich mit den neu vorgestellten graphischen Darstellungen der Entwurfsprozess sowie die Anforderungsspezifikation anschaulicher darstellen. Hiermit bietet SpecScribe für den Nutzer eine Hilfestellung an, systematische Fehler auch ohne konkrete Kenntnisse des einzelnen Anforderungselementes zu erkennen.

Die Evaluierung erfolgte post mortem an einer bestehenden Spezifikation, welche von einem Industriepartner ohne entsprechende Werkzeugunterstützung erstellt wurde. Die darin enthaltenen Anforderungen wurden innerhalb des Werkzeuges SpecScribe erneut erfasst und entsprechend

ISBN 978-3-00-048889-4

annotiert. Unter Verwendung der hier neu vorgestellten Fähigkeiten von SpecScribe und ohne konkretes Fachwissen in Bezug auf das spezifizierte Objekt ließen sich Inkonsistenzen und fehlende Angaben erkennen und beheben.

Eine Evaluierung der neu vorgestellten Fähigkeiten für das Verfassen und Verwalten von Anforderungen im Team konnte mithilfe der Post-mortem-Analyse leider nicht durchgeführt werden. Hierfür streben wir den Einsatz von SpecScribe bei der Erstellung einer noch nicht existierenden Spezifikation im industriellen Umfeld an.

Literatur

[BS01] Broy, Manfred und Ketil Stølen: *Specification and development of interactive systems: focus on streams, interfaces, and refinement*. springer, 2001.

[CdGNFA+12] Gea, Juan M. Carrillo de, Joaquín Nicolás, José L Fernández Alemán, Ambrosio Toval, Christof Ebert und Aurora Vizcaíno: *Requirements engineering tools: Capabilities, survey and assessment*. Information and Software Technology, 54(10):1142–1157, 2012.

[CNA+11] Carrillo de Gea, Juan M., Joaquín Nicolás, José Luis Fernández Alemán, Ambrosio Toval, Christof Ebert und Aurora Vizcaíno: *Requirements engineering tools*. Software, IEEE, 28(4):86–91, 2011.

[Gra] *Homepage Graphviz*. `http://www.graphviz.org/`.

[Har87] Harel, David: *Statecharts: A visual formalism for complex systems*. Science of computer programming, 8(3):231–274, 1987.

[HMvD11] Holtmann, Jörg, Jan Meyer und Markus von Detten: *Automatic validation and correction of formalized, textual requirements*. In: *Fourth International Conference on Software Testing, Verification and Validation Workshops*, Seiten 486–495. IEEE, 2011.

[Hol10] Holtmann, Jörg: *Mit Satzmustern von textuellen Anforderungen zu Modellen*. OBJEKT-spektrum, RE/2010 (Online Themenspecial Requirements Engineering), 2010.

[HT09] Hummel, Benjamin und Judith Thyssen: *Behavioral Specification of Reactive Systems Using Stream-Based I/O Tables*. In: *Seventh IEEE International Conference on Software Engineering and Formal Methods*, Seiten 137–146. IEEE, 2009.

[PML+08] Pross, Uwe, Erik Markert, Jan Langer, Andreas Richter, Chris Drechsler und Ulrich Heinkel: *A platform for requirement based formal specification*. In: *Forum on Specification, Verification and Design Languages*, Seiten 237–238. IEEE, 2008.

[VdB94] Beeck, Michael Von der: *A comparison of statecharts variants*. In: *Formal techniques in real-time and fault-tolerant systems*, Seiten 128–148. Springer, 1994.

ISBN 978-3-00-048889-4

A Counterexample-Guided Approach to Symbolic Simulation of Hybrid Systems

Xian Li, Klaus Schneider
TU Kaiserslautern
xian.li@cs.uni-kl.de
klaus.schneider@cs.uni-kl.de

Abstract

In this paper, we propose a symbolic simulation algorithm for hybrid systems that are specified by parameterized formal models using standard data types (reals, integers and booleans) and non-linear dynamics. To support the proposed algorithm, we develop a prototypical constraint solver for non-linear arithmetic theories over integers and reals with quantifiers. Given a system's symbolic representation, specifications, and finitely many concrete input values, the algorithm computes ranges of the input parameters by extending each concrete value to a range constraint until some parameter valuation violates the specifications. In this way, no concrete value that belongs to the generated ranges need to be considered for subsequent simulations. The algorithm has been prototypically implemented, and its feasibility is proved by a successful experimental evaluation using parameterized hybrid programs.

1. Introduction

Simulation is a commonly used method for validating behaviors of complex dynamical systems. The integration of simulation and symbolic analysis yields the so-called symbolic simulation that improves the simulation coverage of the system model [Alu11]. Symbolic simulation of hybrid systems evaluates a certain dynamic behavior of the system by using parameterized system models.

There are different kinds of formal models for hybrid systems that could be parameterized: Linear Hybrid Automata (LHAs) and Affine Hybrid Automata (AHAs) are special Hybrid Automata, where LHAs are restricted to linear dynamics for continuous state variables, while affine dynamics are allowed for AHAs. Hybrid programs are based on modeling languages, so that the hybrid systems could be encoded with data types and programming statements. Typical examples are KeYmaera [Pla10], HyDI [CMT11], HybridSAL [Tiw12] and *Hybrid Quartz* [Bau12]. For the symbolic simulation, only parameterized input variables are considered.

The hybrid systems discussed in this paper are specified by parameterized formal models supporting standard data types (reals, integers and booleans) and affine dynamics, like *Hybrid Quartz*, HyDI, and HybridSAL. Automated analysis of this kind of models leads to an undecidable satisfiability problem. This is the case since the underlying logic allows boolean combinations of propositional logic atoms as well as atoms from non-linear arithmetic theories over integers and reals with quantifiers. Even though numerous approaches from different backgrounds have been

ISBN 978-3-00-048889-4

proposed in literature, each of them could only solve a subclass of the underlying undecidable satisfiability problem.

Therefore, we analyze various tools, i.e. constraint solvers, and develop a new prototypical constraint solver by integrating the external tool Bonmin [BBC+08] into the BDD package implemented in our *Averest* system (`www.averest.org`). Based on our new solver, we propose a counterexample-guided algorithm for symbolic simulation of hybrid systems. Given a system's symbolic representation, specifications, and finitely many concrete input values, the algorithm computes ranges of the input parameters by extending each concrete value to a range constraint until some parameter valuation violates the specification. In this way, no concrete value that belongs to the generated ranges need to be considered for subsequent simulations. The algorithm has been prototypically implemented in the *Averest* system, and its feasibility is proved by a successful experimental evaluation using parameterized *Hybrid Quartz* programs.

The rest of the paper is organized as follows: Section 2 presents the syntax and semantics of the satisfiability problems that we would like to solve for symbolic simulation of hybrid systems together with a discussion of the available tools. Section 3 gives implementation details of the prototypical constraint solver. The counterexample-guided algorithm for symbolic simulation of hybrid systems will be introduced in Section 4, while Section 5 describes the experiments that we performed for this paper. The paper will be concluded in Section 6.

2. Preliminaries

First, we describe the satisfiability problem that we consider when performing symbolic simulation of hybrid systems that are specified by parameterized formal models using standard data types (reals, integers and booleans) and affine dynamics, so that the decidability of the satisfiability problem and the available tools can be discussed.

2.1. Syntax

We assume a finite set of real, integer and boolean variables $\mathcal{V} = \mathcal{V}_{\mathbb{R}} \uplus \mathcal{V}_{\mathbb{Z}} \uplus \mathcal{V}_{\mathbb{B}}$, and a finite set of real, integer and boolean input parameters $\mathcal{W} = \mathcal{W}_{\mathbb{R}} \uplus \mathcal{W}_{\mathbb{Z}} \uplus \mathcal{W}_{\mathbb{B}}$, where $\mathcal{W}$ is a subset of $\mathcal{V}$, which means $\mathcal{W}_{\mathbb{R}}$, $\mathcal{W}_{\mathbb{Z}}$, and $\mathcal{W}_{\mathbb{B}}$ are subsets of $\mathcal{V}_{\mathbb{R}}$, $\mathcal{V}_{\mathbb{Z}}$, and $\mathcal{V}_{\mathbb{B}}$, respectively. We define boolean expressions and numerical expressions, by the following grammars:

$$\begin{aligned} e_b &:= x \in \mathcal{V}_{\mathbb{B}} \mid \neg e_b \mid e_b \wedge e_b \mid e_b \vee e_b \\ e &:= x \in \mathcal{V}_{\mathbb{R}} \uplus \mathcal{V}_{\mathbb{Z}} \mid e + e \mid e - e \mid e \cdot e \mid e/e \end{aligned} \tag{1}$$

If $X_{\mathbb{B}} \subseteq \mathcal{V}_{\mathbb{B}}$ is the quantifier set, then *boolean expression with $\exists$-quantifiers* is defined as follows:

$$(e_b)_Q := (\exists X_{\mathbb{B}}).\, e_b\,. \tag{2}$$

Similarly, given $X_{\mathbb{R}} \subseteq \mathcal{V}_{\mathbb{R}}$ and $X_{\mathbb{Z}} \subseteq \mathcal{V}_{\mathbb{Z}}$ as quantifier sets, numerical expressions e and e', and operator $\odot \in \{\leq, =\}$, the following formula defines a *constraint* and a *constraint with $\exists$-quantifier*.

$$c_Q := (\exists X_{\mathbb{R}}, X_{\mathbb{Z}}).\, c \text{ where } c := e \odot e' \mid c \wedge c \tag{3}$$

For convenience of analysis and explanation, the syntax of the satisfiability problem that we consider could be defined as the following disjunctive form:

$$C_b := \bigvee_{i \in \mathbb{N}} \left((c_Q)_i \wedge ((e_b)_Q)_i \right) \tag{4}$$

ISBN 978-3-00-048889-4

Formulas without quantifiers are no longer considered. It is reasonable to do that since the constraints and boolean expressions without quantifiers could be represented by those with quantifiers.

For the rest of paper, we assume that the constraint on bound variables that appear in (2-4) are given, denoted as c_v. To have some idea on how the formulas look like, we assume that $\mathcal{V}_{\mathbb{R}} = \{r_1, r_2\}$, $\mathcal{V}_{\mathbb{Z}} = \{z_1, z_2\}$, $\mathcal{V}_{\mathbb{B}} = \{b_1, b_2\}$, and list some examples in Table 1.

Table 1: Formula Examples

Category	Formula
e_b	$E_0 : b_1 \wedge b_2$
c	$E_1 : r_1 \leq r_2 + z_1 * (r_1 - z_2)$
$(e_b)_Q$	$E_2 : (\exists \mathcal{V}_{\mathbb{B}}).\ b_1 \wedge b_2$
c_Q	$E_3 : (\exists \mathcal{V}_{\mathbb{R}}, \mathcal{V}_{\mathbb{Z}}).\ E_1$
C_b	$E_4 : E_2 \wedge E_3$ $E_5 : ((\exists \{r_1\}).\ r_1 \geq 1) \vee ((\exists \{r_1\}).\ r_1 \leq 0.9)$
c_v	$E_6 : (-2.0 \leq r_1 * r_2 \leq 10.0) \wedge (z_1 \leq z_2 \leq 5)$

2.2. Semantics

The semantics of formulas (1-4) are defined by the interpretation $[\![\cdot]\!]_v$, which evaluates the inside formula $\cdot$ to *True* or *False* using the variable valuation function v.

Definition 1 (Variable Valuation) *A variable valuation is a function $v : \mathcal{V} \to \mathbb{R} \cup \mathbb{Z} \cup \mathbb{B}$ that assigns to each variable a real, integer or boolean value. $v_{|^d_x}$ is the x-variant of the variable valuation v. v and $v_{|^d_x}$ agree on everything except possibly the value of variable x, where $v_{|^d_x}(x) = d$ for some $d \in \mathbb{R} \cup \mathbb{Z} \cup \mathbb{B}$.*

Definition 2 (Semantics) *Given variable x, numerical expressions e_1 and e_2, constraints c, boolean expressions e_b, e_{b_1}, and e_{b_2}, together with operator $\circledast \in \{+, -, \cdot, /, \leq, =\}$, to which we assign standard numerical relations of reals and integers, the semantics is defined inductively as follows:*

$$[\![x]\!]_v := v(x)$$
$$[\![e_1 \circledast e_2]\!]_v := [\![e_1]\!]_v \circledast [\![e_2]\!]_v$$
$$[\![e_{b_1} \wedge e_{b_2}]\!]_v := \begin{cases} \textit{True}, \text{ iff } [\![e_{b_1}]\!]_v = [\![e_{b_2}]\!]_v = \textit{True} \\ \textit{False}, \text{Otherwise} \end{cases}$$
$$[\![\neg e_b]\!]_v := \begin{cases} \textit{True}, \text{ iff } [\![e_b]\!]_v = \textit{False} \\ \textit{False}, \text{Otherwise} \end{cases}$$
$$[\![e_{b_1} \vee e_{b_2}]\!]_v := \begin{cases} \textit{False}, \text{ iff } [\![e_{b_1}]\!]_v = [\![e_{b_2}]\!]_v = \textit{False} \\ \textit{True}, \text{Otherwise} \end{cases}$$
$$[\![(\exists x \in X_{\mathbb{B}}).\ e_b]\!]_v := \begin{cases} \textit{True}, \text{ iff there exists a } d \in \mathbb{B} \text{ such that } [\![e_b]\!]_{v|^d_x} = \textit{True} \\ \textit{False}, \text{Otherwise} \end{cases}$$
$$[\![(\exists x \in X_{\mathbb{R}} \uplus X_{\mathbb{Z}}).\ c]\!]_v := \begin{cases} \textit{True}, \text{ iff there exists a } d \in \mathbb{R} \cup \mathbb{Z} \text{ such that } [\![c]\!]_{v|^d_x} = \textit{True} \\ \textit{False}, \text{Otherwise} \end{cases}$$

Taking formulas E_1 and E_6 in Table 1 as an example, where E_6 is the constraint on bound variables, if v_1 is a variable valuation that maps r_1, r_2, z_1 and z_2 to 0.95, 1.0, 1 and -2, respectively, then $[\![E_1]\!]_{v_1}$ evaluates to *True*.

Based on the base cases of our inductive definition, formula (4) is evaluated by $[\![\cdot]\!]_v$ as follows:

$$[\![C_b]\!]_v := \begin{cases} \textit{True}, \text{ iff exists a } i \in \mathbb{N} \text{ such that } [\![(c_Q)_i]\!]_v \wedge [\![((e_b)_Q)_i]\!]_v = \textit{True} \\ \textit{False}, \text{ Otherwise} \end{cases}$$

Considering the variable valuation v_1 in the previous example with E_5 and E_6 in Table 1, $[\![E_5]\!]_{v_1}$ evaluates to *True*, since $v_{1|_{r_1}^{1.5}}$ is a r_1-variant of v_1, with which $[\![r_1 \geq 1]\!]_{v_{1|_{r_1}^{1.5}}} = \textit{True}$ holds.

2.3. Decidability and Tools

The satisfiability problem defined by formula (4) consists of boolean combinations of propositional logic atoms and atoms of non-linear arithmetic theories over integers and reals with $\exists$-quantifiers. However, quantifier alternations are not allowed. Therefore, the satisfiability problem described by formula (4) is undecidable and strictly included in the combination of the first-order logic of the structure $(\mathbb{R}, +, \cdot, \mathbb{Z}, 0, 1, <)$ and propositional logic.

From the point of view of constraint problems [BHZ06], formula (3) is a standard form formula that describes an instance of a MINLP problem. If we get rid of the boolean variables, then the remaining subset problem defined by formula (4) can be reorganized as follows:

$$C_b{}' := \bigvee_{i \in \mathbb{N}} c_{Q_i} \tag{5}$$

Solving the satisfiability problem of the above formula amounts to check whether there exists a solution for a set of MINLP problems, where each MINLP problem corresponds to a subformula c_{Q_i}. The solution satisfies at least one MINLP problem in the set. Therefore, we need effective techniques to check the existence of this solution for the set of MINLP problems generated by $C_b{}'$.

The MINLP problem is one of the most general modeling paradigms in optimization and includes both Non-Linear Programming (NLP) and Mixed Integer Linear Programming (MILP) as subproblems. Linear constraint problems are a subset of NLP problems, while, MILP problems are a subset of MINLP problems.

When classifying the available tools and techniques for hybrid system analysis, according to the constraint problems they could solve, we have the following results:

- Linear constraint problems: HySAT [FH07] and BACH [BLWL08] are designed for linear arithmetic over reals.
- MILP problems: MathSAT5 [CGJS13], CVC4 [CVC], and Z3 [Z3] contain decision procedures for mixed integer linear arithmetic.
- NLP problems: KeYmaera and MetiTarski [Pau12] accept formulas over reals that are non-linear. Originally based on the inverse method [ACEF09], IMITATOR [And09] is a tool for efficient synthesis for Timed Automata [AD94]. In [FK11], the methodology has been extended for LHAs and AHAs. SpaceEx [FLD$^+$11] facilitates quite a lot of algorithms related to reachability and safety verification in the theory of reals. Z3 also presents partial support of non-linear arithmetic.
- MINLP problems: iSAT [EFH08] could solve large boolean combinations of non-linear arithmetic constraints involving transcendental functions.

Some algebraic computation tools developed in different application areas attracted our attention as well.

- NLP problems: QEPCAD [QEP] can produce quantifier-free equivalent formulas for Tarski formulas. Reduce [Redb] can handle integer and real arithmetic. Its package Redlog [Reda] has some quantifier elimination procedures to solve parameterized non-linear real arithmetic problems. Both of them are widely used as external decision procedures for hybrid system verification tools, like KeYmaera. TReX [ABS01], a tool for automatic analysis of automata-

based models, relies on Reduce to solve the constraint problems in the theory of non-linear arithmetic over reals.

- MINLP problems: The survey [BKL+13] introduces a range of approaches to tackle MINLP problems. Bonmin is one of the open source tools mentioned in it. Bonmin contains a combination of techniques that lends itself to be a good candidate for convex MINLP problems.

In general, tools for MINLP problems try to find a solution for a single MINLP problem each time. However, they cannot always find a solution. This might be due to the limitations of the tool itself or due to the fact that satisfiability of MINLP problems is undecidable. No solution returned does not mean no solution exists.

Solving the satisfiability problem requires to map the formula to a logical value (*True* or *False*). It is beyond the ability of the tools to solve the satisfiability problem for the class of formulas defined by formula (5). This is the case even though assigning a logical value is strongly related to whether a tool can find a solution for each individual MINLP problem or not. Taking $C = c_{Q_1} \vee c_{Q_2}$ as an example, if no solution is found for the MINLP problems that correspond to c_{Q_1} and c_{Q_2} respectively, then it is unclear what logical value $[\![C]\!]_v$ should be assigned to.

Moreover, iSAT and the algebraic computation tools are developed in different application areas. It is still unclear which tool performs best to solve the satisfiability problem that we consider. Based on the above analysis, we conclude that currently available tools could solve a certain subclass of the undecidable satisfiability problem, but they all have their own limitations. They could still be used as components in the context of other tools, in accordance with the different synthesis requirements to solve different logical formulas or mathematical problems.

3. The Constraint Solver

To bridge the gap between the existing research challenges and available techniques, we develop a prototypical constraint solver by integrating the external tool Bonmin into the BDD package implemented in our *Averest* system. The new constraint solver has been implemented in our *Averest* system as F# functions so that engineers can check the formulas in an interactive F# session.

Technically speaking, formula (4) is the standard form input formula. $(c_Q)_i$ can be processed as a MINLP problem. Based on the result returned by Bonmin for the MINLP problem, we use the BDD package to evaluate the conjunctive sub-formula $((c_Q)_i \wedge ((e_b)_Q)_i)$.

Given $\mathcal{W}_{\mathbb{R}} = \mathcal{V}_{\mathbb{R}} = \{a\}$, $\mathcal{W}_{\mathbb{Z}} = \mathcal{V}_{\mathbb{Z}} = \{n\}$, and $n \geq 0 \wedge 1.26 \leq a \leq 2.24$ as constraint on parameters, let $(\exists n, a).\ 0.5 * a * (8 * (1 - 0.5^n))^2 \leq 24.5$ be an instance of formula $(c_Q)_i$. The Bonmin input file that encodes the corresponding MINLP problem is the following:

1:	var n integer >= 0; var a >= 1.26 <= 2.24;
2:	minimize cost: n;
3:	subject to
4:	$0.5 * a * (8 * (1 - 0.5^n))^2$ <= 24.5;

Line 1 declares the data types and ranges of variables. The optimal function is given after the statement *minimize cost*, where engineers decide the minimize cost function according to their preferences. The inequality constraint is displayed in Line 4 following the statement *subject to*.

Bonmin returns a solution: $n = 0$ and $a = 1.74997$ for this example. Actually, the solution returned by Bonmin is part of a variable valuation for integer and real variables. Thus, a complete variable valuation that satisfies $((c_Q)_i \wedge ((e_b)_Q)_i)$ can be obtained together with the other part of

the variable valuation decided by $((e_b)_Q)_i$. Later in Section 4.2, the variable valuation produced by Bonmin will be used for Algorithm 1. However, as discussed in Section. 2.3, Bonmin cannot always find a solution for the MINLP problem. Thus, we extend the interpretation $[\![\cdot]\!]_v$ by the following interpretation $[\![\cdot]\!]_3$ for our constraint solver.

Definition 3 (3-Valued Evaluation) *The semantics of formulas (2-3) is redefined by the interpretation $[\![\cdot]\!]_3$, which evaluates the inside formula $\cdot$ to* True*,* False *or* Unkown*.*

$$[\![(\exists X_{\mathbb{B}}).\ e_b]\!]_3 := [\![(\exists X_{\mathbb{B}}).\ e_b]\!]_v$$

$$[\![(\exists X_{\mathbb{R}}, X_{\mathbb{Z}}).\ c]\!]_3 := \begin{cases} \textit{True}, & \text{if Bonmin could obtain a solution of the MINLP problem for } c \\ \textit{Unknown}, & \text{Otherwise} \end{cases}$$

Truth tables for 3-valued logic of conjunction and disjunction (denoted as $\wedge_3$ and $\vee_3$) are shown in Table 2, where T, F and U represent the truth value True*,* False *and* Unknown *respectively.*

Table 2: Truth Tables for 3-Valued Logic

$\wedge_3$	T	F	U
T	T	F	U
F	F	F	F
U	U	F	U

$\vee_3$	T	F	U
T	T	T	T
F	T	F	U
U	T	U	U

Therefore, formula (4) is interpreted by $[\![\cdot]\!]_3$ as below:

$$[\![C_b]\!]_3 := \begin{cases} \textit{True}, & \text{iff there exists a } i \in \mathbb{N} \text{ such that } [\![(c_Q)_i]\!]_3 \ \wedge_3 \ [\![((e_b)_Q)_i]\!]_3 = \textit{True} \\ \textit{False}, & \text{iff for all } i \in \mathbb{N}\ ,\ [\![(c_Q)_i]\!]_3 \ \wedge_3 \ [\![((e_b)_Q)_i]\!]_3 = \textit{False} \\ \textit{Unknown}, & \text{Otherwise} \end{cases}$$

Here is an example to explain the 3-valued valuation. Let $C = (c_Q)_1 \wedge ((e_b)_Q)_1 \ \vee \ (c_Q)_2 \wedge ((e_b)_Q)_2$, where $(c_Q)_1$, $((e_b)_Q)_1$, $(c_Q)_2$, and $((e_b)_Q)_2$ are assigned to *True*, *Unknown*, *False*, and *Unknown*, respectively by interpretation $[\![\cdot]\!]_3$. Then $[\![C]\!]_3$ evaluates to *Unknown*.

In our method, the dual-rail representation is used, where two BDDs represent the three possible values of a node that corresponds to an individual MINLP problem. Thus, the correctness of our constraint solver is assured by the BDD package and Bonmin. Its capability is restricted by the two tools as well. The BDD package could be replaced by other SAT solvers or SMT solvers as mentioned in Section 2.3. Of course, Bonmin is not the only choice for solving MINLP problems. However, the target of our research is to explore a method for the subclass of formulas defined in formula (4), rather than to develop a new SMT solver to compete with other available tools. Thus, improving the efficiency of the tool integration is not the goal of this paper.

4. An SMT-based Algorithm to Determine Ranges for Parameters of Hybrid Systems

Based on our new solver, we propose a counterexample-guided algorithm to compute ranges of input parameters for symbolic simulation of hybrid systems. Section 4.1 gives the general restrictions for parameters that are determined by Algorithm 1. Section 4.2 explains the idea of the algorithm. Specific implementation details are given in Section 4.3.

ISBN 978-3-00-048889-4

4.1. Parameter Restriction

Before introducing Algorithm 1, we explain the few parameter restrictions we make. Only $\mathbb{R}$-typed input parameters are considered. We encode parameters in $\mathcal{W}_{\mathbb{R}}$ as a vector $\vec{p}_r$. The parameter valuation vector $\vec{v}$ is encoded similarly. An initial state set $I := \{\vec{v}_i \mid i \in \mathbb{N}\}$ contains finitely many parameter valuation vectors. It is required that each parameter $p_i \in \mathcal{W}_{\mathbb{R}}$ has a range constraint $\Delta_i := [\delta_{imin}, \delta_{imax}]$, where $\delta_{imin}, \delta_{imax} \in \mathbb{R}$, and $\delta_{imin} \leq \delta_{imax}$. Δ_i could be divided to subranges. Some subranges have parameter valuations that violate the specification, while some other satisfy the specification. Moreover, the parameters are independent so that all the subranges are numerical regions instead of relational regions. For example, given the input parameter set $\mathcal{W}_{\mathbb{R}} = \{p_0, p_1\}$, $\Delta_0 = [0.0, 1.0]$ and $\Delta_1 = [1.0, 2.0]$ are numerical regions for p_0 and p_1 respectively, while ${\Delta_1}' = [1.0, p_0]$ is a relational region for p_1, since the upper bound of region ${\Delta_1}'$ depends on parameter p_0. For the same example, we have the corresponding parameter vector $\vec{p}_r = \begin{pmatrix} p_0 \\ p_1 \end{pmatrix}$ and the range constraint vector $\vec{\Delta} = \begin{pmatrix} \Delta_0 \\ \Delta_1 \end{pmatrix}$. A parameter valuation vector $\vec{v} = \begin{pmatrix} 0.5 \\ 1.5 \end{pmatrix}$ assigns 0.5 and 1.5 to p_0 and p_1 respectively. $\{\vec{v}_0, \vec{v}_1\}$ is an initial state set, where $\vec{v}_0 = \vec{v}$ and $\vec{v}_1 = \begin{pmatrix} 0.3 \\ 1.2 \end{pmatrix}$.

4.2. Algorithm Explanation

Given a system's symbolic representation $\mathcal{G}$, specification C_b, and an initial state set I, the algorithm reuses the solution returned by Bonmin that violates the given specification. The recursive function *ValueRange* computes two subsets of the range constraint vector: Δ_f has range constraint vectors that lead to an over-approximation of the reachable states violating the given specification, while Δ_u has not yet found any parameter valuation that will go against the specification.

Function *Extend* in Line 5 generates a new range constraint vector $\vec{\Delta}_i$ by extending $\vec{v}_i$. For each parameter, the lower and upper bounds of $\vec{\Delta}_i$ are obtained by subtracting and adding $(j+1)$ times ϵ to the value according to $\vec{v}_i$, respectively. Function *Valuation* reorganizes the solution produced by Bonmin for this new range constraint vector. Whenever a parameter valuation vector $\vec{v}$ that violates the specification is found, we first exclude $\vec{v}_i$ from the initial state set, and then classify $\vec{\Delta}_i$ to Δ_f by function *Merge* iff $j = 0$ holds, as shown from Line 8 to Line 11. Otherwise, if function *Valuation* returns an empty set, then $\vec{\Delta}_i$ is classified to Δ_u by function *Merge* in Line 13. The algorithm executes the above steps for all the elements in the initial state set, before performing a recursive call. The recursive procedure terminates when either the initial state set becomes empty or the maximum recursion step N is reached.

Both Δ_f and Δ_u are obtained by extending each parameter valuation in the initial state set to a range constraint vector until some parameter valuation violates the property. The range constraint vector set Δ_f may include some range vectors that should belong to Δ_u, due to the introduced inaccuracy ϵ, while Δ_u provides the candidate ranges for parameters that meet the given specifications. Engineers could adapt the value of ϵ to get more accurate results. However, high accuracy sometimes makes troubles by generating infeasible problems for Bonmin. For the moment, there is no general suggestion to avoid this problem.

ISBN 978-3-00-048889-4

Algorithm 1 Computing Ranges for Input Parameters

Input:
- Initial State Set: $I = \{\vec{v}_i \mid i \in \mathbb{N}\}$
- Specification: C_b
- System's Symbolic Representation: $\mathcal{G}$
- Iteration Length: N

Output:
- Range Constraint Vector Set: Δ_u
- Range Constraint Vector Set: Δ_f

Local:
- Step Size: ϵ
- Parameter Vector: $\vec{p}_r$
- Parameter Valuation Vector: $\vec{v}$
- Iteration Counter : j

```
1: Δ_u ← ∅, Δ_f ← ∅, j ← 0
2: procedure ValueRange(I, C_b, G, N, Δ_u, Δ_f, j)
3:     if I ≠ {} then
4:         for all v_i ∈ I do
5:             Δ_i ← Extend(v_i, ε, j)
6:             {v} ← Valuation(Δ_i, C_b, G, p_r)
7:             if {v} ≠ {} then
8:                 I ← I\v_i
9:                 if j = 0 then
10:                    Δ_f ← Merge(Δ_f, Δ_i)
11:                end if
12:            else
13:                Δ_u ← Merge(Δ_u, Δ_i)
14:            end if
15:        end for
16:        j ← j + 1
17:        if j < N then
18:            ValueRange(I, C_b, G, N, Δ_u, Δ_f, j)
19:        else
20:            return (Δ_u, Δ_f)
21:        end if
22:    else
23:        return (Δ_u, Δ_f)
24:    end if
25: end procedure
```

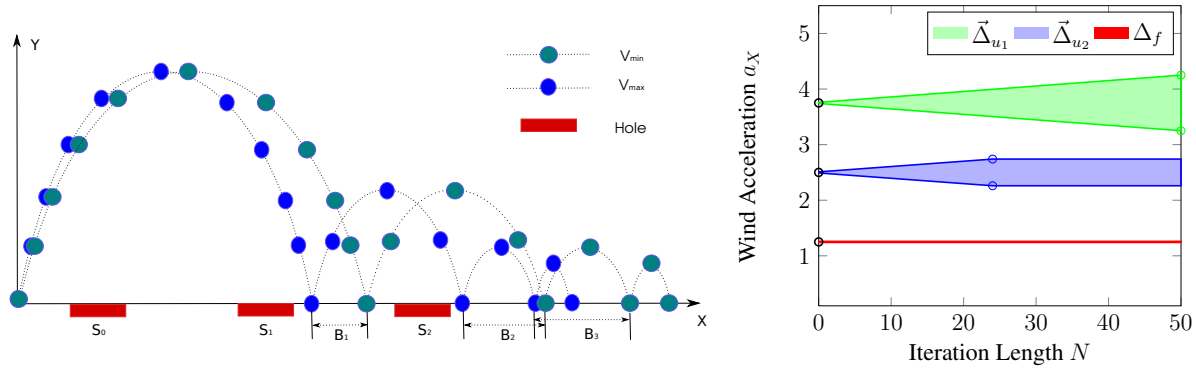
4.3. Implementation details

Algorithm 1 has been implemented on top of our *Averest* system. We benefit from our Averest system, since it offers a constantly evolving infrastructure containing tools for compilation, analysis, synthesis, and different techniques for formal verification. It provides algorithms that translate a *Hybrid Quartz* program to a set of guarded actions denoted as $\mathcal{G}$ [Sch09]. The guarded actions are the basis for the system's symbolic representation [BS10].

Even though the algorithm has been implemented in *Averest* for symbolic simulation of *Hybrid Quartz* programs, it does not mean that the algorithm is applicable only to *Hybrid Quartz* programs. Performing symbolic simulation on different hybrid models, the algorithm is still usable. However, engineers have to make some adaption for the different symbolic representations obtained by the various hybrid models they choose. Since the modeling language is not a prerequisite to understand the algorithm, we do not introduce the *Hybrid Quartz* language further. Instead, a brief overview of the language is given in Appendix A to understand the example.

5. Experimental Evaluation

In this section, we use the *Ball and Holes* scenario to demonstrate the use of our constraint solver and the proposed algorithm. Since both the constraint solver and the proposed algorithm are implemented in the *Averest* system, we encode this scenario in *Hybrid Quartz*. Readers could refer to Appendix B for the complete *Hybrid Quartz* model for this scenario. However, only the scenario description together with the simulation conditions and task will be introduced as prerequisite information. At the end of this section, a comparison between the experimental results and the expected results is given.



(a) The Ball and Holes Scenario

(b) Coverage for 3 Initial States

Figure 1: Experimental Scenario and Results

5.1. Scenario Description

As shown in Fig. 1a, throwing a ball from the ground with a non-zero speed, the ball will bounce continuously according to the environment conditions, e.g. the wind and the gravity. However, there are some holes on the ground where the ball will not be able to bounce again if it falls into the hole. The safe region starts from the right hand side edge of the furthest hole to the infinite.

For convenience of analysis, the speed is divided to V_X and V_Y which stands for the component in the horizontal X and vertical Y dimension, respectively, during the whole bouncing procedure. In the horizontal X dimension, the speed component V_X is influenced by the wind, so its value changes in accordance with the wind acceleration $a_X \in [a_{min}, a_{max}]$. The speed component in Y dimension is controlled by the gravity, which means a_Y is the constant value g. The ball may lose some energy in Y dimension after hitting the ground, where the energy loss coefficient c satisfies $c \in [0.0, 1.0]$. B and H encode the bounce trace in two dimensions that are decided by the speed components Vx and Vy, respectively. Variable n increases its value whenever the ball hits the ground. B, H, and n evolve with physical time t, thus we use $B(t)$, $H(t)$, $n(t)$ to reduce the number of variables appearing in the input formulas for the constraint solver.

5.2. Simulation Conditions and Task

In this scenario, except for $n \in \mathbb{Z}$, all the other variables are real-valued. We would like to know whether the ball could reach the safety region by restricting the number of bounces.

Assume the initial variable values are: $Vx\_init = 0.0$ and $Vy\_init = 19.6$, c is constantly equal to 0.5, and $0.1 \leq a_{min} \leq a_{max} \leq 5.0$. There are three holes in total, $S_0 = [0, 8.0]$, $S_1 = [10, 18]$, and $S_2 = [22.5, 24.5]$. The wind acceleration a_X is the only real parameter.

The simulation task is to check whether the ball could meet the specification that requires it to reach the safety region by bouncing at most twice.

5.3. Experimental Result

The following formula C_0 describes the following two situations that violate the given specification: the ball could not reach the safe region after bouncing twice, described by C_1; the ball falls into the hole region, described by C_2, C_3 and C_4.

ISBN 978-3-00-048889-4

$$C_0 = C_1 \vee C_2 \vee C_3 \vee C_4 \quad \text{where}$$
$$\begin{cases} C_1 = (\exists t).(2 \leq n(t) \wedge B(t) \leq 24.5) \\ C_2 = (\exists t).(n(t) \leq 2 \wedge H(t) \leq 0.0 \wedge 22.5 \leq B(t) \leq 24.5) \\ C_3 = (\exists t).(n(t) \leq 2 \wedge H(t) \leq 0.0 \wedge 10 \leq B(t) \leq 18) \\ C_4 = (\exists t).(n(t) \leq 2 \wedge H(t) \leq 0.0 \wedge 0 \leq B(t) \leq 8.0) \end{cases}$$

Given the parameter vector $\vec{p}_i = \left(a_X\right)$ and an initial state set $I = \{\left(1.25\right), \left(2.5\right), \left(3.75\right)\}$, Fig. 1b plots the symbolic simulation result by applying Algorithm 1 with $\epsilon = 0.01$. The three black circles are the three initial parameter valuations. Δ_u consists of two elements $\vec{\Delta}_{u1}$ and $\vec{\Delta}_{u2}$, depicted by the green and blue regions respectively. The red region stands for Δ_f that contains some parameter valuations violate the property. Two parameter valuations that violate the property are founded by this symbolic simulation procedure. The first one happened at the first iteration step for concrete value $\vec{v}_0 = \left(1.25\right)$. The other one appeared for $\vec{v}_1 = \left(2.5\right)$ at 24-th iteration step in which the two blue circles locate. Moreover, the iteration length has a big effect on the simulation coverage. The coverage increases generally as the increasing of iteration length. 24 iteration steps lead to a 19.6% coverage for this experiment, while 57.2% after performing 50 iteration steps.

Furthermore, formula (6) shows the returned range constraint vector set Δ_u after 50 iteration steps in Fig. 1b. The expected one Δ_t is supposed to contain two vectors $\vec{\Delta}_1$ and $\vec{\Delta}_2$ as shown by formula (7). The experimental result is consistent with the expected result.

$$\begin{cases} \Delta_u = \{\vec{\Delta}_{u1}, \vec{\Delta}_{u2}\} \\ \vec{\Delta}_{u1} = \left([3.25, 4.25]\right) \\ \vec{\Delta}_{u2} = \left([2.26, 2.74]\right) \end{cases} \quad (6) \qquad \begin{cases} \Delta_t = \{\vec{\Delta}_1, \vec{\Delta}_2\} \\ \vec{\Delta}_1 = \left((2.25, 2.8125)\right) \\ \vec{\Delta}_2 = \left((3.0625, +\infty)\right) \end{cases} \quad (7)$$

6. Conclusion

We developed a prototypical constraint solver for an undecidable logic for non-linear arithmetic theories over integers and reals with quantifiers. Based on which, we proposed a symbolic simulation algorithm for hybrid systems that are specified by parameterized models using standard data types (reals, integers and booleans) and non-linear dynamics. The algorithm has been implemented, and its feasibility is proved through a successful experimental evaluation using parameterized hybrid programs.

References

[ABS01] Annichini, A., A. Bouajjani, and M. Sighireanu: *TReX: A tool for reachability analysis of complex systems*. In Berry, G., H. Comon, and A. Finkel (editors): *Computer Aided Verification (CAV)*, volume 2102 of *LNCS*, pages 368–372, Paris, France, 2001. Springer.

[ACEF09] André, Étienne, Thomas Chatain, Emmanuelle Encrenaz, and Laurent Fribourg: *An inverse method for parametric timed automata*. International Journal of Foundations of Computer Science, 20(5):819–836, October 2009.

[AD94] Alur, R. and D.L. Dill: *A theory of timed automata*. Theoretical Computer Science (TCS), 126(2):183–235, 1994.

[Alu11] Alur, R.: *Formal verification of hybrid systems*. In Chakraborty, S., A. Jerraya, S.K. Baruah, and S. Fischmeister (editors): *Embedded Software (EMSOFT)*, pages 273–278, Taipei, Taiwan, 2011. ACM.

[And09] André, Étienne: *IMITATOR: A tool for synthesizing constraints on timing bounds of timed automata*. In *Theoretical Aspects of Computing - ICTAC 2009*, volume 5684 of *LNCS*, pages 336–342. Springer Berlin Heidelberg, 2009.

[Bau12] Bauer, K.: *A New Modelling Language for Cyber-physical Systems*. PhD thesis, Department of Computer Science, University of Kaiserslautern, Germany, Kaiserslautern, Germany, January 2012. PhD.

[BBC+08] Bonami, P., L. Biegler, A. Conn, G. CornuéJols, I. Grossmann, C. Laird, J. Lee, A. Lodi, F. Margot, N. Sawaya, and A. Wächter: *An algorithmic framework for convex mixed integer nonlinear programs*. Discret. Optim., 5(2):186–204, May 2008, ISSN 1572-5286.

[Ber00] Berry, G.: *The Esterel v5 language primer*, July 2000.

[BHZ06] Bordeaux, L., Y. Hamadi, and L. Zhang: *Propositional satisfiability and constraint programming: A comparative survey*. ACM Computing Surveys (CSUR), 38(4), December 2006.

[BKL+13] Belotti, P., C. Kirches, S. Leyffer, J.T. Linderoth, J. Luedtke, and A. Mahajan: *Mixed-Integer Nonlinear Optimization*. In Iserles, Arieh (editor): *Acta Numerica*, volume 22, pages 1–131. Cambridge University Press, 2013.

[BLWL08] Bu, L., Y. Li, L. Wang, and X. Li: *BACH: Bounded reachability checker for linear hybrid automata*. In *Formal Methods in Computer-Aided Design (FMCAD)*, pages 1–4, Portland, Oregon, USA, 2008. IEEE Computer Society.

[BS10] Bauer, K. and K. Schneider: *From synchronous programs to symbolic representations of hybrid systems*. In Johansson, K.H. and W. Yi (editors): *Hybrid Systems: Computation and Control (HSCC)*, pages 41–50, Stockholm, Sweden, 2010. ACM.

[CGJS13] Cimatti, A., A. Griggio, B. Joost Schaafsma, and R. Sebastiani: *The MathSAT5 SMT solver*. In Piterman, N. and S.A. Smolka (editors): *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 7795 of *LNCS*, pages 93–107, Rome, Italy, 2013. Springer.

[CMT11] Cimatti, A., S. Mover, and S. Tonetta: *HyDI: A language for symbolic hybrid systems with discrete interaction*. In *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*, pages 275–278. IEEE Computer Society, 2011.

[CVC] *CVC4*. `cvc4.cs.nyu.edu/web/`.

[EFH08] Eggers, A., M. Fränzle, and C. Herde: *SAT modulo ODE: A direct SAT approach to hybrid systems*. In Cha, S.D., J. Y. Choi, M. Kim, I. Lee, and M. Viswanathan (editors): *Automated Technology for Verification and Analysis (ATVA)*, volume 5311 of *LNCS*, pages 171–185, Seoul, South Korea, 2008. Springer.

[FH07] Fränzle, M. and C. Herde: *HySAT: An efficient proof engine for bounded model checking of hybrid systems*. Formal Methods in System Design (FMSD), 30:179–198, 2007.

[FK11] Fribourg, L. and U. Kühne: *Parametric verification and test coverage for hybrid automata using the inverse method*. In Delzanno, G. and I. Potapov (editors): *Reachability Problems (RP)*, volume 6945 of *LNCS*, pages 191–204, Genoa, Italy, 2011. Springer.

[FLD+11] Frehse, G., C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler: *SpaceEx: Scalable verification of hybrid systems*. In Gopalakrishnan, G. and S. Qadeer (editors): *Computer Aided Verification (CAV)*, volume 6806 of *LNCS*, pages 379–395, Snowbird, Utah, USA, 2011. Springer.

[Pau12] Paulson, L.C.: *MetiTarski: Past and future*. In Beringer, L. and A.P. Felty (editors): *Interactive Theorem Proving (ITP)*, volume 7406 of *LNCS*, pages 1–10, Princeton, New Jersey, USA, 2012. Springer.

[Pla10] Platzer, A.: *Logical Analysis of Hybrid Systems – Proving Theorems for Complex Dynamics*. Springer, 2010.

[QEP] *QEPCAD*. `www.usna.edu/CS/~qepcad/B/QEPCAD.html`.

[Reda] *Redlog*. `www.redlog.eu/`.

[Redb] *Reduce*. reduce-algebra.com/.

[Sch09] Schneider, K.: *The synchronous programming language Quartz*. Internal report 375, Department of Computer Science, University of Kaiserslautern, Kaiserslautern, Germany, December 2009.

[Tiw12] Tiwari, A.: *HybridSAL relational abstracter*. In Madhusudan, P. and S.A. Seshia (editors): *Computer Aided Verification (CAV)*, volume 7358 of *LNCS*, pages 725–731, Berkeley, California, USA, 2012. Springer.

[Z3] *Z3*. z3.codeplex.com/.

A. The Hybrid Quartz Language

Quartz is a synchronous language that is derived from the Esterel language [Ber00]. The execution of a Quartz program is defined by so-called *micro and macro steps*, where macro steps are defined by `pause` statements in the program. A reaction step consists of reading new inputs, and executing the code starting from the active `pause` statements to the next reached `pause` statements as the reaction of the program. Due to parallel statements $S_1 \parallel S_2$, more than one `pause` statement may be active at a time. Each `pause` statement introduces a control-flow location that is given a name by the compiler or the programmer.

The flow statement `flow` $\{S_1; ...; S_n\}$ `until` (σ) can replace a `pause` statement and will then extend the discrete transition by a continuous transition where the continuous variables behave according to the *flow assignments* $S_1; ...; S_n$ that are in the form of either `x` $\leftarrow \tau$ or `drv(x)` $\leftarrow \tau$ (that equate variable `x` or its derivation on time `drv(x)` with the expression τ). In contrast to the discrete transitions, continuous transitions require physical time and terminate as soon as the condition σ becomes true.

The continuous transition of the macro step starts with the variable environment determined by the immediate assignments as initial values. To distinguish between the 'discrete' value at the initial time of the continuous transition and the value during the continuous transitions, a new operator `cont(x)` is introduced: `x` always refers to the discrete value of a variable, whereas `cont(x)` refers to the (changing) value during the continuous evolution. For memorized and event variables `x` and `cont(x)` always coincide as these variables do not change during continuous evolutions.

Due to space limitations, we cannot give an overview of the language here, and refer to [Sch09, Bau12] for full details. Instead, we just list some of the statements used in the Appendix B and give an idea of their meaning:

- $x = \tau$ and `next`$(x) = \tau$ (immed./delayed assignments)
- `assume`(φ), `assert`(φ) (assumptions and assertions)
- ℓ : `pause` (start/end of macro step)
- $S_1; S_2$ (sequences)
- $S_1 \parallel S_2$ (synchronous concurrency)
- `loop` S (iteration)
- `flow` $\{S_1; ...; S_n\}$ `until` (σ) (flow statement)
- $M([params])$ (module call)

B. The Scenario Model

Most of the variable notations in Fig. 2 are the same as the ones introduced in Section 5.1. The *Ball and Holes Hybrid Quartz* model consists of three parts information.

The `macro` part gives the static hole region information, including $N \in \mathbb{Z}$ for the hole number, and two real-valued arrays $S\_min[N]$ and $S\_max[N]$ for the hole locations.

The main `module` starts with the input $\mathbb{R}$-variables, including a_x, Vx_init, Vy_init, Ts, and c. Among them, a_x is the same as a_X in the previous section for the instant wind acceleration that is sampled by the given period Ts, and a_X keeps unchanged until next sample period. Vx_init and Vy_init stand for the initial horizontal and vertical speed components, and c is the energy loss efficiency.

Some local variables are declared to describe the bounce procedure: B and H encode the bounce trace of the two dimensions that are decided by the speed components Vx and Vy, respectively. The other time related variable T works as a timer to stimulate the sampling procedure, so that both a and T should be reset after every Ts time units.

ISBN 978-3-00-048889-4

```
1  macro N =?, S_min[N] = ?, S_max[N] = ?;
2  module ParametricBall(real ?a_x, ?Vx_init, ?Vy_init, ?Ts ?c){
3      hybrid real Vx,Vy,T,B,H; real a; nat n;
4      // Initialization configuration
5      Vx = Vx_init; Vy = Vy_init; a = a_x;
6      // Ball Bounces
7      loop{
8          // Continuous dynamics
9          flow{
10             drv(B) <- cont(Vx); drv(Vx) <- a;
11             drv(H) <- cont(Vy); drv(Vy) <- -9.8;
12             drv(T) <- 1.0;
13         }until((cont(T) >= Ts) or ((cont(H) <= 0.0)&(cont(Vy) <= 0.0)));
14         if((T >= Ts) & ((cont(H)<= 0.0)&(cont(Vy) <= 0.0))){
15         //Case 1: the ball hits the ground at the sample time
16             next(a) = a_x; next(T) = 0.0; next(Vy) = -c*Vy; next(n) = n+1;}
17         else{
18             if((T >= Ts) & !((cont(H)<= 0.0)&(cont(Vy) <= 0.0))){
19             //Case 2: sample time, the ball does not hit the ground
20                 next(a)= a_x; next(T) = 0.0;}
21             else{
22             //Case 3: the ball hits the ground not at the sample time
23                 next(Vy) = -c*Vy; next(n) = n+1;}}
24         pause;}}
25 satisfies{// Specification
26     assert EF (exists(i = 0..N-1) ((S_min[i]<=B)&(B<=S_max[i])&(H<=0.0));}
```

Figure 2: The Ball and Holes Hybrid Quartz Model

Variable n increases its value whenever the ball hits the ground. Except for $n \in \mathbb{Z}$, all the other local variables are $\mathbb{R}$-variables.

The bouncing procedure is represented by a `loop` statement, which contains one continuous macro step for the dynamic evolutions, and another discrete macro step for reseting either the wind acceleration at each sample time or the velocity of the ball when it hits the ground.

Checking whether the ball could avoid all holes, is equal to verify whether there exists a path in the future so that the ball may fall into the hole region. Thus, the last `satisfies` part specifies the specification ignoring the restriction of the bounce number.

For the given scenario, if the wind acceleration is constant, the time duration of each macro step is the time when the ball stays in the air between two sequential bounces. However, the wind acceleration changes its value and is sampled every Ts time units. It separates the macro step with constant wind acceleration to several sub macro steps. It is safe to eliminate the sample-related variables Ts and T to ease the difficulty of synthesis by assuming the wind acceleration is constant. By this, the regions the ball could hit on the ground are over-approximated. If the hole region has no intersection with the approximation regions, then the ball avoids all the holes.

Evaluation of a software-based centralized traffic management inside run-time reconfigurable regions-of-interest of a mesh-based Network-on-Chip topology

Philipp Gorski, Tim Wegner, Dirk Timmermann

University of Rostock, Institute of Applied Microelectronics and Computer Engineering, Rostock
{philipp.gorski2, tim.wegner, dirk.timmermann}@uni-rostock.de

Abstract

This work proposes the introduction of multiple spatially independent network interfaces in order to connect computational resources to mesh-based Networks-on-Chip (NoCs). Furthermore, a flexible system for traffic monitoring is introduced supporting runtime reconfigurability as well as software-based centralized data aggregation and evaluation. These approaches are combined to form a sophisticated hardware/software-framework for run-time traffic management of NoC-based many-core systems exploiting the dual-path options of the underlying network. The conducted simulations show that this framework is capable of significantly decelerating thermal wear-out, while improving performance characteristics (e. g. packet delay) at moderate additional costs. Thereby, the resulting solution is evaluated in a simulation flow that considers the cycle-accurate timing of all hardware/software components and activity related temperature effects regarding the power dissipation as well as the reliability.

1. Introduction

Networks-on-Chip (NoCs) emerged as the next generation of communication infrastructures for modern many-core systems applying packet-based communication inside a networked topology of routers [1]. The most commonly used topology is the two-dimensional $N_X \times N_Y$ mesh (2D-mesh) reverting to XY-routing and wormhole-switching. This supports a regular physical layout as well as scalability and provides the required degree of parallelism [2], [3]. Typically, each computational resource is served by one dedicated router. In many cases routers are enhanced by techniques such as dual-path routing (e. g. XY/YX-routing), which requires the utilization of virtual-channels (VCs) and deepens the logical router pipeline by one stage [4]. The transferred packets consist of data words called flits, which pass the routers and links as smallest entity. This paper introduces different modifications and enhancements for such 2D-meshes. Contributions are as follows:

- Application of a quadrant-based mesh (QMesh) connecting each computational resource to the routers of all surrounding quadrants. This yields a reduction of the average hop distance and traffic interferences as well as it enables the integration of dual-path routing without VCs. The basic structure of the 2D-mesh with deterministic XY-routing is maintained.
- The QMesh is equipped with Flexible Traffic Monitoring (FTM) implemented as a hardware/software (HW/SW) solution. The observation of reconfigurable rectangular clusters is supported by utilizing adaptable algorithms and policies for monitoring and data evaluation. Thereby, comprehensive traffic recording is provided for each cluster including run-time end-to-end traffic analysis.
- Combination of the dual-path capabilities of the QMesh with FTM to enable intra-cluster path updates at run-time.

The resulting NoC infrastructure represents a flexible, configurable solution for adaptive routing at run-time. The overall framework is outlined in Figure 1. As it can be seen, FTM and potential path reconfigurations are performed on a cluster-basis, while monitoring is done individually for each NoC tile within a cluster. Aggregation of monitoring data collected for all tiles of a cluster is performed by a designated master-tile (MT). Traffic evaluation and adaptations are executed by SW agents/modules, while all monitoring functions as well as the aggregation of monitoring data are

ISBN 978-3-00-048889-4

realized in HW. Furthermore, regular network traffic is separated from monitoring and control packets by utilizing two fully independent networks. This QMesh-based FTM approach is expected to yield improvements regarding incidence of network saturation, average packet delay, fault-tolerance and thermally induced wear-out of NoC components. At the same time, additional power dissipation and area overhead shall be kept moderate.

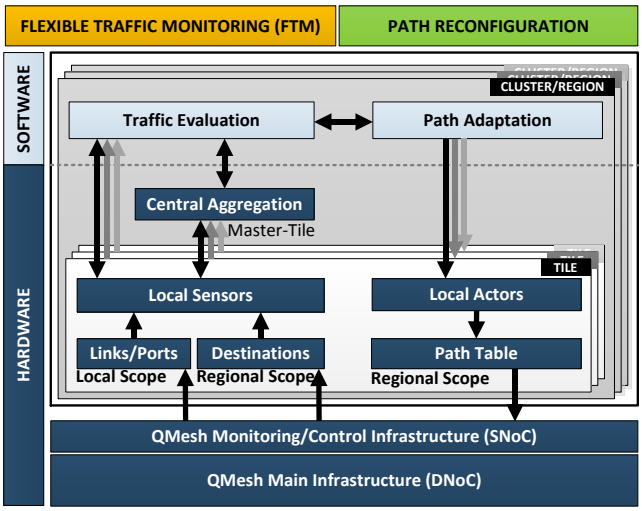


Figure 1: Partition of the NoC-based infrastructure into clusters and tiles for traffic monitoring, evaluation and adaptation of routing paths

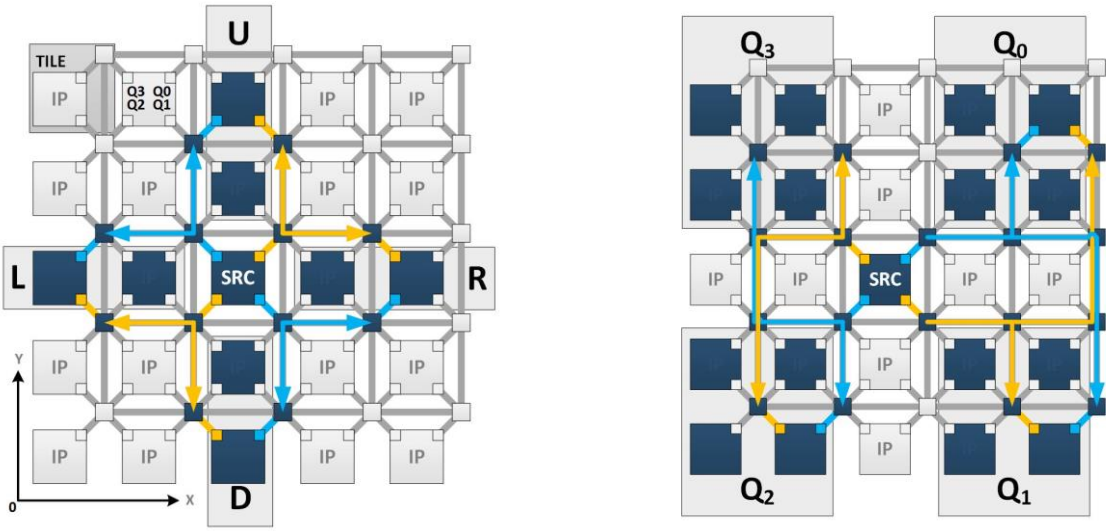
The remainder of this paper is organized as follows. Section 2 introduces the QMesh infrastructure, while in Section 3 the concept of FTM is addressed. Subsequently, in Section 4 the approach for run-time adaptation of routing paths is introduced. Section 5 provides experimental results regarding impact on performance and reliability. The paper is concluded in Section 6.

2. Quadrant-based mesh topology

The basic structure of the QMesh topology is illustrated in Figures 2a and 2b. Furthermore, different packet transfer scenarios and resulting dual-path options with applied XY-routing are depicted. Such NoC infrastructures with multi-ported resources were already proposed and addressed by different works published in diverse fashions [5]–[10] and thereby the QMesh itself can be interpreted as adapted version of the NR-Mesh in [5], [6]. However, the design focus of the QMesh is set to end-to-end (E2E) path configurability through programmable lookup tables at the sources and deterministic routing capabilities to enable predictable traffic management with clearly defined interfaces for the software-based update evaluations. Inside a regular 2D-mesh packets are generated at an arbitrary source (SRC) and are then injected into the NoC at the router located at Q_0 (see Figure 2). Subsequently, packets are forwarded along the X-direction and then along the Y-direction until the router at Q_0 at the destination (DST) is reached. Therefore, simple XY-coordinates of the 2D-mesh are sufficient for unique addressing. For the scenario in Figure 2a DSTs are designated U, R, D, L, while in Figure 2b DSTs are labeled Q_0 to Q_3.

In contrast, the QMesh allows for the injection/ejection of packets via different routers placed around the SRC/DST. For this reason, the routing information comprises not only the XY-coordinates of the DST router, but also the input quadrant $Q_{in,src}$ at the SRC and the correct output port $Q_{out,router}$ at the DST router to address the correct DST tile. Thus, addressing must be extended by 4 bits for $Q_{in,src}$ and $Q_{out,router}$ (2 bits each). The required quadrant information is derived from a programmable path table (PT), which is integrated at every tile representing the basis for the

dynamic path updates. The resulting overhead of $(N_X \times N_Y - 1) \times 4$ bits per tile is acceptable (e.g., around 32 byte for an 8×8 QMesh) and small compared to the kbyte-sized transmission buffers (BUF). For SRC/DST pairs with identical X- or Y-coordinates (see Figure 2a) two different path options with the same hop distance exist. If the DST is located inside one of the quadrants (see Figure 2b) with different X- and Y-coordinates, the path options vary in their distances by two hops. The PT lookup of $Q_{in,src}$ and $Q_{out,router}$ is performed when a message is written into the SRC's transmission buffer prior to packetization and is based on the DST's base address ($x_{dst,base}$, $y_{dst,base}$). The XY-coordinates of the base address are identical to the coordinates of the destination router in a standard 2D-mesh (e.g., router connected to NI at Q_0). In order to address the correct $Q_{out,router}$ the destination address is modified to match the correct endpoint router address.



(a) Same row or column paths (b) Same quandrant paths

Figure 2: Dual-path options inside a 5×5 QMesh for the available constellations of SRC/DST pairs

In comparison to a regular 2D-mesh the resulting path lengths are reduced by up to two hops. Additionally, nearest-neighbor traffic is removed from the links and the average router traffic load is reduced. Furthermore, nearly all destinations can be served via a second path. The combinations for all possible SRC/DST constellations as well as constraints for alternative paths and resulting reductions in hop distance are provided in Table 1. The QMesh provides deterministic dual-path capabilities at increased communication locality representing an optimal candidate for the realization of region-based adaptation strategies at run-time, especially such approaches proposed in [4], [11] but with the focus on software-based evaluations as proposed in [12]–[15].

Table 1: Dual-path options in the QMesh including constraints and hop distances compared to a regular 2D-mesh (from the perspective of a SRC)

Destination	**Path A (Default)**			**Path B**			
	$Q_{in,src}$	$Q_{out,router}$	Hop distance	$Q_{in,src}$	$Q_{out,router}$	Hop distance	Constraint
Up (U)	Q_0	Q_3	$n_{hop\text{-}2D} - 1$	Q_3	Q_0	$n_{hop\text{-}2D} - 1$	$x_{src} > 0$
Right (R)	Q_0	Q_1	$n_{hop\text{-}2D} - 1$	Q_1	Q_0	$n_{hop\text{-}2D} - 1$	$y_{src} > 0$
Down(D)	Q_1	Q_2	$n_{hop\text{-}2D} - 1$	Q_2	Q_1	$n_{hop\text{-}2D} - 1$	$x_{src} > 0$
Left (L)	Q_3	Q_2	$n_{hop\text{-}2D} - 1$	Q_2	Q_3	$n_{hop\text{-}2D} - 1$	$y_{src} > 0$
Q_0	Q_0	Q_0	$n_{hop\text{-}2D} - 2$	Q_1	Q_3	$n_{hop\text{-}2D}$	$y_{src} > 0$
Q_1	Q_1	Q_1	$n_{hop\text{-}2D} - 2$	Q_0	Q_2	$n_{hop\text{-}2D}$	none
Q_2	Q_2	Q_2	$n_{hop\text{-}2D} - 2$	Q_3	Q_1	$n_{hop\text{-}2D}$	$x_{dst,base} > 0$
Q_3	Q_3	Q_3	$n_{hop\text{-}2D} - 2$	Q_2	Q_0	$n_{hop\text{-}2D}$	$y_{src} > 0$ & $x_{dst,base} > 0$

3. Regional Traffic Monitoring

The applied periodic run-time traffic monitoring operates on reconfigurable clusters formed by NoC tiles within regions-of-interest. Furthermore, a centralized data aggregation is provided in every cluster and performed on so called master-tiles (MTs). The proposed approach for Flexible Traffic Monitoring (FTM) is designed for cluster sizes of up to N_{CS}=64 tiles. Multiple rectangular-shaped FTM clusters are permitted but overlapping is prohibited. Every cluster can be configured with an individual timing period selected from a discrete set fitting the requirements of its assigned workload fractions. The centralized processing and evaluation of monitoring data is performed by a software module managing cluster setup and reconfiguration as well. The initial decision for cluster formation and subsequent sizing adaptations derive from higher system-level services (e. g. application mapping), since they possess a more global view.

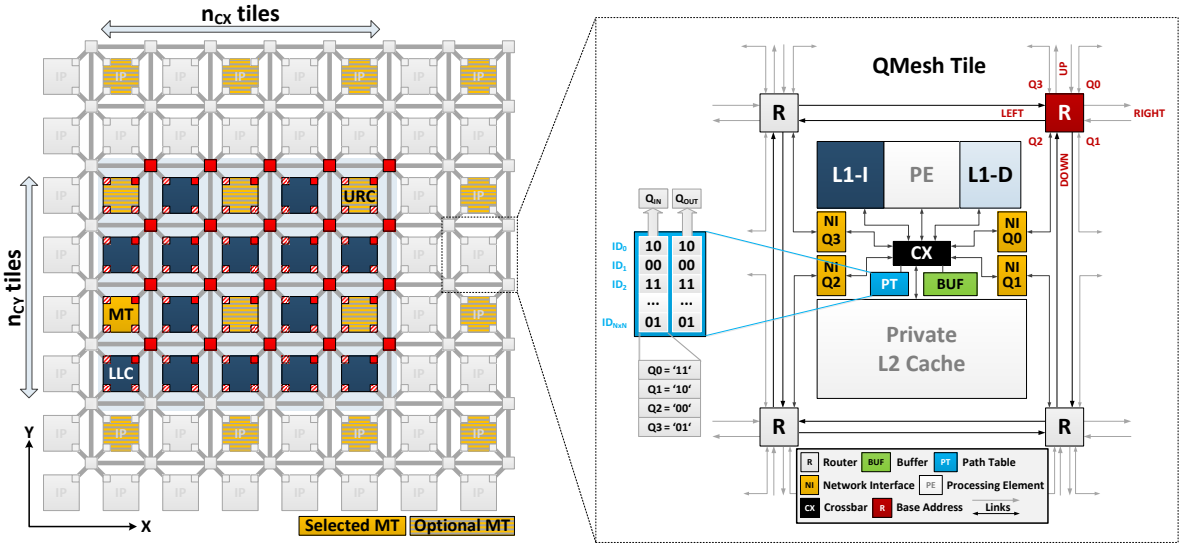


Figure 3: A 4×5 cluster inside an 8×8 QMesh

Figure 3 illustrates a 4×5 FTM cluster inside an 8×8 QMesh NoC ($N_{CS}=n_{CX}\times n_{CY}$). Due to the focus of this paper on traffic management, the description of FTM is kept short focusing on the main capabilities. Detailed explanations and discussions on FTM are provided in [16], [17]. The FTM communication regarding monitoring and control packets is realized via a fully independent network called SNoC, while data transfers of the workload applications are realized via the DNoC. Both networks provide full QMesh connectivity for each tile. Parameters like buffer sizing, link width or operating frequency can be customized independently to meet the requirements of particular traffic classes. Thus, the SNoC can be designed with minimum buffering and reduced link width. Furthermore, it is fully reusable for additional scenarios [16]. The FTM sensing mechanisms are based on simple threshold counters at each tile implemented in hardware. In detail, these counters are sourced by specific handshake logic signals of the DNoC tiles indicating data traversals along the quadrant NIs (Q_0 to Q_3), the eight output ports of the base router at Q_0 (i. e. U, R, D, L as well as Q_0 to Q_3), and for (N_{CS}-1) E2E paths terminating inside the cluster consisting of N_{CS} tiles. After expiration of discretely adjustable sensor periods defined as DNoC clock cycles (2^5 up to 2^{12} cycles) all counters of a tile are checked. In case a threshold is exceeded, the resulting overflow is reported to the selected MT of the cluster via the SNoC. The MT integrates additional groups of hardware-based overflow counters capturing the contents of monitoring packets of each slave-tile (ST) of its associated cluster. In order to improve flexibility and fault-tolerance multiple potential MTs are available within the QMesh allowing for migration of the monitoring software to another MT (see Figure 3). The anticipated fraction of potential MTs within the QMesh is 25 %. At the end of each monitoring cycle (e.g., after 100 sensor periods), DNoC load values for the tile outputs

(T_{load}), the eight outgoing links of the base routers at Q_0 (L_{load}) and for all SRC/DST paths inside the cluster (P_{load}) are provided by the FTM for the complete cluster. The recorded load values represent the utilization of the available bandwidth with maximum and average monitoring errors of 2 % and 0.5 % [17]. To conclude, the introduced FTM represents a run-time configurable solution enabling a holistic and globalized view on the current traffic situation of a specific spatial NoC region. For 4×4 and 8×8 clusters within an 8×8 QMesh the allowed monitoring cycles are $25.6 \cdot 10^3$ and $102.4 \cdot 10^3$ DNoC clock cycles (with parameterization as used in Section 5 from Table 2). As different studies show [15], [18], [19], such timing is sufficient to keep pace with the dynamic behavior of typical workload. The combination of software-based integration of the evaluation logic and flexibility regarding spatial sizing, timing and placement offers similar traffic management capabilities as the hardware-only implementation in [4], [11] with higher usability.

4. Software-based run-time adaptation of routing paths

The QMesh provides deterministic path adaptations through reprogramming of PTs (see Section 2), while FTM data serves to explore existing E2E path options for each SRC/DST pair inside a cluster. In this section an approach for run-time path adaptions is introduced combining the concepts of the QMesh and FTM. The evaluation of potential path updates is performed periodically after expiration of monitoring cycles (see Section 3) and is executed by the currently selected MT of the cluster. The processing order of the cluster tiles for path update evaluation is considered static, beginning at the lower left corner (LLC) and proceeding to the upper right corner (URC) row-by-row (see Figure 3). For every SRC tile along this route the monitoring software evaluates each path to cluster-internal DSTs. The static fashion is applied in order to achieve a convergence in the traffic optimization over several monitoring cycles. The path evaluation operates on copies of the tile output and link utilization maps (T^*_{load} and L^*_{load}). The update evaluation is only performed for valid paths, which requires the availability of at least one more path option and a monitored path utilization that is greater than a defined threshold $P_{load} > 0$. Note that in the QMesh some paths from/to tiles at the left and lower edge do not meet this condition (see Table 1). The flow for path evaluation and potential adaptations is divided into five steps.

- First, the given T^*_{load} and L^*_{load} maps along the currently selected XY-path SP_{old} = (A or B) are prepared by removing the monitored utilization (given by P_{load}) for this path. This is required for a fair comparison of both path options without bandwidth offset.
- Second, the metric for the comparison of both path options (SUM_A and SUM_B) is calculated from the prepared values of T^*_{load} and L^*_{load}. These values are calculated by parsing through these maps along the path coordinates and summing up the utilization data beginning at the tile output quadrant and proceeding over all links along the corresponding XY-paths.
- Third, the values for SUM_A and SUM_B are compared and the new path SP_{new} = (A or B) is selected depending on which path offers the minimal utilization sum. If the chosen path differs from the old one, this is registered by incrementing the number of occurred updates for path evaluations of the current SRC.
- Fourth, T^*_{load} and L^*_{load} maps are updated along the selected XY-path SP_{new} by adding the utilization (given by P_{load}) for the currently evaluated SRC/DST pair. This ensures that successive path evaluations operate on the adapted traffic situation considering prior path adjustments. Furthermore, the PT entry of the corresponding SRC/DST pair is updated to the new path.
- Finally, if all path options of the examined SRC are evaluated for all DSTs inside the cluster, the path setup of the SRC is checked for changes. If paths were adapted, the updated PT is transmitted to the SRC via the SNoC. Afterward, the next SRC is evaluated.

5. Experimental Evaluation

For the experimental evaluation, the QMesh as well as a regular 2D-mesh are integrated into a cycle-accurate SystemC-based simulator [20] under consideration of a standardized reference architecture for the routers/link [1]. This simulator considers traffic-load-driven thermal impacts for NoC routers as well as links and therefore allows for the estimation of temperature-based wear-out effects. Additionally, the average router/link temperature is required to calculate the correct power dissipation due to the dependency of the leakage power on it [21]. Therefore, a thermal power profile was generated via DSENT [22] under consideration of the temperature-throughput-relation from [23] for various CMOS technology sizes (45 nm down to 22 nm) and integrated into this simulator. Furthermore, FTM is integrated under full consideration of the required duration of the path update computations. For this purpose, a C++-based implementation of the FTM software module was profiled at the instruction level using the Sniper simulator [24] at version 5.1, which integrates the accurate Pin tool for this purpose [25]. Unfortunately, Sniper does not support the full integration of the complete simulation flow at the intended accuracy level due to the lack of multi-program workload simulation capabilities (timing of the software module would not be considered), no consideration of activity related temperature effects, as well as the less accurate coverage of the NoC via a simple queuing model with unrestricted resources (such as infinite buffers). For the MT core the built-in Intel Nehalem setup was deployed as processing architecture operating with a clock frequency of 2 GHz in a 45 nm CMOS technology. Network configurations, cluster sizing, and traffic pattern setup for the SystemC-based simulations can be obtained from Table 2. All simulations are executed deploying an 8×8 NoC with cluster sizes of 4×4 and 8×8 and reverting to synthetic traffic patterns. Applied patterns comprise transpose, bit reverse, bit complement and shuffle traffic as exemplary cases for single-program workload scenarios [1] (called bit permutation patterns). Additionally, nearest-neighbor (NN), hotspot (HOT) and rentian (RENT) traffic patterns are examined for the purpose of multi-program scenarios [1], [4], [11], [26] (called probabilistic distribution patterns). For the former, fractions of 20%, 40%, 60% and 80% of the total bandwidth are reserved for NN communication, while for the second these fractions are reserved for 8 hot modules along the perimeter of the NoC. The RENT traffic pattern applies a power law distribution along hop distances (see [26]). Generally, the PTs of the QMesh tiles are configured to select the shortest path A by default (see Table 1). The QMesh (unmanaged and managed via FTM) is compared to a regular 2D-mesh regarding relative improvements of the network saturation Δ_{SAT}, the power overhead Δ_{POWER} and packet delay reduction Δ_{DELAY}. Furthermore, thermal related wear-out effects at minimum and maximum packet injection rates (a_{MTTF} at PIR_{LOW} and PIR_{HIGH}) are analyzed.

Table 2: Basic simulation setup for the evaluation of the traffic management

Parameter	DNoC	SNoC
Link width / Flit size	64 bits	16 bits
Router buffer size	9 flits	1 flit
Clock cycle	1 ns	1 ns
Routing / Switching	XY / Wormhole	
Packet size	20% with 2 flits, 80% with 9 flits (7.6 flits on average)	
Cluster size	4×4 and 8×8	
Technology	45 nm CMOS	

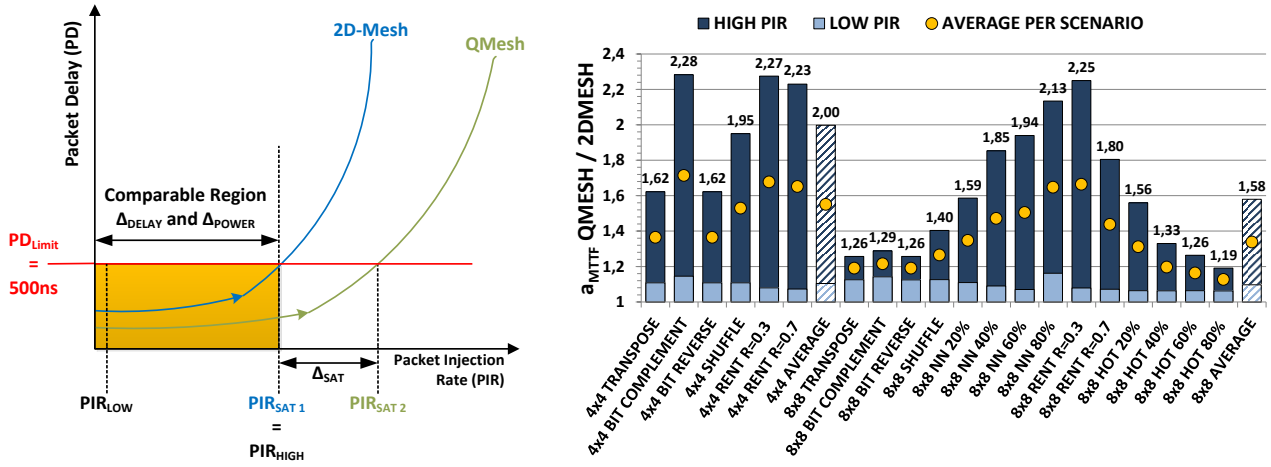
Reliability Results

The mean-time-to-failure (MTTF) describes the average period of time after that a CMOS device fails under constant operating conditions. The acceleration factor a_{MTTF} in turn indicates if the

variation of a parameter induces a decrease or an increase of the MTTF [27]. In this context, Equation 1 represents the relationship between the MTTF for a QMesh (t_{QMesh}) and the MTTF for a regular 2D-mesh (t_{2DMesh}) focusing on the temperature-related increase (a_{MTTF} <1) or decrease (a_{MTTF} >1) of wear-out progress. E_a is the activation energy of a particular wear-out effect in electron volts (0.7 eV at 45 nm), k is the Boltzmann's constant ($8.6 \cdot 10^{-5}$ eV /K), and T_{QMesh} and T_{2DMesh} describe the absolute mesh temperatures in Kelvin. In the following, a_{MTTF} is utilized to compare the QMesh with the 2D-mesh regarding thermal wear-out improvements for router and link components.

$$a_{MTTF} = \frac{t_{QMESH}}{t_{2DMESH}} = e^{\frac{E_a}{k} \cdot \left(\frac{1}{T_{QMESH}} - \frac{1}{T_{2DMESH}} \right)} \quad (1)$$

The results for a_{MTTF} are depicted in Figure 4. As it can be seen, significant improvements for various traffic patterns, tested for cluster sizes of 4×4 and 8×8 inside an 8×8 NoC, are realizable with the application of the QMesh. This applies for both, low and high packet injection rates (PIR), and results in an increase of the MTTF by a factor of almost 0.55 (for 4×4 clusters) and 0.35 (for 8×8 clusters) on average. For specific pattern at high average traffic intensity, the QMesh offers the potential to double the lifetime of the NoC components. This can be mainly attributed to the reduction of the communication locality by the QMesh, which decreases the average data throughput and therewith the activity of the routers and links. Hence, the average operating temperature is reduced. Thereby, the results for the unmanaged QMesh do not vary from those of the managed QMesh with applied FTM, because the path-update periods are not short enough to encounter activity-driven peaks along the temperature changes in the range of 10^3 clock cycles or smaller and the default setup with minimal paths (path A option for all PT positions) has the maximum contribution to the wear-out reduction.



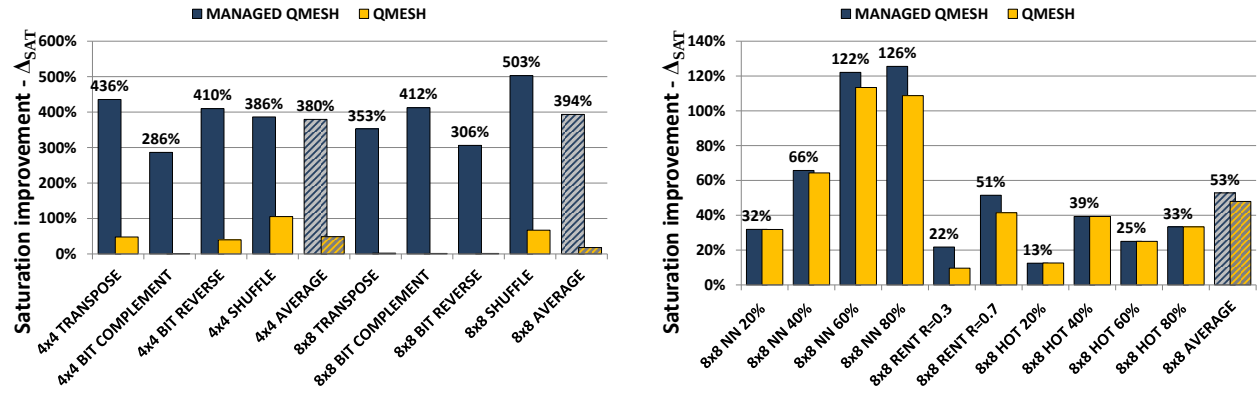
(a) NoC comparison scheme (b) Thermal wear-out improvement of the basic QMesh

Figure 4: NoC comparison scheme and thermal wear-out improvement of the basic QMesh

Performance Results

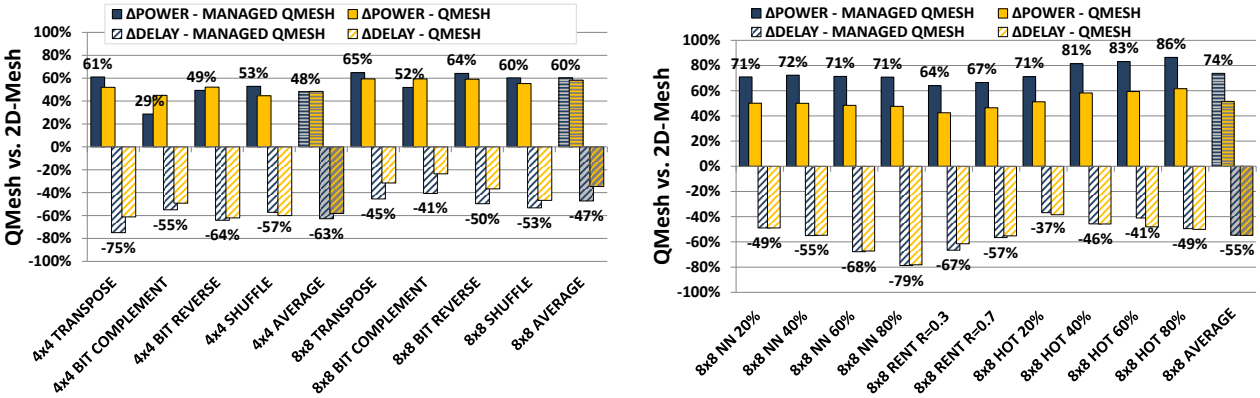
The following charts summarize the impact of the basic and the managed QMesh on run-time costs as well as performance. Regarding the evaluation of power dissipation, the corresponding results of the FTM units as well as the SNoC are integrated for the managed QMesh to provide a comprehensive and realistic analysis. The unmanaged QMesh solely contains the required modifications of eight ports per router (only five in the 2D-mesh) and the three additional NIs. The results in Figures 5a and 5b illustrate the improvement of network saturation Δ_{SAT} of the QMesh variants in relation to the 2D-mesh. Figure 5a indicates that BP patterns at both cluster sizes profit most from the traffic management with average improvements of 380% and 394% for 4×4 and 8×8

cluster sizes. In contrast, PD patterns in Figure 5b reveal a more moderate growth of the NoC saturation limits. The resulting saturation improvements of the managed QMesh are 53% on average in comparison to the 2D-mesh for the 8x8 clustering. Furthermore, the spread between the basic and the managed QMesh is only significant if the pattern provides enough communication locality (i.e., NN > 40% or RENT). Otherwise, the traffic management does not find solutions that provide advanced traffic rebalancing to outperform the basic QMesh with its shortest path PT setup. This results from the presence of many concurrent E2E traffic flows inside the NoCs with such PD workloads. Contrary, the improvements along the BP patterns result of the dominance few E2E connections. Each source has one fixed destination. Realistic workload is expected to be found in between these two corner-case-scenarios as shown in [15] by the evaluation of dominant flows.



(a) Bit permutation (BP) patterns (b) Probabilistic distribution (PD) patterns

Figure 5: Saturation improvements Δ_{SAT} for the analyzed traffic patterns



(a) Bit permutation (BP) patterns (b) Probabilistic distribution (PD) patterns

Figure 6: Relative changes of power dissipation Δ_{POWER} and packet delay Δ_{DELAY}

In Figures 6a and 6b the differences of average total power dissipation Δ_{POWER} and average packet header delay Δ_{DELAY} of the QMesh variants in relation to the 2D-mesh along the comparable regions (see Figure 4a) are depicted. Similar to improvements regarding network saturation, Figure 6a shows that the QMesh provides better results for BP patterns. Generally, the packet delay can be reduced due to application of runtime path adaptation as well as through the basic QMesh, while power consumption is increased by additional hardware in both QMesh variants. Furthermore, the traffic rebalancing along the managed QMesh helps to reduce the power overhead due to thermally-related activity reduction for some BP pattern. For 4×4 clusters, NoC power increases by 48 % averaged over all simulated BP patterns, while packet delay can be reduced by around 63 %. Deployment of 8×8 clusters yields an average increase of power by 60 % and an average reduction of packet delay by 47 %. Figure 6b illustrates that the power overhead is constantly higher for PD

patterns resulting in an average increase of 74 % for the managed QMesh. Furthermore, the offset of around 20% in comparison the basic QMesh due to the additional hardware becomes obvious. In parallel, packet delay is reduced by 55 % on average. Only for PD patterns with high communication locality (NN at 60 % and 80 % and RENT) reductions of packet delay outweigh the related power increase. However, values are restricted to traffic ranges where the 2D-mesh is able to operate at, while the QMesh provides capacities far beyond (see Figure 5a and 5b). Therefore, more beneficial results regarding the ratio of power increase to packet delay reduction are anticipated, if the PIR per tile is increased.

6. Conclusion

This paper introduces two independent approaches to many-core systems based on Networks-on-Chip (NoCs). Firstly, this concerns the concept of multiple spatially independent network interfaces deployed for connecting the computational resources of such systems to mesh-based NoCs. This yields the option of selecting different end-to-end paths for data transfers depending on the current load profile of the mesh. Additionally, the reliability improves further, since computational resources remain accessible in case of faulty routers or network interfaces. Secondly, this concerns a flexible system for traffic monitoring offering the possibility of run-time reconfiguration of local monitoring clusters. Within these clusters a software-based monitoring module is responsible for aggregation and evaluation of recorded communication activities and traffic loads. Moreover, these two approaches are combined to enable run-time traffic management by software-based adaptation of routing paths. Basically, this management utilizes the traffic information gathered from the monitoring system to decide which of the available paths is preferable regarding impact on load distribution. Results indicate that major run-time characteristics of many-core systems can be improved by the proposed run-time traffic management. This especially applies to network saturation and packet transfer delay. Furthermore, temperature-related wear-out effects are decelerated significantly, while additional costs (i. e. power dissipation due to additional hardware) are acceptable.

References

[1] W. J. Dally and B. P. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers, 2004, pp. 1–550.

[2] E. Salminen, A. Kulmala, and T. D. Hämäläinen, "Survey of Network-on-chip Proposals," *WHITE Pap. OCP-IP, MARCH*, no. March, pp. 1–12, 2008.

[3] A. Agarwal, C. Iskander, H. T. Multisystems, and R. Shankar, "Survey of Network on Chip (NoC) Architectures and Contributions," *J. Eng. Comput. Archit.*, vol. 3, no. 1, pp. 1–15, 2009.

[4] R. Manevich, I. Cidon, A. Kolodny, I. Walter, and S. Wimer, "A Cost Effective Centralized Adaptive Routing for Networks-on-Chip," *2011 14th Euromicro Conf. Digit. Syst. Des.*, vol. 9, no. 2, pp. 39–46, Aug. 2011.

[5] J. Camacho, J. Flich, J. Duato, H. Eberle, and W. Olesinski, "A power-efficient network on-chip topology," in *Proceedings of the Fifth International Workshop on Interconnection Network Architecture On-Chip, Multi-Chip - INA-OCMC '11*, 2011, pp. 23–26.

[6] J. Camacho, J. Flich, J. Duato, H. Eberle, and W. Olesinski, "Towards an Efficient NoC Topology through Multiple Injection Ports," in *2011 14th Euromicro Conference on Digital System Design*, 2011, pp. 165–172.

[7] D. Park, C. Nicopoulos, J. Kim, N. Vijaykrishnan, and C. R. Das, "A Distributed Multi-Point Network Interface for Low-Latency, Deadlock-Free On-Chip Interconnects," in *2006 1st International Conference on Nano-Networks and Workshops*, 2006, pp. 1–6.

[8] S. Volos, C. Seiculescu, B. Grot, N. K. Pour, B. Falsafi, and G. De Micheli, "CCNoC: Specializing On-Chip Interconnects for Energy Efficiency in Cache-Coherent Servers," in *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, 2012, no. Nocs, pp. 67–74.

[9] P. Zarkesh-Ha, G. B. P. Bezerra, S. Forrest, and M. Moses, "Hybrid Network on Chip (HNoC): Local Buses with a Global Mesh Architecture," in *Proceedings of the 12th ACM/IEEE international workshop on System level interconnect prediction - SLIP '10*, 2010, no. c, pp. 9–14.

[10] A. E. Zonouz, M. Seyrafi, A. Asad, M. Soryani, M. Fathy, and R. Berangi, "A Fault Tolerant NoC Architecture for Reliability Improvement and Latency Reduction," in *2009 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools*, 2009, pp. 473–480.

[11] R. Manevich, I. Cidon, A. Kolodny, and I. Walter, "Centralized Adaptive Routing for NoCs," *IEEE Comput. Archit. Lett.*, vol. 9, no. 2, pp. 57–60, Feb. 2010.

[12] P. Gorski, C. Cornelius, D. Timmermann, and V. Kühn, "Centralized Adaptive Source-Routing for Networks-on-Chip as HW/SW-Solution with Cluster-based Workload Isolation," in *8th International Conference on Systems (ICONS'13)*, 2013, no. c, pp. 207–215.

[13] J. W. van den Brand, C. Ciordas, K. Goossens, and T. Basten, "Congestion-Controlled Best-Effort Communication for Networks-on-Chip," in *2007 Design, Automation & Test in Europe Conference & Exhibition*, 2007, pp. 1–6.

[14] A. Sharifi and M. Kandemir, "Feedback control for providing QoS in NoC based multicores," in *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, 2010, pp. 1384–1389.

[15] Y. Jin, E. J. Kim, and T. M. Pinkston, "Communication-Aware Globally-Coordinated On-Chip Networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 2, pp. 242–254, Feb. 2012.

[16] P. Gorski, C. Cornelius, D. Timmermann, and V. Kühn, "RedNoCs: A Runtime Configurable Solution for Cluster-based and Multi-objective System Management in Networks-on-Chip," in *Eighth International Conference on Systems (ICONS'13)*, 2013, no. c, pp. 192–201.

[17] P. Gorski and D. Timmermann, "Centralized traffic monitoring for online-resizable clusters in Networks-on-Chip," in *2013 8th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*, 2013, pp. 1–8.

[18] W. Heirman, J. Dambre, D. Stroobandt, and J. Van Campenhout, "Rent's Rule and Parallel Programs: Characterizing Network Traffic Behavior," in *Proceedings of the tenth international workshop on System level interconnect prediction - SLIP '08*, 2008, p. 87.

[19] A. K. Mishra, O. Mutlu, and C. R. Das, "A heterogeneous multiple network-on-chip design: an application-aware approach," in *Proceedings of the 50th Annual Design Automation Conference on - DAC '13*, 2013, pp. 1–10.

[20] M. Gag, T. Wegner, and P. Gorski, "System level modeling of Networks-on-Chip for power estimation and design space exploration," in *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV 2013)*, 2013, pp. 25–34.

[21] W. Huang, "HotSpot — A Chip and Package Compact Thermal Modeling Methodology for VLSI Design - Dissertation," University of Virginia, 2007.

[22] C. Sun, C.-H. O. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L.-S. Peh, and V. Stojanovic, "DSENT - A Tool Connecting Emerging Photonics with Electronics for Opto-Electronic Networks-on-Chip Modeling," in *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, 2012, pp. 201–210.

[23] K. Aisopos, "Fault Tolerant Architectures for On-Chip Networks - Dissertation," Princeton University, 2012.

[24] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-Core Simulation," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '11*, 2011, pp. 1–12.

[25] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation," *ACM SIGPLAN Not.*, vol. 40, no. 6, pp. 190–201, Jun. 2005.

[26] R. Manevich, I. Cidon, and A. Kolodny, "Handling global traffic in future CMP NoCs," in *Proceedings of the International Workshop on System Level Interconnect Prediction - SLIP '12*, 2012, pp. 40–48.

[27] M. White, "Scaled CMOS technology reliability users guide," Pasadena, California, 2010.

Ein Verfahren zur Bestimmung eines Powermodells von Xilinx MicroBlaze MPSoCs zur Verwendung in Virtuellen Plattformen

Sören Schreiner, Kim Grüttner, Sven Rosinger
OFFIS e.V., Oldenburg
{schreiner | gruettner | rosinger}@offis.de

Wolfgang Nebel
Carl von Ossietzky Universität Oldenburg
wolfgang.nebel@uni-oldenburg.de

Zusammenfassung

Durch die Einführung von *Multi-Processor-System-on-Chips* werden die steigenden Anforderungen an die Performanz für moderne Eingebettete Systeme erfüllt. Oft sind diese aber im mobilen Einsatz und müssen besondere Anforderungen an ihre Leistungsaufnahme erfüllen. Die vorliegende Arbeit stellt einen Ansatz vor, mit dem durch *Charakterisierung* einer realen Hardware eine *Virtuelle Plattform* des gleichen Systems mit gemessenen Verlustleistungsdaten annotiert werden kann. So können im finalen Stadium echte Anwendungen auf der *Virtuellen Plattform* ausgeführt und Vorhersagen über die Leistungsaufnahme des realen Systems getätigt werden. Zur ersten Evaluation des beschriebenen Ansatzes wurde ein Xilinx MicroBlaze *MPSoC* Design auf einem FPGA implementiert. Mit Hilfe von *Micro-Benchmarks* wurde die Verlustleistung charakterisiert und aus den erhaltenen Daten ein *Power-Modell* des Designs erstellt. Dieses wurde in einer *Virtuellen Plattform* des gleichen Designs integriert und bezüglich seiner Verlustleistungsvorhersage in einer Co-Simulation mit der Messung der Verlustleistung am FPGA-Design verglichen. Für die durchgeführten Experimente hat sich ein akkumulierter Fehler kleiner als 1% ergeben.

1. Einleitung

Moderne *Multi-Processor-System-on-Chips* (MPSoCs) erfüllen durch ihre Performanz und vielfältigen Konfigurationsmöglichkeiten die stetig steigenden Anforderungen auf dem Gebiet der *Eingebetteten Systeme*. Sie können eine Vielzahl an unterschiedlichen Komponenten enthalten, wie mehrere Prozessorkerne, Digitale Signal Prozessoren (DSPs), flüchtige und nicht-flüchtige Speicher, Field Programmable Gate Arrays (FPGAs), etc. Diese Gegebenheiten machen *MPSoCs* besonders interessant für komplexe Anwendungen. Durch die fortschreitende Miniaturisierung der Transistorgröße und die stetig steigende Komplexität sowie die größer werdende Anzahl der verbauten Komponenten im Chip, kommt es allerdings zu einer immer größeren Energiedichte[2]. Dies führt zu einem Anstieg der Leistungsaufnahme und der Wärmeentwicklung des Chips. Durch die herrschenden Ströme kann die Temperatur soweit ansteigen, dass die Möglichkeit der Beschädigung oder gar Zerstörung der Systeme besteht. Weiterhin kann die Steigerung der Energiedichte von modernen Akkumulatoren nicht mit der wachsenden Leistungsaufnahme der Chips mithalten, was insbesondere bei mobilen Geräten, wie Smartphones und Tablets, durch kurze Laufzeiten im mobilen Betrieb ins Gewicht fällt. Um diesen Umständen entgegenzuwirken, müssen die Leistungsaufnahme und so auch die auftretenden Ströme im Chip gesenkt werden. Zur Realisierung werden verschiedene Techniken eingesetzt, wie *Dynamic Voltage and Frequency Scaling* (DVFS) [5], *Clock Gating* [4] oder *Power Gating* [10]. Solche Techniken machen es möglich, dass die einzelnen Komponenten eines *MPSoCs* in unterschiedliche Power-Modi versetzt werden können, um Eigenschaften wie Performanz, Leistungsaufnahme oder Wärmeentwicklung an

Diese Arbeit ist im Rahmen des Europäischen Forschungsprojekts CONTREX - „Design of embedded mixed-criticality CONTRol systems under consideration of EXtra-functional properties“ von der Europäischen Kommission unter der Fördernummer FP7-247999 unterstützt worden.

die aktuellen Anforderungen des Systems anzupassen. Um eine nichtinvasive Simulation des Systems durchführen zu können, wäre eine *Virtuelle Plattform* (VP) als Prototyp, welche mit den genannten Eigenschaften annotiert wird, von großem Vorteil. Mit einer solchen Technik könnten Entwickler auch ohne präsente Hardware und aufwendige Messeinrichtungen Aussagen über die Charakteristika eines Systems und die Einhaltung der dazugehörigen Anforderungen treffen. In dieser Arbeit wird der Fokus auf die Leistungsaufnahme eines zu analysierenden Systems gesetzt. In der vorgestellten Methodik wird die reale Hardware über ein messtechnisches Verfahren in ihren einzelnen Power-Modi charakterisiert. Die ermittelten Daten der Leistungsaufnahmen werden im Anschluss in Form eines Power-Modells an eine VP angebunden, welche dann als virtuelle Messstation bei der Ausführung realer Anwendungen dient. Ein erstelltes Beispiel auf Basis eines FPGA-Designs dient einer ersten Evaluation und zeigt einen über die Zeit akkumulierten, vorzeichenbehafteten Fehler kleiner 1%.

2. Verwandte Arbeiten

In [9] von der Uni Erlangen-Nürnberg in Kooperation mit Intel wird das Framework MAESTRO beschrieben, welches es ermöglicht nicht-funktionale Eigenschaften, wie Zeit, Leistungsaufnahme und Temperatur von *MPSoCs* zu modellieren und zu analysieren. Um das gesamte System zu modellieren wird die hauseigene Sprache *SysteMoC* verwendet, welche auf *SystemC* basiert. Das Modell besteht dabei aus mehreren Ebenen, welche unabhängig voneinander modelliert werden und so ein schnelles Austauschen für eine *Design Space Exploration* ermöglichen. Funktionale Modelle der ausgeführten Applikationen werden ebenfalls in *SysteMoC* entwickelt. Komponenten, wie CPUs, erhalten eine weitere Architekturebene, welche deren internen Komponenten feiner darstellen. Diese erhalten dann Interaktionsmodelle, welche bspw. das zeitliche, thermische oder leistungsaufnehmende Verhalten abbilden. Sie leiten ihre aktuellen Zustände an konkrete Modelle weiter, welche die finalen Ergebnisse der Simulation liefern. Größter Unterschied zu der hier vorliegenden Arbeit ist somit schon bei der auszuführenden Applikation zu finden. Hier werden lediglich funktionale Modelle der Applikation auf der entwickelten Plattform ausgeführt. Bei dem hier vorliegenden Ansatz soll aber die finale Applikation des realen Systems, mit nur möglichst minimalen Anpassungen, auf einer VP ausgeführt werden. Weiterhin wird nicht erläutert wie die konkreten Modelle für die Verlustleistung entstehen und wie ihre eigentliche Charakterisierung vorgenommen wird.
In [7] und [8] wird ein sehr ähnlicher Ansatz zu dem hier Vorliegenden vorgestellt. Bei der hybriden Verlustleistungsabschätzung wird eine Hardware-Plattform mit der später in Abschnitt 3 beschriebenen Messmethodik auf Funktionsebene charakterisiert und anhand dessen ein Verlustleistungs-Modell generiert. Dieses Modell wird dann an eine VP gekoppelt, welche die charakterisierte Hardware abbildet, um auf dieser die jeweilige Applikation laufen zu lassen. Als VP kommt *Open Virtual Platform* (OVP) zum Einsatz. Durch die Annotation mit dem Verlustleistungsmodell können aus der VP Daten über die Leistungsaufnahme des simulierten Systems extrahiert und ausgewertet werden. Im Vergleich zur in dieser Arbeit Methodik lässt sich ein Unterschied in den Verlustleistungsmodellen finden. Während in unserem Ansatz *Power State Machines* (PSMs) [1] verwendet werden, welche mit ihren einzelnen Zuständen die Power-Modi der einzelnen Komponenten des *MPSoC* abbilden, werden in [7] und [8] mathematische Funktionen verwendet, welche parametrisiert werden müssen. Hierfür werden im Anschluss an die Charakterisierung die gewonnen Daten mit Hilfe von Regressionsverfahren in die parametrisierbaren Polynome umgewandelt. Die Parameter sind bspw. Prozessorfrequenz, Busfrequenz, Cache-Miss-Rate, etc.
In [11] wird ein rein modellgetriebener Ansatz zur Verlustleistungsabschätzung von *Systems on Chips* (SoCs) vorgestellt. Auf Basis von Modellen, Metamodellen und Modelltransformationen werden Modelle des SoCs und der Applikation auf hoher Abstraktionsebene im MARTE Standard (*Modeling and Analysis for Real-Time and Embedded systems*) erstellt, die dann durch Transformationen weiter verfeinert werden. Ergebnis ist ein zyklenakkurates Modell des Systems in *SystemC*, das simuliert

ISBN 978-3-00-048889-4

werden kann. Zur Verlustleistungsabschätzung werden die transformierten Modelle auf Höhe des *Cycle-Accurate Bit-Accurate* (CABA) Levels auf zwei unterschiedliche Weisen mit Power-Modellen annotiert. Bei den Komponenten des Systems wird zwischen White-Box IPs und Black-Box IPs differenziert. Bei White-Box IP Komponenten, bei denen der Quellcode verfügbar ist, werden Event-Zähler eingefügt, welche die relevanten Aktivitäten für die Power-Modelle sichtbar machen. Bei Black-Box IP Komponenten, ohne verfügbaren Quellcode, wird ihre Kommunikation mit anderen Komponenten durch die Power-Modelle beobachtet. So können Rückschlüsse auf den aktuellen Power-Zustand der jeweiligen Komponente gezogen werden. Innerhalb der Power-Modelle können PSMs oder analytische Funktionen verwendet werden, um die aktuelle Leistungsaufnahme einer Komponente abbilden zu können. Die Daten für diese Power-Modelle werden durch Low-Level Simulationen gewonnen. Als Ergebnis kann das System simuliert und so die Leistungsaufnahme beobachtet werden. Der Unterschied zu der hier vorliegenden Arbeit liegt in der Anbindung der Power-Modelle. In [11] sind sie fest in der simulierbaren Umgebung eingebunden. In unserer Methodik hingegen werden sie von außen an die VP angebunden.

3. Methodik

Die im Folgenden vorgestellte Methodik zur Verlustleistungsabschätzung von Systemen basierend auf *MPSoCs* zeigt einige Gemeinsamkeiten zu denen in Abschnitt 2 vorgestellten Arbeiten. Der Fokus in unserer Arbeit wird auf die Charakterisierung und Erstellung der Power-Modelle sowie deren Anbindung an die Simulationsumgebung gesetzt. Insbesondere die Charakterisierung der Verlustleistungsdaten eines Systems wird in vielen verwandten Arbeiten nicht beschrieben und gilt als gegeben. In Abbildung 1 wird ein Überblick über das gesamte Vorgehen gegeben.

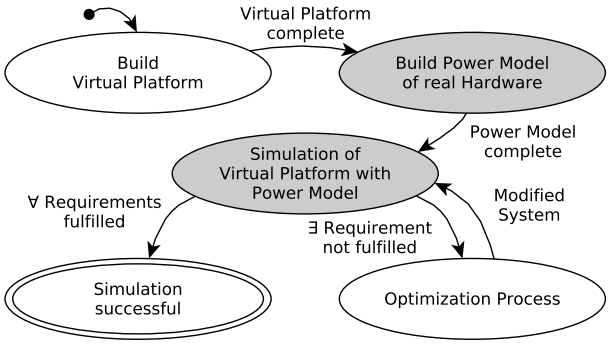


Abbildung 1: Ablauf der Methodik

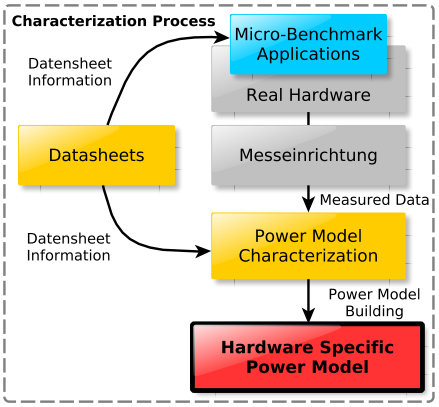


Abbildung 2: Charakterisierungsprozess

Begonnen wird mit dem Erstellen der *Virtuellen Plattform*. Diese bildet die vorhandene, reale Hardware im gewünschten Detailgrad nach. Im nächsten Schritt werden die entscheidenden Komponenten deren Leistungsaufnahme simuliert werden soll charakterisiert. Das so erhaltene *Power-Modell* wird dann an die *Virtuelle Plattform* angebunden und diese mit der jeweiligen Applikation simuliert. Während der Simulation können die gestellten Anforderungen[1] überwacht werden. Bleiben diese durchgängig erfüllt, ist die Simulation erfolgreich. Werden hingegen Anforderungen verletzt, so muss das System (Hardware oder Software) optimiert werden. Die reale Hardware wird charakterisiert, da Netzlisten oder Power-Modelle durch die Chip-Hersteller in den meisten Fällen nicht für eine Verlustleistungssimulation herausgegeben werden. Durch den Schwerpunkt unserer Arbeit wird der Fokus auf die grau hinterlegten Schritte gelegt.
Abbildung 2 zeigt das Vorgehen zur Charakterisierung. Durch sie werden die realen Hardwarekomponenten des Systems in ihren einzelnen Power-Modi charakterisiert. Dieser Prozess ist essentiell für

[1] Anforderungen in Bezug auf die Leistungsaufnahmen der charakterisierten Komponenten.

die Genauigkeit und Aussagekraft des in der Simulation verwendeten *Power-Modells*. Durch gezielt erstellte *Micro-Benchmarks* werden die Komponenten des Systems in alle benötigten Power-Modi und Auslastungen versetzt. Um Aussagen über die verfügbaren Power-Modi zu erhalten, wird auf Informationen aus den jeweiligen Datenblättern zurückgegriffen. Über ein messtechnisches Verfahren kann dann die Leistungsaufnahme des realen Systems im jeweiligen Modus und der Auslastung bestimmt werden. Hierzu werden Shunt-Widerstände in den Spannungsversorgungen der Komponenten genutzt [12, 6, 3, 8]. Die erhaltenen Daten gehen dann in die Modellcharakterisierung ein. Sind nützliche Daten aus Datenblättern der Komponenten vorhanden, werden diese mit einbezogen. Mit den gesammelten Daten wird dann ein *hardwarespezifisches Power-Modell* erstellt. Es besteht aus einzelnen PSMs [1] für jede abzubildende Komponente. Jeder Power-Modus einer Komponente erhält in dem jeweiligen Automaten einen Zustand. Diese Zustände können mit einer konstanten Verlustleistung oder auch einer parametrisierbaren Verlustleistungsfunktion annotiert werden. Auch die Zustandsübergänge können mit spezifischen Verlustleistungen gekennzeichnet werden. Darüber hinaus in dir Integration eines einfachen Temperaturmodells geplant. Ein Beispiel für ein Power-Modell wird bei der Evaluation in Abschnitt 4 gezeigt.
In Abbildung 3 ist das geplante Vorgehen zur Simulation in einer Übersicht gezeigt. Wie in der Abbildung zu erkennen ist, lassen sich die einzelnen Elemente der vorgeschlagenen Methodik in zwei Gruppen gliedern. Diese werden nun näher erläutert. Wie bereits vorher erwähnt, soll der Optimierungsprozess hier nicht weiter behandelt werden.

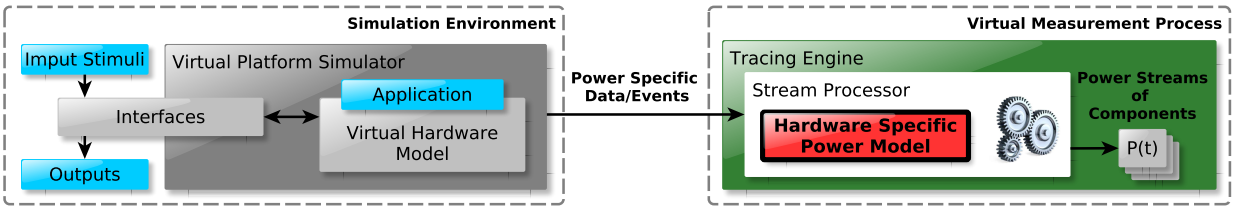


Abbildung 3: Veranschaulichung der Simulationsumgebung und des virtuellen Messprozesses

Simulation Environment: Im Zentrum der Simulationsumgebung steht der eigentliche Simulator, welcher das virtuelle Hardware-Modell samt einer Applikation ausführt. Dabei können über Schnittstellen Eingaben getätigt und Ausgaben erhalten werden. Bei der Applikation soll es sich möglichst um das später im realen System verwendete *Binary* handeln. Daher muss auch das virtuelle Modell der Hardware dem realen System möglichst genau nachempfunden werden. Denn nur so kann eine aussagekräftige Verlustleistungsabschätzung getätigt werden. Der Simulator liefert während der Ausführung die benötigten Daten, um die virtuellen Messungen durchführen zu können. Als VP-Simulator wird OVPSim[2] verwendet werden. Es handelt sich um einen instruktionsakkuraten Simulator. Der Quellcode der virtuellen Modelle ist frei verfügbar. Auf diese Weise können eigene virtuelle Plattformen erstellt und zusätzliche IP-Cores implementiert werden.
Virtual Measurement Process: Der virtuelle Messprozess enthält eine *Tracing Engine*, die die Resultate der Simulation als *Timed-Value-Streams* zur Verfügung stellt. Zum Verarbeiten der *Basis-Streams* wird ein *Stream Processor* verwendet. Dieser verarbeitet ebenfalls die Ausgaben der Simulationsumgebung und leitet sie in das *hardwarespezifische Power-Modell*. Die benötigten Daten der Simulationsumgebung sind bspw. die aktuelle Zeit des simulierten Systems und Events welche Rückschlüsse über die aktuell genutzten Komponenten, ihre Power-Modi sowie deren Auslastung geben. Ergebnis des *Stream Processors* sind einzelne *Power Streams* für jede Komponente, welche während der Simulation betrachtet und überwacht werden können.

[2]Webseite: http://www.ovpworld.org, letzte Ansicht: 16.01.2015

4. Evaluation

Zur Evaluation der vorgestellten Methodik wurde das in Abbildung 4 gezeigte Design auf einem Digilent ATLYS-Board, mit einem Xilinx Spartan 6 FPGA, realisiert. Dieses Design dient vorrangig der Evaluation des Charakterisierungsprozesses der realen Hardware.
In der Ebene *Hardware Architecture* sind insgesamt sechs Xilinx MicroBlaze Prozessoren (SC0 bis SC5) und ein selbst entwickelter *Clock Gating* IP-Core dargestellt. SC0 ist der Master, welcher den *Clock Gating IP-Core* ansteuern kann. Dieser *IP-Core* kann die Softcores SC1 bis SC5 jeweils in drei unterschiedliche Power-Modi versetzen. **Running**: Der Softcore führt sein Programm in normaler Form aus. **Reset**: Der *Clock Gating IP-Core* hält den Softcore im Reset-Zustand. **Clock Gated**: Der *Clock Gating IP-Core* trennt die Taktleitung des Softcores. Der *Clock Gating* IP-Core wird durch die Software S0 (Software-Ebene) vom Master aus gesteuert. Die Software S1 bis S5 der Softcores wurde in unterschiedlichen Konfigurationen getestet. Darunter „Idle“ und „Full Load“. Die Konfiguration aller Softcores unter voller Auslastung wurde in den späteren Experimenten verwendet.

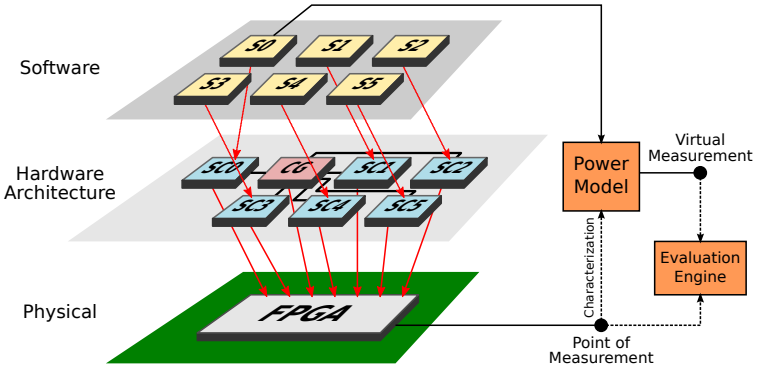


Abbildung 4: Aufbau des Designs der Evaluation

4.1. Charakterisierung

Zur Charakterisierung des Power-Modells wurde die auf dem Digilent ATLYS-Board befindliche Messelektronik verwendet. Sie kann alle vier Board-Spannungen (3.3 V, 2.5 V, 1.8 V und 1.2 V) überwachen. Der dafür verbaute Analog-Digital-Wandler LTC2481C von Linear Technologies (vertikale Auflösung von ca. 2.5 mA) besitzt eine Messfrequenz von lediglich ca. 6.25 Hz, da typisch 160 ms für eine Messung benötigt werden. Bei der Charakterisierung der Power-Modi war darauf zu achten, dass die Zeiträume für die Aktivierung der einzelnen Modi groß genug gewählt wurden, um eine akkurate Messung zu ermöglichen. So wurden die einzelnen Power-Modi mit insgesamt 18 *Micro-Benchmarks* für je 60 Sekunden gehalten und die Werte im Anschluss gemittelt. Die Temperatur des Systems wird in dieser Evaluation nicht berücksichtigt. Vor den Messungen wurde die Hardware ca. eine Stunde im Vorfeld eingeschaltet, um sich aufzuwärmen. So wird ein Driften der Messwerte aufgrund der temperaturabhängigen statischen Verlustleistung (Leakage) nach oben durch den Temperaturanstieg weitestgehend umgangen. Die Änderung der Umgebungstemperatur der Testumgebung[3] im Labor sowie die Temperaturänderungen durch die unterschiedlichen Leistungsaufnahmen treten als Störfaktor auf. Die Ergebnisse sind in Tabelle 1 gezeigt. Bei diesen wurde ausschließlich die 1.2 V-Spannung betrachtet, da dies die interne Versorgungsspannung des FPGAs für die verwendete Logik ist.
Für die Grundlast (*Base*) wurden alle Komponenten des Designs in den Power-Modus mit der jeweils geringsten Leistungsaufnahme versetzt. Der Master ist im Zustand „Reset“, während die Slaves im Zustand „Clock Gated“ sind. Zur Charakterisierung der Power-Modi des Masters wurden alle Slaves im Zustand „Clock Gated“ gehalten, während der Zustand des Masters angepasst wurde. Für die Messungen in den Power-Modi der Slaves hat der Master die Softcores einzeln in die jeweiligen Modi versetzt, während die weiteren Slaves im Zustand „Clock Gated“ gehalten wurden. Durch Subtraktion der Grundlast und dem Zustand „Run“ des Masters wurden die einzelnen Werte der Slaves berechnet.

[3] Umgebungstemperatur hier bei 24 °C als fix angenommen.

Core	Clock Gated [mA]/[mW]	Reset [mA]/[mW]	Idle (Running) [mA]/[mW]	100% (Running) [mA]/[mW]
Base		327 / 392		
SC0	-	0 / 0	19 / 23	150 / 180
SC1	0 / 0	19 / 23	47 / 56	85 / 102
SC2	0 / 0	18 / 22	44 / 53	63 / 77
SC3	0 / 0	17 / 21	45 / 54	64 / 78
SC4	0 / 0	19 / 23	46 / 55	66 / 79
SC5	0 / 0	18 / 22	45 / 54	64 / 78

Tabelle 1: Charakteristische Daten des Designs

4.2. Analyse der Charakterisierung

Es fällt auf, dass die Grundlast (*Base*) einen großen Teil der maximal möglichen Leistungsaufnahme ausmacht. Dies steht in Zusammenhang damit, dass in diesem Wert die statischen Leistungsaufnahmen des gesamten FPGAs sowie die verbleibenden Grundlasten aller Komponenten enthalten sind. Da die Taktleitung des Masters nicht getrennt werden kann, ist der Power-Modus „Clock Gated“ nicht vorhanden. Weiterhin besitzt der Master eine andere Hardware-Konfiguration als die Slaves. Dies erklärt die signifikanten Unterschiede bei den Modi „Idle“ und „100%“. Die Slaves hingegen besitzen die gleiche Hardware-Konfiguration. Slave SC1 zeigt sich hier im Modus „100%“ mit ca. 1/3 mehr Leistungsaufnahme. Bei Betrachtung des *Floorplans* des Designs (hier nicht abgebildet) fällt auf, dass SC1 eine weitläufigere Verteilung über die Fläche des FPGAs zeigt. So müssen durch längere Verbindungen größere Kapazitäten geschaltet werden, was zu einer größeren dynamischen Verlustleistung führt, was diesen Umstand erklären könnte. Dieses Verhalten führt ebenfalls zu dem Schluss, dass für eine akkurate Simulation der Leistungsaufnahme von FPGAs jeder Bitstream ein eigenes Power-Modell erhalten muss. Denn die Lage der einzelnen Komponenten im *Floorplan* des FPGAs sowie dessen Konfiguration haben einen großen Einfluss auf dessen Leistungsaufnahme. Die weiteren Daten in der Tabelle zeigen das erwartete Verhalten. Je niedriger die Aktivität in den einzelnen Power-Modi, desto niedriger wird auch die dynamische Verlustleistung.

4.3. Power-Modell

Mit Hilfe der charakterisierten Daten wurde ein Power-Modell für das vorliegende Design erstellt. Aufgrund der Tatsache, dass die Softcores nur mit je zwei unterschiedlich auslastenden *Micro-Benchmarks* getestet wurden, sind nicht genügend Daten für eine ausreichend akkurate Regression vorhanden, um den Power-Mode „Running“ mit einer dynamischen Funktion zu annotieren. Aus diesem Grund wurde das Power-Modell sowie die folgenden Experimente für eine 100%ige Auslastung der Softcores ausgelegt. Das erstellte Power-Modell ist in Abbildung 5 dargestellt.

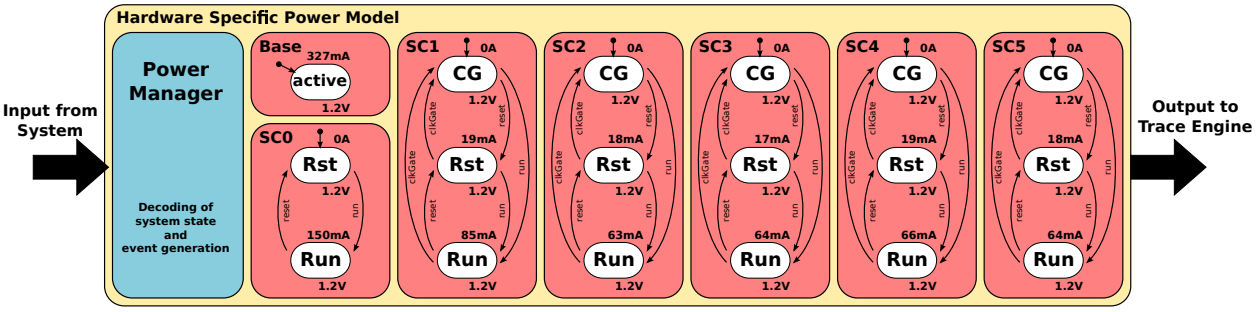
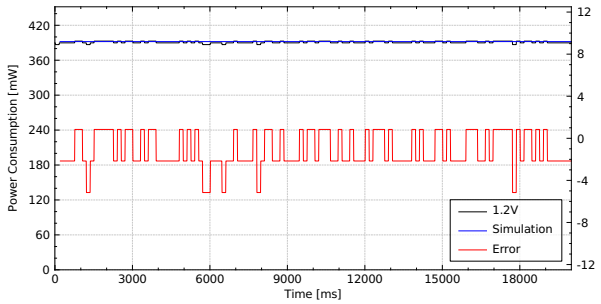
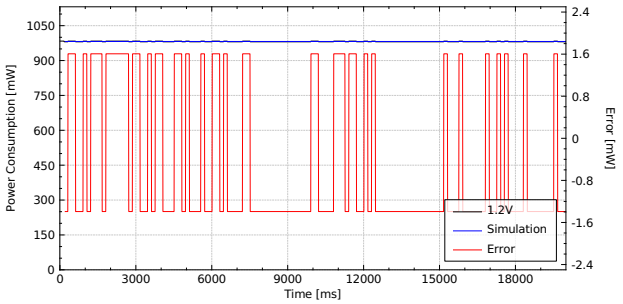


Abbildung 5: Power-Modell des FPGA-Designs

Wie zu erkennen, besteht es aus einem Power-Manager, welcher den Zustand des Systems dekodiert und an insgesamt sieben PSMs weiterleitet: eine Base-PSM und sechs PSMs für jeden Softcore. Die

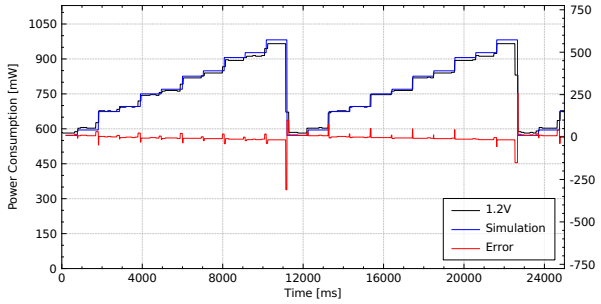


(a) Experiment 1: SC0 reset, SC1-SC5 „clock gated“

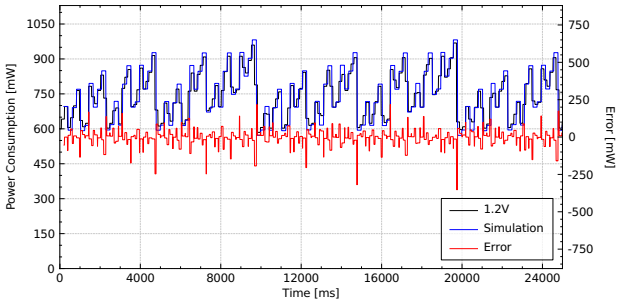


(b) Experiment 2: SC0-SC5 „running“

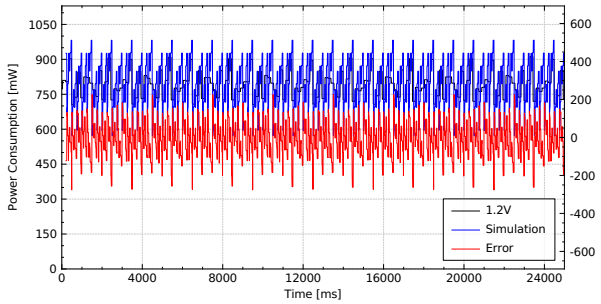
Abbildung 6: Evaluation der statischen Power-Modi



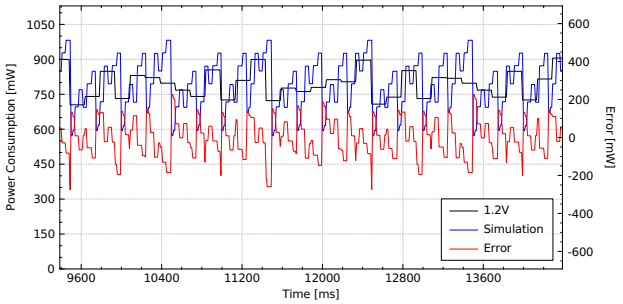
(a) Experiment 3



(b) Experiment 4



(c) Experiment 5



(d) Experiments 5 (Ausschnitt)

Abbildung 7: Evaluation der dynamischen Power-Modi

Zustände der einzelnen PSMs wurden mit den Werten aus Tabelle 1 annotiert. Es ist zu sehen, dass hier die Werte für die Ströme und zusätzlich die Spannungen annotiert wurden. Diese Entscheidung bietet für spätere Charakterisierungen von Hardware, welche bspw. *Dynamic Voltage and Frequency Scaling* verwendet, mehr Flexibilität bei Änderungen der Modi durch Annotationen der Zustände mit Funktionen. Der Power-Manager erhält über eine serielle Schnittstelle den jeweils aktuellen Zustand des Systems, dekodiert diesen und generiert die notwendigen Events für die PSMs. Deren annotierte Outputs werden an eine *Tracing Engine* weitergeleitet, die die Outputs verarbeitet und an eine grafische Benutzungsoberfläche gibt. Die erhaltenen Daten werden als Graphen zur Analyse dargestellt.

4.4. Experimente

Im Anschluss wurden mehrere Experimente durchgeführt, um das entwickelte Verfahren einer ersten Evaluation zu unterziehen. In den Abbildungen 6a bis 7d sind sechs durchgeführte Experimente gezeigt. Die ersten Graphen zeigen statische Zustände in Ausschnitten von jeweils 20 Sekunden. Darauf folgen weitere Visualisierungen, welche unterschiedliche Programme zum Wechseln der Power-Modi abfahren. Der Testaufbau für die Experimente ist ebenfalls in Abbildung 4 zu sehen. Während

ISBN 978-3-00-048889-4

der Master des Systems durchgängig den Status der Komponenten an das Power-Modell weitergibt, erhält die *Evaluation Engine* zusätzlich die aktuellen Messwerte des Analog-Digital-Wandlers des Digilent ATLYS Boards und vergleicht diese mit der Simulation des Power-Modells. Alle Experimente wurden unter der Annahme gleichbleibender Umgebungstemperatur durchgeführt. Der *schwarze* Graph zeigt die realen Messwerte der Versorgungsspannung, der *blaue* Graph zeigt die simulierten Werte des Power-Modells und der *rote* Graph zeigt den Fehler (Differenz) zwischen Messwerten und Simulation. Mit den Experimenten der statischen Power-Modi des Systems kann die Langzeitstabilität des Verfahrens veranschaulicht und evaluiert werden. Dabei wurden die folgenden Konfigurationen der Power-Modi durch speziell angepasste *Micro-Benchmarks* für jedes Experiment vorgenommen:
Experiment 1 (Abbildung 6a): Der Master wurde in den Reset-Zustand versetzt, nachdem die Slaves in den Zustand „Clock Gated“ versetzt wurden. Dieser Zustand stellt die Grundlast des Systems dar.
Experiment 2 (Abbildung 6b): Der Master und alle Slaves befinden sich im Power-Modus „Runnig“. Dieser Zustand des Systems ist jener mit der größten gemessenen Leistungsaufnahme.
Für die dynamischen Wechsel zwischen den einzelnen Power-Modi wurde eine spezielle Firmware geschrieben. Der Master steuert dabei das System über den *Clock Gating IP-Core*, so dass beliebige Profile abgefahren werden können. Mit den dynamischen Wechsel der Power-Modi können die Zustandsänderungen zwischen diesen evaluiert werden.
Experiment 3 (Abbildung 7a): Der Master schaltet jede Sekunde einen Slave in den nächst höheren Modus. Sind alle Slaves im höchsten Power-Modus, werden alle Slaves wieder in „Reset“ versetzt.
Experiment 4 (Abbildung 7b): Zum Umschalten der Power-Modi der Slaves wendet SC0 „binäres Zählen“ an: SC1 wird alle 200 ms in den nächsten Power-Modus geschaltet, SC2 alle 400 ms, SC3 alle 800 ms, SC4 alle 1600 ms und SC5 alle 3200 ms.
Experiment 5 (Abbildung 7c und 7d): Es handelt sich um das Experiment 4 in beschleunigter Form. Alle Slaves schalten 10-mal schneller als in Experiment 4 um. Hier zeigt sich die geringe zeitlichen Auflösung der verbauten Messeinrichtung. In Abbildung 7d ist ein Ausschnitt von 5 Sekunden gezeigt. Er verdeutlicht, dass Messkurve und Simulationskurve nicht mehr deckungsgleich übereinanderliegen.

4.5. Analyse der Experimente

In Tabelle 2 werden die Ergebnisse der Experimente gezeigt. Besonderes Augenmerk für die Evaluation liegt hierbei auf den letzten vier Zeilen der Tabelle, die Informationen über den Fehler zwischen Messungen und Simulationen geben. Der Fehler wird durch den Mittelwert des „Error“-Graphen gebildet. Dieses Vorgehen ist an dieser Stelle legitim, da sich so positive und negative Fehler relativieren. Sie treten, wie später erwähnt, direkt hintereinander und charakteristisch bei Zustandsübergängen auf.

Experiment	**1**	**2**	**3**	**4**	**5**
Art	statisch		dynamisch		
Dauer [s]	20	20	25	25	25
Durchschnitt [mW]	390.878	981.576	758.543	738.152	791.233
Reales System [mWh]	2.17263	5.45320	5.26766	5.12811	5.49468
Simulation [mWh]	2.17887	5.45556	5.29636	5.14324	5.51482
Fehler [mWh]	0.00623	0.00235	0.02871	0.01514	0.02015
Fehler [%]	0.287	0.043	0.545	0.295	0.36654
Max. Fehler neg. [mW]	5.2	1.4	310.2	352.4	262.2
Max. Fehler pos. [mW]	0.9	1.6	258.6	216.7	225.2

Tabelle 2: Ergebnisse der Experimente am FPGA-Design

Bei den statischen Experimenten des Systems zeigen sich Fehler zwischen den Messungen und den Simulationen, die sich lediglich in einem sehr schmalen Band ober- und unterhalb der Simulationskurve bewegen. So ergibt sich eine sehr genaue Simulation von diesen Zuständen, welche hier mit einem relativen Fehler über die Zeit von maximal ca. 0.3 % aufwarten. Diese Genauigkeit lässt sich durch die Charakterisierung des Systems mit statischen Power-Modi erklären. Auch weitere durchgeführte,

allerdings hier nicht präsentierte, statische Experimente in weiteren Power-Modi zeigten Ergebnisse gleicher Genauigkeit gegenüber des verwendeten Messsystems. Ein bereits erwähnter Störfaktor bleibt die Umgebungstemperatur des realen Systems. Stimmt die Umgebungstemperatur während der Experimente nicht mit jener überein, welche zum Zeitpunkt der Charakterisierung des Systems vorherrschte, so ergeben sich teils starke Fehler. Bei einer $5\,^{\circ}\mathrm{C}$ niedrigeren Umgebungstemperatur liegt ein Fehler im Bereich von ca. $2.2\,\%$ vor, da der FPGA ca. $25\,\mathrm{mW}$ weniger Leistungsaufnahme benötigt. Aus diesem Grund wurde bei der Durchführung darauf geachtet die Umgebungstemperatur bei der Charakterisierung des Systems einzustellen. Um dieser Tatsache zu begegnen müsste die temperaturabhängige statische Verlustleistung in den Power-Modellen mit in Betracht gezogen werden.
Betrachtet man die Werte der Experimente 3 und 4 mit dynamischen Zustandswechseln, dann fallen die extrem großen maximalen Fehler auf. In Abbildung 7a und 7b ist zu erkennen, dass diese Fehler insbesondere bei den Zustandsübergängen auftreten. Dies steht im Zusammenhang damit, dass die Simulation ein Rechtecksignal erzeugt, während in der Realität an- und absteigende Flanken gemessen werden. Dies führt häufig zu Fehlern mit einem positiven und einem negativen Anteil während eines Wechsels zwischen zwei Power-Modi. Stimmt die zeitliche Synchronisation zwischen realen Messwerten und Simulation nicht genau überein, so verlagert sich der Fehler in die positive, bzw. negative Richtung. Dies wird bspw. durch die niedrige zeitliche Auflösung den verwendeten Analog-Digital-Wandlers auf dem Digilent ATLYS Board hervorgerufen. Dennoch ist in den Graphen gut zu erkennen, dass die Simulationskurven akkurat und mit nur geringen Fehlern den realen Messwerten in den unterschiedlichen Power-Modi folgen. Hierfür sprechen auch die kleinen relativen Fehler, von ca. $0.55\,\%$ (Experiment 3) und $0.3\,\%$ (Experiment 4). Auch bei längeren Experimenten über mehrere Minuten blieben die relativen Fehler in dem hier angegebenen Rahmen.
In Experiment 5 wurde das Experiment 4 mit 10-facher Geschwindigkeit durchgeführt. Anhand von Abbildung 7c und 7d ist eindeutig zu sehen, dass hier die Grenze der zeitlichen Auflösung des Analog-Digital-Wandlers überschritten ist und die Messwerte nur noch sporadisch und sehr abstrakt mit der Form der Simulationskurve übereinstimmen. Doch betrachtet man auch hier den relativen Fehler, fällt dieser vergleichsweise mit $0.37\,\%$ sehr gering aus. Trotz der durchgängig präsenten positiven und negativen Fehler weichen die Kurven im Mittel über die Zeit betrachtet nur minimal voneinander ab.

5. Fazit und Ausblick

In dieser Arbeit wurde ein Verfahren zur messbasierten Powercharakterisierung eines Multi-Core FPGA-Designs vorgestellt. Aus der Charakterisierung wurde ein einfaches PSM Modell erstellt, welches in eine Evaluationsumgebung integriert werden konnte. Die durchgeführte Evaluation hat ergeben, dass das erzeugte PSM Modell, innerhalb des physikalischen Messbereichs eine sehr gute Genauigkeit liefert. Dadurch das reale Hardware für eine VP charakterisiert wird geht ein Vorteil der VP verloren, die parallele Entwicklung von Hardware und Software, aber es werden auch wichtige Punkte gewonnen. So wird es dem Entwickler möglich, viel detailierter und nichtinvasiv in die Zusammensetzung der Leistungsaufnahme durch die einzelnen Komponenten hineinzuschauen. Ein Bezug zwischen den Funktionen der Applikation und ihrer Leistungsaufnahme kann hierbei ebenfalls hergestellt werden. Bei realer Hardware ist dies nur mit großem Aufwand mit invasiven Methoden möglich. Wir kommen damit zu dem Fazit, dass das vorgestellte Verfahren grundsätzlich geeignet ist.
Allerdings wurden in dieser Arbeit nur synthetische Benchmarks benutzt, was die Aussagekraft zunächst einmal einschränkt. In der nächsten Evaluation sollen Programme aus einer Benchmark Suite mit dem vorgestellten Verfahren getestet werden. Das beinhaltet auch die Verwendung komplexerer Software-Stacks mit den Betriebssystemen FreeRTOS und Linux. Außerdem wurde die temperaturabhängige statische Verlustleistung bisher nicht betrachtet. Hierzu soll das vorgestellte PSM Modell um eine temperaturabhängige Komponente erweitert werden. Damit sollen dann die Umgebungstemperatur und dessen Veränderung (z.B. durch aktive Kühler) sowie die durch die dynamische Verlustleistung

ISBN 978-3-00-048889-4

induzierte Temperaturveränderung im inneren des Chips in dem Modell repräsentiert werden.
Weiterhin müssen zum Charakterisieren und Simulieren von MPSoCs weitere Punkte erarbeitet werden. Die On-Chip-Kommunikation verursacht innerhalb des Systems eine große Leistungsaufnahme. Hier müssen Wege zur Charakterisierung von Kommunikationskanälen gefunden und erarbeitet werden. Ebenso muss eine geeignete Anbindung von *Power-Modellen* an VPs entwickelt werden.

Literatur

[1] Benini, Luca, Robin Hodgson und Polly Siegel: *System-level Power Estimation and Optimization*. In: *Proceedings of the 1998 International Symposium on Low Power Electronics and Design*, ISLPED '98, Seiten 173–178, New York, NY, USA, 1998. ACM, ISBN 1-58113-059-7.

[2] Esmaeilzadeh, Hadi, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam und Doug Burger: *Dark Silicon and the End of Multicore Scaling*. In: *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, Seiten 365–376, New York, NY, USA, 2011. ACM, ISBN 978-1-4503-0472-6.

[3] Joseph, Russ und Margaret Martonosi: *Run-time Power Estimation in High Performance Microprocessors*. In: *Proceedings of the 2001 International Symposium on Low Power Electronics and Design*, ISLPED '01, Seiten 135–140, New York, NY, USA, 2001. ACM, ISBN 1-58113-371-5.

[4] Lam, Tak Kei, Steve Yang, Wai Chung Tang und Yu Liang Wu: *Logic Synthesis for Low Power Using Clock Gating and Rewiring*. In: *Proceedings of the 20th Symposium on Great Lakes Symposium on VLSI*, GLSVLSI '10, Seiten 179–184, New York, NY, USA, 2010. ACM, ISBN 978-1-4503-0012-4.

[5] Le Sueur, Etienne und Gernot Heiser: *Dynamic Voltage and Frequency Scaling: The Laws of Diminishing Returns*. In: *Proceedings of the 2010 International Conference on Power Aware Computing and Systems*, HotPower'10, Seiten 1–8, Berkeley, CA, USA, 2010. USENIX Association.

[6] Mesa-Martinez, Francisco J., Michael Brown, Joseph Nayfach-Battilana und Jose Renau: *Measuring Performance, Power, and Temperature from Real Processors*. In: *Proceedings of the 2007 Workshop on Experimental Computer Science*, ExpCS '07, New York, NY, USA, 2007. ACM, ISBN 978-1-59593-751-3.

[7] Rethinagiri, Santhosh Kumar, Rabie Ben Atitallah, Jean Luc Dekeyser, Eric Senn und Smail Niar: *An Efficient Power Estimation Methodology for Complex RISC Processor-based Platforms*. In: *Proceedings of the Great Lakes Symposium on VLSI*, GLSVLSI '12, Seiten 239–244, New York, NY, USA, 2012. ACM, ISBN 978-1-4503-1244-8.

[8] Rethinagiri, Santhosh Kumar, Oscar Palomar, Rabie Ben Atitallah, Smail Niar, Osman Unsal und Adrian Cristal Kestelman: *System-level Power Estimation Tool for Embedded Processor Based Platforms*. In: *Proceedings of the 6th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, RAPIDO '14, Seiten 5:1–5:8, New York, NY, USA, 2014. ACM, ISBN 978-1-4503-2471-7.

[9] Rosales, Rafael, Michael Glass, Jürgen Teich, Bo Wang, Yang Xu und Ralph Hasholzner: *MAESTRO - Holistic Actor-Oriented Modeling of Nonfunctional Properties and Firmware Behavior for MPSoCs*. ACM Trans. Des. Autom. Electron. Syst., 19(3):23:1–23:26, Juni 2014, ISSN 1084-4309.

[10] Shin, Youngsoo, Jun Seomun, Kyu Myung Choi und Takayasu Sakurai: *Power Gating: Circuits, Design Methodologies, and Best Practice for Standard-cell VLSI Designs*. ACM Trans. Des. Autom. Electron. Syst., 15(4):28:1–28:37, Oktober 2010, ISSN 1084-4309.

[11] Trabelsi, Chiraz, Rabie Ben Atitallah, Samy Meftali, Jean Luc Dekeyser und Abderrazek Jemai: *A Model-Driven Approach for Hybrid Power Estimation in Embedded Systems Design*. EURASIP Journal on Embedded Systems, 2011(1):569031, 2011, ISSN 1687-3963.

[12] Viredaz, Marc A., Deborah A. Wallach, Marc A. Viredaz und Deborah A. Wallach: *Power Evaluation of Itsy Version 2.4*. Technischer Bericht TN-59, Compaq Western Research Laboratory, 2001.

Modeling Power Consumption for Design of Power- and Noise-Aware AMS Circuits

Xiao Pan, Javier Moreno, Christoph Grimm
Design of Cyber-Physical Systems
Technische Universität Kaiserslautern
{pan,moreno,grimm}@cs.uni-kl.de

Abstract

Power estimation and noise performance analysis are very important in designing low power and robust embedded system devices. A particular challenge is to analyze potential disturbances in the power distribution grid, e.g. due to crosstalk or ground bounce. We propose an extended power state machine that models power consumption of components that are considered as "aggressors" at a high level of abstraction. For this purpose, we model power consumption in power states by a statistical model, and in state-changes by a transfer function. As such kind of power modeling lacks experiences, focus of the paper is on experimental validation of this modeling approach.

1. Introduction

The design of Analog/Mixed Signal (AMS) systems has grown significantly in the recent years. The explosion of the wireless embedded systems market leads to an extraordinary challenge in the electronics integration technology. Wireless devices must be small, consuming very low power, be energy autonomous and integrate all possible kind of sensors and actuators to interact with the environment. This results into integrated circuits (ICs) with a digital part, an RF part for wireless connectivity and microelectromechanical systems (MEMS), all included in the same die.
The coexistence of digital and analogue systems in the same chip involves risks and pitfalls. Analogue subsystems can be very sensitive to signal integrity problems. Cross-talk from the digital part may interfere with the analogue signals and compromise the correct operation of the system. In order to enable the verification of this kind of issues and detect any critical problem before manufacturing, the simulation of both the analogue and the digital parts is required.
However, joint simulation of analogue and digital subsystems is a challenging task. Circuit-level simulators permit the evaluation of signal integrity problems. However, simulation performance is insufficient to deal with the complexity of the systems that are being designed nowadays. Using system-level models, the non-critical parts of the system can be abstracted, improving simulation performance.
Energy and power aware design involves in particular cross-talk and signal integrity problems. Energy and power awareness has become a critical design question, in order to fulfill the requirements for energy autonomy. Chip architectures implement separated power domains, where the different

ISBN 978-3-00-048889-4

functional parts of the system have independent power supplies, which can be switched on and off depending on the system's needs.
The switching between different power modes involves fluctuations in the power supply. If these fluctuations or their harmonics incidentally fall within the analogue system operational bandwidth, they can put the system functionality at risk. It is therefore required to model this kind of effects at one hand accurately, and at the other hand at a high level of abstraction
In this paper, an approach to model power consumption at the system level is proposed. It is based on an extension of the power state machine, which was widely used to estimate energy consumption. However, instead of just including the average power consumption value, which is sufficient to estimate energy consumption, the extended model must characterize the instant variation of power in order to identify the spectral characteristics that may endanger the operation of the analogue subsystems.

2. Related Work

Modeling/simulation of the coupling effects in mixed-signal circuits has a long history. Verghese [VAM93] proposed a modification to SPICE to efficiently and accurately model coupling effects in mixed-signal ICs. However, complexity of mixed-signal ICs has increased extremely since then, and newer techniques are required in order to achieve efficient modeling and simulation.
When SPICE-based simulation became unfeasible, higher level methods were developed, such as the one proposed in [vHBD$^+$00], which uses VHDL to extract switching events. A similar approach is followed in [SJK$^+$11], which uses an RTL power estimator in FPGA-based system design.
Finite State Machines (FSMs) have been widely used to model power consumption at the system level in digital systems. The system-level power state machine was first proposed in [BHS98]. Many tools have successfully applied this method to estimate power and energy consumption during the design cycle, such as IDEA1 [DMN10] and PowerDept [HLF$^+$11]. These power state machines assume average power consumption values as constant within the state duration. Average power consumption values can be used to estimate energy consumption, but they are not sufficient to analyze cross-talk and ground-bounce. In these effects, a block ("aggressor") produces noise in operation, and power variations in particular during power state transitions. These disturbances are propagated via the power supply rails to all blocks of a system. In each block they modify its properties, depending on its sensitivity to power supply variations.
Concerning the power modeling of AMS systems, the discrete model based on state machines and power averages is not applicable. Circuit-level power models are often used for simple analog circuits [LG02] [BCS11]. Casinovi [CY03] presents a probabilistic power estimation technique for switched-capacitor circuits that is applicable for circuit level power estimation. Analytic power estimators are an alternative solution. Formulas for power estimation can be obtained for individual functional blocks. For instance, the power estimation of an analog digital converter (ADC) is demonstrated in [Mur08], [SMS09]. However, complex systems demand models with higher simulation performance. Furthermore, in the circuit-level approach, the designer needs to have access to the hardware architecture and even to the processing technology, which are usually not given by the IP vendors.
Therefore, we extend the general finite state machine approach to model instant power variations that may cause cross-talk or signal integrity issues in an abstract way.

ISBN 978-3-00-048889-4

3. Extended Power State Machine (x-PMS)

AMS circuits consist of a variety of very different components, among them at a high level in particular microprocessors, receivers, transmitters, PLL, DAC, ADC, and PLL. A single power model for all such components would not provide realistic and accurate estimations. For this purpose, we aim at capturing the major components of power consumption, and model them abstractly as follows:

1. A probabilistic model rep. colored noise in the frequency domain, describing the small signal behavior. This captures particular RF behavior of aggressors that introduce spurious tones into sensitive components.

2. Low-pass behavior specified by a transfer function. This captures the large signal behavior, e.g. due to charging or discharging capacities etc. when starting a subsystem.

3. A function of input and output signals that describes, e.g. power consumption of class B amplifiers or PWM.

We introduce the first two kinds of behaviors into the Power State Machine (PSM) model and validate the modeling approach by measurements.

3.1. Power State Machine (PSM-) Model

The behavior of components can be modeled at a high level by operation modes that are switched by external inputs. This can be specified by a finite state machine with n states $s_i, i \in \{1, \dots n\}$. In the PSM-model, the power consumption of each state is described by its average value P_i. Usually, the average power consumption at specific operation states (power states) can be taken from data sheets. Assuming a PSM model, the power consumption at time t is the power at state s_n of PSM would be estimated as constant value

$$P(t) = P_{avg}(s_n) = \Delta E(s_n)/\Delta t(s_n)$$

which is not true, as power consumption is highly variant, and not – as suggested above – piecewise constant with changes only when power states change. However, it is appropriate to estimate energy consumption.
Accounting state transitions is another issue. Depending on the abstraction level, many approaches haven been proposed in the recent years. For peripheral devices modeled with registers (e.g. UART), state transition can be detected by monitoring the contents in the control registers. For the micro-controller modeled at the instruction-level, counting instructions provides enough granularity [MHG10]. State transition for Black-Box IPs can be determined by detecting these activities though the signal that the components exchange [CTJ11], or observed by events generated through Protocols (PrPSM) [LHGN13].

3.2. Extensions to allow estimation of instantaneous power consumption

Extension 1: Large signal model of power consumption Advancements in energy aware and ultra-low power design have led to very aggressive power management techniques, with different

power domains that are switched on and off by power gating. Power consumption during state transitions does not change instantaneously. It requires a settling phase to reach a more or less steady state. Often, there are delays, boosting/overshoots, oscillation, glitches or spikes, over the power signals. They are due to the linear behavior of capacities to be loaded and the inductance of wires. In worst cases, there is a risk of suffering a brown-out. However, even very small disturbances can have critical effects.
To capture the large signal behavior of the power consumption at state-changes of the PSM, we assume that delays, overshooting and small oscillations over the power signal can be generally specified by a transfer function $H(s)$. Then the power consumption is (assuming that multiplication of $H(s)$ with a function in time domain is implemented as in most modeling languages):

$$P(t) = H(s) \cdot P_{avg}(s_n) = H(s) \cdot I_{in}(t) \cdot U$$

Note, that $P_{avg}(s_n)$ is a stepwise constant function that changes only when power state change. Measuring the step response of the electric current on the examined component is the simplest way to identify the model, which can be extracted from low level (e.g. RTL) simulations (bottom-up) or measurements on the real hardware (top-down).

Extension 2: Small signal (noise-) model of power consumption The power behavior of the AMS circuit is highly dependent on its functionality and internal design. The design of often not known or would introduce a complexity that is inappropriate for system-level modeling.
Therefore, we use a probabilistic approach, assuming that power consumption is a stationary signal when a system is in a power state. We assume that power consumption is colored noise, resp. a random variable (correlated to previous values) in time domain. Furthermore, we assume that the parameters of the noise resp. random process depend on (non-random) parameters, such as operation frequency, supply voltage or input signals that should not be changed for the purpose of characterization of a component.

$$P(s) = H_{noise}(s) \Rightarrow P(t) = H_{noise}(s) \cdot \text{white noise(t)}$$

The noise on the power supply rails $H_{noise}(s)$ of a component can be determined easily via laboratory experiments and measurements, or simulation at circuit-level. Its spectral properties are constant as we assume stationary signals. To determine – e.g. for system simulation – white noise in time domain can be generated and filtered with $H(s)$; again we assume for convenience that the multiplication $\cdot$ of a transfer function with a signal in time domain is implemented as in most simulators.

Extension 3: Modulation from an output or internal signal In most analog circuits power consumption is a function of a signal $s(t)$ that has particular impact, e.g. for amplifiers the power consumption is mostly a function of its output voltage. This can be modeled by a function $f : \{s(t)\} \rightarrow \{P(t)\}$, hence:

$$P(t) = f(s(t))$$

The function $f(t)$ and the specific signal $s(t)$ are often known by a designer based on the known structure of a circuits. For example, the power consumption of an amplified can be modeled as $P(t) = out(t)^2 \cdot const_1 + const_2$, where the constants must be determined by characterization.

ISBN 978-3-00-048889-4

The instantaneous power consumption in a power state is then modeled as a linear combination of all three extensions:

$$P(t) = H(s) \cdot P_{avg}(s_n) + H_{noise}(s) \cdot \text{white noise(t)} + f(s(t))$$

4. Validation of the modeling approach by a temperature monitoring system

To validate the xPSM modeling approach, we use an embedded application for temperature measurement using our BAConLab hardware framework (Building Automation and Control Laboratory) . The hardware components are integrated in two separate PCB boards: the MCU board that consists of a micro-controller, and the sensor board that has an ADC with I2C communication interface and a temperature sensor.
Figure 1 shows the overall architecture of the temperature monitoring system.

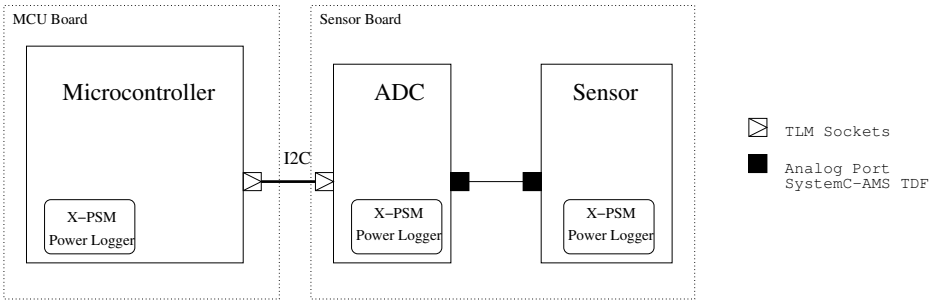


Figure 1: Architecture of temperature measurement system.

The monitoring system measures the environment temperature within certain period. The operation works as described in Figure 3:

1. The hardware sends, after powered on, a “start conversion command” to the ADC via I2C.
2. It waits until conversion is done.
3. It reads the converted data from ADC.

The two PCB boards are powered by a high-performance programmable power supply. The voltage of each power line is assumed to be constant according to the specification.

4.1. Power Measurements from Hardware

The power consumption is measured using a Hall-effect current probe and recorded using an oscilloscope. Table 1 lists the equipment and the ICs used in the experiments.
Power models of the demo system are explored from the measurements of the real hardware (Top-Down). The measurements provide the power consumption profiles for each hardware subsystem. In the experiment, we wrap a few wires around the current probe to increase the accuracy and run multiple times to reduce experiment error. The noise coming from the power supply and the oscilloscope is considered to be included in the power models as the extrinsic noise, which is inevitable in the real application.

ISBN 978-3-00-048889-4

Table 1: Experiment's hardware

Equipment/IC	Model	Specifications/Settings
Oscilloscope	Tektronic MOS5034	Sampling Rate = 500kS/s
Current Probe	Tektronic TCP0030A	Accurately = 1mA
DC Power Supply	Hameg HMP2030	Output Voltage/Power = 3.3V/500mW
MCU Board	Micro-Controller	ATSAM3S
Sensor Board	ADC	MCP3424
	Temp-Sensor	MCP9700A

Power profiles (in measured current, as voltage is constant) of our experiment is shown in Figure 2. The programmable power supply in our experiment requires a start-up time to provide stable output voltage (about 30ms in the figure). The large oscillations during this period also come from the power source. However, modeling the programmable power supply is not to be discussed in this work.

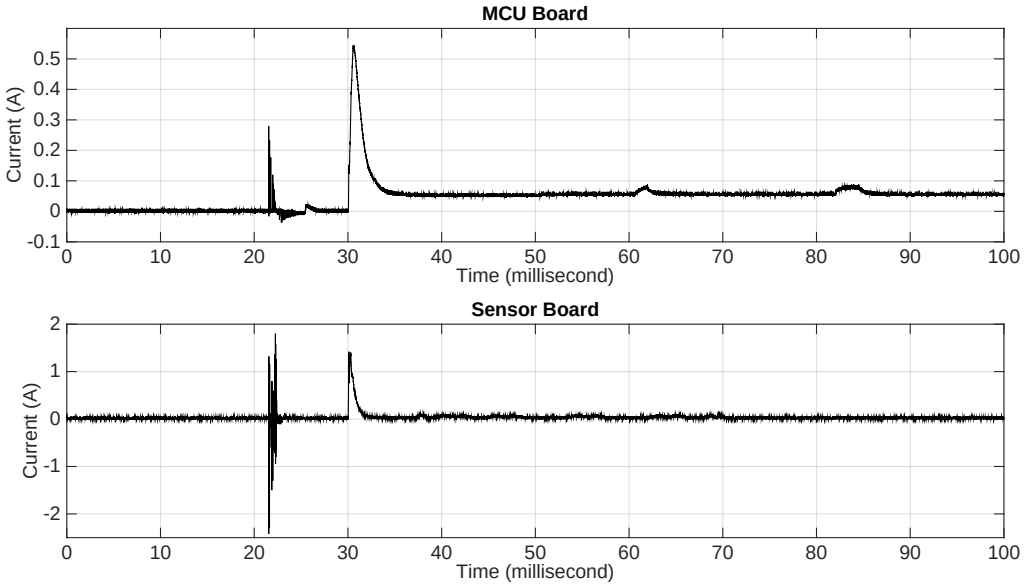


Figure 2: Measured current of MCU (top) and sensor board (bottom).

4.2. Modeling with Extended Power State Machine

During the system activities of TX/RX, both the MCU and I2C are working together, but the power consumption does not significantly vary. The MCU in the experiment system has four operation modes and therefore four power states are needed in the PSM model, as shown in Figure 3 (right). Figure 4 gives an example of power estimation using PSM model based on the use-case scenario.

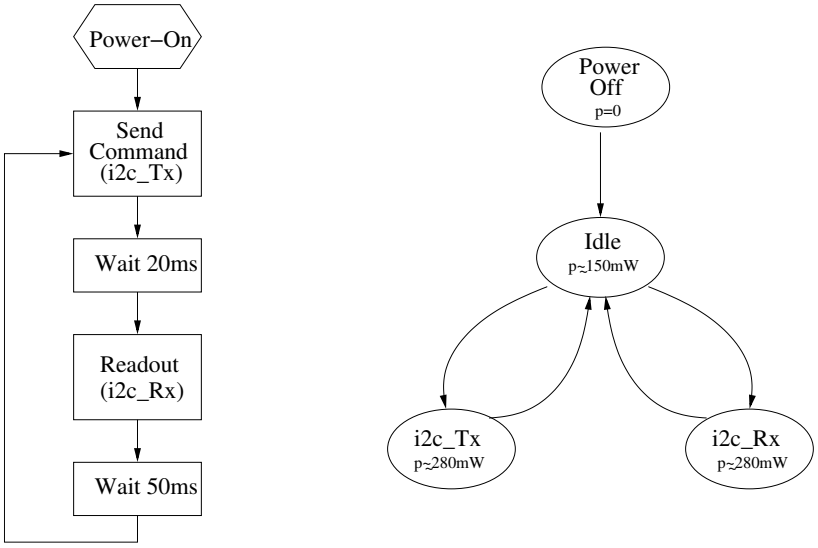


Figure 3: System activities flowchart (left) and MCU power state machine (right)

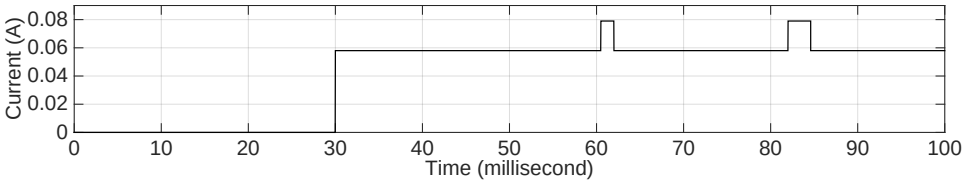


Figure 4: Estimated current using PSM model (MCU board).

Large Signal Model of Power Consumption We first evaluate the approach for modeling large signal behavior of power consumption caused by state transitions. The inputs are the step signals obtained from the PSM-model of the demo system shown in Figure 4. Therefore, the model parameters are identified from the characteristics of the time domain step response that we logged from the experiment. The power consumption during state transitions is modeled using first-order and second order band-pass models.
The large signal models used to model the power transitions are given in Equation 1 for the first order model, and Equation 2 for the second order model. The parameters and the corresponding values used for evaluation in the use-case are listed in Table 2.

$$H_s = H_{(1st)} \frac{1}{T_s \cdot s + 1} \tag{1}$$

$$H_s = H_{(2nd)} \frac{2\zeta\omega s}{s^2 + 2\zeta\omega s + \omega_2} \tag{2}$$

Small Signal (Noise) Model of Power Consumption The noise on the power line is considered a stationary process that produces colored noise. To characterize the model, a spectrum analyzer can be used to capture, in particular, RF behavior; however, for simplicity and proof-of concept we employ the transfer functions from the large signal model for state transitions and $H(s) = 1$ for during a power state, multiply by white noise as the small signal model.

Table 2: Parameters of the transfer function from characterization by measurement.

Parameter Name	Adjustment	PowerOn State Transition	TX/RX State Transition
$H_{(1st)}$	peak value	1	1
T_s	convergence speed	0.00063	0.00047
$H_{(2nd)}$	peak value	11.7	-
ζ	oscillates	1.05	-
ω	convergence speed	2000	-

4.3. Results and discussion

Figure 5 shows a comparison of the estimated power using the large signal model with the measurements. The comparison shows very clearly that the large signal modeling approach can simulate the realistic behavior of power consumption at the system level. The small deviations that are visible are mostly due to the power supply components that are not part of the model.

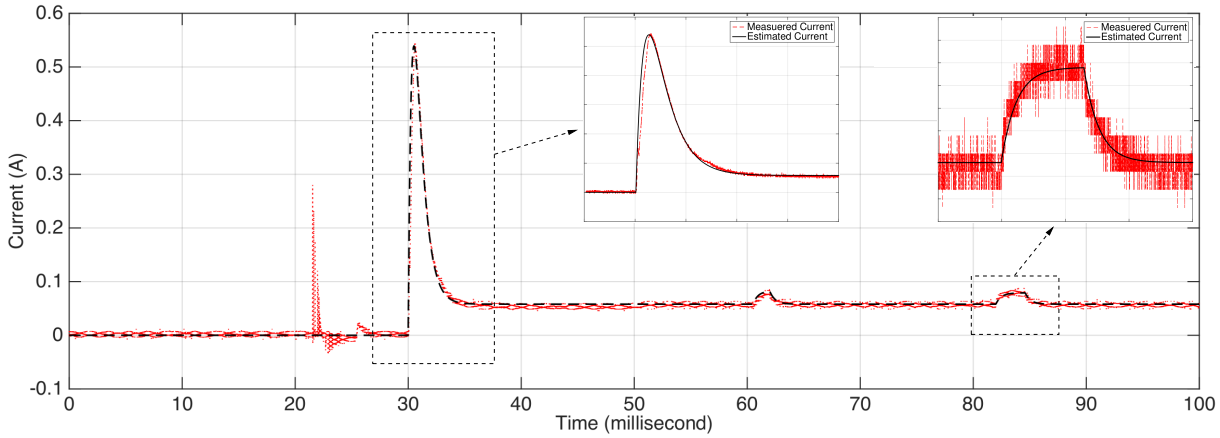


Figure 5: Comparison of measured power and estimated power consumption using the large signal modeling approach.

Note, that we did not include the small signal approach in the comparison. Considering the assumption of stationary, probabilistic signals, a comparison of waveforms would not make sense. We could have added some simulated noise to the large signal model of power consumption, so that it would intuitively look more similar, but it would not have been equal. Furthermore, the spectral properties of the simulation would be trivially the same (if implemented correctly in the simulator) as the measured ones; hence, additional measurements would not add credibility to the approach.
However, in particular, the high frequency behavior of the noise that is created by components is a candidate to create spurious tones in other components. Therefore, as part of our future work, we intend to demonstrate the value of the small-signal modeling approach by a more complex example including RF subsystems.

ISBN 978-3-00-048889-4

5. Conclusion

We proposed and evaluated an approach that for the first time allows to estimate instantaneous power consumption at a high level of abstraction.
The parameters for modeling small and large signal power behavior are easy to determine by measurements or circuit-level simulation. The modeling approach extended power state machine is intended to permit analysis of the power distribution grid and its infrastructure (e.g. DC-DC converters) and its dimensioning. Furthermore, in particular the small signal power behavior in frequency domain is useful to analyze cross-coupling via power supply rails in a very early development stage.
The experimental results show that the modeling approach is accurate. Where it is, as we postulated, useful to show potential issues in a design has to be shown by a more complex case study. This is subject of future work, based on the concept presented here. Also, a further formalization of the model is part of future work.

References

[BCS11] Bousquet, L., F. Cenni, and E. Simeu: *Systemc-ams high-level modeling of linear analog blocks with power consumption information*. In *Test Workshop (LATW), 2011 12th Latin American*, pages 1–6, March 2011.

[BHS98] Benini, Luca, Robin Hodgson, and Polly Siegel: *System-level power estimation and optimization*. In *Proceedings of the 1998 International Symposium on Low Power Electronics and Design*, ISLPED '98, pages 173–178, New York, NY, USA, 1998. ACM, ISBN 1-58113-059-7. http://doi.acm.org/10.1145/280756.280881.

[CTJ11] Chiraz Trabelsi, Rabie Ben Atitallah, Samy Meftali Jean Luc Dekeyser and Abderrazek Jemai: *A model-driven approach for hybrid power estimation in embedded systems design*. EURASIP Journal on Embedded Systems, 2011(1):15, February 2011. Article ID 569031.

[CY03] Casinovi, G. and C. Young: *Estimation of power dissipation in switched-capacitor circuits*. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 22(12):1625–1636, Dec 2003, ISSN 0278-0070.

[DMN10] Du, Wan, Fabien Mieyeville, and David Navarro: *Modeling Energy Consumption of Wireless Sensor Networks by SystemC*. Systems and Networks Communication, International Conference on, 0:94–98, 2010.

[HLF+11] Hsu, Chen Wei, Jia Lu Liao, Shan Chien Fang, Chia Chien Weng, Shi Yu Huang, Wen Tsan Hsieh, and Jen Chieh Yeh: *Powerdepot: Integrating ip-based power modeling with esl power analysis for multi-core soc designs*. In *Proceedings of the 48th Design Automation Conference*, DAC '11, pages 47–52, New York, NY, USA, 2011. ACM, ISBN 978-1-4503-0636-2. http://doi.acm.org/10.1145/2024724.2024736.

[LG02] Lauwers, E. and G. Gielen: *Power estimation methods for analog circuits for architectural exploration of integrated systems*. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 10(2):155–162, April 2002, ISSN 1063-8210.

[LHGN13] Lorenz, Daniel, PhilippA. Hartmann, Kim Gruettner, and Wolfgang Nebel: *Non-invasive power simulation at system-level with systemc*. In Ayala, JoseL., Delong Shang, and Alex Yakovlev (editors): *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation*, volume 7606 of *Lecture Notes in Computer Science*, pages 21–31. Springer Berlin Heidelberg, 2013, ISBN 978-3-642-36156-2. http://dx.doi.org/10.1007/978-3-642-36157-9_3.

[MHG10] Molina, J.M., J. Haase, and C. Grimm: *Energy consumption estimation and profiling in wireless sensor networks*. In *Architecture of Computing Systems (ARCS), 2010 23rd International Conference on*, pages 1–6. VDE, 2010.

[Mur08] Murmann, B.: *A/d converter trends: Power dissipation, scaling and digitally assisted architectures*. In *Custom Integrated Circuits Conference, 2008. CICC 2008. IEEE*, pages 105–112, Sept 2008.

[SJK+11] Schumacher, P., P. Jha, S. Kuntur, T. Burke, and A. Frost: *Fast rtl power estimation for fpga designs*. In *Field Programmable Logic and Applications (FPL), 2011 International Conference on*, pages 343–348, Sept 2011.

[SMS09] Sundstrom, T., B. Murmann, and C. Svensson: *Power dissipation bounds for high-speed nyquist analog-to-digital converters*. Circuits and Systems I: Regular Papers, IEEE Transactions on, 56(3):509–518, March 2009, ISSN 1549-8328.

[VAM93] Verghese, N.K., D.J. Allstot, and S. Masui: *Rapid simulation of substrate coupling effects in mixed-mode ics*. In *Custom Integrated Circuits Conference, 1993., Proceedings of the IEEE 1993*, pages 18.3.1–18.3.4, May 1993.

[vHBD+00] Heijningen, Marc van, Mustafa Badaroglu, Stéphane Donnay, Marc Engels, and Ivo Bolsens: *High-level simulation of substrate noise generation including power supply noise coupling*. In *Proceedings of the 37th Annual Design Automation Conference*, DAC '00, pages 446–451, New York, NY, USA, 2000. ACM, ISBN 1-58113-187-9. http://doi.acm.org/10.1145/337292.337539.

Architectural System Modeling for Correct-by-Construction RTL Design

Joakim Urdahl, Dominik Stoffel, and Wolfgang Kunz
Technische Universität Kaiserslautern
{urdahl,stoffel,kunz}@eit.uni-kl.de

Abstract

This paper works towards a design flow where a design model at an architectural system level is refined into an RTL implementation in such a way that architectural model and RTL implementation stand in a well-defined formal relationship to each other. Functional properties valid at the system level are guaranteed to hold also in the concrete implementation without any additional verification efforts at the RTL. The contribution of this paper is an "architectural modeling language (AML)". It is intended to be used together with (and can be automatically derived from) standardized high level description languages such as SystemC as an intermediate description. In particular, it is needed to overcome the limitations of SystemC in precisely defining the semantics of the design model and its interfaces. Based on an AML description of the architectural model, the paper will show how properties in a standard language like SVA can automatically be generated that guarantee the correctness of the implementation when proven on the design after all refinement steps in the design and the property set have been completed. Two cases studies are presented that demonstrate the usefulness of the proposed language and the generated properties to achieve the objectives of the new design flow.

1. Introduction

In recent years, descriptions at high abstraction levels have become established as a means of modeling digital systems in early design phases. These descriptions, often referred to as *electronic-system-level* (ESL) descriptions, are commonly used in modern design flows as virtual prototypes for parallelizing hardware and software development or as intermediate conceptual models while developing a specification. The benefits of this include design exploration, evaluation and verification of the overall system already in an early design phase. The subsequent Register-Transfer-Level (RTL) design phase benefits, however, only little from available ESL descriptions. The creation of system-level models in the large majority of applications constitutes extra design and verification efforts that must be added to the efforts required in a conventional design flow.
In contrast to the abstraction layers below, i.e., RT level, gate level and transistor level, where each of these levels constitutes a sound abstraction for the next lower level, the developed ESL descriptions are not fully trusted. When verifying system-level functionality, therefore, verifying the ESL model is not sufficient. Instead, chip-wide simulations at the RTL must be performed. This is one of the main bottlenecks in current design flows and accounts for the largest portion of the overall verification costs.

ISBN 978-3-00-048889-4

Synthesis from descriptions at abstraction levels higher than RTL, *high level synthesis*, has been a subject of research for about two decades. A number of commercial tools are available today that can produce well optimized designs. The input languages of these tools allow for a compact and intuitive description of complex algorithms. This can be of great benefit when describing strongly datapath oriented designs such as in certain applications of signal processing. Other application areas of hardware design are, however, mostly control oriented. Complicated protocols and interfaces must be developed while datapaths are fairly simple. High-level synthesis is known to perform poorly in such cases. Therefore, methods are required that better support the system level verification of designs with complex communication schemes.

In this paper, a novel design methodology is proposed where the bottom-up approach in [USK14] for creating sound system level models for given RTL implementations is inverted into a top-down procedure. A design flow is proposed where descriptions at high abstraction levels are systematically refined into RTL implementations ensuring that the original high level description represents a sound abstraction of the final implementation. Verification results obtained at the system level can therefore be trusted for the actual implementation. It is envisioned that the tremendous costs currently spent for chip wide verification on the RT-level could be drastically reduced or even be avoided completely if such a design flow can be made practical.

Sound models at the system level require abstract and yet clearly defined descriptions of the system's modules and their communication. Existing system description languages such as SystemC, due to their universal applicability and their origin as programming languages, are not immediately adequate as a basis for establishing a formal relationship between the system level and the RTL. This paper therefore proposes a language for architectural models that can be used as an intermediate format in system level design flows.

In our envisioned methodology, the refinement from system level models into RTL will be a largely manual process although we expect that the method can also be exploited for various automatic refinement steps. In general, however, we argue that a fully automatic synthesis from high level models is, in many cases, not wanted since engineering decisions are still to be made in the refinement to the cycle-accurate and bit-accurate RTL descriptions.

Design flows based on a stack of sound functional models above the RTL have been investigated extensively in the context of formal verification by theorem proving, for example in [RH04, MS08, BMB]. An important differentiator between these previous works and sound abstractions based on *path predicate abstraction (PPA)*, as proposed in [USK14], lies in the fact that the methodology developed in our work is entirely based on standardized property languages such as System Verilog Assertions (SVA) and relies exclusively on fully automatic property checking using a bounded circuit model [CBRZ01, NTW$^+$08]. This does not only support an intuitive formulation of design properties but also facilitates proof procedures that can handle industrial RTL designs of realistic complexity. The proposed approach is expected to also complement system level verification approaches targeting specific high level coverage metrics [DSW14] by providing a framework to map system level verification results to the RTL.

The article is structured as follows. The syntax and semantics of the developed architectural language is presented in Sec. 3. In Sec. 4 it is shown how a model described in this language is used as a starting point to automatically generate properties for ensuring a correct design refinement into an RTL implementation. Sec. 5 presents two cases studies demonstrating the feasibility of the envisioned design flow.

ISBN 978-3-00-048889-4

2. Design Flow

The envisioned design flow is depicted in Fig. 1. After initial phases of concept building it is proposed that ideas and concepts are formalized as executable model descriptions, e.g., written in SystemC. The refinement of these descriptions will then result in a model at an abstraction level that will here be referred to as the *architectural level*. The same high level languages can be used for describing this architectural model as are used for the conceptual models.

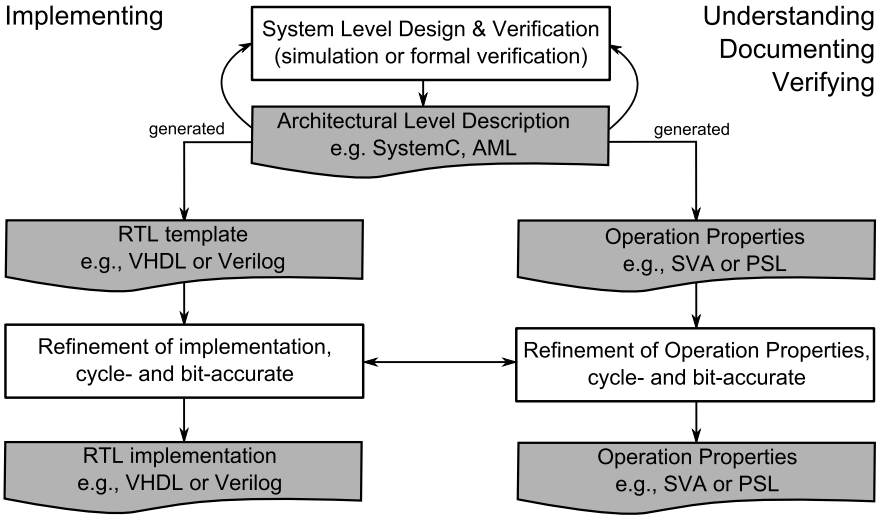


Figure 1: Novel design flow based on path predicate abstraction

The architectural level description plays a key role in our methodology. If the proposed design flow is followed it will be guaranteed that this description is in a well-defined relationship with the concrete design implementation. Specifically, the architectural model represents a *path predicate abstraction* [USK14] of the underlying RTL implementation. This has important consequences: verification results obtained at the architectural level will be valid also for the implementation without any further verifications steps. In other words, the architectural model is a *sound abstraction* of the RTL design.

In previous work it was shown how a path predicate abstraction can be created bottom-up for an existing RTL description [USK14]. The technique requires the design behavior to be structured by "operations" which are formulated as properties. These *operation properties* are expressed in terms of abstract objects. Features of the chosen property language for supporting a hierarchical code structure, such as macros or functions, are used to encapsulate all direct references to elements of the RTL design into low level objects of the property code hierarchy. This provides not only an intuitive and well readable design documentation, but, if following the methodology of [USK14] the resulting property description creates a well-defined abstraction of the RTL design. The atomic objects of this abstraction, i.e., the input alphabet, output alphabet, and states, are identified by predicates evaluated at the RTL on concrete input sequences, output sequences, and states, respectively.

Operations are understood as finite sequences of behavior between important control modes of the design. An abstract transition function can be defined as the transitions between such modes at the occurrence of abstract input symbols. For the abstraction to be fully specified all possible behaviors of the design must be described by operation properties. Such *completeness* of a property set can be proven in a computationally tractable check, as described in [BB05, USB+10].

ISBN 978-3-00-048889-4

Instead of using a bottom-up verification process the design flow in Fig. 1 creates a path predicate abstraction top-down by starting from an architectural design description. Contributing a modeling language which gives this architectural design description a clear semantics is the main objective of this paper. The full syntax of the new language is described in Backus-Naur form and is made available through our project page at http://www.eit.uni-kl.de/en/eis/research/ppa. Note that the proposed language is intended merely as an intermediate language to which the standardized high level languages preferred in industry can be automatically mapped. We consider such an intermediate language necessary because most existing high level languages are actually primarily software programming languages. Models described in these languages, e.g., SystemC, therefore lack a clear semantics as a design model. Obviously, when considering executable descriptions, their semantics is clearly defined through the corresponding assembler code. For modeling purposes this is, however, not sufficient since such semantics lack the high level constructs of the original description.

The proposed *architectural modeling language* (AML) is intended for a bit- and time-abstract design description. It precisely describes the decomposition of the design into its sub-components. The interaction between these components is modeled by event-driven communication. AML models are intended as the starting point for the RTL design phase. They specify the functionality that each module fulfills in the system without regards to timing, synchronization mechanisms and bit-widths.

In the proposed flow, from the final architectural description a complete set of *abstract operation properties*, written in SVA [Acc09] or PSL [Acc05], is generated automatically. This will be detailed in Sec. 4. These properties describe the behavior of the architectural model in terms of *abstract operations*. Using macros, functions or related language features for introducing a code hierarchy the abstract design behavior is described in terms of high level objects in a hierarchical property code. The detailed encoding of these high level objects in terms of RTL design elements, for example by formulating the actual body of an SVA function, is left to later stages of the design flow. Any design implementation that later fulfills these properties, for any encoding of the abstract property objects, is ensured to preserve the relationship of a path predicate abstraction with the original architectural description. In this sense, the resulting implementation is "correct by construction" and represents a *correct refinement* of the architectural model.

On the implementation side of Fig. 1 an arrow is drawn from the architectural description to an RTL template. It is intended to reduce efforts for implementing the design by generating an implementation template from the architectural description containing an entity definition and a state machine for the main control. This is, however, only a feature for improved convenience. All implementations fulfilling the properties, also those not using the template, by construction of our methodology are *correct refinements*.

Both the implementation and the property set must be refined into bit- and cycle-accurate descriptions. Design refinements must match the concretizations of properties to be proven. The horizontal arrow in Fig. 1, pointing in both directions between the refinement on the "implementation side" and the refinement on the "understanding, documenting, verifying side", is supposed to indicate this correspondence. Note that the refinement can be "driven" from both sides. In fact, it may be an attractive option to start with the refinement of the properties. Especially the property specifications for input and output sequences are good candidates as starting points since they represent explicit protocol descriptions and should be subject of early and conscious design decisions.

At the current state of development, a tool has been implemented which successfully generates the

ISBN 978-3-00-048889-4

set of operation properties from an architectural description in the defined language. Note that by virtue of this generation step the set of properties resulting in the right part of the flow in Fig. 1 is automatically complete. The completeness check of [BB05] is only necessary when creating an abstraction bottom-up from an existing implementation [USK14]. It is however not required in the proposed design flow. The generation of the suggested RTL template, as mentioned above, only serves convenience purposes and has not yet been implemented. Also an automatic mapping from SystemC or other standardized ESL languages to the proposed AML language is not yet developed.

3. Architectural Modeling Language

The language can be used to describe system architecture in a hierarchical manner. Systems can be composed of sub-systems and/or behavioral descriptions (modules). The full description of the language is available on our website. In the following we will sketch important aspects and features, partly using an example.

3.1. Global Scope

An AML description is composed of definitions of enumeration types, constants, functions, modules and systems. These definitions reside in the global definition scope.
System definitions are primarily intended for system modeling and verification purposes. Future development could also exploit system analysis to simplify the operational structure of the generated property suite. In a system definition behavior is described as a collection of connected module instances and/or instances of (sub-)systems. **Connections** can be made between pairs of input and output from different instances, as well as "forwarding" connections from input of the system to input of an instance and from output of an instance to output of the system.
A **constant** is a symbolic representation of an already defined data type value and may be used anywhere in the description in place of that value. Constants allow creating configurable descriptions and improve readability of both the AML as well as the operation property suite.
Enumeration types define custom data types that can be used in later declarations. The symbolic values of the data types will be present in the operation property suite as macros and are later given an encoding on the RTL. (This is in contrast to most programming languages where values are assigned to the enumeration symbols already in the definition.)
Also, **functions** can be defined in the global scope. Functions are defined in the mathematical sense: they compute a result from a set of values given as arguments. Unlike in many programming languages, functions are not subroutines and a function "call" neither affects the state of the model nor does it have any side effects. The purpose of function definitions is to encapsulate computation for code reuse and readability.
Module definitions contain the behavioral description of the modules which will later be implemented in RTL. For each module a complete operation property suite will be generated.

3.2. Example

Listing 1 will serve as an example in the following discussion. It describes a performance monitoring circuit for a telecom protocol framer. The monitor counts the number of times the framer is synchronized with the data stream ("in frame") or not ("out of frame"). The functional behavior

ISBN 978-3-00-048889-4

is inspired from a similar device implemented in the industrial framer reported in Sec. 5.2. An intuitive understanding of the functionality should be easily gained from the code.
The framer reports de-synchronization through its output `oof` ("out of frame"), which is an input to the monitor. The monitor updates `lof` ("loss of frame") as a performance metric collected over a longer period of time. The exact setting of this metric can be configured through a microprocessor interface, which is abstracted here by the input `setup`.

```
1  MODULE monitor {
2    in<bool> oof;
3    in<bool intmod, int set, int reset> setup;
4    out<bool> lof;
5
6    FSM behavior {
7      states = {REGULAR, INTEGRATING};
8      var<int> OOFcnt; var<int> IFcnt;
9      var<int> set; var<int> reset;
10   @init:
11     nextstate = REGULAR;
12     OOFcnt = 0;
13     IFcnt = 0;
14     lof = false;
15     set = 7;
16     reset = 7;
17   @REGULAR:
18     read(oof);
19     if (oof) {
20       IFcnt = 0;
21       if (OOFcnt >= set) {
22         lof = true;
23       } else {
24         OOFcnt++; }
25     } else {
26       OOFcnt = 0;
27       if (IFcnt >= reset) {
28         lof = false;
29       } else {
30         IFcnt++; }
31     }
32     write(lof);
33     if (pending(setup)){
34       read(setup);
35       set = setup.set;
36       reset = setup.reset;
37       if (setup.intmod) {
38         nextstate = INTEGRATING; }
39     }
40   @INTEGRATING:
41     read(oof);
42     if (oof) {
43       IFcnt = 0;
44       if (OOFcnt >= set) {
45         lof = true;
46       } else {
47         OOFcnt++; }
48     } else {
49       if (IFcnt >= reset) {
50         OOFcnt = 0;
51         lof = false;
52       } else {
53         IFcnt++; }
54     }
55     write(lof);
56     if (pending(setup)){
57       read(setup);
58       set = setup.set;
59       reset = setup.reset;
60       if (not setup.intmod) {
61         nextstate = REGULAR;}
62     }
63   }
64 };
```

Listing 1: AML description of monitor

3.3. Module Definitions

We will explain module definitions at the example shown in Listing 1. The syntax is quite intuitive but there are some fundamental differences when compared to RTL descriptions. The most important difference originates in the modeling of communication. A communication interface is called a *port* and it is declared using the `in` and `out` keywords (see lines 2–4). Communication is realized through these ports as `read` and `write` calls. The syntax and the semantics is that of a function call, similar to communication modeling in SystemC. The data type carried on the port is defined within angle brackets. A port may also be of a composite data type, as shown for `setup`. It "carries" data at the *event* of a write call. Both, `read` and `write` functions, block until the communication is completed.
Models at the architectural level are event-driven. No clock or similar construct is present that drives the behavior through the sequential description. Between communication points (read/write calls) behavior is only ensured to be executed within some finite time. For the system, all behavior described between communication points is unobservable and can be treated as a single atomic expression.

ISBN 978-3-00-048889-4

The behavior of a module is described as a finite state machine within an `FSM` block (see lines 6–63 in the example). The first part of the definition holds declarations used to describe the state of the module. The control states are enumerated in `states` while data variables are declared using the keyword `var` followed by the data type, again within angle brackets.
The remainder of the definition holds the behavior of the module for each of the control states. The initial state of the module is always called `init`. This keyword is implicitly added to the set of `states`. The *init section* (lines 10–16) is mandatory. No read/write calls to the ports are allowed within the init section.
The behavior of the module is specified for each of the states defined in the `states` set. A standard set of operators and syntax elements exists for defining behavior. Most of it is borrowed from the C programming language. Assignments are defined using the = operator. Conditional execution is declared using if/else.
The special keyword `nextstate` can be understood as a variable of the same enumeration type as defined by `states`. Assigning a control state to `nextstate` defines the state the FSM assumes after the execution of the current state section has finished.
A `for` loop construct is also present. It serves, however, only for conveniently defining a constant number of repetitions. This is useful when iterating over arrays or in order to create generic designs. Actual behavioral loops cannot be modeled using `for` but can instead be described as repeated executions of a section.
Read and **write** operations model, per default, *blocking* communication. A `read` is applied to an `in` port, a `write` is applied to an `out` port. After a read, the `in` port contains the received data. It can be viewed as a variable that cannot be assigned to, (it never appears on the left side of an assignment). It holds its value until the next read on this port. Analogously, an `out` port can only be written and never read, i.e., it never appears on the right hand side of an assignment. It must be assigned a value before the corresponding `write` can be called. The AML parser enforces that `in` ports are read before used and that `out` ports are assigned a value before written.
Non-blocking communication can also be modeled. The predicate `pending` applied to a port name returns *true* if data is available at the port. Unlike `read` or `write`, it does not block. If `pending` returns *true*, a subsequent `read` or `write` will return immediately (see lines 33–39).

4. Correct Refinement

The sound relationship between the architectural level and the implementation is that of a *path predicate abstraction* [USK14]. This relationship is formally proven by describing the complete behavior of the implementation as a set of operation properties where each property describes a transition between abstract control states in a module. A transition is triggered by abstract input, and it is accompanied by the abstract output produced by the module. In practice, the abstract objects are defined using constructs for defining macros or functions. Such constructs are available in all standard property languages. The macro name is considered an atomic object in the abstraction, while the macro body defines its bit-accurate and cycle-accurate implementation on the RTL. The macro body can thus be viewed as an *encoding* of the abstract object on the RTL.
Using the example of Listing 1, we discuss in the following how the architectural-level description enforces a *correct refinement* of the abstraction into the RTL. An automatic tool called *refinement synthesizer* takes as input the AML description of the system and produces a set of operation properties together with macro skeletons for the abstract objects. (See our project website at http://www.eit.uni-kl.de/en/eis/research/ppa for the property suite generated for the example

ISBN 978-3-00-048889-4

above.) Fig. 2 shows a graph representation of the operational structure of the example. Nodes in the graph represent control states, edges represent operation properties. The numbers attached as edge labels refer to the corresponding operation property in the example suite.

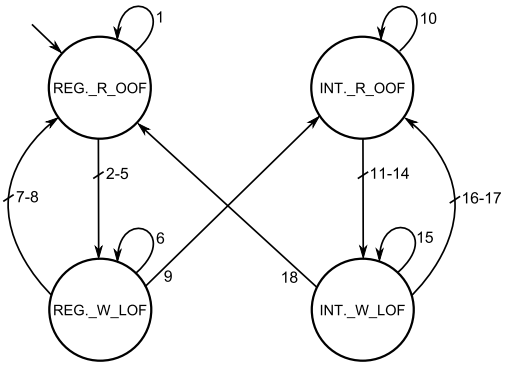


Figure 2: Control Mode Graph of the Monitor

4.1. Objects of the Abstraction

The macro skeletons consist of only a name with a return type. The macro body is empty and must be filled by the designer to represent how the abstract object is encoded in the RTL implementation. For example, an abstract state macro needs to be encoded by a set of Boolean constraints that characterize the the set of implementation states corresponding to the abstract control state.
In our example the generated property suite has four control states, one for each call to a (blocking) communication, i.e., a read or a write. In general, a control state may also be required to define the start of a section. The tool, however, recognizes the cases where this state can be merged with the first read/write state to keep the property suite as simple as possible. (For our example, this check is trivial because both abstract state sections immediately begin with a read.) Also, the abstract state information represented by the module's variable set are directly mapped to macros skeletons.
Also, macros are created for the input and output ports of the module, for receiving and sending the actual data, for synchronization, and, if needed, also for storing of data. Synchronization signal macros are generated for each port. The macro names ending in `_notify` represent incoming synchronization signals, the ones ending in `_sync` represent outgoing synchronization signals.
The set of generated communication macros also includes datapath macros which are given the ending `_sig`. The encoding of these macros may be spread over a finite number of clock cycles. These macros describe input and output sequence predicates, and they must refer only to input or output signals of the module, respectively. When modeling a system, this requirement is met automatically because connected ports are forced to share the same encoding, by construction.
Ports may be referenced throughout the entire architectural description, not only in the read/write calls. State datapath macros may therefore also have to be created for the ports. Note that this does not imply that such ports will actually have additional RTL state variables for every port. The encoding may even be the same as the for the corresponding `_sig` macro.
In the example only the `lof` port causes the creation of a state macro (the macro with name `lof`). For the two other ports, the `oof` port and the `setup` port, it holds that any reading reference to a value is preceded by a `read` call to this port, and any writing reference to a value is followed by a `write` call to this port. Therefore, no data must be stored across operations and no data storage macro must be created for these cases.

ISBN 978-3-00-048889-4

4.2. Operational Structure

The refinement synthesizer creates operation properties by, effectively, enumerating all "execution paths" that can be taken between abstract control states. Consider, for example, the `@REGULAR` section: after the `read` four such execution paths exist which all end in a `write`. These paths correspond to the operations 2 through 5 in the generated suite. They differ in the evaluation of the conditions of the if-else blocks. Each execution path is characterized by the conjunction of all condition expressions along the path. This conjunction forms the assumption of the operation property corresponding to the execution path.
The translation from ports with blocking read/write calls is reflected in the operational structure. (In its current state, the tool only supports clock-synchronous communication.) The read/write calls to `oof` and `lof` (lines 18, 32, 41, and 55) are blocking calls. The operation property suite is structured in a way such that the modeled blocking behavior is imposed on the RTL implementation. In particular, a *waiting state* is generated, modeling a mode where the module waits for an incoming synchronization signal (`_sync`). The wait is modeled by a waiting operation (e.g., operations 1, 6, 10, and 15 in the example). This operation is "triggered" by the absence of the `_notify` flag of the corresponding port.
The read calls to `setup` (line 34 and 57), are both immediately preceded by a `pending` if-condition and are therefore examples of non-blocking communication. This is exploited to simplify the operation property suite. No additional wait state is required. Note that the blocking communication scheme does not force any wait operation to actually be executed. Non-blocking communication could therefore also be modeled in this way but the wait operation would never be triggered. When the non-blocking read/write is executed the `_notify` flag is active for one clock cycle to inform the communication partner of the communication event. In a synchronous system the event is always safely captured by the communication partner. The incoming `_sync` (being the outgoing `_notify` of the paired port) ensures that the communication partner is in a state where it can react to the call. Due to the common clock any signal value kept stable for one clock cycle will be captured.

4.3. Modeling Communication at the RTL

The transfer of data must be soundly modeled by the property suite. In a read call the input data, encoded in `_sig`, is captured at the time when `_sync` is set active. In the following operation the read value is referenced as the input at this time point. If a reference is made to this port value also in other operations the value must be stored within the abstract state. This is realized as an additional datapath state macro which, at the ending control state of the operation, is proven to encode the read value.
A write call works just in the opposite way. The outgoing data transported in the `_sig` macro must have the value of the state datapath macro at the time point of the `_sync` event.
By a proper encoding of the communication data macro, `_sig`, the actual data transfer may be specified to occur also at any (fixed) time later in the actual RTL description. The structure of the properties simply ensures that the data macros explicitly specify the data encoding with time reference "anchored" at the synchronization event.
The generated macros and operations together form a communication framework that forces the RTL to implement a communication infrastructure with proper synchronization and correct blocking behavior. In RTL design, however, there exist many different signaling, synchronization and

ISBN 978-3-00-048889-4

data transfer mechanisms. The generated communication framework must allow the designer to implement these by simply encoding them properly into the macro bodies. It is therefore important that the generated property set is "generic" enough and does not lead to redundant structures on the RTL.
The paper [USWK12] discusses various common communication schemes and their modeling through synchronization and data transfer sequence predicates. All these schemes can be modeled using blocking read or write in AML and translated into corresponding mechanisms on the RTL, e.g. by event-signalling or handshaking. This is encoded through two synchronization macros for each port, an outgoing, `_notify`, and an incoming, `_sync`.

5. Experimental Results

In order to evaluate the novel design methodology we conducted two case studies. The first study was a student group design project. The second study used the new methodology for a redesign of an industrial telecommunications IP component.

5.1. Student Project

Four students were given the task to use AML and the new methodology for designing, implementing and verifying a musical game to be realized on an FPGA evaluation board. Prior to the actual design phase, the students were trained in formal property checking and completeness checking with a commercial tool suite (OneSpin 360 DV).
The system to be designed was a game called "Perfect Pitch" in which the user has to guess the musical note corresponding to a random tone played by the device. All system components were to be designed purely as hardware descriptions; no processor was used. The task was focused on the design of the central game controller. A set of hardware IP blocks for input and output (keyboard, LCD, audio) were given.
The students were asked to describe the game controller as an AML module communicating with peripherals through freely chosen ports. As a next step, the AML description was used as an informal specification while implementing the RT level using VHDL. The RTL implementation consists of the given IPs, a "main game controller", a pseudo-random generator, and several communication modules providing interfaces between the game controller and the given IPs.
In contrast to the suggested design flow, the operation property suite was first given after the implementation was done. In the case study we did not want the implementation to be affected by the details of the operational structure. The purpose was to investigate the possible discrepancies and their cause, and to check if the working communication structures of the implementation could be encoded within the boundaries of the abstract operation property suite.
The case study showed that the students could identify a suitable encoding of the abstract objects in the RTL implementation quite easily. The property suite that was created and refined from the architectural description initially failed on the implementation, pointing to an actual design discrepancy between the manual RTL design and the architectural specification. After these issues were resolved, the property suite could be proven on the RTL design.
The students were exchange students from American University of Beirut and had no previous experience with formal verification and the design methods developed in this research at the host university. They had previous experience with VHDL-based RTL design from courses in their home university. It was encouraging to see that the students, who were not biased by years of

ISBN 978-3-00-048889-4

"classic" RTL design practice, quickly picked up the new concepts, learned to use AML and adopted the new methodology. The team of four students took about two weeks for acquiring property checking skills before they actually began design. They then used about four weeks of team effort to complete the design. The students communicated their impression that the existence of AML models greatly eased the integration of the developed modules and the pursued design flow significantly reduced their work effort.

5.2. Alcatel Lucent SONET/SDH framer

The second experiment dealt with an industrial telecommunications design from Alcatel-Lucent, namely a SONET/SDH framer. In prior work, this design had been verified completely and a path predicate abstraction had been created "bottom-up" according to [USK14]. The effort attributed with this were six person months. In this work, we re-designed the circuit from scratch, following the new top-down design methodology, and starting from the path predicate abstraction created earlier. The effort for creating the new design together with an accompanying refined property suite was less than two person months. The new implementation is provably correct by construction. It is a sound refinement of the same path predicate abstraction as the original design.
In addition to being a "clean refactoring" of the old IP block, the new implementation also includes aggressive RTL optimizations for minimizing the power consumption of the circuit. The particular optimizations include manifold sharing of a single counter for various purposes (as opposed to several specific instances in the original design), reduction of input buffering, and clock gating of large combinational circuit portions. These measures lead to substantial reduction of circuit activity. Actually including them into the RTL design was only possible because their functional correctness could be immediately verified using the accompanying property suite.
The new design consumes a lot less energy than the original one. We measured a power reduction of about 50%.

6. Conclusion

The proposed design flow results in an easily understandable formal specification which is available throughout the entire RTL development phase. Any changes made to the RTL code can easily be checked against the properties. The property suite with its hierarchical description of the design behavior serves as a well readable documentation and facilitates the maintenance of the design code. Moreover, the suggested methodology allows for an efficient exploration of possible architectural choices and supports aggressive optimization on the RTL.
Our work shows that it is possible to bridge the "semantic gap" between time abstract system level models and the RTL using only standardized property languages together with bounded model property checking approaches. These approaches are known to be tractable in practice also for large industrial implementations.
In our future work, we intend to make standardized ESL languages usable in our design flow also for the architectural description itself. The constructs of the proposed language AML are therefore created to resemble the constructs in such languages, in particular in SystemC.
It is also planned to develop a module library of commonly used communication buffers. The library is intended to serve as a design aid for the typical communication protocols and should be linked with the built-in support for "channels" in SystemC.

ISBN 978-3-00-048889-4

References

[Acc05] Accellera: *IEEE Standard for Property Specification Language (PSL)*. IEEE Std 1850-2005, 2005. http://www.eda.org/ieee-1850/.

[Acc09] Accellera: *IEEE Standard for SystemVerilog – unified hardware design, specification, and verification language*. IEEE Std. 1800-2009, 2009. http://www.systemverilog.org/.

[BB05] Bormann, Joerg and Holger Busch: *Verfahren zur Bestimmung der Güte einer Menge von Eigenschaften (Method for determining the quality of a set of properties)*. European Patent Application, Publication Number EP1764715, September 2005.

[BMB] BMBF: *Verisoft XT*. http://www.verisoftxt.de.

[CBRZ01] Clarke, Edmund, Armin Biere, Richard Raimi, and Yunshan Zhu: *Bounded model checking using satisfiability solving*. Form. Methods Syst. Des., 19(1):7–34, July 2001, ISSN 0925-9856. http://dx.doi.org/10.1023/A:1011276507260.

[DSW14] Drechsler, Rolf, Mathias Soeken, and Robert Wille: *Formal Specification Level*, pages 37–52. Springer, 2014.

[MS08] Manolios, Panagiotis and Sudarshan K. Srinivasan: *A refinement-based compositional reasoning framework for pipelined machine verification*. IEEE Transactions on VLSI Systems, 16:353–364, 2008.

[NTW+08] Nguyen, Minh D., Max Thalmaier, Markus Wedler, Jörg Bormann, Dominik Stoffel, and Wolfgang Kunz: *Unbounded protocol compliance verification using interval property checking with invariants*. IEEE Transactions on Computer-Aided Design, 27(11):2068–2082, November 2008.

[RH04] Ray, Sandip and Warren A. Hunt, Jr.: *Deductive verification of pipelined machines using first-order quantification*. In *Proceedings of the* 16*th International Conference on Computer-Aided Verification (CAV 2004)*, pages 31–43, Boston, MA, 2004. Springer.

[USB+10] Urdahl, Joakim, Dominik Stoffel, Joerg Bormann, Markus Wedler, and Wolfgang Kunz: *Path predicate abstraction by complete interval property checking*. In *Proc. International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, pages 207–215, 2010.

[USK14] Urdahl, Joakim, Dominik Stoffel, and Wolfgang Kunz: *Path predicate abstraction for sound system-level models of RT-level circuit designs*. IEEE Transactions On Computer-Aided Design of Integrated Circuits and Systems (TCAD), 33(2):291–304, Feb. 2014.

[USWK12] Urdahl, Joakim, Dominik Stoffel, Markus Wedler, and Wolfgang Kunz: *System verification of concurrent RTL modules by compositional path predicate abstraction*. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pages 334–343, New York, NY, USA, 2012. ACM, ISBN 978-1-4503-1199-1.

On the Influence of Hardware Design Options on Schedule Synthesis in Time-Triggered Real-Time Systems

Alexander Biewer, Peter Munk, Jens Gladigau
Corporate Sector Research, Robert Bosch GmbH, Germany
{alexander.biewer,peter.munk,jens.gladigau}@bosch.com

Christian Haubelt
Applied Microelectronics and Computer Engineering, University of Rostock, Germany
christian.haubelt@uni-rostock.de

Abstract

In electronic system-level design, allocation, binding, routing, and scheduling heavily depend on each other. For cost-driven markets, a low-cost design of a system contains a selection of cheap hardware resources with limited parallelism. Consequently, with increasing utilization, scheduling for these cheap shared resources becomes more complicated and hence more time consuming. The time spent on solving the scheduling problem impacts the monetary cost of the system's design and thus reduces aspired cost savings.
In this paper, we present how different cost-driven hardware design options impact the scheduling problem on time-triggered tile-based hardware architectures. We introduce a refined platform model that enables us to reflect selected design options of hardware blocks from the domain of architectures including a network-on-chip (NoC). We provide a symbolic scheduling encoding to derive time-triggered schedules for platform instances with different design options. Our experiments quantify the impact of low-cost design choices on the time to find time-triggered schedules for different case studies based on periodic control applications.

1. Introduction

The computational demand of real-time embedded systems in the automotive domain is increasing due to feature requests of customers and demanding environmental regulations. Sophisticated and computationally intensive control algorithms become inevitable, e. g., in order to comply with mandatory emission targets. Many-core processors with a network-on-chip (NoC) interconnecting a large number of processing elements (PEs) are deemed to offer scalable performance [BM06]. The concept of time-triggered architectures seems attractive to execute safety-critical applications from the hard real-time domain on many-core processors, since it offers guaranteed performance [KB03]. To guarantee performance, time-triggered schedules synthesized at design-time assign each computation or communication a dedicated starting time and ensure contention-free access to shared hardware resources.
During the design phase of a new hardware platform, the architectural complexity is often handled by assembling hardware blocks. In cost-driven domains with large volumes such as the automotive sector, hardware costs are of paramount importance. Precious chip-area can be saved by reducing the number of hardware blocks as well as using hardware block design options with just enough capabilities to perform the job. Thereby, the design options might differ in their capabilities to handle independent transactions concurrently. For example, the more area consuming crossbar switch in a router enables establishing simultaneous connections between several input and output ports whereas the less area consuming bus can only be used by one input and one output port at

ISBN 978-3-00-048889-4

the same time and hence enforces arbitration of concurrent requests.
In electronic system-level design, the decisions for allocation, binding, routing, and scheduling are heavily dependent. For instance, allocating resources that implicate a high arbitration effort translates to a more complicated time-triggered scheduling of these resources. In a worst-case scenario, a prolonged design phase of a system reduces the potential cost savings offered by cheaper hardware blocks by postponing start of development.
In this paper, we investigate how different hardware design options impact the scheduling of a time-triggered real-time system. To demonstrate this, we introduce two design options with different capabilities and chip-area costs for each tile and router of the hardware platform. Given an allocation and a binding and routing of an application set onto the allocated resources, we present a refined platform model of the allocation that captures the capabilities of the selected tile and router design options. Based on the refined platform model, a time-triggered scheduling problem is formulated. Our experimental results from three different case studies show that a low-cost hardware design can increase schedule synthesis time up to 28%.
The remainder of this paper is structured as follows: Section 2 surveys related work. In Section 3 we introduce the considered hardware design options. Section 4 presents the refined platform model. Section 5 introduces our application model. In Section 6 we present the encoding of time-triggered schedules considering different design options. Experimental results that quantify the impact of hardware design choices on schedule synthesis time are presented in Section 7. Our conclusions are drawn in Section 8.

2. Related Work

The problem of symbolic time-triggered scheduling has been studied in the past. Steiner [Ste10] introduces a symbolic time-triggered scheduling encoding for messages in TTEthernet [SBHM11] and presents a scalable algorithm that extends the context of an SMT-solver incrementally in order to solve scheduling problems. Steiner [Ste11] extends his work by integrating rate-constrained event-triggered messages into his synthesis approach. Huang et al. [HBR+12] present a symbolic encoding to define the routing and scheduling of messages in the time-triggered network-on-chip (TTNoC) [PK08]. In contrast to TTEthernet, TTNoC does not allow to delay messages by buffering them in the routers of the network. Huang et al. also compare a purely SMT-based approach with incremental heuristics to improve scalability.
The literature discussed so far focuses on the scheduling of messages in time-triggered architectures. This paper considers the co-synthesis of computational task and message schedules based on an adapted encoding by Lukasiewycz and Chakraborty [LC12], who investigate the automotive bus system FlexRay. Zhang et al. [ZGSC14] and Craciunas and Oliver [CO14] present comparable scheduling encodings. Zhang et al. additionally consider (multi-objective) optimization of schedules, whereas Craciunas and Oliver present a method to improve the scalability of symbolic scheduling by introducing an incremental approach based on a demand bound test. To the best of our knowledge, this is the first work that investigates the influence of different hardware design options on co-synthesis of computational task and message schedules.

3. Hardware Design Options

In this section we introduce design options that can be selected for allocated resources. Given an allocation on the resources of a topology, e. g., the resources in Figure 1, one out of two design options for a tile or a router can be selected. One of each design options consumes less area and hence is the first choice for a low-cost hardware design. We assume that an allocation is performed on tile-based hardware architectures including an NoC. While we in general do not constrain the topology of the platform, we assume full-duplex bidirectional connections between resources (cf.

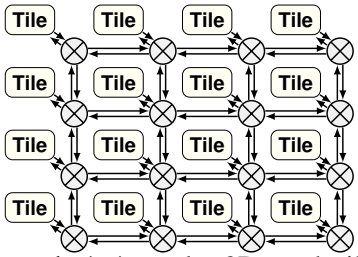
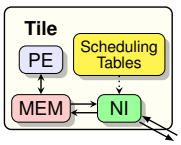
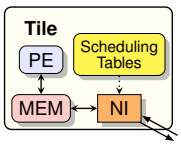


Figure 1: 4x4 regular 2D mesh tile-based hardware platform



(a) Tile design option T_a implementing an NI that can serve incoming and outgoing messages concurrently.



(b) Tile design option T_b. Due to the reduced connectivity to the memory either a incoming or outgoing message can be processed by the NI.

Figure 2: Different design options for the tiles of the NoC

the links of mesh topology in Figure 1). Concerning the transport of data through the NoC, we assume a store-and-forward routing protocol [BM06], i. e., the messages are transferred on a per-hop basis as atomic entities. Time-triggered scheduling of all resources is enabled by a global time base, e. g., a global clock.

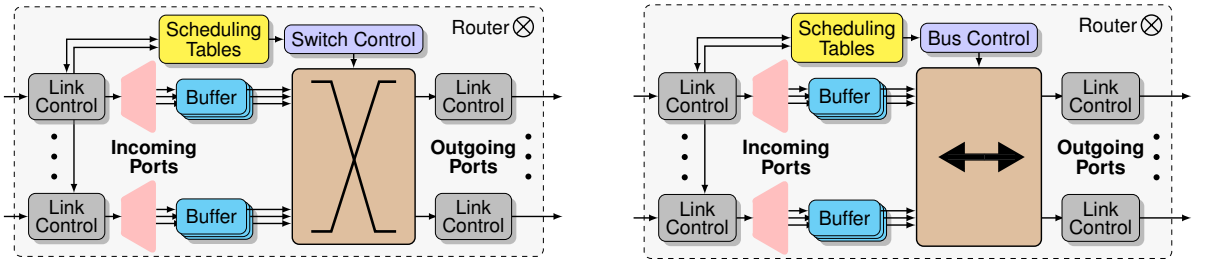
Tile Design Options. Figure 2(a) and Figure 2(b) illustrate the two design options for the tiles of the platform that we consider in this paper. Each tile contains a processing element (PE), a local memory (MEM), a network-interface (NI), and scheduling tables for messages. A PE executes computational tasks and can access the MEM to fetch code or data. The MEM can also be accessed by the NI. A multi-ported memory ensures interference-free accesses to the MEM with one port being assigned to the PE and at least one port being assigned to the NI.

Scheduling tables for messages store start times at which the NI initializes the transfer of messages on the NoC. While initializing the asynchronous transmission of a message on the outgoing link of the tile, the NI accesses the MEM to obtain all necessary data, e. g., the payload of the message. Concerning incoming messages, the NI is responsible for storing an incoming message in the tile's MEM. In this paper, we assume NIs of "simple" design, i. e., messages cannot be queued in the NI. Thus, the NI has to store incoming message immediately in the MEM of the tile.

The two design options illustrated in Figure 2 differ in the capabilities of the NI to process incoming and outgoing messages concurrently. As mentioned earlier, a multi-ported memory interface strictly decouples computation on the tile with communication on the NoC. In both design options one port of the tile's MEM is assigned to the PE of the tile. In design option T_a (cf. Figure 2(a)), the MEM provides three ports and the NI interface can access the memory via two ports. With this tile design, an incoming and an outgoing message can be processed simultaneously. In contrast, design option T_b (cf. Figure 2(b)) uses a dual-ported memory interface. While the dual-ported design T_b can be seen as the low-cost design compared to the triple-ported design T_a, the NI can access the MEM only once at a time. Ultimately, either an incoming message can be transferred on the incoming link of the tile or an outgoing message on the outgoing link of the tile. This fact is reflected in the symbolic encoding of the time-triggered schedules (cf. Section 6).

Router Design Options. Figure 3 depicts two different design options for routers. If a message is transferred on an incoming link of a router, the link control in both design options selects an incoming buffer defined at design-time and stores the message in this buffer. Similar to the scheduling tables of a tile, a router contains scheduling tables in order to enable a time-triggered forwarding of messages. A buffer is switched at predefined points in time to the associated outgoing link of the router. Subsequently, a message is transferred to the incoming buffer of the next router. Note that a connection between the incoming buffer and the outgoing link of a router has to stay established until the complete message is stored in the incoming buffer of the downstream router.

In design option R_a (cf. Figure 3(a)), a crossbar is assumed to switch between the incoming buffers of a router and the outgoing links such that each router is able to concurrently forward messages from different input buffers as long as two messages do not request the same output port. The time-triggered schedules ensure that one outgoing link is not requested more than once at the same



(a) Router design option R_a implementing a crossbar that switches between buffers and outgoing links.

(b) Router design option R_b implementing a shared bus that connects incoming buffers with outgoing links.

Figure 3: Different design options for the routers of the NoC

time. In contrast to design option R_a, the router design option R_b (cf. Figure 3(b)) implements a bus interconnect. Like tile design option T_b, the router design R_b can be seen as the more cost-efficient design. However, the router can only serve one connection at a time. As a consequence, scheduling becomes more complicated since the resource contention of all messages routed on this router design has to be resolved at once.

4. Refined Platform Model

In this section, we present a refined platform model that captures all selected design options. We assume that an allocation and a binding and routing of an application set onto the allocated resources is given. The subsequent selection of the design options are explicitly captured in the refined platform model. On the basis of the refined platform model, we formulate a time-triggered scheduling problem that respects the capabilities of the selected design options (cf. Section 6).

An allocation of resources with selected design options for tiles and routers is modeled as a directed graph $g^P = (\mathbf{R}, \mathbf{E}_p)$ (cf. Figure 4 and Figure 5). Nodes $\mathbf{R} = \mathbf{R}_{pe} \cup \mathbf{R}^a_{ni} \cup \mathbf{R}^b_{ni} \cup \mathbf{R}_{rsu}$ model shared resources of the hardware platform. Shared resources can be distinguished in processing elements (PEs) $\mathbf{R}_{pe}$, network interfaces (NIs) $\mathbf{R}^a_{ni}$ or $\mathbf{R}^b_{ni}$, and router switching units (RSUs) $\mathbf{R}_{rsu}$. As an auxiliary element, a tile $t \in \mathbf{R}_t$ merges a PE $p \in \mathbf{R}_{pe}$ and an NI $n \in \mathbf{R}^a_{ni} \cup \mathbf{R}^b_{ni}$, whereas a router $r \in \mathbf{R}_{rtr}$ merges several RSUs from the set of RSUs $\mathbf{R}_{rsu}$. The RSUs captured in a router of a refined platform model are used to represent the capabilities of a router to serve outgoing links concurrently. The router represented in the partial refined platform model in Figure 4 corresponds to the router design option R_a. Thus, a crossbar in the router is modeled. In a 2D mesh (cf. Figure 1), five RSUs represent the router's functionality to switch in each of the four directions in a 2D mesh leaving one RSU that switches to the connected NI. Figuratively, one RSU of the router is assigned to one outgoing port. Thus, an RSU is only shared by messages that request the same outgoing port in a router.

The router modeled in Figure 5 represents the router design option R_b. It only contains one RSU that is shared by all messages independent of the requested outgoing port.

Comparing the refined platform model with Figure 1, links between resources are not explicitly modeled as shared resources. Representing links as shared resources is not necessary, since we assume full-duplex bidirectional links between resources (cf. Section 3) and the fact that an RSU stays in a switched state until a message is completely transferred in a subsequent buffer. Since the availability of RSUs limits a straight forward transfer of a message on the NoC, a refined platform model with explicit representation of the RSUs is sufficient.

Directed edges $e \in \mathbf{E}_p \subseteq (\mathbf{R} \times \mathbf{R})$ in the refined platform graph model possible valid transitions between shared resources such that the given route of a message $\mathbf{R}_m$ from source to sink can be represented as an orderd set of shared resources, e. g., $\mathbf{R}_m$ = {NI 1, RSU 2, RSU 3, ..., RSU 42, NI 43}. While the graphical representations in Figure 4 and Figure 5 might seem non-intuitive in

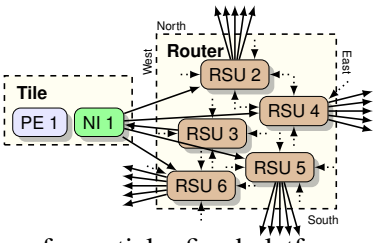


Figure 4: Illustration of a partial refined platform model representing a platform implementing the design options R_a and T_a.

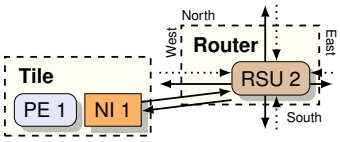


Figure 5: Illustration of a partial refined platform model representing a platform implementing the design options R_b and T_b.

the first place, the functionality of the router design options is explicitly captured in the refined platform model allowing for a succinct scheduling encoding (cf. Section 6).
For an RSU $r \in \mathbf{R}_{rsu}$, the set of outgoing edges $\{e \mid e = (r, \tilde{r}) \in \mathbf{E}_p, \tilde{r} \in \mathbf{R}_{rsu}\}$ corresponds to a link of the NoC that is arbitrated to transfer a message from a router to a router or from a router to an NI. Note that these edges are connected to all the RSUs of the downstream router. In Figure 4 and Figure 5, some incoming edges of an RSU from next neighbour routers are indicated by dashed lines.
Concerning the design options of tiles, a tile implementing the design option T_a is illustrated in Figure 4. Figure 5 represents the tile design option T_b. We introduce two types of nodes $\mathbf{R}^a_{ni}$ and $\mathbf{R}^b_{ni}$ to represent the capacities of each NI design in a tile explicitly. If tile design option T_b is selected, it is represented by a node $r \in \mathbf{R}^b_{ni}$ in the refined platform model and the symbolic encoding adds additional constraints to the scheduling problem. These constraints capture the inability of design option T_b to process incoming and outgoing message simultaneously.

5. Application Model

In this section, we provide an overview of the model of applications that are assumed to be bound and routed on allocated resources of a given topology. The formal application model is derived from periodic control applications, e. g., from the automotive domain. Each application $A_i \in \mathbf{A}$ from the set of applications $\mathbf{A}$ is specified by the tuple $A_i = (g^A_i, P_i, D_i)$. An application A_i requires to be executed periodically with the period P_i. All computation and communication of an application A_i has to be completed before the relative deadline $D_i \leq P_i$.
The connected directed acyclic graph $g^A_i = (\mathbf{T}_i, \mathbf{E}^A_i)$ modeling an application A_i specifies the computations and data dependencies in the application (cf. Figure 6). The set of nodes $\mathbf{T}_i = \mathbf{T}^t_i \cup \mathbf{T}^m_i$ of the application graph g^A_i is the union of the set of computational tasks $\mathbf{T}^t_i$ and the set of messages $\mathbf{T}^m_i$ of the application A_i. The directed edges $\mathbf{E}^A_i \subseteq (\mathbf{T}^t_i \times \mathbf{T}^m_i) \cup (\mathbf{T}^m_i \times \mathbf{T}^t_i)$ of the graph g^A_i specify data dependencies between computational tasks and messages or vice versa.
We assume that each message $m \in \mathbf{T}^m_i$ represents an atomic entity that is transferred via the interconnect of the platform on a per-hop basis, e. g., a packet in an NoC that implements a store-and-forward routing protocol [BM06].
Each computational task $t \in \mathbf{T}^t_i$ is associated with a worst-case execution time (WCET) C^r_t for each processing element (PE) $r \in \mathbf{R}_{pe}$, i. e., a tile of a heterogeneous hardware platform. For messages $m \in \mathbf{T}^m_i$, we specify the worst-case transfer time C^r_m on a resource of the NoC $r \in \mathbf{R} \setminus \mathbf{R}_{pe}$. For the sake of clarity, the period P_i of an application $A_i \in \mathbf{A}$ translates to the period P_t of a computational task $t \in \mathbf{T}^t_i$ and period P_m of a message $m \in \mathbf{T}^m_i$, such that $P_t = P_m = P_i$ (i. e., we do not consider communication between computational tasks with different periods).

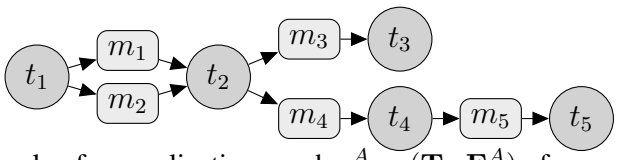


Figure 6: An example of an application graph $g_i^A = (\mathbf{T}_i, \mathbf{E}_i^A)$ of an application $A_i \in \mathbf{A}$.

6. Symbolic Scheduling Encoding

In this section we present the symbolic scheduling encoding that is based on the application model introduced in the previous section and the refined platform model introduced in Section 4. With a formulation based on the refined platform model, our encoding can be used to formulate time-triggered scheduling problems for platform instances with different selected design options.

Preliminaries. Each computational task $t \in \mathbf{T}_i^t$ of an application $A_i \in \mathbf{A}$ is assigned a start time[1] $\mathbf{s}_t^r$ on a PE $r \in \mathbf{R}_{pe}$ in a time-triggered schedule. During the system's runtime, a computational task is executed without preemption in the time frame $[k \cdot P_t + \mathbf{s}_t^r, k \cdot P_t + \mathbf{s}_t^r + C_t^r]$ with $k \in \mathbb{N}$. With this definition, a start time is a constant offset with respect to the period. Similar to the computational tasks, each message has a start time $\mathbf{s}_m^r$ on each shared resource $r \in (\mathbf{R} \setminus \mathbf{R}_{pe})$ on its route through the NoC.

The symbolic encoding assumes a given binding

$$\mathbf{B} \subseteq \left(\bigcup_{A_i \in \mathbf{A}} \mathbf{T}_i^t\right) \times \mathbf{R}_{pe}$$

of each computational task to exactly one PE of the allocated tiles ($\forall A_i \in \mathbf{A}, t \in \mathbf{T}_i^t : |\{(t,r)|(t,r) \in \mathbf{B}\}| = 1$). Furthermore, an ordered route $\mathbf{R}_m = \{r_0, r_1, r_2, \ldots, r_{|\mathbf{R}_m|}\}$ from source to sink is assumed to be given for all messages on allocated resources of a topology with selected design options. Note that $r_0, r_{|\mathbf{R}_m|} \in \mathbf{R}_{ni}^a \cup \mathbf{R}_{ni}^b$ and $(\mathbf{R}_m \setminus \{r_0, r_{|\mathbf{R}_m|}\}) \subseteq \mathbf{R}_{rsu}$. If a message is not routed on the NoC, $\mathbf{R}_m = \emptyset$ holds.

For the sake of clarity, we express all values of parameters that adhere to a notion of time, e. g., the WCETs C_t^r, the periods P_t/P_m, and the start times $\mathbf{s}_t^r/\mathbf{s}_m^r$, as multiples of the global clock in the system, i. e., in cycles. Thus, $C_t^r, P_t, P_m, \mathbf{s}_t^r, \mathbf{s}_m^r \in \mathbb{N}$, holds.

As an auxiliary function, the binary function $path(\lambda, \widetilde{\lambda}) \rightarrow \{0,1\}$ returns 1 if there exists a path in the connected directed graph g_i^A of application $A_i \in \mathbf{A}$ from node $\lambda \in \mathbf{T}_i$ to node $\widetilde{\lambda} \in \mathbf{T}_i$.

Variable Bounds. For all computational tasks, the latest point in time each computational task $t \in \mathbf{T}_i^t$ can be started on a PE $r \in \mathbf{R}_{pe}$ without risking to miss its deadline is equal to $D_i - C_t^r$, hence

$$\forall A_i \in \mathbf{A}, \forall t \in \mathbf{T}_i^t, (t,r) \in \mathbf{B} : 0 \leq \mathbf{s}_t^r \leq D_i - C_t^r. \tag{1}$$

The start times of messages do not need to be bounded due to the reasonable assumption that all messages represented in the application graphs are consumed by computational tasks, i.e., all messages are connected via an outgoing edge to a computational task. Implicitly, the domain of start times of messages is bounded by constraints between computational tasks and messages introduced later in this section.

Variable Constraints. For each PE of the platform, it has to be ensured that a PE is utilized at most by one computational task at the same time. This is ensured by the following constraint that in addition ensures non-preemptive scheduling.

$\forall r \in \mathbf{R}_{pe}, \forall t, \tilde{t} \in \{t|(t,r) \in \mathbf{B}\}, path(t,\tilde{t}) = path(\tilde{t},t) = 0, H_{t\tilde{t}} = lcm(P_t, P_{\tilde{t}}),$
$i = \left\{0, 1, \ldots, \frac{H_{t\tilde{t}}}{P_t} - 1\right\}, j = \left\{0, 1, \ldots, \frac{H_{t\tilde{t}}}{P_{\tilde{t}}} - 1\right\} :$

$$(i \cdot P_t + \mathbf{s}_t^r + C_t^r \leq j \cdot P_{\tilde{t}} + \mathbf{s}_{\tilde{t}}^r) \oplus (j \cdot P_{\tilde{t}} + \mathbf{s}_{\tilde{t}}^r + C_{\tilde{t}}^r \leq i \cdot P_t + \mathbf{s}_t^r) \tag{2}$$

[1] To ease the readability, the variables determined by an arithmetic solver are displayed as lowercase bold characters.

Here, $\oplus$ is the exclusive or operator (XOR). Equation (2) ensures that each instance i of a computational task t does not overlap with any instance j of computational task $\tilde{t}$ on the PE during the hyper-period $H_{t\tilde{t}}$. The hyper-period is the least-common multiple (LCM) of the period P_t and $P_{\tilde{t}}$. The left hand side of (2) states the necessary constraint if instance i of t starts before instance j of $\tilde{t}$. The right hand side states the reverse (assuming instance j of $\tilde{t}$ starts before instance i of t). Note that both sides of (2) can never be satisfied at the same time.
Equation (2) is not evaluated if there is a path in the application graph between two computational tasks that are bound to the same PE. However, it still has to be ensured that two computational tasks do not utilize a PE at the same time. Furthermore, if two computational tasks that exchange one or more messages are bound to the same PE, these messages are not routed on the NoC. The following constraint satisfies the data dependency between the two tasks and satisfies (2).
$\forall A_i \in \mathbf{A}, \forall t, \tilde{t} \in \mathbf{T}_i^t,\ (t,m),(m,\tilde{t}) \in \mathbf{E}_i^A,\ (t,r),(\tilde{t},r) \in \mathbf{B}$:

$$\mathbf{s}_t^r + C_t^r \leq \mathbf{s}_{\tilde{t}}^r \tag{3}$$

In case there exists only one path between two computational tasks and no immediate message is exchanged, constraint (2) is implicitly satisfied by constraints introduced later in this section. These constraints ensure a chronological sequence via the other computational task or messages on the path in the application graph.
Similar to PEs, resources of the NoC can be utilized at most by one message at the same time. This results in the following constraint, similar to (2).
$\forall r \in (\mathbf{R} \setminus \mathbf{R}_{pe}), \forall m, \widetilde{m} \in \{m | r \in \mathbf{R}_m = \{r_0, r_1, r_2, \ldots, r_{|\mathbf{R}_m|}\}, r \neq r_{|\mathbf{R}_m|}\}, path(m,\widetilde{m}) = path(\widetilde{m}, m) = 0, H_{m\widetilde{m}} = lcm(P_m, P_{\widetilde{m}}), i = \left\{0, 1, \ldots, \frac{H_{m\widetilde{m}}}{P_m} - 1\right\},\ j = \left\{0, 1, \ldots, \frac{H_{m\widetilde{m}}}{P_{\widetilde{m}}} - 1\right\}$:

$$(i \cdot P_m + \mathbf{s}_m^r + C_m^r \leq j \cdot P_{\widetilde{m}} + \mathbf{s}_{\widetilde{m}}^r) \oplus (j \cdot P_{\widetilde{m}} + \mathbf{s}_{\widetilde{m}}^r + C_{\widetilde{m}}^r \leq i \cdot P_m + \mathbf{s}_m^r) \tag{4}$$

Different values for the delays $C_m^r / C_{\widetilde{m}}^r$ for different resources can be used to model heterogeneous delays on the resources. However, in an NoC implementing a store-and-forward routing protocol, the delay on the same resource for each message is assumed to be equal. Note that the values of $C_m^r / C_{\widetilde{m}}^r$ implicitly include the delay on the links of the NoC. Furthermore, the values include the switching delay of an RSU (if $r \in \mathbf{R}_{rsu}$) or the delay of an NI (if $r \in \mathbf{R}_{ni}^a \cup \mathbf{R}_{ni}^b$).
Note that (4) does not instantiate constraints on the start times of a message if the selected shared resource r corresponds to the last resource $r_{|\mathbf{R}_m|}$ of the message on its route on the NoC, i. e., the NI of its destination. If this were not the case, all NIs would be treated as the tile design option T_b. We will introduce an additional constraint that captures the limited capabilities for allocated tiles implementing the tile design option T_b.
$\forall \tilde{r} \in \mathbf{R}_{ni}^b, r = \{r | (r,\tilde{r}) \in \mathbf{E}_p, r \in \mathbf{R}_{rsu}\}, \forall \widetilde{m} \in \{m | \tilde{r} \in \mathbf{R}_m = \{r_0, r_1, \ldots, r_{|\mathbf{R}_m|}\}, \tilde{r} = r_0\}, \forall m \in \{m | r \in \mathbf{R}_m = \{r_0, r_1, \ldots, r_{|\mathbf{R}_m|}\}, r = r_{|\mathbf{R}_m|-1}\}, path(m,\widetilde{m}) = path(\widetilde{m}, m) = 0, H_{m\widetilde{m}} = lcm(P_m, P_{\widetilde{m}}), i = \left\{0, 1, \ldots, \frac{H_{m\widetilde{m}}}{P_m} - 1\right\},\ j = \left\{0, 1, \ldots, \frac{H_{m\widetilde{m}}}{P_{\widetilde{m}}} - 1\right\}$:

$$(i \cdot P_m + \mathbf{s}_m^r + C_m^r \leq j \cdot P_{\widetilde{m}} + \mathbf{s}_{\widetilde{m}}^{\tilde{r}}) \oplus (j \cdot P_{\widetilde{m}} + \mathbf{s}_{\widetilde{m}}^{\tilde{r}} + C_{\widetilde{m}}^{\tilde{r}} \leq i \cdot P_m + \mathbf{s}_m^r) \tag{5}$$

The constraints added by (5) synchronize the RSU that switches to the NI $\tilde{r} \in \mathbf{R}_{ni}^b$ with the outgoing messages of the same NI, since the NIs $\mathbf{R}_{ni}^b$ cannot process an incoming and outgoing message at the same time.
While (4) and (5) deal with the utilization of shared resources of the NoC, the path dependency of messages between resources has also to be satisfied. At the earliest, the transfer of a message m on a route $\mathbf{R}_m = \{r_0, r_1, \ldots, r_i, r_{i+1}, \ldots, r_{|\mathbf{R}_m|}\}$ can be started on resource r_{i+1} after the message

ISBN 978-3-00-048889-4

was transferred on resource r_i.
$\forall A_i \in \mathbf{A}, \forall m \in \mathbf{T}_i^m, \forall r \in \mathbf{R}_m = \{r_0, r_1, r_2, \ldots, r_{|\mathbf{R}_m|}\}, i = \{0, 1, 2, \ldots, |\mathbf{R}_m| - 1\}$:

$$\mathbf{s}_m^{r_i} + C_m^{r_i} \leq \mathbf{s}_m^{r_{i+1}} \tag{6}$$

By formulating the constraint with "$\leq$" instead of "=", we model buffering in the incoming buffers of a router. Note that the NI of the sink of a message is omitted in (6). With the assumption of the simple NI that cannot queue messages, cf. Section 3, it is not necessary to introduce a start time of the NI of a message's destination tile. The (periodic) availability of a message m on a tile can be computed by the start time of the message and the delay on the RSU $r_{|\mathbf{R}_m|-1} \in \mathbf{R}_m$ that switches to the tile: $\mathbf{s}_m^{r_{|\mathbf{R}_m|-1}} + C_m^{r_{|\mathbf{R}_m|-1}} + k \cdot P_m$. Note that if the NIs in an NoC are designed such that they introduce additional delays, this has to be reflected in $C_m^{r_{|\mathbf{R}_m|-1}}$ for incoming messages of NIs.
Similar to the data dependencies between computational tasks in (3), the data dependencies between messages and computational tasks have to be satisfied if messages are routed on the NoC.
$\forall A_i \in \mathbf{A}, \forall (t, m) \in \mathbf{E}_i^A, t \in \mathbf{T}_i^t, (t, r) \in \mathbf{B}, m \in \mathbf{T}_i^m, \tilde{r} = r_0 \in \mathbf{R}_m$:

$$\mathbf{s}_t^r + C_t^r \leq \mathbf{s}_m^{\tilde{r}} \tag{7}$$

At the earliest, the transfer of a message m on the NoC can be initialized if the associated sender computational task finished its execution. By formulating the constraint with "$\leq$" instead of "=", we model buffering in the tile of the sender.
Similar to (7), a computational task can start its execution at the earliest if all data necessary for its computation is available. This implies that all the incoming messages of a computational task, deduced from the graph g_i^A, have been transferred completely via the last RSU on their route $\mathbf{R}_m$.
$\forall A_i \in \mathbf{A}, \forall (m, t) \in \mathbf{E}_i^A, m \in \mathbf{T}_i^m, t \in \mathbf{T}_i^t, (t, r) \in \mathbf{B}, \tilde{r} = r_{|\mathbf{R}_m|-1} \in \mathbf{R}_m$:

$$\mathbf{s}_m^{\tilde{r}} + C_m^{\tilde{r}} \leq \mathbf{s}_t^r \tag{8}$$

7. Experiments

In this section, we present experimental results that quantify the influence of different hardware design options on the time to derive a time-triggered schedule for the system. All experiments are performed on a Linux workstation with two quad-core 2.4 GHz Intel Xeon E5620 and 48 GB RAM. We use the SMT-solver `yices`[2] [Dut14] to derive the time-triggered schedules.
For each of our three case studies CS_i ($i = 1, 2, 3$) we build a synthetic application set $\mathbf{A}$ by generating random applications $A_i \in \mathbf{A}$. New applications are added to the application set until the overall computational utilization was greater than 6400% (i. e., a platform with 16 PEs is utilized at least 40%). Each application $A_i = (g_i^A, P_i, D_i)$ has a random period and deadline $D_i = P_i \in \{5\,\text{ms}, 10\,\text{ms}, 20\,\text{ms}, 40\,\text{ms}, 80\,\text{ms}\}$ and consists of a random number of tasks $|\mathbf{T}_i^t| \in \{3, 4, 5, 6\}$. All computational tasks have a homogeneous WCET on all PEs ($\forall r, \tilde{r} \in \mathbf{R}_{pe}, t \in \mathbf{T}_i^t : C_t^r = C_t^{\tilde{r}}$). The WCET of each computational task is defined by a random utilization $C_t/P_i \in \{3\%, 4\%, 5\%, 6\%\}$. Concerning the messages of each application, the overall number of messages is set to $|\mathbf{T}_i^t| - 1$. The exchange of messages between computational tasks in an application is specified such that the resulting application graph g_i^A is a task chain. The transfer time of all messages on all resources was set to 7 cycles ($\forall r \in (\mathbf{R} \setminus \mathbf{R}_{pe}), \forall m \in \mathbf{T}_i^m : C_m^{\tilde{r}} = 7\,\text{cycles}$). In total, the three case studies contain 145 ± 4 computational tasks and 112 ± 3 messages.
The topology of the allocation of all case studies is set to a regular 4x4 2D mesh with 16 tiles (cf. Figure 1). We use the approach presented in [BAG+15] to compute the binding of computational tasks and the routing of messages of the each case study's application set on the allocated resources. The binding and routing approach is based on the work presented in [AGS+13] and allows us to perform load balancing of the PEs. We ensure that all messages between computational

[2] Version 2.2.2 with command-line arguments `-logic=QF_IDL` and `-arith-solver=floyd-warshall`.

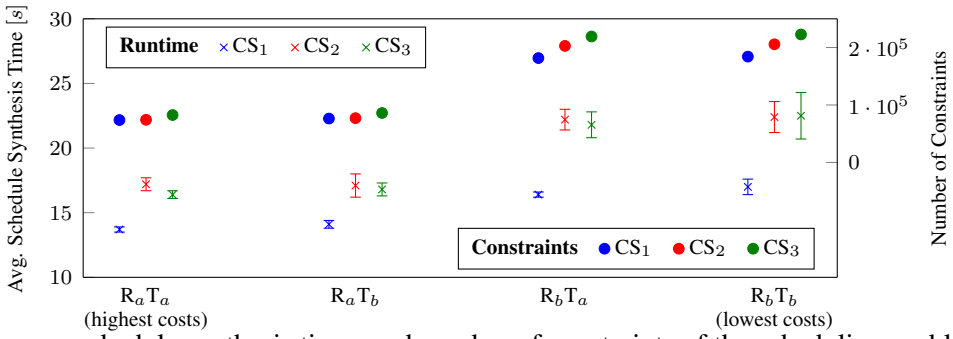


Figure 7: Average schedule synthesis times and number of constraints of the scheduling problems over the selected design options for the three case studies CS_i. Error bars show the standard deviation.

tasks are routed on the NoC by adding the constraint that every task of one application is mapped to another PE. In the resulting routing of the messages, the average number of hops is 1357 ± 40.
Given the binding and routing of a case study's application set onto the allocation, we configure the routers and tiles of the allocation with the design options from Section 3. We differentiate four design option selections: R_aT_a, R_aT_b, R_bT_a, and R_bT_b. In R_aT_a, the design option R_a is selected for all routers and the design option T_a is selected for all tiles. The same holds for the remaining design option selections with respect to their indexes. Design option selection R_aT_a represents the most expensive selection in terms of chip area and reduces the scheduling effort to a minimum. Design option selection R_bT_b is the low-cost selection with tile and router design options that enforce a high scheduling effort due to the limited parallelism of the resources.
Given allocation, binding, and routing for each of the case studies, we generate a scheduling problem for each of the four design option selections using the encoding presented in Section 6. The average number of variables, i. e., start times of computational tasks and messages, of scheduling problems is 1498 ± 39. Note that the number of variables is independent of the design option selection for a fixed allocation, binding and routing.
Figure 7 presents the mean schedule synthesis time and the number of constraints of the scheduling problem of each case study CS_i for all design option selections. Note that all scheduling problems are feasible and the synthesis runtimes per design option selection is averaged over five runs of the SMT-solver with different seeds. Figure 7 shows that the difference in the number of constraints is relatively small ($2,978 \pm 509$ constraints) if tile design option T_b instead of T_a is selected for all tiles (independent of the router design option selection). Consequently, the difference of the average schedule synthesis time of each case study is small ($0.3s \pm 0.3s$) if the design options change from $R_aT_a \rightarrow R_aT_b$ and $R_bT_a \rightarrow R_bT_b$.
In contrast, if all routers are selected to be of option R_b instead of R_a, we observe a considerable increase in the number of constraints and an increase in the schedule synthesis time. On average, the number of constraints increases by $124,389 \pm 14,875$ constraints due to additional constraints to represent the limited parallelism of the bus in router design option R_b (cf. Figure 3(b)). The increased time for schedule synthesis is consistent with the increase in the number of constraints. On average, the synthesis time increases by $4.5s \pm 1.4s$ ($\approx 28\%$) if the design options change from $R_aT_a \rightarrow R_bT_a$ and $R_aT_b \rightarrow R_bT_b$.
We summarize the results of our experiments as follows: (1) selecting design option T_b for all tiles of the allocation instead of design option T_a results only in a reduced number of additional constraints and hence a small decrease in the schedule synthesis times. (2) Selecting design option R_b for all routers of the allocation can considerably increase the additional number of constraints of the time-triggered scheduling problems and hence noticeably increase the synthesis times. For our case studies we found that choosing the low-cost router design option R_b over the more expen-

sive option R_a can increase the schedule synthesis time by approximately 28%. Thus, during the development of a system, projections towards a low-cost design should be treated carefully such that potential cost savings are not mitigated by a prolonged design phase.

8. Conclusions

In this paper, we introduced two different design options for tiles and for routers of a hardware platform including an NoC. One design option represents an efficient design in terms of hardware costs while the other exposes better capabilities in terms of parallel execution of transactions. Given a binding and routing of an application set onto a given allocation and a design option selection for the allocation, we introduced a refined platform model that explicitly captures the capabilities of the selected design options. Based on the refined platform model, we presented a symbolic scheduling encoding to compute time-triggered schedules for platform instances. We quantified the influence of design option selections on schedule synthesis time by three case studies in our experiments. We found that, due to limited parallelism in resources, the schedule synthesis time can be up to 28% higher if hardware block design options with limited capacities are selected, mitigating the potential cost savings the cheaper hardware block might offer. In system-level design, the dependencies of allocation, binding, routing, and scheduling have to be considered such that cost-saving designs of hardware resources do not introduce additional costs due to challenging problems in schedule synthesis.

References

[AGS+13] Andres, B., M. Gebser, T. Schaub, C. Haubelt, F. Reimann, M. Glaß: *Symbolic system synthesis using answer set programming*. In *Proc. of LPNMR*, pages 79–91, 2013.

[BM06] Bjerregaard, T. and S. Mahadevan: *A survey of research and practices of network-on-chip*. ACM Comput. Surv., 38(1), 2006.

[CO14] Craciunas, S. and R. Oliver : *SMT-based task- and network-level static schedule generation for time-triggered networked systems*. In *Proc. of RTNS*, pages 45–54, 2014.

[Dut14] Dutertre, B.: *Yices 2.2*. In *Proc. of CAV*, pages 737–744, 2014.

[HBR+12] Huang, J., J. Blech, A. Raabe, C. Buckl, A. Knoll: *Static scheduling of a time-triggered network-on-chip based on SMT solving*. In *Proc. of DATE*, pages 509–514, 2012.

[KB03] Kopetz, H. and G. Bauer: *The time-triggered architecture*. Proc. of the IEEE, 91(1):112–126, 2003.

[LC12] Lukasiewycz, M. and S. Chakraborty: *Concurrent architecture and schedule optimization of time-triggered automotive systems*. In *Proc. of CODES+ISSS*, pages 383–392, 2012.

[BAG+15] Biewer, A., B. Andres, J. Gladigau, T. Schaub, and C. Haubelt: *A symbolic system synthesis approach for hard real-time systems based on coordinated SMT-solving*. In *to be published in Proc. of DATE*, 2015.

[PK08] Paukovits, C. and H. Kopetz: *Concepts of switching in the time-triggered network-on-chip*. In *Proc. of RTCSA*, pages 120–129, 2008.

[SBHM11] Steiner, W., G. Bauer, B. Hall, and Paulitsch M.: *Time-triggered ethernet*. In Obermaisser, Roman (editor): *Time-Triggered Communication*. CRC Press, 2011.

[Ste10] Steiner, W.: *An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks*. In *Proc. of RTSS*, pages 375–384, 2010.

[Ste11] Steiner, W.: *Synthesis of static communication schedules for mixed-criticality systems*. In *Proc. of ISORCW*, pages 11–18, 2011.

[ZGSC14] Zhang, L., D. Goswami, R. Schneider, and S. Chakraborty: *Task- and network-level schedule co-synthesis of ethernet-based time-triggered systems*. In *Proc. of ASP-DAC*, pages 119–124, 2014.

Symbolic Message Routing for Multi-Objective Optimization of Automotive E/E Architecture Component Platforms

Sebastian Graf, Michael Glaß, Jürgen Teich
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
{sebastian.graf,glass,teich}@cs.fau.de

Abstract

The work proposes an enhanced edge-based symbolic routing encoding strategy applied for a valid static message routing during a Design Space Exploration of automotive E/E architecture component platforms. Especially as the extent of automotive networks and their network diameters are increasing, a multi-variant optimization requires a compact and scalable message routing encoding to be applicable for the new demands and extents of the E/E architecture. Within the applied symbolic encoding, experiments show that the proposed routing encoding significantly reduces the required number of variables to guarantee a valid static routing during system synthesis, while still covering the same design space. Moreover, the approach increases the performance of the optimization of E/E architecture component platforms.

1. Introduction and Related Work

In recent years, the automotive industry forces to reuse equal parts in different cars and, thus, heavily tends towards platform-based product family development in various areas [SSJ06]. Starting from using identical mechanical parts across various car variants based on platforms, also parts of the electric and electronic (E/E) architecture of the car, i. e., hardware components, are candidates for a reuse. But, to set up such a scalable *E/E architecture component platform* that covers several car variants—possibly from entry level cars up to premium cars—potentially multiple *manifestations* of the involved *components* like *Electronic Control Units (ECUs)* have to be developed to optimally meet all given requirements. Moreover, while the component platform has to (a) cover all car variants, (b) guarantee certain design constraints and optimized objectives, and (c) still implement each car as cheap as possible, the number of component manifestations that have to be developed should be kept small and, thus, be reused whenever this is possible and beneficial.
To deal with the challenging task of E/E architecture and component design and, even more, with the architecture component platform design, system-level design methodologies are of increasing importance for the development. Therefore, to ponder between several design objectives, multi-objective *Design Space Exploration (DSE)* approaches for the challenging task of E/E architecture design (e. g., see [GLT+09, SVDN07]) as well as for multi-variant optimization of component platforms, see [GGTL14b, GGTL14a], were proposed recently. Beside approaches designed for the use within the automotive domain, several other approaches for system-level design of embedded systems, e. g., MPSoCs [LKH10], exist. Beside handling of allocation and binding, they all face with the problem of guaranteeing valid message routings, i. e., it has to be ensured that for each message a valid path from the sending to the receiving resource(s) is existent. Therefore, they typically include static message routing determination within the optimization model.
As future automotive E/E architectures comprise more and more resources, multiple different bus systems, and upcoming switched networks, the exact routing of the messages at design time is

ISBN 978-3-00-048889-4

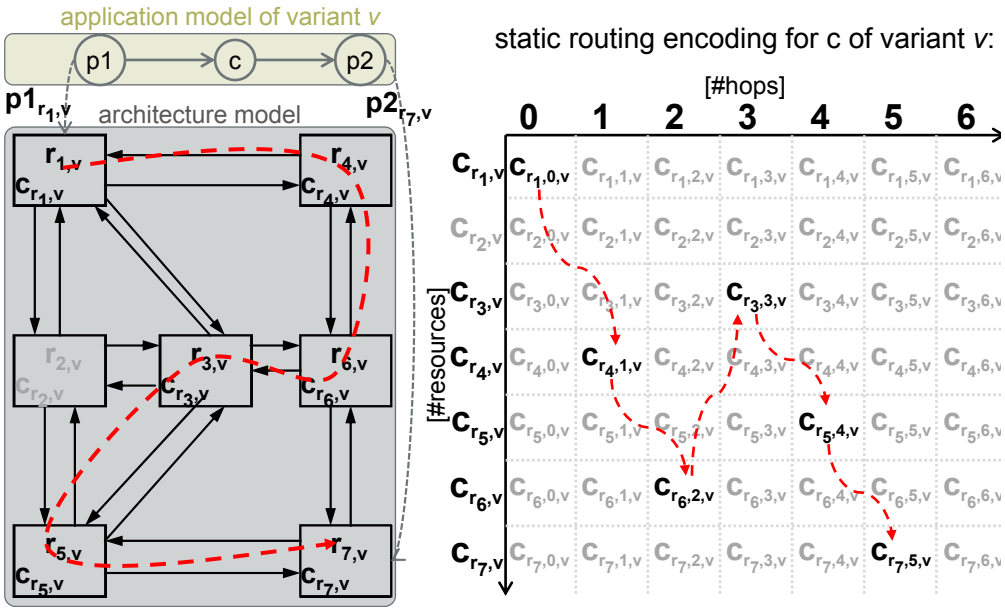


Figure 1: Basic symbolic encoding for a system-level DSE as proposed in [GGTL14b]. Whereas resource allocation and task binding for a variant v is encoded by binary variables $\boldsymbol{r}_v$ and $\boldsymbol{p}_{r,v}$, the routing of message c comprises multiple binary variables. As given for message c, if assuming a network diameter of 6, the hop-based message routing encoding strategy results in 56 routing-related (7 $\boldsymbol{c}_{r,v}$ and 49 $\boldsymbol{c}_{r,t,v}$) binary variables. One variable assignment for a valid routing (path $r_1 \rightarrow r_4 \rightarrow r_6 \rightarrow r_3 \rightarrow r_5 \rightarrow r_7$) is marked in the matrix as well in the architecture.

increasingly complex. Caused by safety critical applications requiring guaranteed delays or redundant paths, all messages are routed statically across the whole network enabling a proper real-time analysis, bandwidth allocation, and even guaranteed bus loads. Thus, opposed to large scale network infrastructure with fully dynamic routing or Networks on Chips (NoCs) used in MPSoCs (e. g., with trivial routing algorithms like x/y-routing [HM03]), the automotive domain requires an a-priory static message routing for the whole system at design time.

Due to the rising extent of the E/E architecture—especially regarding future networks integrating switched Ethernet—the involved static routes are getting longer and the flexibility for their determination at design time increases. Thus, to handle this, DSE approaches integrate message routing, beside resource *allocation* and task *binding* and *scheduling*, in the step of *system synthesis* [GHP+09]. One popular approach to deal with static message routing during system synthesis is a hop-based routing encoding strategy proposed in [LSG+09] and adapted for an E/E architecture component platform optimization in [GGTL14b]. As an example for the problem encoding, also applied within this work, left hand side of Fig. 1 gives a rough overview on the used symbolic model encoding of one single car variant expressed as a functional variant $v \in V$ using binary variables[1] as presented in [GGTL14b] and based on [LSG+09]. There, for a variant $v \in V$, the resource allocation is encoded by variables $\boldsymbol{r}_v \in R$ set to (1) if the resource is included in an allocation for variant v and (0) if not. For the task binding, mapping edges $\boldsymbol{p}_{r,v} \in E_M$ are used to signal the binding of a task $p \in P^v$ to resource $r \in R$ (set to (1)) or not (0). The routing of message $c \in C^v$ is decided within each variant v with the help of binary routing variables $\boldsymbol{c}_{r,v}$ for each resource $r \in R$ and hop variables $\boldsymbol{c}_{r,t,v}$ for each resource $r \in R$ and hop $t = [0; n]$. The latter one is used to give the exact position of a resource r in the routing path.

This encoding allows to extract static message routing, but it has one major drawback: The number

[1] Throughout this work, all binary variables used for the symbolic encoding are given in boldface.

of routing-related binary variables is very high. As each resource can possibly be used at any position within the route, the encoding has to integrate all possibilities, but at most one can be activated. I. e., for the example in Fig. 1, 7 $\boldsymbol{c}_{r,t,v}$ variables—one for each hop position—for each resource possibly used for the static routing of message c are required. But, most of these variables or even all of them are set to (0) (in Fig. 1 given in gray font) when determining a fixed static message routing. This results in a matrix-based encoding for each message $c \in C^v$, see right hand side of Fig. 1. Moreover, to lower the extent of the encoding, the encoding needs to define the maximum number of allowed hops that defines the length n of the routing, i. e., to limit the network diameter. As could easily be realized, the complexity of the overall system encoding heavily depends on this network parameter. In the worst case, the routing of x messages over a network with y resources and an upper bound for the network diameter of $z \leq y$ comprises $[x{*}y{*}z]$ routing-related $\boldsymbol{c}_{r,t,v}$ variables, just for guaranteeing the correctness of a message routing. Thus, if permitting full routing flexibility—meaning all y resources can be passed within one route and, thus, $(z = y)$—this encoding scales quadratically with the number of resources y, i. e., $[x * y * y]$. But, for larger examples, z may have to be chosen smaller than the network diameter. In this case, possible parts of the design space are lost due to message routing restrictions.
Overall, the previously applied hop-based routing encoding strategy requires a very high number of variables for the message routing to encode the optimization problem. Beside unneeded complexity, this causes scalability problems towards the optimization of whole E/E architecture component platforms of premium class cars with lots of involved electronic control units and multiple bus systems. Thus, to explore E/E architecture component platforms for multiple cars in parallel, as proposed in [GGTL14b, GGTL14a], where the routing of a message has to be determined for each variant, such an inefficient encoding aggravates and limits the optimization process.
To overcome these drawbacks, the work at hand proposes an edge-based message routing encoding strategy, see [GRGT14], that was adapted for the multi-variant component platform design and optimization. This encoding a) does not require to limit the number of hops, thus, to unnecessarily limit the routing capabilities by restricting the design space, b) significantly reduces the number of routing-related variables due to a more efficient encoding, and c) increases the optimization performance in case of scalability and convergence.
The rest of the paper is outlined as follows: Section 2 presents the proposed multi-variant edge-based message routing encoding. In Section 3 experimental results show the scalability as well as the impact in optimization quality compared to existing work. Section 4 concludes the work.

2. Multi-Variant Edge-based Routing Encoding

The proposed message routing encoding strategy enhances the optimization principles proposed in [GGTL14b]. The component platform optimization problem is given as a *multi-variant specification* that comprises functional variants $v \in V$ defined on an application graph G_T with processes $p \in P^v \subseteq P$ and messages $c \in C^v \subseteq V$. Additionally, an architecture model is given as an architecture graph G_R with resources $r \in R$ connected via edges $e \in E_R$, and mapping edges $p_r \in E_M$ for each task $p \in P$. For the optimization, the symbolic model encoding for allocation and binding is taken from [GGTL14b], with adapted routing encoding by principles proposed in [GRGT14].

2.1. Basic Edge-based Routing Encoding

The proposed routing encoding strategy for each variant $v \in V$ is based on encoding directed edges $e \in E_R$ between adjacent resources. It avoids the necessity to express each resource with its

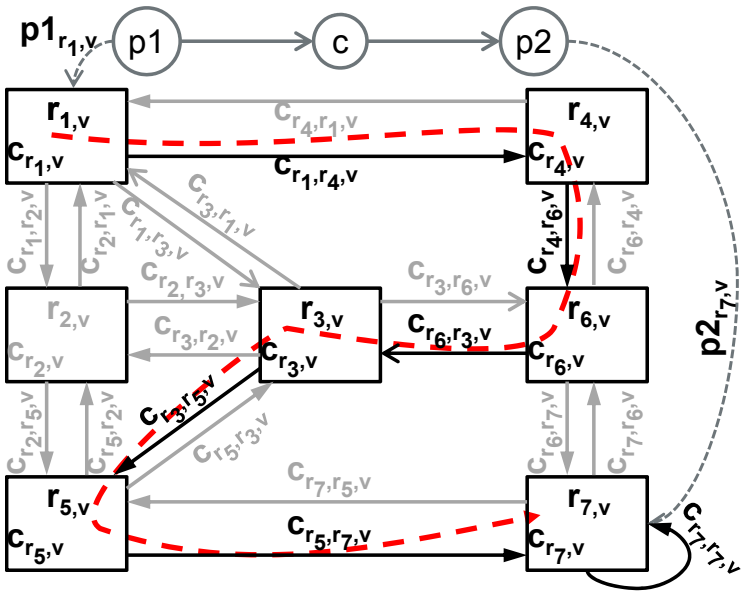


Figure 2: The routing of message c of variant v is encoded with the help of edge-based routing variables $\boldsymbol{c}_{r,r',v}$ and variables $\boldsymbol{c}_{r,v}$. For the given example, 28 variables are required for the routing encoding (20 $\boldsymbol{c}_{r,r',v}$ edges, 1 $\boldsymbol{c}_{r,r,v}$ self-loop, and 7 $\boldsymbol{c}_{r,v}$ for the involved resource). As in Fig. 1, the variable assignment for one valid routing for c (path $r_1 \rightarrow r_4 \rightarrow r_6 \rightarrow r_3 \rightarrow r_5 \rightarrow r_7$ given by the red dashed line) in variant v is marked gray (0) and black (1).

position and, thus, removes the notion of hops. Moreover, this enables a *hop-less* routing encoding being independent from the network diameter but still allows to extract the same *valid* routing paths as the hop-based approach. This significantly decreases the number of variables required to encode the routing while still offering the full flexibility for the system design.

As depicted in Fig. 2, compared to the variables required to encode the hop-based strategy, the principles of the edge-based routing encoding only requires at most $(|C^v| * |R| + |C^v| * |E_R|)$ routing-related variables. As automotive E/E architectures, where a static multi-hop routing has to be determined at design time, are typically not fully meshed, the number of edges typically is much smaller then the cross product $R \times R$ required for the hop-based strategy when offering full routing flexibility. Thus, the number of variables to encode the routing of message $c \in C^v$ is reduced by at least the factor of $\frac{|R|+|R|^2}{|R|+|E_R|}$ compared to the hop-based encoding. In the example in Fig. 2, the edge-based routing encoding requires only 28 variables compared to 56 for the hop-based approach and is further improved if the number of involved resources or hops is increased. In case of sparsely connected architectures with many involved resources, this can easily tend towards more than one order of magnitude. E. g., a single daisy chain of $n = 30$ hops requires $n^2 = 900$ $c_{r,t}$ variables (hop-based) vs. $n = 30$ $c_{r,r}$ variables (edge-based).

For the encoding, we introduce a binary variable $\boldsymbol{c}_{r,r',v}$ for each edge $e = (r, r') \in E_R$ and variant $v \in V$, indicating whether a message $c \in C^v$ is routed (1) over the link between adjacent resources r and r' in variant v or not (0). Furthermore, a self-loop variable $\boldsymbol{c}_{r,r,v}$ indicates if one or more receiver tasks of message c in variant v are bound (1) to resource r or not (0). Thus, a self-loop denotes if resource r is a feasible endpoint of an routing path for message c in variant v. As shown in Fig. 2 for message c of variant v, the self-loop $c_{r_7,r_7,v}$ is activated at the receiving resource r_7, forcing an incoming edge—here $c_{r_5,r_7,v}$—to be activated, too.

First, as a receiving resource r is marked by a self-loop, the corresponding $\boldsymbol{c}_{r,r,v}$ variable has to be set if one or more receivers of message c in variant v are bound to r, see Eq. (1a), and is not allowed to be set if no receiving task is bound within this variant, see Eq. (1b).

$$\forall r \in R, \forall p \in P^v, (p,r) \in E_M; (c,p) \in E_T : \boldsymbol{p}_{r,v} - \boldsymbol{c}_{r,r,v} \leq 0 \tag{1a}$$

$$\forall r \in R : \sum_{(p,r)\in E_M : p \in P^v, (c,p) \in E_T} -\boldsymbol{p}_{r,v} + \boldsymbol{c}_{r,r,v} \leq 0 \tag{1b}$$

To fulfill data-dependencies, each self-loop requires an active incoming edge or the sender itself bound to the resource, see Eq. (2a). Additionally, a sending resource requires at least one outgoing edge if its self-loop is not set, i. e., no receiving task is bound to it, which is ensured by Eq. (2b).

$$\forall r \in R : \boldsymbol{c}_{r,r,v} - \sum_{r' \in R, (r',r) \in E_R} \boldsymbol{c}_{r',r,v} - \sum_{\substack{p_r \in E_M : p \in P^v, \\ (p,c) \in E_T}} \boldsymbol{p}_{r,v} \leq 0 \tag{2a}$$

$$\forall r \in R, p \in P^v, (p,r) \in E_M; (p,c) \in E_T : -\boldsymbol{p}_{r,v} + \sum_{\forall r' \in R, (r,r') \in E_R} \boldsymbol{c}_{r,r',v} + \boldsymbol{c}_{r,r,v} \geq 0 \tag{2b}$$

We have to ensure that in each variant v each activated edge between r and r' has a preceding edge from r'' to r or it starts from a sending resource, i. e., the sender task is bound to r, (Eq. (3a)). Additionally, an incoming edge from r'' to r requires at least one outgoing edge from r to an adjacent resource r' or it ends at a receiving resource r (Eq. (3b)), signaled by an self-loop $\boldsymbol{c}_{r,r,v}$.

$$\forall r \in R, \forall r' \in R, (r,r') \in E_R : -\boldsymbol{c}_{r,r',v} + \sum_{\substack{\forall r'' \in R, \\ (r'',r) \in E_r}} \boldsymbol{c}_{r'',r,v} + \sum_{\substack{\forall p \in P^v, (p,r) \in E_M, \\ (p,c) \in E_T}} \boldsymbol{p}_{r,v} \geq 0 \tag{3a}$$

$$\forall r \in R : - \sum_{\substack{\forall r'' \in R, \\ (r'',r) \in E_R}} \boldsymbol{c}_{r'',r,v} + \sum_{\substack{\forall r' \in R, \\ (r,r') \in E_R}} \boldsymbol{c}_{r,r',v} + \boldsymbol{c}_{r,r,v} \geq 0 \tag{3b}$$

As multicast communication allows receiver tasks $\widetilde{P^v}$ to be bound to the sender as well as to other resources (the set of all mappings $p_r \in E_M$ for tasks $p \in \widetilde{P^v}$ is noted by $M_{\widetilde{P^v}}$), Eq. (4a) forces at least one outgoing edge at the sender, if there is any $p \in \widetilde{P^v}$ not bound to the resource itself. Furthermore, to avoid loops and crossing routes, a resource is allowed to have at most one incoming edge or even none if it is the sending resource, see Eq. (4b).

$$\forall r \in R, (p,r) \in E_M; p \in \widetilde{P^v} = \{p | p \in P^v : (c,p) \in E_T\} : \sum_{\forall p' \in P^v, (p,c) \in E_T} (|M_{\widetilde{P^v}}| \cdot \boldsymbol{p'}_{r,v}) - \sum_{\substack{\forall p \in \widetilde{P^v}; \\ \forall r \in R | r' \neq r}} \boldsymbol{p}_{\boldsymbol{r'},\boldsymbol{v}} + \sum_{\forall r' \in R, (r,r') \in E_R)} |M_{\widetilde{P^v}}| \cdot \boldsymbol{c}_{r,r',v} \geq 0 \tag{4a}$$

$$\forall r \in R, p \in P^v, (p,c) \in E_T : \sum_{\forall (r',r) \in E_r} \boldsymbol{c}_{r',r,v} + \boldsymbol{p}_{r,v} \leq 1 \tag{4b}$$

Finally, to be compatible to the basic model, Equations (5a) and (5b) guarantee to set the variable $\boldsymbol{c}_{r,v}$ if a self-loop or an edge from or to r is activated in variant v.

$$\forall r \in R : \sum_{\substack{\forall r' \in R, \\ (r',r) \in E_r}} (\boldsymbol{c}_{r,v} - \boldsymbol{c}_{r',r,v}) + \sum_{\substack{\forall r' \in R, \\ (r,r') \in E_R}} (\boldsymbol{c}_{r,v} - \boldsymbol{c}_{r,r',v}) - \boldsymbol{c}_{r,r,v} \geq 0 \tag{5a}$$

$$\forall r \in R : \boldsymbol{c}_{r,r,v} - \boldsymbol{c}_{r,v} + \sum_{\substack{\forall r' \in R, \\ (r',r) \in E_r}} \boldsymbol{c}_{r',r,v} + \sum_{\substack{\forall r' \in R, \\ (r,r') \in E_r}} \boldsymbol{c}_{r,r',v} \geq 0 \tag{5b}$$

These instantiation of the previously proposed constraint sets for every $c \in C^v$ and each $v \in V$ allows to efficiently encode unicast as well as multicast communication and guarantee a valid static unicast message routing. But, due to the waiver of expressing hops explicitly, it is not guaranteed for a multicast message that each receiver has a valid route from the sender but may be included in an independent closed loop, see Fig. 3. Thus, in the following the encoding is extended to correctly handle messages with multiple receiving resources.

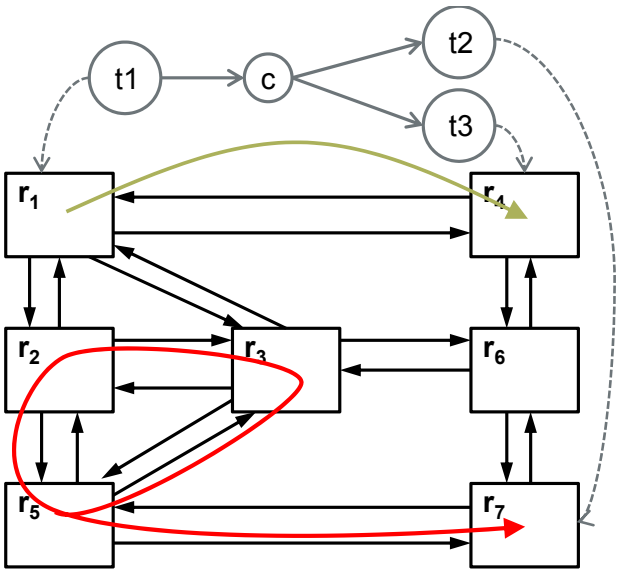


Figure 3: Visualization of the multicast encoding problem. Only one path to one receiving task (here $r_1 \rightarrow r_4$) can be guaranteed to be valid. For a second receiver, it can happen that its message routing may start from a routing-loop as is shown in $r_5 \rightarrow r_3 \rightarrow r_2$, caused by a message multicast in r_5.

2.2. Multicast Edge-based Routing Encoding

To ensure a correct multicast routing encoding, we need to consider each receiving task of a multicast message individually. Therefore, binary variables $\boldsymbol{c}_{r,r',v,p}$ are introduced, marking the usage of an edge $\boldsymbol{c}_{r,r',v}$ for the routing of message c to receiving task $p \in P^v, (c,p) \in E_T$ (1) or not (0). First, Eq. (6a) guarantees that the variable $\boldsymbol{c}_{r,r',v}$ is set if any $\boldsymbol{c}_{r,r',v,p}$ is set.

$$\forall r \in R, (r', r) \in E_R : \boldsymbol{c}_{r',r,v} - \boldsymbol{c}_{r',r,v,p} \geq 0 \tag{6a}$$

Additionally, most of the required constraints are related to the ones presented in the basic edge-based routing encoding with slight adaptions and the usage of the new $\boldsymbol{c}_{r,r',v,p}$ variables. Thus, Equations (7a)–(7g) have the same purpose as Equations (1a)–(4a). Therefore, each multicast message is additionally encoded as multiple individual single-cast communications to each receiver, while the basic encoding guarantees their correct merge to a valid multicast routing. The following linear constraints need to be instantiated for all $v \in V$ and for every $c \in C^v$ that has more than one receiving task and are applied for all receiving tasks $p \in P^v, (c,p) \in E_T$ of message c:

$$\forall r \in R, (p, r) \in E_M : \boldsymbol{p}_{r,v} - \boldsymbol{c}_{r,r,v,p} \leq 0 \tag{7a}$$

$$\forall r \in R : -\boldsymbol{p}_{r,v} + \boldsymbol{c}_{r,r,v,p} \leq 0 \tag{7b}$$

$$\forall r \in R : \boldsymbol{c}_{r,r,v,p} - \sum_{\substack{\forall r' \in R, \\ (r',r) \in E_R}} \boldsymbol{c}_{r',r,v,p} - \sum_{\substack{\forall (p,r) \in E_M, \\ (p,c) \in E_T}} \boldsymbol{p}_{r,v} \leq 0 \tag{7c}$$

$$\forall r \in R, (p', r) \in E_M : (p', c) \in E_T : -\boldsymbol{p'}_{r,v} + \sum_{\forall r' \in R, (r,r') \in E_r} \boldsymbol{c}_{r,r',v,p} + \boldsymbol{c}_{r,r,v,p} \geq 0 \tag{7d}$$

$$\forall r \in R, (r, r') \in E_R : -\boldsymbol{c}_{r,r',v,p} + \sum_{\forall (r',r) \in E_r} \boldsymbol{c}_{r',r,v,p} + \boldsymbol{p}_{r,v} \geq 0 \tag{7e}$$

$$\forall r \in R : \boldsymbol{c}_{r,r,v,p} - \sum_{\forall (r',r) \in E_r} \boldsymbol{c}_{r',r,v,p} + \sum_{\substack{\forall r' \in R, \\ (r,r') \in E_r}} \boldsymbol{c}_{r,r',v,p} \geq 0 \tag{7f}$$

$$\forall r \in R, (p,r) \in E_M : \sum_{\substack{\forall r' \in R, \\ (r,r') \in E_R}} |M_p| \cdot \boldsymbol{c}_{r,r',v,p} + \sum_{\substack{\forall p \in P^v; \\ (p,c) \in E_T}} (|M_p| \cdot \boldsymbol{p}_{r,v}) - \sum_{\substack{\forall r' \in R; \\ r' \neq r}} \boldsymbol{p}_{\boldsymbol{r'},\boldsymbol{v}} \geq 0 \tag{7g}$$

In overall, the edge-based routing encoding guarantees a correct combined multicast routing with absence of routing loops, multiple paths to the same resources, etc. while being much more compact than the hop-based approach. The only weakness is that it is not capable of prohibiting independent closed cycles that are not in contact with any used (multicast) paths. These static cycles can be easily removed by a trivial repair step. Thus, the advantages of the proposed edge-based routing encoding easily outweigh this minor weakness.
As the experimental results in the next section show, the proposed edge-based routing encoding strategy gives convincing optimization results and significantly reduces the number of variables required to encode the multi-variant specification for a multi-objective optimization of an E/E architecture component platform for future car series.

3. Experimental Results

Comparable to the work proposed in [GGTL14b], we define our use case as a multi-variant specification by explicitly integrating multiple functional variants and an architecture template giving the overall freedom for the component platform design [2]. In overall, our presented use case $u1$ has 8 predefined functional variants defined upon 49 tasks and 47 exchanged messages. Additionally, the architecture template consists of 141 routing-related resources, i. e., they are able to route messages, like communication controller, processors, and actuators, e. g., defining the resources within an *airbag ECU* component and a *2nd ECU* component not further discussed here, but element of the architecture component platform that has to be optimized globally. For the comparison and to limit the overall complexity of the hop-based routing encoding, we restrict its used network diameter n to 15. Beside the use case $u1$, we further provide some results for an additional internal use-case $u2$.
For the E/E architecture component platform optimization, we used the following four important design objectives that are relevant for the real-world development to be minimized:

overall monetary hardware cost
: We calculate the overall hardware cost to implement all variants with their estimated equipment rates by building a weighted sum of the individual hardware cost per variant.

number of *airbag ECU* manifestations
: The number of manifestations of the *airbag ECU* is one important impact factor for the component platform.

number of *2nd ECU* manifestations
: The number of manifestations of a *2nd ECU* within the component platform.

[2] Due to reasons of secrecy, we cannot give detailed information on the used application and architecture models as well as the equipment rate for each variant. This detailed use case descriptions have to be excluded from this work. But, as the proposed approach was tested with multiple real use cases, the results were confirmed by all of them.

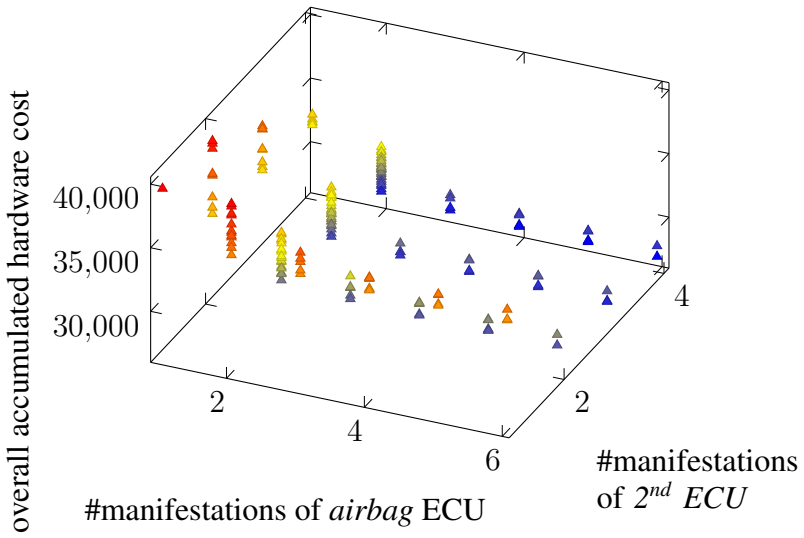


Figure 4: Pareto-plot of the results of the presented use case as a three-dimensional projection to the both manifestation dimensions and overall costs. Due to the projection, each pareto point is colored based on its overall cost: from blue for low cost to red representing a high overall cost. The component platform optimization approach enables to ponder between different number of manifestations for the inspected hardware components like ECUs and their effect to the overall cost and is significantly enhanced by the newly proposed edge-based message routing encoding.

difference from overall allocation

To guide the optimization towards solutions with a low number of manifestations, we add the number of resources that are not used within each functional variants allocation as an optimization goal.

Results for the component platform optimization of $u1$ are given in Fig. 4 as a three-dimensional projection of the pareto-front to the design objectives of *manifestations of the airbag ECU*, the regarded *2nd ECU*, and, as most important, *the overall accumulated monetary hardware cost*. It could be seen that, despite the defined 8 functional variants, the optimization results in several architecture component platforms that all require less manifestations of the *airbag ECU*. Also, a small number of manifestations to implement the 2nd ECU are sufficient to reach proper overall costs. Therefore, developers are able to ponder between their overheads to develop and maintain several manifestations and the overall hardware cost to build all car variants based on the optimized E/E architecture component platform. As it can be seen that the impact of a very low number of manifestations of the two components is very high, it is definitely worth developing at least two manifestations per ECU component and, thus, set up a E/E architecture component platform.
Beside the powerful extraction of optimized *solutions* given as a proper pareto-front, we compare the proposed edge-based symbolic routing encoding with the hop-based approach known from literature [GGTL14b, LSG+09]. To compare both, we applied the encodings to our used case study and evaluate their optimization quality by taking into account the average ϵ-dominance [LTDZ02] of 10 optimization runs. This is an indicator representing the convergence of the optimization, i. e., the average difference to the best known design points during the optimization process. Lower values means that the results of the optimization (for the current optimization step) are closer to the optima and, thus, outperform an approach that results in higher values. Additionally, we give quality numbers for two use cases based on the number of required variables and constraints to encode the whole optimization problem as an indicator for scalability and efficiency of the newly proposed symbolic static message routing encoding.

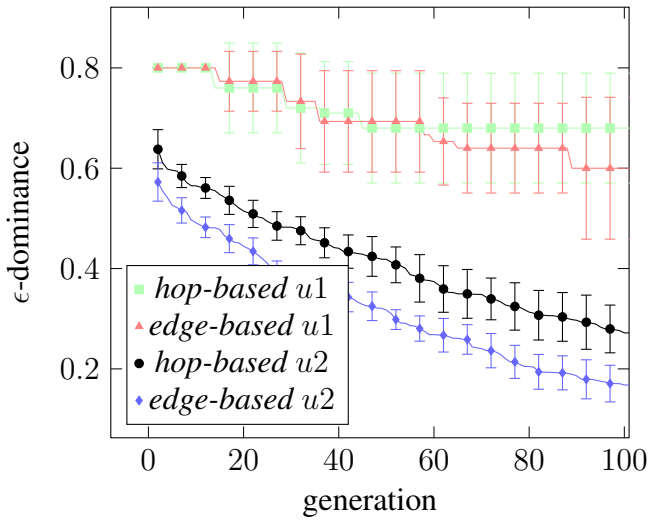


Figure 5: Convergence of the *hop-based* [GGTL14b] and the proposed *edge-based* routing encoding for both E/E architecture component platform use cases $u1$ and $u2$ as the ϵ-dominance for 100 generations, i. e., iterative optimization steps of an evolutionary algorithm, using SAT-decoding [LGHT07]. The optimization quality for the *hop-based* encoding is outperformed by the *edge-based* routing encoding approach.

As is shown in Fig. 5, the proposed encoding outperforms previous approaches by faster converging, i. e., the average ϵ-dominance goes down faster compared to the hop-based approach. I. e., after 60 generations of the optimization heuristics, edge-based already gives better average results for $u2$ than hop-based at the end of the whole optimization run (in this case 100 generations, i. e., iterations of the heuristic). Additionally, due to the reduced complexity of the encoding, our approach requires less runtime to setup the constraints as well as to synthesize solutions.
For the encoding complexity, the results are given in Table 1. It can be seen that for our real-world automotive use cases with lots of routing possibilities, our approach significantly reduces the number of required variables and constraints by more than 90%, while still representing the same design space due to the more effective routing determination. Moreover, as all the omitted variables would have been relevant for the static routing determination, our approach removes lots of unneeded variables, i. e., the proposed encoding is much more compact and efficient. Beside positive effects to the optimization process (as seen before), this also enlarges the processable problem sizes of the overall optimization process. Just as a short reminder, as a constraint typically consists of several instances of variables, the given example easily results in a memory demand to only store the constraints of the hop-based static message routing encoding of more then 1 gb. Thus, our new message routing encoding approach offers a much better encoding scalability and therefore is an enabler towards an automatic E/E architecture component platform optimization of the full automotive E/E architecture comprising much more binding, allocation, and of course static message routing tasks during system synthesis.

Table 1: Required number of variables and constraints to encode the multi-variant use-cases.

	presented use-case $u1$		internal use-case $u2$	
routing encoding strategy	#variables	#constraints	#variables	#constraints
hop-based	546,359	2,006,589	1,711,861	5,380,121
edge-based	49,082	65,216	66,649	89,409
seach-space reduction	91%	96.7%	96.1%	98.3%

ISBN 978-3-00-048889-4

4. Conclusion

The work at hand proposes an edge-based symbolic routing encoding enabling multi-objective optimization of future multi-variant automotive E/E architecture component platforms with their rising demands for the static message routing and increasing lengths of the paths. As shown in the experimental results, the edge-based approach allows to optimize multiple components in parallel while tremendously decreasing the number of variables, i. e., covering the same design space with a smaller search space. This more compact encoding improves the optimization quality, but even more enlarges the area of use towards the whole E/E architecture of upcoming cars.

References

[GGTL14a] Graf, Sebastian, Michael Glaß, Jürgen Teich, and Christoph Lauer: *Design Space Exploration for Automotive E/E Architecture Component Platforms*. In *Proc. of DSD*, pages 651–654, 2014.

[GGTL14b] Graf, Sebastian, Michael Glaß, Jürgen Teich, and Christoph Lauer: *Multi-Variant-based Design Space Exploration for Automotive Embedded Systems*. In *Proc. of DATE*, pages 7:1–7:6, 2014.

[GHP+09] Gerstlauer, A., C. Haubelt, A.D. Pimentel, T.P. Stefanov, D.D. Gajski, and J. Teich: *Electronic System-Level Synthesis Methodologies*. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 28(10):1517–1530, 2009.

[GLT+09] Glaß, Michael, Martin Lukasiewycz, Jürgen Teich, Unmesh D. Bordoloi, and Samarjit Chakraborty: *Designing heterogeneous ecu networks via compact architecture encoding and hybrid timing analysis*. In *Proc of DAC*, pages 43–46, 2009.

[GRGT14] Graf, Sebastian, Felix Reimann, Michael Glaß, and Jürgen Teich: *Towards scalable symbolic routing for multi-objective networked embedded system design and optimization*. In *Proc. of the CODES+ISSS*, pages 2:1–2:10, 2014.

[HM03] Hu, Jingcao and R. Marculescu: *Exploiting the routing flexibility for energy/performance aware mapping of regular noc architectures*. In *Proc. of DATE*, pages 688–693, 2003.

[LGHT07] Lukasiewycz, Martin, Michael Glaß, Christian Haubelt, and Jürgen Teich: *SAT-Decoding in Evolutionary Algorithms for Discrete Constrained Optimization Problems*. In *Pro. of CEC*, pages 935–942, 2007.

[LKH10] Lee, Choonseung, Sungchan Kim, and Soonhoi Ha: *A systematic design space exploration of mpsoc based on synchronous data flow specification*. J. SPS, 58(2):193–213, 2010.

[LSG+09] Lukasiewycz, Martin, Martin Streubühr, Michael Glaß, Christian Haubelt, and Jürgen Teich: *Combined System Synthesis and Communication Architecture Exploration for MPSoCs*. In *Proc. of DATE*, pages 472–477, 2009.

[LTDZ02] Laumanns, Marco, Lothar Thiele, Kalyanmoy Deb, and Eckart Zitzler: *Combining convergence and diversity in evolutionary multiobjective optimization*. Evolutionary computation, 10:263–282, 2002.

[SSJ06] Simpson, TimothyW., Zahed Siddique, and Jianxin (Roger) Jiao: *Platform-based product family development*. In *Product Platform and Product Family Design*, pages 1–15. 2006.

[SVDN07] Sangiovanni-Vincentelli, A. and M. Di Natale: *Embedded system design for automotive applications*. IEEE Computer, 40(10):42–51, 2007.

ISBN 978-3-00-048889-4

Model-based Systems Engineering with Matlab/Simulink in the Railway Sector

Alexander Nitsch
Universität Rostock
Alexander.Nitsch@uni-rostock.de

Benjamin Beichler
Universität Rostock
Benjamin.Beichler@uni-rostock.de

Frank Golatowski
Universität Rostock
Frank.Golatowski@uni-rostock.de

Christian Haubelt
Universität Rostock
Christian.Haubelt@uni-rostock.de

Abstract

Model-based systems engineering is widely used in the automotive and avionics domain but less in the railway domain. This paper shows that Matlab/Simulink can be used to develop safety-critical cyber-physical systems for railway applications. To this end, an executable model has been implemented which allows for train movement simulation such as automatic emergency braking.

Keywords: model-based, cyber-physical system, Matlab/Simulink, safety-critical, executable model

Acknowledgement: This work was funded by the German Federal Ministry of Education and Research (Grant No. 01IS12021) in the context of the ITEA2 project openETCS.

1. Introduction

Model-based system engineering has proven to be a well suited methodology to develop embedded systems and especially safety-critical cyber-physical systems. Model-based approaches are widely used in the automotive and avionics domain but still uncommon in the railway sector. The increasing complexity of software in locomotive on-board units renders software development with traditional methods nearly impossible. We propose model-based engineering techniques as a means to ease this process.
One critically safety-relevant software module is the control system of the train. Protection systems like this are getting developed since the very beginning of railway operation. Consequently, trains run by different countries use mostly non-interoperable train control systems. Especially in the converging European Union this leads to a problem: all trains that need to cross borders also need to be equipped with several expensive train control systems.
The European Train Control System (ETCS), which was designed in the early 1990s, is the designated solution to overcome this problem within the European borders. ETCS includes a set of modern concepts for train control to achieve high speed and high utilization of the rail. Besides this, ETCS aims to be flexible to address all requirements of the national railway operators. The resulting ETCS standard became rather complex and, as the standard currently is only available as natural, non-formal language document, is very difficult to implement. Consequences of this are

ISBN 978-3-00-048889-4

high development costs and incompatible implementations of different vendors caused by ambiguities of the specification.
In this environment, the openETCS project was created with the goal of an open source implementation of the on-board unit software. To achieve this, model-based systems engineering methods are employed. In this paper we present our efforts to analyze and develop the Speed and Distance Monitoring, which is part of the ETCS standard. The chosen modelling tool is Simulink, which is widely used in industrial applications especially in the automotive sector. The result of this paper is an executable model of the braking curve calculation, which is part of the Speed and Distance Monitoring. Moreover, the developed model offers a simulation framework for the train movement characteristics.
This paper is structured as follows. After the motivation of the topic in section 1, section 2 gives an overview of railway related publications. Section 3 introduces the basics of the European Train Control System and presents the Speed and Distance Monitoring as a subsystem of ETCS. Braking curves are used to predict the movement behavior of a train especially in case of emergency. The principle of the Emergency Brake Deceleration (EBD) curve and its calculation in the form of an executable model are described in section 4. As a case study, the model of the EBD calculation is used in section 5 to simulate the braking behavior of a moving train. Section 6 summarizes the acquired knowledge and briefly discusses future work.

2. Related Work

Since the first release of the ETCS standard several publications examined different aspects of the ETCS specification. Many of them deal with real-time properties and reliability of the communication link between train and track-side equipment. In [zHHS13, JzHS98, ZH05, HJU05] Petri net extensions are used to investigate the functional properties and stochastic guarantees of the communication. Modeling and calculation of speed and distance monitoring of ETCS were covered in [BT11, Fri10]. These works focus on the functional properties of the computation and use of an application-specific modeling methodology.
Other publications in the ETCS context focus on formalization and safety analysis. The authors in [CPD+14] show in three case studies how formal languages can ease the verification process of safety-critical systems. They show how the SPARK language and its toolset can be integrated into the existing development process to decrease the effort of system certification in the railway domain. In [CA14] a formal model in form of a Time Extended Finite State Machine is developed. This model is used to represent safety properties of the ETCS requirements and allows to derive tests for checking these properties. Within the scope of model-based systems engineering, SysML is a widely used language for graphical system description [OMG12]. A large number of publications use SysML to describe, test and verify architectural and functional system properties in context of ETCS and generally railway, e.g. [BFM+11, MFM+14, BHH+14].However, SysML cannot produce executable code. Simulink is another graphical programming environment for model-based design and in contrast to SysML, it enables simulation of dynamic systems and provides automatic code generation for the integration into other applications [Mat14]. This paper focuses on Simulink to develop a ETCS related model which is executable and therefore usable for dynamic analysis tasks such as train movement.
To our best knowledge, no comparable solution exists for train movement analysis and simulation with respect to conformity of the ETCS standard.

ISBN 978-3-00-048889-4

3. ETCS - Speed and Distance Monitoring

One of the main tasks of ETCS is to supervise the speed and position of trains to ensure that the train stays in the permitted speed ranges. Because of the low friction between steel wheels and rail and the relative high mass of the train, the braking distance is very large compared with e.g. automobiles. As a consequence, a human train driver is not even able to perceive the brake distance on the track including railway signals or other trains.
An established approach in train control systems are track side equipment like multiple combined signals and mutual exclusive track usage of trains. The size of the track segments significantly effects the utilization and possible throughput and therefore the profitability of a track. Since the signal equipment is fixed at the track side, a customization for different rolling stock is effectively impossible. This becomes a serious problem if trains with significant different maximum speed and braking abilities are used on a track.
To prevent a human failure of the perception of such safety-critical information, all modern train control systems must have an automatic intervention possibility for dangerous situations. More sophisticated train control systems like ETCS make usage of customized signaling with displays within the train cab. This so called "cab signalling" helps to customize the speed and distance limits for every train. The challenge of such a calculation on the onboard unit of the train control system is to ensure the safe operation of the train. This includes the functional safety and the time critical aspects of this calculation speed and distance limits.

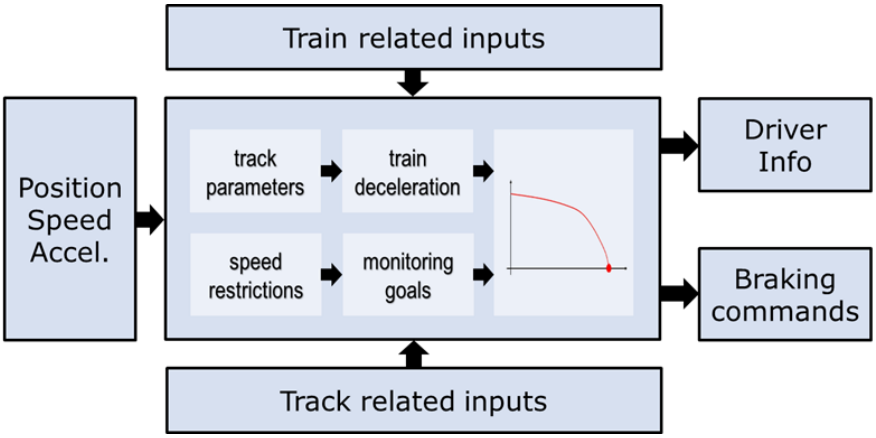


Figure 1: Overview of the ETCS Speed and Distance Monitoring

An overview of the SaDM is shown in Figure 1. The tasks of the Speed and Distance Monitoring (SaDM) are defined within the System Requirements Specification [UNI12] within chapter 3.13. The main results of the SaDM are information for the driver, e.g. the current permitted speed, monitoring targets and for critical situations the SaDM issues automatic braking commands.
In order to determine this information, SaDM needs several inputs such as the dynamic values of current position, speed and acceleration of the train. Moreover a certain number of other train and track related inputs are needed, which have a lower dynamic as position or speed. The most important train related inputs are the braking abilities of a train.
Modern trains have multiple sets of brakes, which have different operating principles, and are used in several combinations according to various conditions. According to this the applicable braking deceleration in a dangerous situation needs to be defined for all possible combinations.

ISBN 978-3-00-048889-4

Other important characteristics such as curve tilt abilities, maximum train speed or the train length are also needed to be considered in order to calculate the train dependent impact on the speed and distance limits. All train related inputs are combined to a function called A_{safe}, that assigns a braking acceleration to the two independent parameters of speed and distance. All described inputs are piece-wise constant functions or so called step function of speed or position, so that the $A_{safe}(v, d)$ have also the characteristics of a step function in a two dimensional manner.
Beside the train characteristics, the track related information is the other important input data. A train equipped with ETCS receives information about the track properties while moving on it. This includes a profile of the track slopes and a set of static speed restrictions, which are caused by the shape of a track. Furthermore dynamic speed restrictions (e.g. in areas which are under maintenance) are transmitted to the train. This collection of location based data defined speed restrictions are compressed to a single data structure called Most Restrictive Speed Profile (MRSP), which contains a single allowed speed for every position on the track ahead. From this profile the particular targets for the supervision are derived by getting all points with a decreasing allowed speed. An additional special target is derived from the limited permission of a train to move on the track. This End of Authority is derived from the Movement Authority, which needed to be transmitted to the train while operation.
Every of the described supervision targets are forwarded to the calculation of the target specific braking curve. To predict the behavior of the train in an emergency case the Emergency Brake Deceleration (EBD) curve is one of the most important calculations, which is therefore in the focus of following sections.

4. Braking Curve Calculation

In this section, we detail the braking curve calculation, which is part of the Speed and Distance Monitoring and presented as Simulink model. To best of our knowledge, this is first executable model derived from ETCS SRS.

4.1. EBD Calculation

The Emergence Brake Deceleration curve (EBD) is called the parachute of ETCS because this curve represents the braking behavior in case of emergency. In case of emergency and from a certain speed the system has to use all available brakes to reach zero speed at a concrete location. In addition, there exist several constraints, e.g. there is a slippery track, which leads to a reduced brake performance, or the system is not able to use all brakes but only a specific combination, which also results in a reduced brake performance, the system has to calculate the position of brake initiation to reach the target position under any circumstances. The influence of the brake performance on the braking distance is shown in Figure 2.
By a lower brake performance (1) the train will not stop at the desired position on track, that means the system has to brake earlier to stop at the desired position. In contrast to that, a higher brake performance will earlier stop the train (2) or the initial braking can be done later. The braking distance depends on a brake deceleration value (3).
If the stop location and the brake performance on each section of the track are known, the latest possibility of braking can be calculated to stop at the desired position. Hence, there is a need of a

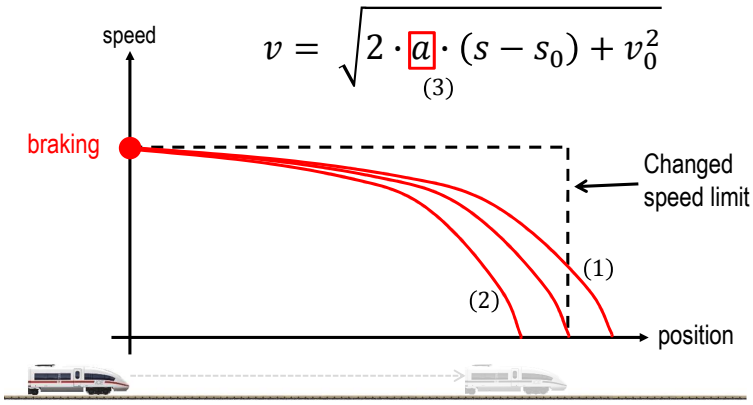


Figure 2: Brake performance and its influence on the brake distance

backward calculation algorithm, which starts its calculation from the target location and calculates backwards to the front end of the train Figure 3.

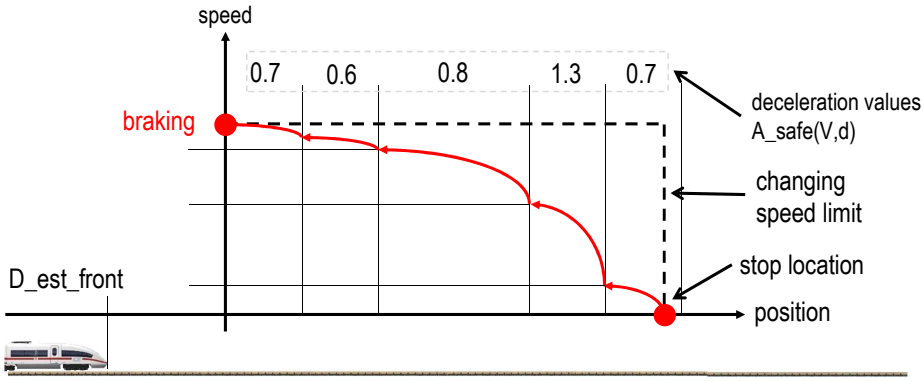


Figure 3: Backward calculation of the brake initiation depending on brake performance

The result of the algorithm is the maximum speed of the train on a specific position on track. By exceeding this speed limit the train will fail to stop at the desired location. This information is known as EBD. After knowing the maximum speed in comparison to the actual speed, the ETCS onboard computer can intervene and brake automatically.

4.2. Simulink Model for the EBD Calculation

For the calculation of the EBD curve a Simulink model has been implemented. The algorithm for the calculation process can be seen in Figure 4. For a given target distance the algorithm calculates the maximum allowed speed of the train to stop at that target location. The Simulink model use numerous inputs, provided by a balise, which is integrated in the rail bed in front of a possible target. The inputs are: the distance to the target location (`d_target`), the desired speed at the target location (`V_target`) and the estimated front end (`d_est_front`) of the train (distance covered yet).
Another input is a two dimensional step function organized as an array named `A_safe(V,d)` containing information about deceleration values (`curAcc`) of the train depending on the track position (`curDis`) and the speed of the train (`curVel`). Therefore a specific deceleration value depending on both, a particular speed and position category (`dis_cat`, `vel_cat`) is returned.

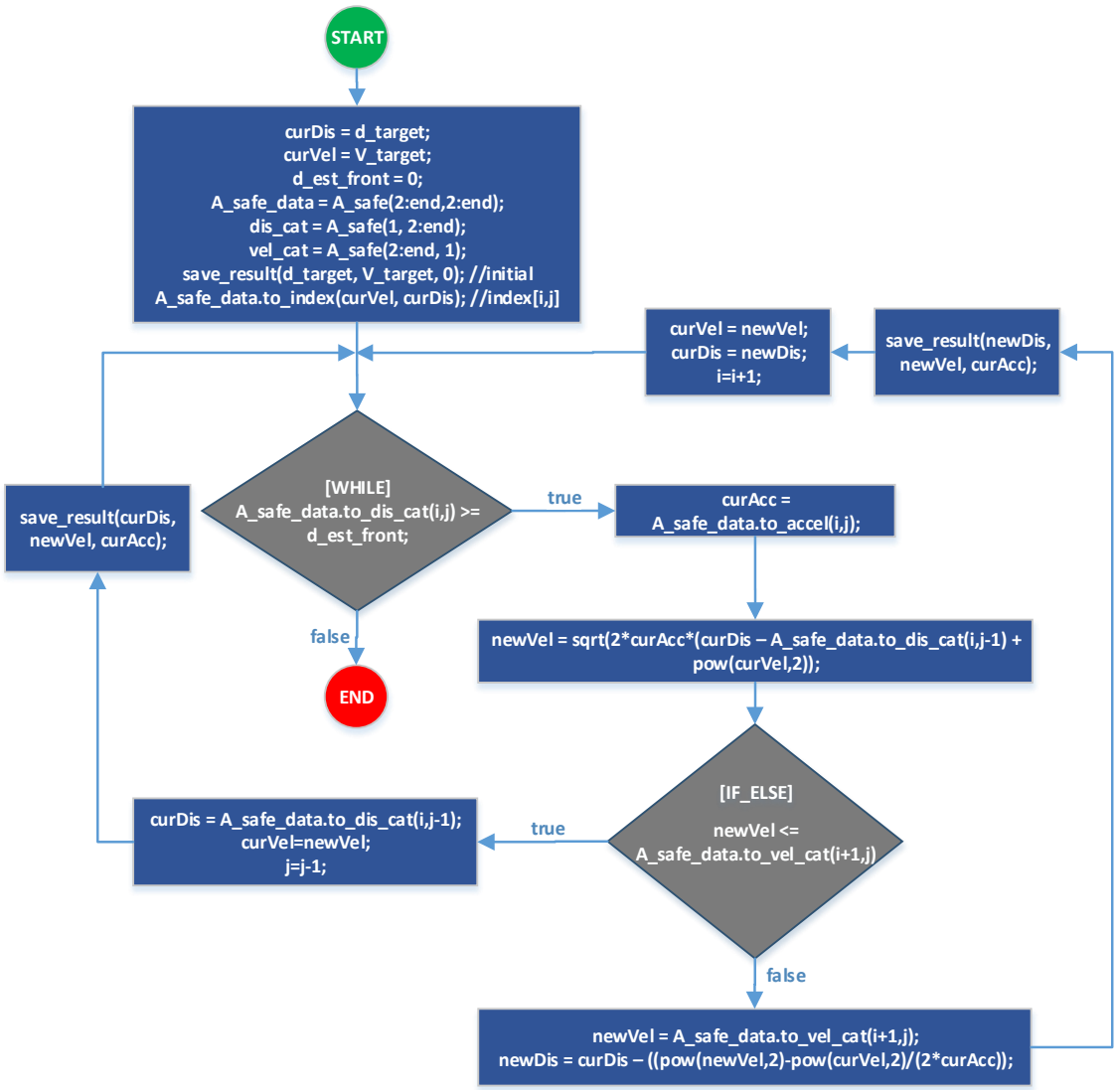


Figure 4: Algorithm of the EBD curve calculation

Based on the given inputs the Simulink model calculates iteratively the maximum allowed speed of the train regarding to a specific position on track, which must not been exceeded by the train to stop at the desired target location.

The structure of an example `A_safe`-data-set is depicted in Figure 5. This function contains a matrix (gray) which represents the deceleration values of the train in m/s^2, for a distance category in m (blue) and a speed category vector m/s (orange). Additionally, the left axes represents the speed category index (`i`) and the upper axes represents the distance category index (`j`).

Now the EBD calculation starts as follows. The deceleration value in the target region has to be determined at first. Therefore within a Simulink look-up-table the given target distance in the distance vector is approximated (`A_safe_data.to_index(curVel,curDis)`) and also the relevant index is returned. Because zero speed at the target is considered, the first deceleration value can be derived at the speed index zero and the distance category index selection which depends on the target distance. Thereafter, the the maximum allowed speed is calculated by the formula which is given in Figure 2. The resulting speed (`newVel`) is calculated until the next lower distance

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
		0	15	30	45	60	75	90	105	120	135	150	165	180	195	210
0	0,00	0,8	0,8	0,8	1	1	1	0,8	0,8	0,8	1	1	1	1,2	1,2	1,2
1	5,55	0,8	0,8	0,8	1	1	1	0,8	0,8	0,8	1	1	1	1,2	1,2	1,2
2	11,11	0,8	0,8	0,8	1	1	1	0,8	0,8	0,8	1	1	1	1,2	1,2	1,2
3	16,66	0,8	0,8	0,8	1	1	1	0,8	0,8	0,8	1	1	1	1,2	1,2	1,2
4	22,22	0,8	0,8	0,8	1	1	1	0,8	0,8	0,8	1	1	1	1,2	1,2	1,2
5	27,77	0,7	0,7	0,7	0,9	0,9	0,9	0,7	0,7	0,7	0,9	0,9	0,9	1,1	1,1	1,1
6	33,33	0,7	0,7	0,7	0,9	0,9	0,9	0,7	0,7	0,7	0,9	0,9	0,9	1,1	1,1	1,1
7	38,88	0,7	0,7	0,7	0,9	0,9	0,9	0,7	0,7	0,7	0,9	0,9	0,9	1,1	1,1	1,1
8	44,44	0,7	0,7	0,7	0,9	0,9	0,9	0,7	0,7	0,7	0,9	0,9	0,9	1,1	1,1	1,1

Figure 5: Example `A_safe`-data

category index because at this position may the deceleration value change and consequently the brake performance also change.
Due to the two dimensional characteristic of the `A_safe`-data the brake performance also depends on speed. The algorithm has to check whether the new calculated maximum speed matches into the related speed category, which has been realized through an Simulink If-Condition-Block. There are two possible scenarios. If the calculated maximum speed is lower or equal to next speed stage, the speed category index is incremented by one, the distance category index will be decremented and its value represents the new distance as input to the next iteration of the algorithm. The else-case is triggered if the calculated maximum speed is greater than the next speed stage. It must be assumed that there is change in deceleration value on that speed level. In consequence of that, the concrete position regarding to the actual speed level have to be calculated by transposing the formula which is given in Figure 2. The results are saved in a table and the backward calculation algorithm processes as long as the actual train position is reach.

5. Case Study

In this section, the Simulink model of the EBD calculation is used to simulate the braking behavior of a moving train in case of a speed limit change to zero speed Figure 6. The simulation is interactive, therefore the user is able to manipulate the train movement by setting the actual acceleration value of the train. A positive value will accelerate the train to a desired speed, zero acceleration leads to a constant speed and a negative acceleration is used to decelerate the train until zero speed is reached and the train stands still. The outputs of the train movement block are the actual train speed and the actual train position on the track.
At the start of the simulation, the braking curve of the whole track is calculated. This curve represents the upper speed limit of the train to really stop at the desired location. By reaching the EBD curve the train will automatically brake by using the deceleration of the `A_safe`-data, which depends on the actual speed and track position of the train. The output of the EBD Calculator is a matrix which consists of 4 column vectors: start and end position of a deceleration section, its corresponding deceleration value and the calculated maximum speed at the end of each section.
The EBD Sampler calculates, corresponding to actual train position, a maximum speed value by using the formula which is presented in Figure 2. The maximum speed regarding to a specific position is given as an output to the Speed Limiter. The Speed Limiter compares the actual speed of the train with the braking curve speed limit and feeds back a boolean value to the train movement.

ISBN 978-3-00-048889-4

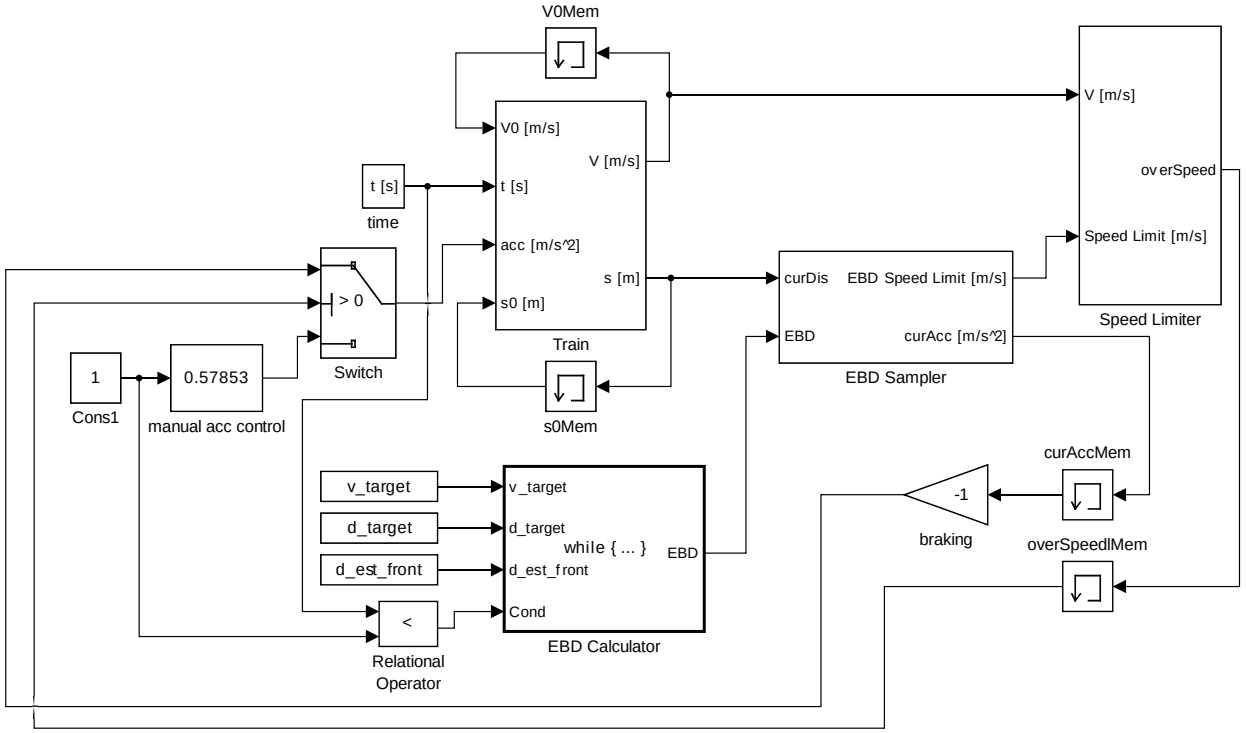


Figure 6: Train movement simulator block diagram

In case of equality of both values, the boolean is set to one. The acceleration input switches from manual user acceleration control to automatic braking. Consequently the train slows down and stops at the target location. The result of an example test case is depicted in Figure 7. Due to several impacts like the brake built up time and possible other tractional acceleration a certain safety margin is subtracted from the EBD. Therefore the both curves are not congruent while the automatic intervention.

6. Conclusion and Future Work

This paper has shown that model-based system engineering is suitable to develop complex safety-critical cyber-physical systems of the railway domain. We have proven that the desired functionality can be realized with Matlab/Simulink. Simulink was used to implement an executable model to calculate the Emergency Brake Deceleration curve, which is an important outcome of the Speed and Distance Monitoring. Additionally a simulation framework for the analysis of train movement was realized. With that result, we are now able to test different scenarios for automatic braking. To the best of our knowledge, this is the first executable model of the Speed and Distance Monitoring of the new European Train Control System. This model will be used in future experiments to verify and design parts of ETCS trains onboard unit on model-based approaches.

ISBN 978-3-00-048889-4

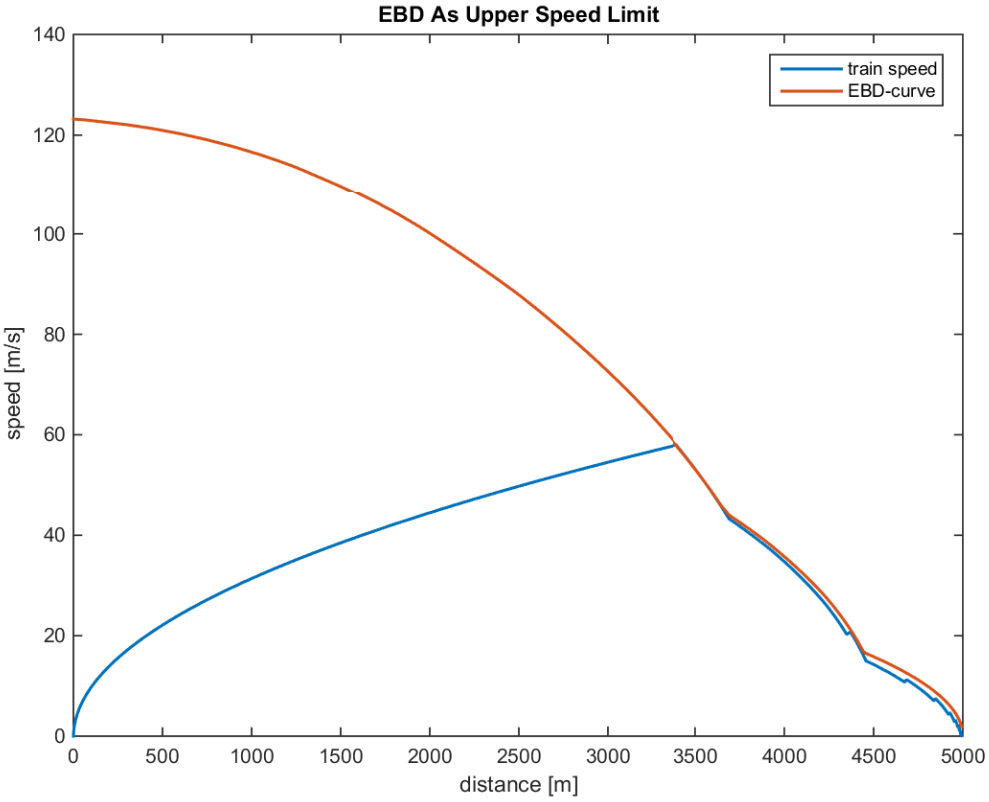


Figure 7: Simulation result of an example test case

References

[BFM+11] Bernardi, Simona, Francesco Flammini, Stefano Marrone, José Merseguer, Camilla Papa, and Valeria Vittorini: *Model-driven availability evaluation of railway control systems*. In Flammini, Francesco, Sandro Bologna, and Valeria Vittorini (editors): *Computer Safety, Reliability, and Security*, volume 6894 of *Lecture Notes in Computer Science*, pages 15–28. Springer Berlin Heidelberg, 2011.

[BHH+14] Braunstein, Cécile, AnneE. Haxthausen, Wen ling Huang, Felix Hübner, Jan Peleska, Uwe Schulze, and Linh Vu Hong: *Complete model-based equivalence class testing for the etcs ceiling speed monitor*. In Merz, Stephan and Jun Pang (editors): *Formal Methods and Software Engineering*, volume 8829 of *Lecture Notes in Computer Science*, pages 380–395. Springer International Publishing, 2014.

[BT11] B.Vincze and G. Tarnai: *Development and analysis of train brake curve calculation methods with complex simulation*. Advances in Electrical and Electronic Engineering, 5(1-2):174–177, 2011.

[CA14] C. Andrés, A. Cavalli, N. Yevtushenko J. Santos R. Abreu: *On modeling and testing components of the european train control system*. In *International Conference on Advances in Information Processing and Communication Technology - IPCT 2014*. UACEE, 2014.

[CPD+14] Claire Dross, Pavlos Efstathopoulos, David Lesens, David Mentré, and Yannick Moy: *Rail, space, security: Three case studies for SPARK 2014*. Toulouse, February 2014.

[Fri10] Friman, B.: *An algorithm for braking curve calculations in ertms train protection systems*. Advanced Train Control Systems, page 65, 2010.

[HJU05] Hermanns, H., D.N. Jansen, and Y.S. Usenko: *A comparative reliability analysis of etcs train radio communications*, February 2005. AVACS Technical Report No. 2.

[JzHS98] Jansen, L., M. M. zu Hörste, and E. Schnieder: *Technical issues in modelling the european train control system*. Proceedings of the workshop on practical use of coloured Petri Nets and Design /CPN 1998, pages 103–115, 1998.

[Mat14] Mathworks: *Simulink - Simulation and Model-based Design, Version R14*. 2014.

[MFM+14] Marrone, Stefano, Francesco Flammini, Nicola Mazzocca, Roberto Nardone, and Valeria Vittorini: *Towards model-driven v&v assessment of railway control systems*. International Journal on Software Tools for Technology Transfer, 16(6):669–683, 2014.

[OMG12] OMG, Object Management Group: *Systems Modeling Language (SysML), Version 1.3 Reference Manual*. 2012.

[UNI12] UNISIG: *SUBSET-026 – System Requirements Specification*. Srs 3.3.0, ERA, 2012.

[ZH05] Zimmermann, A. and G. Hommel: *Towards modeling and evaluation of etcs real-time communication and operation*. Journal of Systems and Software, 77(1):47–54, 2005.

[zHHS13] Hörste, M.M. zu, H. Hungar, and E. Schnieder: *Modelling functionality of train control systems using petri nets*. Towards a Formal Methods Body of Knowledge for Railway Control and Safety Systems, page 46, 2013.

A new Mapping Method from Fuzzy Logic System into Fuzzy Automaton

Lei Yang, Erik Markert, Ulrich Heinkel
Professur Schaltkreis- und Systementwurf, TU Chemnitz
Chemnitz, Germany
lei.yang@etit.tu-chemnitz.de

Abstract

The fuzzy method is a good way to solve some complex classification or control problems. This paper introduces a new method which maps a fuzzy logic system into a fuzzy automaton, terminateand defines a new type of automaton: the fuzzy logic tree automaton. It manages to involve fuzzy sets definition, fuzzy rules and fuzzy inference methods into a new type of tree automaton. This will lead to a fast calculation of a fuzzy logic system with more inputs and outputs. It also provides a new way for easily adding or deleting input/output data of a fuzzy automaton.

1. Introduction

With the fast increase of the complexity in modern systems, a fast and accurate solution is highly required when dealing with control, classification or matching problems. Classical methods commonly base on system models. The accuracy of the models has a big influence of the results. But it is very hard to get an precise model for a large and complex system. It requires not only a comprehensive knowledge of the aimed system, but also exact mathematic descriptions of the effect of different aspects to the aimed system. Meanwhile, a complex algorithm also needs more calculation time, which may lead to a fail of real time requirements during execution.

Fuzzy methods are established methods to handle systems with uncertainties. They also provide a solution of solving the problems without accurate system models. Two main research directions are fuzzy logic and fuzzy automata. Detailed explanation is contained in the following part. Fuzzy logic systems are very easy to build and understand. Due to the algorithm used for fuzzy reference, the calculation complexity will grow exponentially when the input number is increased. This problem is referred to "curse of dimensionality". So in the practice, usually the number of input signals for a fuzzy system will be limited within 3 [Yao03]. This also limits the application of fuzzy logic in the real system.

Finite automaton technique is efficient for dealing with multi input signals and complex states systems. A combination with fuzzy definition is fuzzy automaton, which support fuzzy transitions during the process. Unlike fuzzy logic which is well developed, fuzzy automaton is still a freshman in the fuzzy family. It provides a way to express fuzzy transitions, but lack of representation of

ISBN 978-3-00-048889-4

fuzzy inputs and outputs. Fuzzy cellular automaton provides a way to involve fuzzy inputs and outputs into a cellular automaton, but it lack of support of fuzzy rules in the system[CFM+97].

To find a suitable way to solve the curse of dimensionality problem when using more inputs is an important task of fuzzy engineers. In this paper a new type of automaton is proposed. It uses the fuzzy logic method to build a tree automaton. It uses the cellular states concept to deal with the fuzziness of inputs and outputs; it uses the layer conception from neural network to build the automata structure; it uses the originally fuzzy automaton definition to represent the fuzzy transition rules. In this way, a so called fuzzy logic tree automaton (FLTA) is generated. Compared with the current fuzzy methods, it has several advantages. First, it keeps the benefit of fuzzy logic definition interface to build a tree automaton, which is easy to understand and convenient for the designer. Meanwhile, it also keeps the benefit of automata, which could rise the calculation speed of multi inputs multi outputs (MIMO) system. Secondly, it merges the cellular conception to fuzzy automata theory that support the fuzziness of the inputs and outputs and fuzzy transitions. Thirdly, the layered structure solves the problem of curse of dimensionality, then it makes it possible to build a MIMO fuzzy system. Lastly, adding or removing an input/output is more easier in a FLTA. This could lead to a further extension in formalisation algorithm. A washing machine example is built in this paper to explain the FLTA, and an algorithm analysis is given at the end to compare with the normal fuzzy logic method.

2. Brief Introduction to Fuzzy methods

Fuzzy logic was first introduced by Lotfi A. Zadeh[Zad65]. It imitates a human like way of thinking, involves fuzzy set definition instead of a crisp set. This means using non-precise description like "tall", "not too large" etc. instead of a precise value. A Fuzzy logic system is a system which takes the precise input values, processes data based on some fuzzy methods and then returns a precise output value. Basically, the core of a fuzzy logic system consists of three parts: First the fuzzification, which translates the input and output into a serial of membership functions, thus a fuzzy input is built. Secondly, an inference interface, which is used to record the fuzzy rules and build the fuzzy output. Thirdly, a defuzzification method, which used to calculate a classic "crisp" output value from the fuzzy output.

There are some special concepts in fuzzy field. A *fuzzy set* is used to define an unclear notion. In order to describe the elements in the set, instead of using a character function for a normal set, *membership functions* (mf) are used for a fuzzy set description. Each mf represents a fuzzy status of fuzzy definition. For a crisp input value, it may belong to several membership functions, the degree of membership is a value between 0 and 1, which is named *membership value* (mv). For example, a temperature set is defined as a fuzzy set, it contains three status: cold, normal and hot. Each status relates to a membership function. If the given input temperature is 18 degree, then from the membership functions, its mv for cold is 0.3 and its mv for normal is 0.7.

In the **Deterministic Finite Automaton (DFA)**, two main structures are very important: The transition structure, which represents the internal behavior of an automaton, and the output structure, which represents the external behavior. A typical fuzzy automaton involves fuzzy techniques into these two parts. [DK05] gives a common definition of fuzzy automata, and it is also used as the start point of this paper.

ISBN 978-3-00-048889-4

The fuzzy embedded in two fields according to this definition: fuzzy transitions and fuzzy outputs. Fuzzy transitions mean that with the same input symbol, there could be more than one transition triggered at the same time. A membership value is assigned to each of these transitions, this mv is also called the weight of the transition. Multi transitions at the same time lead to multi active states at the same time. Then, the notion of current state and next state should be extended to a state set and the name seemed to be inaccurate. [DK05] introduces two new terms successors and predecessors for it. For each successor state, an mv should also be considered. Usually this mv for the successor takes the same value as the weight of the transition, which is called transition based membership [OGT99]. But in some case, without considering the previous transition weight may cause some unreasonable result. For example, if a state q_i has the mv 0.01, and it has only one successor q_j with mv 1.0. This means q_j is the most possible final state with current input, which obviously is unreasonable. Thus, [DK05] defined a function to take the history transition weight into account.

A membership assignment function $F_1(\mu, \delta)$ is "a mapping function which is applied via augmented transition function $\tilde{\delta}$ to assign mv's to the active states" [DK05]. It is influenced by two parameters:

- μ: the mv of a predecessor;
- δ: the weight of a transition.

For a transition happening from q_i to q_j, with the input a_k, the mv of q_j could be represent as:

$$\mu^{t+1}(q_j) = \tilde{\delta}((q_i, \mu^t(q_i)), a_k, q_j) = F_1(\mu^t(q_i), \delta(q_i, a_k, q_j)) = F_1(\mu, \delta) \tag{1}$$

In practice, usually some simple methods are used to calculate the mv of a state, such as using arithmetic mean, geometric mean, max/min value etc.

Since the successors are not identified, it may happen that different predecessors lead to the same successor. Thus, several mv may be set to one successor at the same time. This is called multi-membership problem. [DK05] also defines a multi-membership resolution function F_2 for calculating the membership value of each state. Simply speaking, first calculating all the possible mv caused by different predecessor; then using some calculation rules to get one single value from these possible membership values. Common used calculation rules are similar when dealing with mv of a state.

For a common fuzzy automaton, there is also another big problem for choosing one suitable final state, which is called output mapping. Some methods are introduced in [DK04]. All the methods used in defuzzification could also be used in output calculation.

3. Fuzzy logic tree automaton

3.1. Definition

In section 2, a common definition of a fuzzy automaton is introduced. From the definition it can be seen that the fuzzy concept lies on the transitions and the final state. But the input symbols of the system are still crisp. This is different from a fuzzy logic system, which begins with fuzzy inputs. The basic idea of this paper is to merge fuzzy logic technique into an automaton. Although fuzzy logic and fuzzy automaton both use fuzzy definition, the structures and execution mechanisms are

quite different. A fuzzy automaton supports fuzzy transitions and fuzzy states, but the inputs can't be fuzzy. Fuzzy cellular automaton [CFM$^+$97] involved the fuzzy inputs, but the transitions are deterministic. Although the fuzzy logic rules represent in a determined way, it is different when the status of the inputs is changed. Furthermore, the number of rules will also largely increase when adding more inputs. Besides, the expression of the rules depends on the related inputs. This means the outputs depend on all the inputs status that contains in the rules. For the same inputs, different status combinations usually result different outputs.

So in order to merge the two fuzzy methods together, three main problems need to be solved. First, the fuzziness process of inputs should be represented properly. This means to build a fuzzy set of each input in a form of membership functions. Secondly, all the rules should be expressed in the structure. Thirdly, the expressions of the rules in the structure should be decoupled from the expression of the status of the inputs. A new type of tree automaton is defined as follows to solve these three problems. A complete FLTA definition could be given as follows:

Definition 1 A Fuzzy Logic Tree Automaton (FLTA) F is a 7-tuple denoted as:

$$F = (I, N, O, T, \delta, \omega, M), where \tag{2}$$

I: Non fuzzy inputs set I= {a, b, c ...}. Each input is a set of cellular states, each cellular state represents one membership function. a = {a_1, a_2, a_3 ...}; b = {b_1, b_2, b_3 ...} ...

N: Non-terminate states set, contains all cellular states of all inputs. N={ a_1, a_2, a_3, b_1, b_2, b_3 ... }. N is divided into layers, the number of the layer is the same of inputs. Each layer N_i contains all the cellular states from one single input. Eg: N_0 = {a_1, a_2, a_3 ...}; N_1= {b_1, b_2, b_3 ...} ...

O: Non-fuzzy output set, O={x, y, z ...}. The same as inputs, each output is a set of cellular states, each cellular state represents one membership function. Eg. x = {x_1, x_2, x_3 ...}; y = {y_1, y_2, y_3 ...} ...

T: terminate states set. Each terminate state is a finite set. The number of the elements in the set is the same as outputs. Each element represents the status of one particular output, if the output status is not defined by the rules, then use ϕ to present this undefined element. Eg: T_0={x_1, ϕ, z_2}

δ: $n_i \times n_j \rightarrow [0, 1]$ Transition weight. The assignment of value δ is using the same definition as a membership assignment function. The value of δ_k is calculated as follows:

- if $n_i \in N_0$, then $\delta_k = \tilde{n_i}$. This means when the predecessor belongs to the start states set, then the transition weight is the same as predecessor's state weight $\tilde{n_i}$
- if $n_i \notin N_0$, then $\delta_k = F_1(\tilde{n_i}, \delta_{k-1})$. When the predecessor does not belong to the start states set, then the transition weight is the result of the previous transition weight δ_{k-1} and state's weight $\tilde{n_i}$. Function F_1 could be defined by designer. The one used in this paper is $\delta_k = F_1(\tilde{n_i}, \delta_{k-1}) = \frac{1}{2}(\tilde{n_i} + \delta_{k-1})$

ω: Fuzzy IF-THEN rules set.

M: Outputs mapping function. The output mapping process is the same as defuzzification process in fuzzy logic. All the methods which are used for defuzzification could be used in the output mapping.

In order to give an intuitive impression, a washing machine control unit is built step by step using FLTA in the following. The washing time is determined by the dirtiness of the clothes and the

weight of the clothes. The dirtiness estimation depends on two aspects, the sludge dirty and the oily dirty. Obviously, the more dirty the clothes are, and more heavy they are, a longer washing time is required. Meanwhile, dirty clothes which are more oily also need longer washing time than the dirty ones with more sludge. It is very hard to build an accurate mathematic model to determine the washing time with these three input data, and there is also no need to build such an accurate model. Thus, fuzzy method would be a perfect solution in this situation. In this example, the input set I is defined as {S (Sludge dirty degree), Oi (Oily dirty degree), W (Weight of clothes)}, the output set O is defined as { Ti (washing Time) }.

3.2. FLTA structure

The structure of the fuzzy logic tree automaton looks like a tree automaton with strict layers. The nodes of the tree automaton are all the non-terminate states and terminate states. Each layer represents a particular input and the last layer contains all the terminate states. As mentioned in Definition.1, each input is a fuzzy set, the elements in this fuzzy set are cellular states with membership functions. The order of the inputs is not important, the whole trees may look different with different order, but the results are always the same. The starting point is not one state, it is a start layer N_0 , with all the cellular states of the first input. Each non-terminate state takes the next input set as its child, until the last input is reached. Each state in the last input layer is connected to a specific terminate state. The content of the terminate state is determined by the fuzzy rules. If one output is related to multiple status, an output mapping with defuzzification algorithm will be used to get a crisp result.

In the washing machine control unit, the input and output could defined as follows:

- S: Sludge dirty degree, value range is (0, 100). Fuzzy set S= {Small (SS), Middle (SM), Large (SL)}.
- Oi: Oily dirty degree, value range is (0, 100). Fuzzy set Oi= {Small (OS), Middle (OM), Large (OL)}.
- W: Weight of clothes, value range is (0, 10kg). Fuzzy set W= {Light (WL), Normal (WN), Heavy (WH)}.
- Ti: Washing time, value range is (0, 150min). Fuzzy set Ti= {Very Short (VS), Short (VS), Medium (M), Long (L), Very Long (VL)}.

The membership functions for each variable are given as in Fig.1. According to the above definitions, the system could contain $3 \times 3 \times 3 + 3 \times 3 + 3 = 39$ rules. Normally, not all of these rules are required. In this example, 24 rules are assigned to calculate the washing time. Table.1 lists part of the rules used in the system. With all these rules, an FLTA could be built. Since the rules covered all the definitions, there is no empty state for the output. Because of the limited space, Fig.2 shows only part of the FLTA.

3.3. FLTA execution

The execution of an FLTA is divided into two steps, the off line tree building and on line processing. During the off line step, the inputs and outputs membership functions should be defined. The fuzzy rules should also be given off line. Another important process which is finished in this step is building the structure of the FLTA according to the above information. During this process, the

ISBN 978-3-00-048889-4

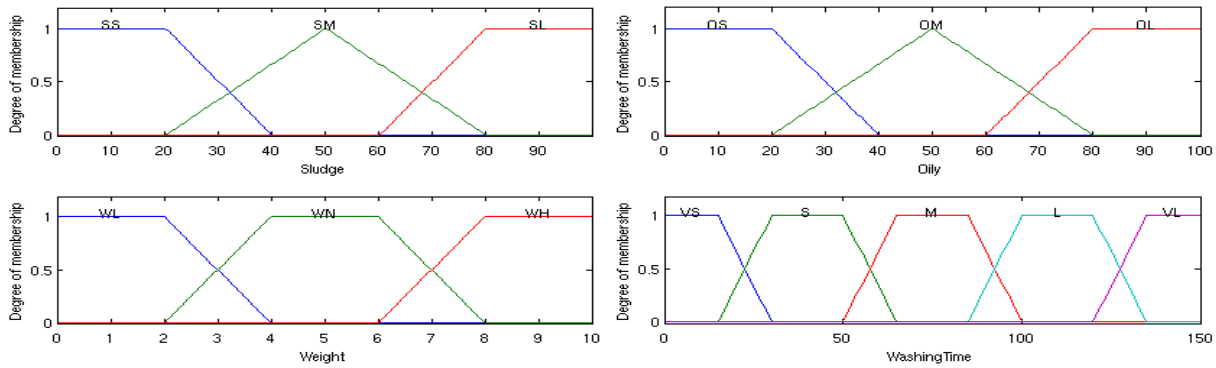


Figure 1: Membership functions of inputs and output

Table 1: Fuzzy rules of washing machine control method

Sludge dirty (1-100)	Oily dirty (1-100)	Wight (0-10kg)	Washing time (0-150m)
SS	OS	WL	VS
SS	OL	WN	L
	OL	WH	VL
...	...	...	...
SL	OS	WL	M

system should use the definition of the inputs to build the layers, the order of the input has no influence of the complexity of the FLTA. The terminate states status is defined by the fuzzy rules. All the steps mentioned above of the washing machine control example should be finished in this process.

The on line part is the main execution process of an FLTA. Each input set takes the related input data, and each cellular state calculates the membership value of this input parallel with the given membership function. The mv is also taken as state weight in the FLTA definition. For the washing machine example, suppose the input data from the sensor is that the sludge dirty degree is 35%, the oily dirty degree is 70% and the weight of the washing clothes is 3.5kg. Then the execution could be illustrated as Fig.3.

After the state values are calculated, the tree is traversed from the first layer until reaching the terminate states. As shown in Fig.3, during the traversal,

- function δ is used to assign the transition values. In the example, we suppose $\delta_k = F_1(\tilde{n_i}, \delta_{k-1}) = \frac{1}{2}(\tilde{n_i} + \delta_{k-1})$
- when a state weight is 0, then it will skip this branch.
- the reached terminate states take the transition weight as their states weight. But the output membership value is decided by all the terminate states which are reached. A multi-membership resolution function F_2 could be used to represent the calculation rules. In order to reduce the complexity of the calculation, normally F_2 is suggested to choose one of the following forms: taking the min / max value, or taking the average value. The state value of empty status is not calculated. The function used in this example is max value assignment.

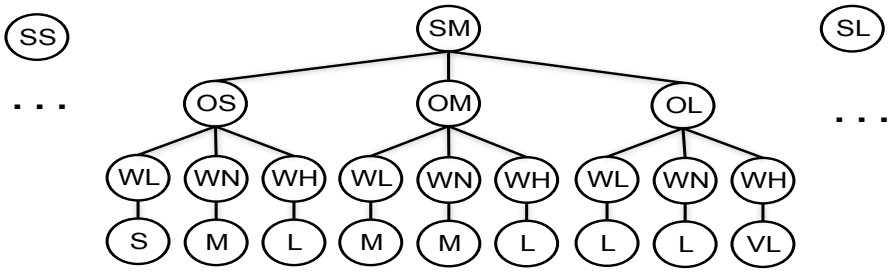


Figure 2: FLTA of a washing machine control algorithm

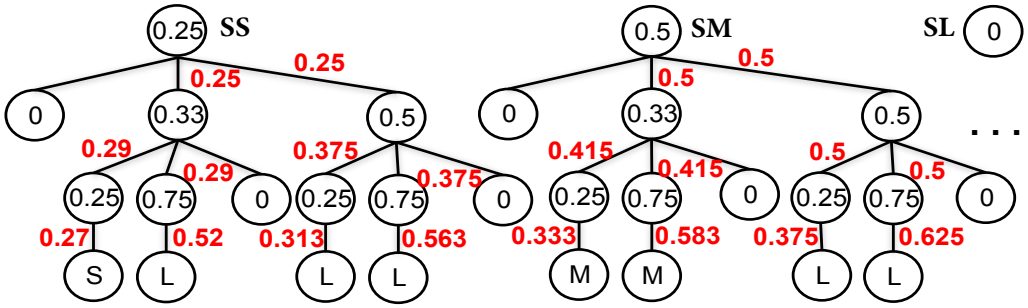


Figure 3: Execution process of a FLTA for a washing machine control example

It could be seen from Fig.3 that the reached terminate states contains three states of the output. By choosing the maximum value as the function to assign the terminate state weight, the status weight could be calculated as follows:

TS: $TS = max(0.27) = 0.27$
TM: $TM = max(0.333, 0.583) = 0.583$
TL: $TL = max(0.52, 0.313, 0.563, 0.375, 0.625) = 0.625$

After getting the membership values of each status of an output, the final result is calculated by the defuzzification methods. All the methods which are available for a fuzzy logic defuzzification process could be also used in an FLTA output mapping. In the example, a prevalent and accurate method centroid method is chosen as the output mapping function M, the final result is shown in Fig.4. The shaded part is the fuzzy output with the state weight, the vertical line in the middle marks the result of centroid defuzzification method. It could be seen that the resulting washing time should be 83 minutes.

4. Analysis of the FLTA algorithm

Analysis of an algorithm contains different aspects. The basic and also important part is algorithm correctness checking, and computational complexity analysis. The complexity mainly focus on the time complexity and memory complexity. In this part, a basic analysis of FLTA in these three aspects is given.

4.1. Correctness checking

In order to check the correctness of FLTA, the same washing machine control example is implemented in Matlab using fuzzy logic tool box with mamdani inference algorithm. When the fuzzy

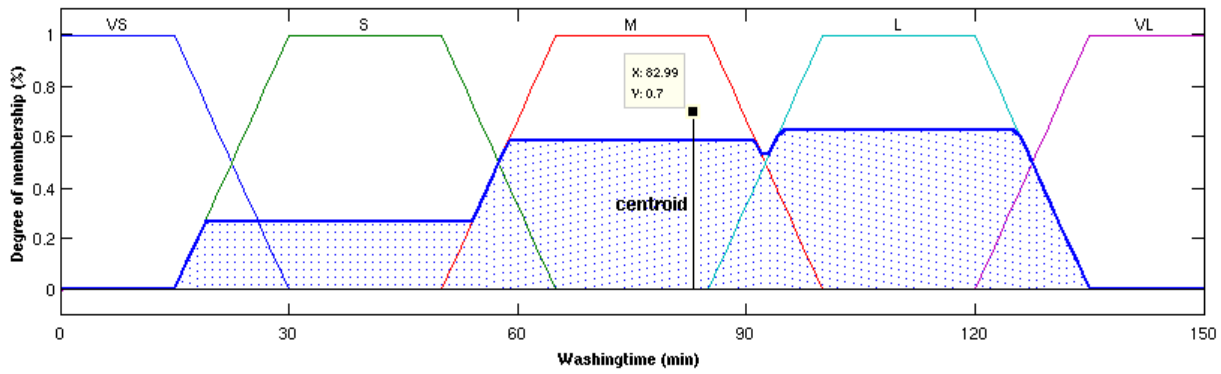


Figure 4: Defuzzification result with centroid method

logic system is defined with similar methods, which means to use the minimum value of an AND method, the maximum value for aggregation, and centroid method for defuzzification, then the result showed in Fig.5 is also 82.9. This means, when choosing the same function during the execution process, an FLTA could reach the same result as a traditional fuzzy logic system.

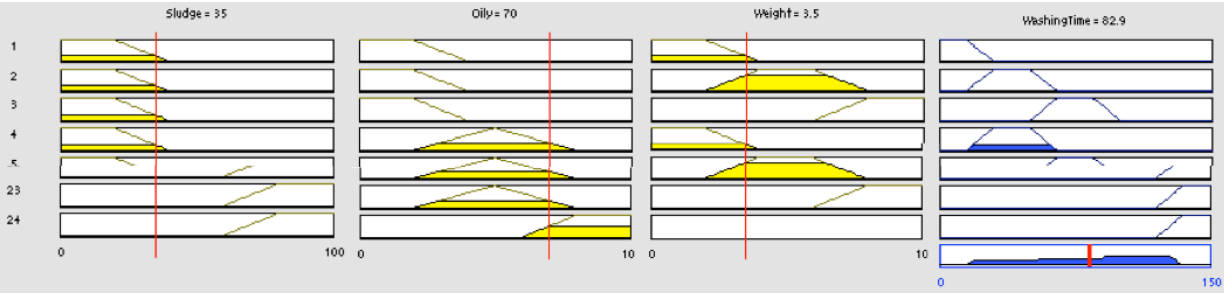


Figure 5: Implementation in Matlab fuzzy logic tool box (only part of the rules is shown)

4.2. Computational complexity analysis

Both fuzzy logic and FLTA contain off-line definitions and on-line execution. In this paper, only the on-line process is analysed for computational complexity. According to [Zad65], the essence of fuzzy inference is the calculation of fuzzy relationship among the inputs and outputs. Mamdani proposed a method in [Mam77], it uses min operation instead of AND and max operation represent OR. It could largely simplify the calculation and also decrease the complexity in the inference process [Koc95]. The Mamdani method is a common used inference algorithm in fuzzy logic currently, and in this paper, the calculation complexity of Mamdani is used to compare with the FLTA method.

Suppose a fuzzy system is defines as follows:

Input set: $X = (X_1, X_2, ...X_k)$, the number of mf of X_i is m_i, $k, m_i \in N$, $i \leq k$, $m_i \leq m$;

Output set: $Z = (Z_1, Z_2, ...Z_g)$, the number of mf of Z_j is n_j, $g, n_j \in N$, $j \leq g$, $n_j \leq n$;

Fuzzy rules R_l is "IF X_1 is A and X_2 is B and ... THEN Z_1 is C and Z_2 is D ....", $l \in N$ and $l \leq r$

A Mamdani inference is showed in Fig5, [Koc95] gives a detailed calculation of Mamdani method for a multi input single output system, extended to a multi input and multi output system with the

definition above, the computational complexity of Mamdani could be written as $O(rg(k+1)m)$. In order to get a complete definition space, the number of the rules is similar to the number of possible combinations of all input variables, which could be written as $\mid R \mid= O(m^k)$ [Koc95]. Thus, the complexity of Mamdani is

$$O(gm^{k+1}(k+1)) \tag{3}$$

The worst case in an FLTA is that all the states need to be visited and all the terminate states will be triggered. Then the algorithm could be summarised as:

Each non-terminate state calculate its state mv with the given input;
Checking the state-mv and calculating the transition-mv;
The fuzzy output C' of one single output is calculated from all the triggered terminate states.

Then the complexity of FLTA could be calculated as

$$O = mk + \frac{m(1-m^k)}{1-m} + ng = O(mk + m^k + ng) \tag{4}$$

When $m \geq 2$ and $k \geq 2$, the complexity of FLTA is $O(m^k)$. Compare Equation.3 and Equation.4, it could be seen that even in the worst case, the complexity of FLTA is still smaller than Mamdani, and the difference is even larger when the input and output number increase.

Usually in practice, one crisp input may be related to at most 2 mf. According to the definition of FLTA, only part of the states need to be visited, this leads to a normal case of FLTA. But this situation does not influence the normal fuzzy logic method. So the complexity of Mamdani in this case is still the same as Equation.3. The complexity of FLTA in the normal case could be calculated as:

$$O = mk+m+2m+...+2^{k-1}m+ng = mk+(2^{k-1}-1)m+ng = O(mk+(2^{k-1}-1)m+ng) \tag{5}$$

When $k \geq 3$, the complexity of FLTA is $O(2^{k-1}m)$. Compare with the worst case,

$$\frac{O_{normal}}{O_{mamdani}} = \frac{2^{k-1}m}{gm^{k+1}(k+1)} = (\frac{2}{m})^k \frac{2}{g(k-1)}$$

It shows that the more complex a fuzzy system is, the less calculation is needed in FLTA than normal fuzzy logic system.

4.3. Space complexity analysis

The memory space required of Mamdani in the execution mainly contains four aspects: the mv of given input variables in each mf; the rule space (separated in "IF" part and "THEN" part); the mv of each output in each rule; and the mf of each output. Then a rough calculation could be

$$O_{Mamspace} = mk + 2r + rg + ng = mk + 2m^k + m^k g + ng = O(mk + ng + (2+g)m^k)$$

The rules are already included in the terminate states, so during the execution, the memory needed for an FLTA contains only the mv of given input variables in each mf; mv of transitions in current level; and the mf of each output.

$$O_{FLTAspaceWorst} = mk + \frac{m(1-m^k)}{1-m} + ng = O(mk + ng + m^k)$$

ISBN 978-3-00-048889-4

$$O_{FLTAspaceNormal} = mk + (2^{k-1} - 1)m + ng = O(mk + (2^{k-1} - 1)m + ng)$$

It could be seen from the above calculation that for a MIMO system, the space needed in an FLTA is changed according to the mf definitions. In the normal case, where one crisp input only related to 2 mf, the memory needed in FLTA is quite less than a Mamdani method. As a conclusion, FLTA could not solve the exponential problem in fuzzy logic, but it still largely decreases the complexity of a MIMO system.

5. Conclusion and further work

The fuzzy logic tree automaton is a new algorithm that provides a way to combine fuzzy logic and automata together. This paper explains the details of this method, the definitions, structures and execution methods. This algorithm focuses on dealing with the system of a large number of inputs and outputs, which is non-efficient with fuzzy logic only. And it also involve the fuzzy input sets into a fuzzy automaton. It provides a possibility to solve most control and classification problems with fuzziness. The structure is independent of the rules, only the terminate states depend on them. So an FLTA could be decoupled from the inputs, then if a new input/output is added, the whole automaton could be easily updated. Although it seems to have a huge tree, the execution is parallel. And the on line calculations are not complex. This means the execution time is not a big problem. For estimating the real execution time, a complete implementation of FLTA is required, which will be the next step of this work.

References

[CFM+97] Cattaneo, C., P. Flocchini, G. Mauri, C.Q. Vogliotti, and N. Santoro: *Cellular automata in fuzzy backgrounds*. Physica D, 105:105–120, 1997.

[DK04] Doostfatemeh, Mansoor and Stefan C. Kremer: *The significance of output mapping in fuzzy automata*. Proceedings of the 12th Iranian Conference on Elecectrical Engineering, 2004.

[DK05] Doostfatemeh, M. and S.C. Kremer: *New directions in fuzzy automatadirection*. Interna tional journal of approximate reasoning, 38(2):175–214, 2005.

[Koc95] Koczy, Laszlo T.: *Algorithmic aspects of fuzzy control*. International Journal of approximate reasoning, 12:159–219, 1995.

[Mam77] Mamdani, Ebrahim H.: *Application of fuzzy logic to approximate reasoning using linguistic synthesis*. IEEE transactions on computers, Vol. C-26(No.12):1182–1191, 1977.

[OGT99] Omlin, Christian W., C. Lee Giles, and K.K. Thornber: *Equivalence in knowledge representation: Automata, recurrent neural networks, and dynamical fuzzy systems*. PROCEEDINGS OF THE IEEE, pages 1623–1640, 1999.

[Yao03] Yaochu, Jin: *Advanced Fuzzy Systems Design and Applications*. ISBN 3-7908-1537-3. Physica-Verlag, 2003.

[Zad65] Zadeh, L.A: *Fuzzy sets*. Inform. Control, 8:338–353, 1965.

Framework for Varied Sensor Perception in Virtual Prototypes

Stefan Mueller, Dennis Hospach, Joachim Gerlach, Oliver Bringmann, Wolfgang Rosenstiel
University of Tuebingen
Tuebingen
{stefan.mueller,dennis.hospach}@uni-tuebingen.de

Abstract

To achieve a high test coverage of Advanced Driver Assistance Systems, many different environmental conditions have to be tested. It is impossible to build test sets of all environmental combinations by recording real video data. Our approach eases the generation of test sets by using real on-road captures taken at normal conditions and applying computer-generated environmental variations to it. This paper presents an easily integrable framework that connects virtual prototypes with varying sensor perceptions. With this framework we propose a method to reduce the required amount of on-road captures used in the design and validation of vision-based Advanced Driver Assistance Systems and autonomous driving. This is done by modifying real video data through different filter chains. With this approach it is possible to simulate the behavior of the tested system under extreme conditions that rarely occur in reality. In this paper we present the current state of our virtual prototyping framework and the implemented plug-ins.

1. Introduction

In recent years advances in embedded systems and sensors technology have lead to a tight integration of the physical and digital world. Such systems, having connections to the physical world through sensors and actors and also having an embedded system for communication and processing, are often considered as Cyber-Physical Systems (CPS). These CPS are accompanied by new challenges in design and verification. Many examples for CPS can be found in a modern car, especially in the range of Advanced Driver Assistance Systems (ADAS), appearing more and more in the latest car generations. These ADAS heavily rely on sensor perception. The major problem is to guarantee functional safety requirements especially if ADAS are taking over more and more active control over the vehicle. These systems need to operate correctly in very different environmental conditions which are strongly influenced by the traffic situation, weather conditions, illumination, etc. This requires a high amount of on-road captures to test all combinations of environmental influences. Nevertheless, a total coverage is impossible. To address the issue of the amount of needed on-road captures, this paper presents an approach to reduce the number of captures by adding synthetic weather conditions to real captures. The presented framework allows to explore the virtual prototype, the Electrical/Electronic (E/E) architecture and the software running on it in the scope of many different use cases. In this manner it is possible to generate variations of environmental conditions of a traffic situation or add rarely occurring weather to existing on-road captures.

ISBN 978-3-00-048889-4

2. Related Work

The challenges in safety evaluation of automotive electronics using virtual prototypes are stated in [BBB+14]. With current validation methods, $10^7 - 10^8$ hours of on-road captures are needed to verify the functional safety of a highway pilot system in an ISO 26262 compliant way [Nor14]. Most vision-based ADAS techniques heavily rely on machine learning algorithms such as neural networks and support vector machines (SVM) as presented in [MBLAGJ+07, BZR+05] and/or Bayesian networks [BZR+05]. All these approaches have in common that they need to be trained with well selected training data [GLU12, SSSI11]. There are approaches to generate synthetic training data, where image degradation [ITI+07] or characteristics of the sensor and the optical system [HWLK07] are used to enlarge the training data. Most ADAS employ several sensors, networks and Electrical Control Units (ECU) to fulfill their work, which results in a complex scenario that can be considered as a cyber-physical system [Lee08]. A methodology to generate virtual prototypes from such large systems and keep a maintainable speed is shown in [MBED12]. It uses different abstraction levels to reach high performance of the controller and network models which are connected to a physical environment simulation. Another paper that covers virtual prototyping in the scope of ADAS comes from Reiter et al. [RPV+13]. They show how robustness and error tolerance of ADAS can be improved with error effect simulation. None of these works has presented a consistent way to connect the test and training methods to virtual prototypes.

3. Connecting Environment to Virtual Prototypes

To consider a virtual prototype under varying conditions, it is necessary to connect the virtual prototype to the environment. This connection is realized through a virtual sensor which models the properties of the corresponding real sensor. The advantage of using virtual sensors is that they can be applied in every design stage.
In the succeeding chapters we focus on vision-based virtual prototypes and refer to the following example applications:

- an in-house developed Traffic Sign Recognition (TSR) which is targetable to different hardware platforms. The Framework is used to train and verify the results of the integrated SVM.
- the Caltech Lane Detection Software (LD) which is presented in [Aly08]. We use this software as a second independent implementation of an ADAS algorithm to show the ease of integration of this framework into existing virtual prototypes.

As this work is focused on vision-based systems, the connection between virtual prototype and environment is done by a virtual camera. The input of the camera is an image of the environment. In our approach, this image is dynamically generated by modifying an existing video stream as shown in figure 1. These modifications may include different environmental impacts, which are sent to the virtual prototype and can on the one hand be used for detection of faults in hardware or software and on the other hand for fitness evaluation of the implemented algorithms.
The camera model and the dynamic generation of environment data is described in detail in the next chapter.

ISBN 978-3-00-048889-4

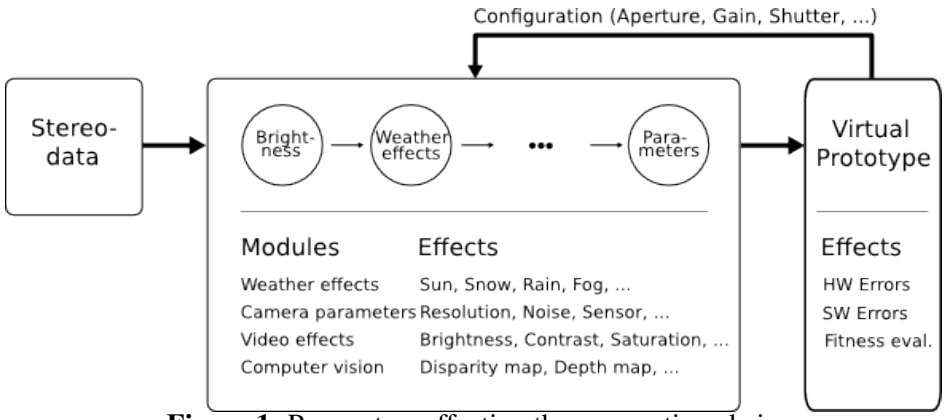


Figure 1: Parameters affecting the preparation chain

3.1. Data Acquisition with Virtual Sensors

The data acquisition with virtual sensors is different from the data acquisition of real sensors. The real environment provides all available information at a time, whereas virtual environment cannot deliver all this information simultaneously due to limited computing power. For example, a digital camera can vary in aperture, exposure time, gain, etc., which all influence the appearance of the captured image. To address this issue, the virtual sensor has to communicate with the virtual environment and request the desired values. The virtual environment then generates the desired result. The parameters which influence the requested value can be divided into two groups: sensor parameters and environment parameters. Examples for typical environment parameters are brightness, rain rate or the amount of fog. Examples for typical sensor parameters are given in figure 1. To ease the handling of the communication between sensor and environment, a generic, modular, plug-in based framework was created. It was designed under the following premise:

- generic data format and flexibility
- C/C++ interface because most simulation environments allow the call of C/C++ functions or can co-simulate it
- lightweight interface for data exchange
- easy to integrate in virtual prototypes

To achieve maximum flexibility, the framework uses three types of plug-ins for preparing the input data: source, filter and sink. These can be connected to preparation chains. Sources read a recorded scenario and pass it to the first filter or the sink. Filters modify the incoming data and pass it to the next stage, which can be a filter or a sink. There are two categories of sinks - online sinks and offline sinks. Both sink types provide the synchronization between one or more filter chains and the output destination. Online sinks are used for real-time capable processing chains and communicate directly with the prototype, where real-time is determined by the needs of the connected prototype. Let T_{cycle} be the time consumed by the prototype for processing one frame, N the number of filters in the chain, T_i the time that the i-th filter needs for processing and $T_{transport}$ the time consumed for the transport between the data generation and the processing unit of the prototype. The following equation must hold if the system has to run in real-time:

$$T_{cycle} > \sum_{i=1}^{N} T_i + T_{transport}$$

Depending on the used abstraction level of the virtual prototype, the sink can be connected via different communication channels like a Transaction Level Modeling (TLM) interface or a network protocol. In contrast, offline sinks are used to prepare computationally intensive filtering jobs. Offline sinks store the received results for subsequent online runs. The preparation filter chain works with a pull mechanism, where the sink triggers the processing of each frame. This behavior is important for online sinks because it allows to work in sync with the simulation of the virtual prototype. Online sinks can also receive resulting data from the virtual prototype and can act as a source for an evaluation chain. This chain works with a push mechanism, which is important for systems with an asynchronous processing behavior, because it allows the evaluation to run on its own frequency.
The communication between these plug-ins is done by integration of the boost iostream library [ios]. By using the boost asio library [Koh] as a tunnel for the iostreams, it is possible to distribute the work over several computational resources. The generic data format is designed to abstract the data from all dependencies. This allows to build and run the framework on different platforms and the only library dependency that must be fulfilled by the simulation environment is the commonly used boost library. The communication between the different plug-ins is package-based. Each package has a type-id, which defines its content. This ensures the correctness of the chain. For example, a source that sends packages containing a rectified stereo image cannot be connected to a filter that expects packages containing a depth map.
To ease test creation and execution of such plug-in chains we created a Qt-based GUI which is designed to run independent of the server process. This allows to remote control the data preparation on computational resources. Prepared test cases can be stored as XML file and may be used to start the server process from the command line in batch processes.

3.2. Plug-in Chain for Environmental Variations

In the following we present the already implemented plug-ins which can be combined to chains to apply environmental variations on the input data. For illustration, figure 2 shows a full setup of a virtual prototype with preparation and evaluation chains in an online scenario.

3.2.1. Sources

By now we implemented image source plug-ins for image file sequences and video files. Both exist in a single and stereo image version. Each image source reads the specified data source using the ffmpeg library [ffm] and passes the data frame by frame to the succeeding plug-in. The only difference between single and stereo sources is that the stereo sources transfer two images per data package. Besides the normal image formats, there are sources for the images which are converted into scene radiance values after acquisition and for the output of the CarMaker software from IPG [Car]. The radiance value images are used by the plug-in which changes the sensor characteristics and the brightness. These are described later on.

3.2.2. Brightness

Probably the simplest variation to introduce to images is brightness. With real cameras, variations in brightness often occur due to the relatively slow adaption of the camera to changing incident

ISBN 978-3-00-048889-4

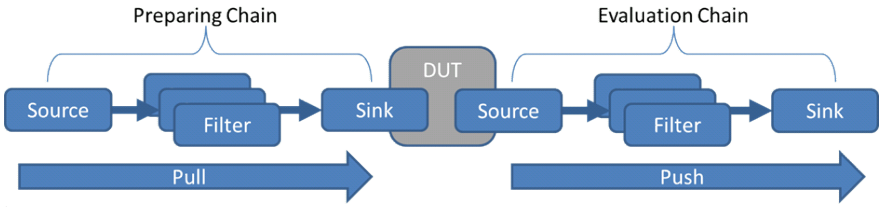


Figure 2: Work flow using preparation and evaluation chain in an online setup

illumination. An often-observed effect is over- or underexposure of the images. The reader may imagine driving into a tunnel or coming out of it. During the time the camera adapts the shutter and gain, all images will suffer from bad exposure.
To simulate this effect, we chose the following method: Saturating multiplication of the image in pixel space will lead to similar effects as described above. Let I denote a 2D image and $I(x, y)$ be the intensity value at position (x, y). Then this operation may be described as

$$I(x,y) = \begin{cases} a * I(x,y) & \text{if } a * I(x,y) < 255 \\ 255 & \text{else.} \end{cases}$$

where $a \in \mathbb{R}^+$. This leads to a spreading of the histogram with loss of information, where the pixel intensities run into saturation (see figure 3).
An alternative way of changing brightness is by doing it in the scene radiance space as presented in [HMB+14]. Changing the incident scene radiance before the virtual prototype remaps it into pixel space is a more physically correct way of changing brightness and is the way to do it if one would like to simulate the virtual camera as an adapting system. The pixel values of the virtual camera will only run into saturation, if the parameters of the model are set accordingly.

3.2.3. Depth Information

For more complex variations like rain, fog or other effects that reduce the visibility, the image itself is not sufficient. Per pixel depth information is also necessary. This information can be sourced from stereo images. To convert the stereo images into a depth map, a chain of three plug-ins is used. The first filter generates a disparity map from the stereo images. This disparity map is generated by the Semi-Global Block Matching algorithm (StereoSGBM) of the OpenCV library [ope]. Afterwards, the following filter refines the disparity map by closing holes in the map. The holes in the disparity map are filled with the values of the neighbor pixels as discussed in [Sti13]. At last, the third filter calculates the depth information from the disparity map and supplies the left input image and the depth map to the succeeding filter.

3.2.4. Rain

The simulation of rain is a very complex task by itself and simulating every little aspect of it is quite impossible. The simulation of rain has mainly been addressed in the scope of computer vision, aiming at generating visually convincing results. We have created a rain filter that is physically grounded, rendering rain streaks that follow the mathematical probability distributions of real rain. We are currently still evaluating the performance of the results with respect to the effects a sensor perceives.

3.2.5. Sensor characteristics

The virtual prototype acting as the connection of the physical world to the cyber-generated image heavily depends on the parameters of the simulation. The outcome can be significantly different when parameters of color channel sensitivity to illumination, color balance, noise characteristics at different gain levels or sensor size changes. To test the simulation with different optical parts of the virtual prototype, we developed a filter to map the image data from the camera system with which we recorded our sample data to another camera system. This virtual camera system can be based on a real camera, that has been calibrated against our recording system, or it could as well be a pure virtual system, exposing the possibility of simulating whichever parameter set is worthwhile testing. Color processing and color balance of a target system are also implemented. Further, we would like to be able to model the addition of sensor noise and possibly temperature drift of the sensor. Latter effects are under current development but need further evaluation.

3.2.6. Fog

For an initial rendering of particle effects like fog, there is a plug-in to pipe the data through the scene rendering tool Blender [ble]. This plug-in takes the scene image and transforms the image into a Blender scene according to the information from the depth map. Within this generated blender scene, various effects could be applied to the scene, rendered and returned to the next element in the chain. The current rendering of fog uses the standard Blender effect and is not evaluated in matters of its quality.

3.2.7. Sinks

As described earlier there are two kind of sinks. The offline sink just writes out the resulting images to a video stream, image sequence or to the display. The online sink transports the images to a co-simulated virtual prototype. This online sink establishes a bi-directional connection to the simulation and allows to transfer the images in sync with the simulation time. The online sink can also act as a source for an evaluation chain and return measures from the virtual prototype over the bi-directional connection. These received measures can be evaluated in several ways and lead to a new parameterization of the preparation chain for a new test run and is intended to build up a parameter space exploration.

3.3. Integration

In order to include all the aspects of our TSR, we use SystemC to model the microcontroller, the memory hierarchy and the on-chip busses as well as the automotive-networks like MOST, FlexRay and CAN. This is used to evaluate robustness of the embedded software with respect to the entire chain of effects from the sensor via the E/E architecture to the ECU architecture under varying environment conditions. As proof of the ease of integration into third-party prototypes we connected our system to the Caltech Lane Detection Software [Aly08]. This software is written in C++ and uses the OpenCV library. The basic integration of this software to our framework took about 1 hour and needed 25 lines of code for the communication. The conversion of the images delivered by the framework into the OpenCV format required 18 lines of code.

ISBN 978-3-00-048889-4

Figure 3: Brightness variations

Figure 4: Scene at normal environmental conditions

4. Results

In this chapter we present first promising pictures generated by this framework. In figure 3 several brightness variations are shown. Figure 4 shows the original scene whereas figure 5 shows the same scene with the modification of rain at a rate of $30.0\ mm/hr$. Brightness was left unchanged. The stereo images used for the results are captured with a camera, which consists of three image sensors with a maximum resolution of 1280x960 pixels. In total, over 330 km of city and interurban on-road captures have been acquired.
The used SVM-based TSR system can discriminate between 12 classes of speed signs. For the evaluation three test sets are used. A route at bright weather (track 1), the same route at rainy weather (track 2), and a different route with many road signs at a diffuse light (track 3) for training purposes. The initial training of the TSR was done with track 3 and used to evaluate track 1 to show that the training works for the chosen route. Then track 2 is evaluated to measure its performance on rainy conditions. After that we modified track 3 by applying different rain rates like in figure 5 and used it to enhance the training of the TSR. The newly trained TSR is used to evaluate track 2 again. The recognition results are shown in table 1. The difference in the number of total recognitions between track 1 and track 2 is caused by two factors: the driven velocities and the acutance. The test routes were driven at different velocities and therefore a traffic sign

ISBN 978-3-00-048889-4

Figure 5: Image with artificially added rainfall (rain rate: $30.0\ mm/hr$) and unchanged brightness

may be visible in more or less frames. The acutance is important for the circle detection. Rain adds some more or less heavy blur to the images, so that more parts of the image are detected as circles. The currently used TSR application does not do any kind of circle aggregation previous to the classification step.
Comparing the performance of both trained TSRs on the rainy day test set (track 2) shows that the number of correct recognitions rises from 44.8 % to 64.2%. Even though the result is not as high as for the bright day (track 1), it has increased significantly.

Test set	Training set	True	False	Sum	Percent
Bright day(track 1)	track 3	52	11	63	82.5%
Rainy day (track 2)	track 3	43	53	96	44.8%
Rainy day (track 2)	track 3 with add. rain	61	34	95	64.2%

Table 1: Evaluation results

5. Conclusion

In this paper, we introduced a platform independent framework for simulation of environmental conditions and the use of virtual prototypes to evaluate their effect on the embedded software and the underlying E/E architecture. The platform is based only on free and open C/C++ libraries: boost::iostreams, boost::asio, OpenCV and ffmpeg. Due to the flexible streaming concept, we are able to add various effects to video material and thus simulate many different combinations of environmental effects and sensor characteristics of the targeted optical system. This allows us to generate many different training sets from the same on-road captures. Furthermore, our system

supports the requirement to evaluate different environmental conditions on a given trajectory to compare efficiency and effectivity of different algorithms for specific ADAS problems. An evaluation step may then rate the overall performance of the virtual prototype and give feedback to conduct automatic parameter space explorations on a higher level of accuracy.

6. Future work

The next steps will be to validate the quality of each synthetic environmental effect related to the sensor perception and the requirements of the different ADAS applications in more detail. For an automatic parameter space exploration different search algorithms will be implemented to speed up the search of specific functional limits of the system.

Acknowledgment

This work was partially funded by the State of Baden Wuerttemberg, Germany, Ministry of Science, Research and Arts within the scope of Cooperative Research Training Group and has been partially supported by the German Ministry of Science and Education (BMBF) in the project EffektiV under grant 01IS13022.

References

[Aly08] Aly, M.: *Real time detection of lane markers in urban streets*. 2008 IEEE Intelligent Vehicles Symposium, June 2008.

[BBB+14] Bannow, N., M. Becker, O. Bringmann, A. Burger, M. Chaari, S. Chakraborty, *et al.*: *Safety Evaluation of Automotive Electronics Using Virtual Prototypes: State of the Art and Research Challenges*. In *Design Automation Conference*, 2014.

[ble] *blender.org - Home of the Blender project - Free and Open 3D Creation Software*. `http://www.blender.org/`, visited on 23/05/14.

[BZR+05] Bahlmann, Claus, Ying Zhu, Visvanathan Ramesh, Martin Pellkofer, and Thorsten Koehler: *A system for traffic sign detection, tracking, and recognition using color, shape, and motion information*. In *IEEE Intelligent Vehicles Symposium (IV 2005)*, 2005.

[Car] *IPG: CarMaker*. `http://ipg.de/simulationsolutions/carmaker/`, visited on 23/05/14.

[ffm] *FFmpeg*. `http://www.ffmpeg.org/`, visited on 21/05/14.

[GLU12] Geiger, Andreas, Philip Lenz, and Raquel Urtasun: *Are we ready for autonomous driving? the kitti vision benchmark suite*. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[HMB+14] Hospach, D., S. Mueller, O. Bringmann, J. Gerlach, and W. Rosenstiel: *Simulation and evaluation of sensor characteristics in vision based advanced driver assistance systems*. In *Intelligent Transportation Systems, 2014 IEEE 17th International Conference on*, pages 2610–2615, Oct 2014.

[HWLK07] Hoessler, Hélène, Christian Wöhler, Frank Lindner, and Ulrich Kreßel: *Classifier training based on synthetically generated samples*. In *The 5th International Conference on Computer Vision Systems*, 2007, ISBN 9783000209338.

[ios] *The Boost Iostreams Library*. `http://www.boost.org/doc/libs/1_54_0/libs/iostreams/doc/`, visited on 22.08.2013.

[ITI+07] Ishida, H., T. Takahashi, I. Ide, Y. Mekada, and H. Murase: *Generation of Training Data by Degradation Models for Traffic Sign Symbol Recognition*. IEICE Transactions on Information and Systems, pages 1134–1141, 2007.

[Koh] Kohlhoff, C.: *Boost.Asio*. `http://www.boost.org/doc/libs/1_54_0/doc/html/boost_asio.html`, visited on 22.08.2013.

[Lee08] Lee, Edward A.: *Cyber physical systems: Design challenges*. Technical report, EECS Department, University of California, Berkeley, Jan 2008.

[MBED12] Mueller, W., M. Becker, A. Elfeky, and A. DiPasquale: *Virtual prototyping of cyber-physical systems*. In *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, pages 219–226, Jan 2012.

[MBLAGJ+07] Maldonado-Bascon, S., S. Lafuente-Arroyo, P. Gil-Jimenez, H. Gomez-Moreno, and F. Lopez-Ferreras: *Road-Sign Detection and Recognition Based on Support Vector Machines*. IEEE Transactions on Intelligent Transportation Systems, 8(2):264–278, June 2007, ISSN 1524-9050.

[Nor14] Nordbusch, Stefan, Robert Bosch GmbH: *Vision or Reality – The Way to Fully Automated Driving*, 2014. `https://www.edacentrum.de/veranstaltungen/edaforum/2014/programm`.

[ope] *OpenCV*. `http://opencv.org/`, visited on 06/09/13.

[RPV+13] Reiter, S., M. Pressler, A. Viehl, O. Bringmann, and W. Rosenstiel: *Reliability assessment of safety-relevant automotive systems in a model-based design flow*. In *Design Automation Conference (ASP-DAC), 2013 18th Asia and South Pacific*, pages 417–422, Jan 2013.

[SSSI11] Stallkamp, J., M. Schlipsing, J. Salmen, and C. Igel: *The german traffic sign recognition benchmark: A multi-class classification competition*. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 1453–1460, July 2011.

[Sti13] Stickel, Christoph: *Simulation und Modellierung von Witterungsbedingungen in der Auswertung videobasierter Umfeldsensorik*. 2013.

HOPE: Hardware Optimized Parallel Execution

Aquib Rashid and Prof. Dr. Hardt

Abstract: **Feature points are required for matching different images of a single scene, taken from different viewpoints. Hardware implementation of feature point detectors is a challenging task as an optimal balance between localization accuracy and detector efficiency has to be met. Renowned methods like SIFT, SURF, and ORB provide promising results, but with their sequential character and complexity, they require high-level computing resources. These approaches are not appropriate for hardware-based parallel execution. This paper presents a novel algorithm for parallelized feature point detection. It uses a P-controller, which iteratively enhances the detection quality. The work focuses on minimization of logic resource consumption while generating feature detections with high matching scores.**

I. Introduction

Feature matching algorithms comprise of various steps particularly feature detection, orientation computation, descriptor extraction, and descriptor matching. The detection of features in an image is just the initial step in the process, followed by orientation computation, descriptor extraction and descriptor matching. A feature represents an image location which could be unambiguously recognized in different image, possibly taken from a different perspective, angle or position. Features can be edges, corners or small image blocks. Feature points are less ambiguous compared to corners or edges. The main characteristic qualifying a feature is its repeatability and distinctness over various image representations of the same scenario.

In hardware implementation of computer vision algorithms the main focus is to find balance between high-speed and better performance with a minimum of resource utilization. Various feature detectors like SIFT, SURF, and ORB generate a large number of features, though it is beneficial in the object recognition applications, but requires high resources. As for each feature point, its orientation and corresponding descriptor have to be computed and stored in the memory. This in turn results in large computational power requirements and memory.

The quantity of detected features can be easily reduced by using Harris response to retain only the best features. ORB, though not being scale invariant compared to SIFT and SURF is simpler and faster than the later with lower localization error [1]. Software implementation of ORB performs sorting of all the detected FAST-12 features and retains only the required number of best features. Sorting, however, is another problem for hardware implementation. Though it can be implemented, it is computationally expensive. We address this problem by utilizing iterations to obtain similar results as that from sorting.

HOPE is designed to exploit the parallel processing of FAST and Harris corner detector. Harris corner detector is used to iteratively select high quality features from all the detected features within FAST detector at the end of each frame [7].

II. Related work

A lot of research has been done in the field of feature point detectors in the last decade. Harris Corner detector is one of the earliest feature point detectors which reduces the computation time and increases repeatability drastically compared to edge detection algorithms. Harris algorithm is based on auto-correlation function which captures the essence of a region [5]. Scale invariant version of Harris was introduced later but was overshadowed with the advent of SIFT. Scale Invariant feature transform (SIFT) detects features by generating a scale-space pyramid. This is obtained by first down-sampling the image and then convolving it with Gaussian kernel. This

ISBN 978-3-00-048889-4

process is repeated, resulting in a stack of blurred images from higher to lower sizes, thus forming a pyramid shape. A candidate key-point is detected as a feature if all its surrounding 8 and corresponding 9 key-points in the upper and lower layers each are all lower or higher than it. Thus, features generated from SIFT are scale invariant. Speed-up robust feature detector (SURF), uses integral images for generating different box filter kernels which are approximation of Gaussian kernel. These box filters are then used to convolve with the original high resolution image to generate various scaled and blurred image stack. Thus, instead of down-sampling the image, the filter is altered to generate image stack. SIFT and SURF which were later introduced, produce higher quality of features which are scale-, translation-, rotation-invariant, but are computationally expensive and require high resources for execution [2][3]. Later FAST detector was introduced which is simpler to implement and generates features in real time [4]. However, FAST generates high number of features for heterogeneous images and thus results in high resource utilization. ORB a relatively new algorithm utilizes the simplicity of FAST detector and quality measure from Harris response to generate feature points. However, sorting in hardware is computationally expensive and our target is to reduce the complexity [1].

III. HOPE Architecture

HOPE generates features only when both FAST and Harris detector agree on a candidate key-point. FAST-12 detects features when in a circle of radius 3 pixels, at least 12 of the contiguous pixels are all higher or lower than the center pixels by some threshold [4]. Harris on the other hand detects features when corner response is higher than some given threshold. Depending on this threshold the number of detected features can be higher or lower [5]. Evaluator, as shown in Figure 1, checks if both the detectors are agreeing on any candidate to be a feature and only then declares it as detected HOPE feature.

If the number of features generated is higher than the required number of features then P-controller correspondingly sets a higher threshold for the next frame. In a video stream of sufficient similarity, the consecutive frames have very less variation from each other. Thus, threshold computed from the previous frame can be used for the next frame. This results in reduced number of features as compared to the previous frame. If the number is lower than expected then the P-controller updates the threshold correspondingly for the next frame. Thus, parallel processing model combined with P-controller, for adaptive thresholding, provides required number of high quality features in real time [7].

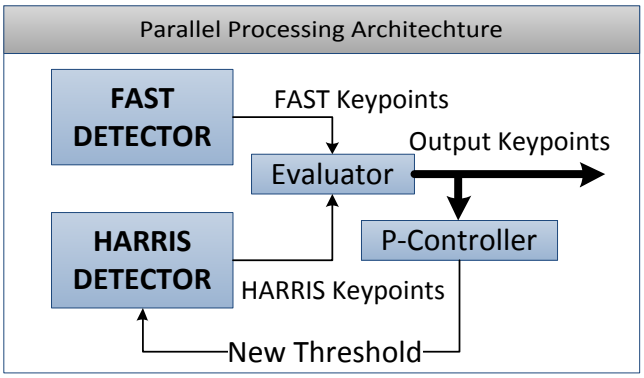


Figure 1: Modular design of HOPE [7]. FAST detector features and Harris detector features are evaluated in Evaluator. Numbers of feature points generated after each frame completes, are given as actuating variable to P-Controller which generates appropriate New Threshold for the next frame.

A. Implementation

HOPE has been implemented in MATLAB 2011a. Proportional constant of 0.5 and threshold of 100 is set for required number of features in P-controller. OpenCV 2.4.6 is used for SIFT, SURF and ORB implementations. Default threshold of 20 is used in FAST detector [7].

IV. Experiments

The FAST implementation is re-written in a modular design in MATLAB which is easier to be translated into hardware implementation. FAST is later coupled with Harris and P-controller

ISBN 978-3-00-048889-4

implementations. The dataset and evaluation techniques used for this implementation were proposed by Mikolajczyk and Schmid. For simplicity only first two images of Boat, Graffiti and Cars have been used for testing. These are of sufficient similarity. Each dataset consists of 6 images of the same scene taken with increasing level of disturbance either in scale, rotation or illumination [6].

Figure 2: First two images of Graffiti and Boat. Images from Oxford dataset. Graffiti dataset is used to test variations in view point and Boat is used for Scale and Rotation Variations

A. Evaluation Criteria

Correspondence and repeatability are used to evaluate detector performance. Correspondences are the point correspondences; true positives and false negatives; which are found in different scenes of the same object of interest [6]. When the translation and rotation between two images is known, then we can mathematically find for each key-point location in one image a corresponding key-point in another image. This mathematical approach, called homography or perspective transformation, is used to locate key-points in other scenes. It is thus used to compute correspondences between two images. Homography is not completely accurate. Therefore, a small tolerance, i.e., homography threshold is added to the interface. Only a portion of original image is matched for correspondences with the second image as some portions of the original image can be out of the frame. Correspondence is expressed as:

$$\text{Correspondence} = \text{True Positives} + \text{False Negatives} \quad (1)$$

Repeatability is the measure of number of features which are detected repeatedly in different scenes. Repeatability, thus, is the measure of accuracy of the detector [6]. It can be expressed as:

$$\text{Repeatability} = \frac{\text{Number of Correspondces}}{\text{Total Number of Features}} \quad (2)$$

V. Results

Graffiti images represent the view point change, Cars represent the illumination variation and Boat represents the scale and rotation change. This is a general scenario for SLAM application. Although large number of features is generated by ORB detector as compared to HOPE, similar repeatability scores are obtained. Figure 3 illustrates the repeatability scores between first two images of the Graffiti, Cars and Boat dataset with HOPE and ORB. Repeatability scores are comparable with large variation in the utilization of resources.

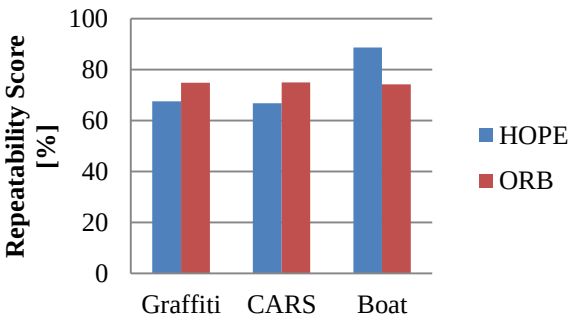


Figure 3: Repeatability scores of HOPE and ORB on first two images of Graffiti, Cars and Boat dataset

Table 1 includes the results of state of art detectors in comparison with HOPE on Graffiti images. The total number of features generated on source image by ORB is almost 13 times greater

than by HOPE, but the results generated by HOPE are similar to that of ORB.

Dataset	Graffiti			
Detector	**HOPE**	**ORB**	**SIFT**	**SURF**
Key-points found	219	3000	2679	2434
Repeatability [%]	67.58	74.86	56.9	57.5

Table 1: Scores of key-points found, repeatability and correspondences by HOPE, ORB, SIFT and SURF detectors on first two images of Graffiti dataset in our reduced interface

SIFT and SURF provides similar results to the image set. Repeatability of HOPE outperforms ORB detector on Boat image set. Figure 4 shows the highly repeatable HOPE features, generated by comparing Harris and FAST detector output. The red and green key-points in FAST detector image output are called positives and negatives. They represent candidate key-points which are higher than at least 12 contiguous pixels and candidate key-points which are lower than at least 12 contiguous pixels, respectively [4]. In Harris detector image output, large amount of key-points are generated initially at lower thresholds and with increase in threshold by P-controller the number of key-points are reduced. HOPE detector evaluates FAST key-points with high threshold candidate key-points from Harris. Thus, key-points or features generated from HOPE have combined properties of FAST and Harris. HOPE key-points are translation and rotation invariant.

As illustrated in Figure 5 and Figure 6, the ratio of number of key-points to correspondences by HOPE and ORB is similar.

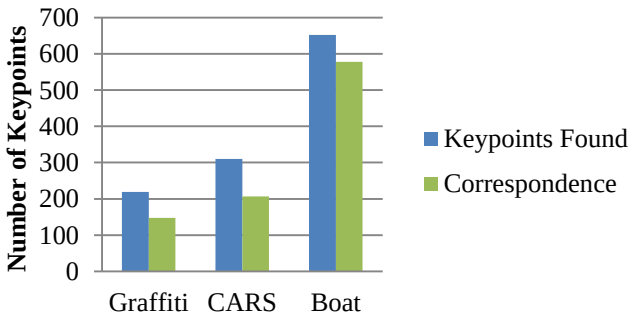


Figure 5: Number of key-points and correspondences generated by HOPE on different images

Key-points generated by ORB are relatively much higher than HOPE, but yield similar results in repeatability.

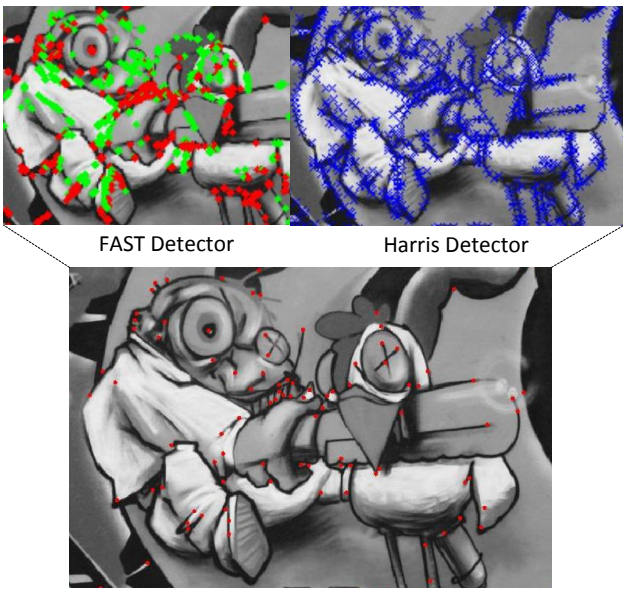


Figure 4: Combination of the FAST and Harris detector key-points results in HOPE key-points which are lesser in number but highly repeatable in different images [7]

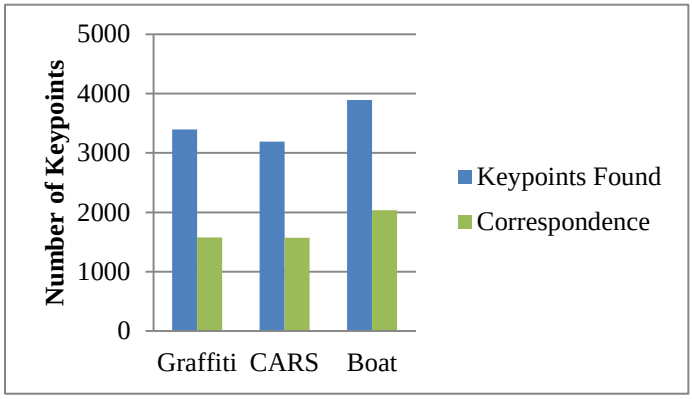


Figure 6: Number of key-points and correspondences generated by ORB on different images

HOPE requires more time because the iterations need to adjust the threshold for Harris. However, this increase in time is a small price to pay in comparison to the minimized amount of resources. This approach adaptively adjusts the Harris threshold for changing scenes, which in turn results in a robust feature point detector.

ISBN 978-3-00-048889-4

VI. CONCLUSIONS

In this paper we have proposed and tested a new feature point detector which utilizes FAST and Harris detectors in parallel with P-controller for providing adaptive threshold for each new frame. At the end of each frame the threshold is reset with respect to the difference in required and obtained number of feature points. Further steps in this work will include analysis of HOPE performance in combination with various binary descriptors. In future HOPE can be used in automotive, robotics and aerospace industry [8].

ACKNOWLEDGMENT

We acknowledge the efforts of department of Computer Engineering at Technische Universitaet Chemnitz for collaboration in this work. Special thanks to Mr. Arne Zender, Mr. Hamzah Ijaz and Mr. Sahil Sholla for their feedback and suggestions.

REFERENCES

[1] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In Computer Vision (ICCV), 2011 IEEE International Conference on, pages 2564-2571, Nov 2011.

[2] David G. Lowe. Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vision, 60(2):91-110, November 2004.

[3] Herbert Bay. From wide-baseline point and line correspondences to 3D. PhD thesis, Swiss Federal Institute of Technology, ETH Zurich, 2009.

[4] Edward Rosten and Tom Drummond. Fusing points and lines for high performance tracking. In IEEE International Conference on Computer Vision, volume 2, pages 1508-1511, October 2005.

[5] Chris Harris and Mike Stephens. A combined corner and edge detector. In Proc. of Fourth Alvey Vision Conference, pages 147-151, 1988.

[6] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaf-falitzky, T. Kadir, and L. Van Gool. A comparison of affine region detectors. Int. J. Comput. Vision, 65(1-2):43-72, November 2005.

[7] Aquib Rashid. Hardware Iimplementation of a Robust Feature Point Detector. Master Thesis, Technische Universitaet Chemnitz, Department of Computer Engineering, Faculty of Computer Science, Strasse der Nationen 62, 09111 Chemnitz, Germany, July 2014.

[8] Stephan Blokzyl and Matthias Vodel and Wolfram Hardt. FPGA-based Approach for Runway Boundary Detection in High-resolution Colour Images. In Proceedings of the Sensors & Applications Symposium (SAS2014), IEEE Computer Society, February 2014.

Execution Tracing of C Code for Formal Analysis
(Extended Abstract)*

Heinz Riener* Michael Kirkedal Thomsen* Görschwin Fey*‡
*Institute of Computer Science ‡Institute of Space Systems
University of Bremen, Germany German Aerospace Center, Germany
{hriener,kirkedal,fey}@informatik.uni-bremen.de

Abstract

Many formal tools in verification and debugging demand precise analysis of execution traces. In this paper, we describe Tracy, an adaptable framework for execution tracing of imperative C code that executes a program on given inputs and logs the corresponding execution trace. The logged execution trace is a sequence of symbolic expressions describing the data-flow transformations from the initial state to the final state of the execution. Tracy is easily customizable and allows for defining code annotations without affecting the semantics of the C code; thus normal execution is possible. In the current version, Tracy supports an expressive subset of C. Additionally, Tracy offers API language bindings for C++. To show the extensibility of Tracy, we have used it in combination with *Satisfiability Modulo Theories* (SMT) solvers to enable trace-based reasoning like concolic execution or fault diagnosis.

1. Introduction

Lazy reasoning is a key technique in formal analysis of large-scale software programs. Formal analysis, however, demands good abstraction. On the one hand, the abstraction should not be too coarse to avoid false positives. On the other hand, the abstraction should not be too precise to avoid scalability issues. A prominent approach to lazy reasoning about software, is to abstract by considering a subset of the possible execution traces of a program. This is essentially the idea in path-based white-box reasoning [GKS05, SA06, CDE08, CGP+08, McM10]: the program source is path-wise traversed and analyzed; new paths are considered on demand to refine the analysis. In order to make path-based reasoning effective in practice, learning techniques guide the path exploration by previously derived information.

Extracting detailed information from a program execution is a common task in program analysis, testing, and debugging. Existing tools, however, fail to provide sufficient tracing information to the user-side that could be leveraged for automated formal analysis. This hardens the development of new tools and forces developers to re-engineer parsing and analyzing the source code of a program.

In this paper, we propose, Tracy, an adaptable framework for execution tracing of imperative C programs that separates execution tracing from reasoning. Given a C program and an assignment

*This work was supported by the *German Research Foundation* (DFG, grant no. FE 797/6-1) and *European Commission* under the 7*th* *Framework Programme*.

ISBN 978-3-00-048889-4

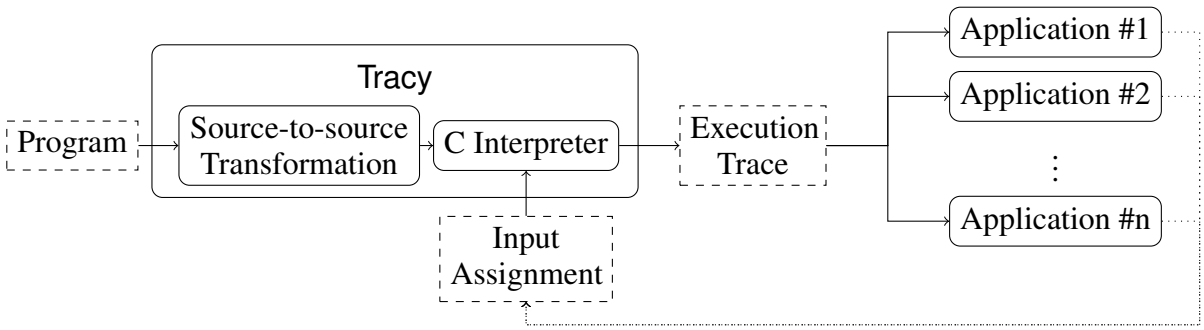


Figure 1: Outline of Tracy's interfacing with application tools.

to the program inputs, Tracy runs the program on the given inputs and logs the corresponding execution trace. The logged execution trace is a sequence of symbolic expressions and symbolically describes the data-flow transformations from the initial state to the final state along the execution. The concrete inputs are used to fix the control-flow path and guide the analysis to a specific program path.

Tracy uses a simple trace grammar to describe execution traces and dumps them in a readable XML format. Moreover, API language bindings for C++ are provided, which can be used to interface Tracy with reasoning tools written in C++.

Our contributions in this paper are threefold:

1. We present, Tracy, an adaptable framework for execution tracing of C programs to separate execution tracing from reasoning.

2. We provide an easy to use API interface for C++ to interface Tracy with reasoning tools written in C++.

3. We provide a simple trace grammar for execution traces, which may be used as a starting point for standardizing the output format of software analysis tools.

The tool (including source code) is publicly available at:
`http://github.com/kirkedal/tracy/`

2. The Tracy Execution Tracer

The overall architecture of Tracy interfaced with path-based reasoning tools, called *applications*, is shown in Figure 1. Dashed boxes denote inputs, whereas solid boxes denote tools.

Tracy semantically interprets an imperative C program statement by statement and dumps the corresponding execution trace in a symbolic representation. We assume deterministic programs. As a consequence, the input assignment provided to Tracy unambiguously fixes the execution trace. Optionally, the C program is first normalized, e.g., by applying source-to-source transformations. As output, Tracy dumps the execution trace. This logged execution trace can be read by one or more application tools. Also, the applications may provide new input assignments such that Tracy in combination with applications implements an iterative path-based abstraction refinement loop.

The trace grammar used by Tracy is shown in Figure 2. A trace is a sequence of actions. Each action is of one of the following forms:

- Declaration (`declaration`) of a variable with a given *type* and a *name*.

```
Trace  ::= Action*
Action ::= <declaration type=Type name=Identifier />
         | <assign lhs=Identifier rhs=Expr />
         | <assume condition=Expr />
         | <assert condition=Expr />
         | <annotation label=Label>
           Trace
           </annotation>
         | <debug information=String />
```

Identifier: variable name.
Type: type name.
Expr: an expression formatted in the respective output format.
Label: text string excluding white spaces.
String: text string including white spaces.

Figure 2: Trace grammar.

- Assignment (`assign`) of a variable on the *left-hand side* (lhs) to the result of an expression on the *right-hand side* (rhs).
- Assumption (`assume`) of a *condition* of Boolean type. The input assignment to Tracy fixes the control-flow path of the execution trace. Conditions of branches and loops are added as assumptions to the execution trace.
- Assertion (`assert`) of a *condition* of Boolean type. Assertions are generated from local assertions in a program's source code. Local assertions are commonly used to express local invariants corresponding to safety specifications.
- Annotation (`annotation`) can be used to group sequences of actions and mark them with a *label*. We exploit this feature for marking specification code or faulty parts of a program and pass this information to the application-side.
- Debug information (`debug`) provides additional *information* to the user-side. For instance, Tracy allows for dumping the computed concrete values during tracing, which aids in understanding Tracy's results.

The expression format used by Tracy is designed to be equal to SMT-LIB2 [BST10] strings to allow for simple interfacing with SMT solvers. In the current version, Tracy supports different SMT theories including `QF_(A)BV` and `QF_NIA`. Extensions to other formats are possible by providing additional pretty printers for Tracy's abstract syntax tree.

3. Example: Trace-Based Fault Diagnosis

To demonstrate the ease of using Tracy in applications, we have used Tracy in combination with SMT solvers to implement a trace-based fault diagnosis tool similar to [MSTC14].

The implemented approach to trace-based fault localization uses sequencing interpolants. Notice that the approach consists of two steps which demonstrate the separation of execution tracing and formal reasoning. Firstly, an execution trace is logged, i.e., a faulty program and an input

ISBN 978-3-00-048889-4

assignment that corresponds to a counterexample is passed to Tracy. Secondly, the logged execution trace is parsed and loaded with Tracy's C++ API. Our implementation of the fault diagnosis algorithm uses the API of an SMT solver to build an unsatisfiable SMT instance and to compute a sequencing interpolant for this SMT instance [McM11]. The sequencing interpolant describes the progress of the execution towards the error and is a sequence of inductive invariants that have to hold along the execution trace to reach the error. From the sequencing interpolant, a set of fault candidates is derived which serves as a diagnosis of the error. Adjacent invariants of the sequencing interpolant that do not change do not affect the progress towards the error and thus all statements enclosed by these invariants are considered correct. On the other hand, statements enclosed by adjacent invariants that change are marked as fault candidates.

Figure 3 shows a simple introductory example program and the execution trace produced by Tracy for the input assignment `(a=1,b=0,c=1)`. The program, `minmax`, is taken from Groce et al. [GCKS06]. On the top left, the C source code is shown. The source code of the program is faulty; the program location `L7` should read `least = b;`. On the top right, the execution trace dumped by Tracy is shown in XML. On the bottom, the same execution trace is shown as a graph. For the sake of clearness, we avoid using the XML tag syntax. The nodes of the graph represent actions, whereas the edges of the graph represent control-flow. We used the SMT solver Z3 [dMB08] to compute a sequencing interpolant for the execution trace and leveraged the SMT-LIB2 theory QF_BV corresponding to the quantifier-free fragment of first-order logic modulo bit-vector arithmetic. To reduce the size of the logic formulae, the bit-width of an integer in C has been reduced to 2 bit. This can be automatically done utilizing Tracy's bit-width reduction feature. The sequencing interpolant is shown as edge labels in the graph. The fault diagnoses, i.e., changing adjacent invariants of the sequencing interpolant, are marked in the graph with gray color.

4. Conclusion

We presented Tracy, an adaptable framework for execution tracing of C programs, supporting an expressive subset of C. Tracy allows for separating execution tracing from reasoning and provides C++ API language bindings which can be used to interface Tracy with applications written in C++. Moreover, a simple grammar has been proposed to describe execution traces, This grammar can be seen as a starting point for standardizing the input/output-format of trace-based reasoning tools.

References

[BST10] Barrett, Clark, Aaron Stump, and Cesare Tinelli: *The SMT-LIB standard version 2.0*, 2010.

[CDE08] Cadar, Cristian, Daniel Dunbar, and Dawson R. Engler: *KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs*. In *Operating Systems Design and Implementation*, pages 209–224, 2008.

[CGP+08] Cadar, Cristian, Vijay Ganesh, Peter M. Pawlowski, David L. Dill, and Dawson R. Engler: *EXE: Automatically generating inputs of death*. ACM Transactions on Information and System Security, 12(2):322–335, 2008.

[dMB08] Moura, Leonardo de and Nikolaj Bjørner: *Z3: An efficient SMT solver*. In *Tools and Algorithms for Construction and Analysis of Systems*, pages 337–340, 2008.

ISBN 978-3-00-048889-4

```
void minmax(int a,int b,int c)
{
/*L00*/int least = a;
/*L01*/int most = a;
/*L02*/if (most < b)
/*L03*/  most = b;
/*L04*/if (most < c)
/*L05*/  most = c;
/*L06*/if (least > b)
/*L07*/  most = b; // ERROR!
/*L08*/if (least > c)
/*L09*/  least = c;
/*L10*/assert(least <= most);
}
```

```
<declaration type="(_ BitVec 2)" name="least" />
<declaration type="(_ BitVec 2)" name="most" />
<declaration type="(_ BitVec 2)" name="a" />
<declaration type="(_ BitVec 2)" name="b" />
<declaration type="(_ BitVec 2)" name="c" />
<assign line="0"  lhs="least" rhs="a" />
<assign line="1"  lhs="most" rhs="a" />
<assume line="2"  condition="(not (bvslt most b))" />
<assume line="4"  condition="(not (bvslt most c))" />
<assume line="6"  condition="(bvsgt least b)" />
<assign line="7"  lhs="most" rhs="b" />
<assume line="8"  condition="(not (bvsgt least c))" />
<assert line="10" condition="(bvsle least most)" />
```

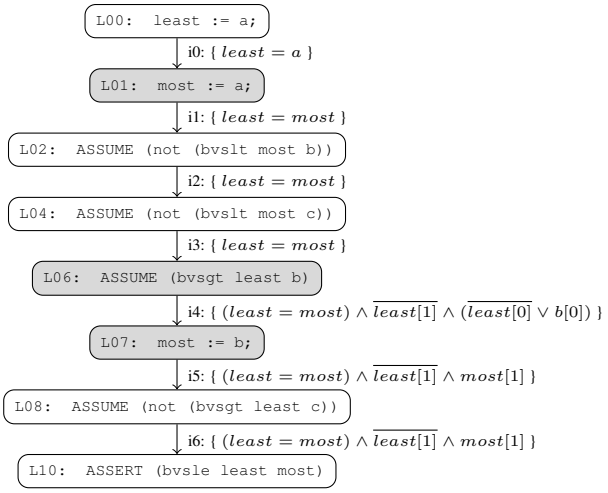


Figure 3: Trace-based error diagnosis.

[GCKS06] Groce, Alex, Sagar Chaki, Daniel Kroening, and Ofer Strichman: *Error explanation with distance metrics*. International Journal on Software Tools for Technology Transfer, 8(3):229–247, 2006.

[GKS05] Godefroid, Patrice, Nils Klarlund, and Koushik Sen: *DART: Directed automated random testing*. In *Programming Language Design and Implementation*, pages 213–223, 2005.

[McM10] McMillan, Kenneth L.: *Lazy annotation for program testing and verification*. In *Computer Aided Verification*, pages 104–118, 2010.

[McM11] McMillan, Kenneth L.: *Interpolants from Z3 proofs*. In *Formal Methods in Computer-Aided Design*, pages 19–27, 2011.

[MSTC14] Murali, Vijayaraghavan, Nishant Sinha, Emina Torlak, and Satish Chandra: *What gives? A hybrid algorithm for error explanation*. In *Verified Software: Theories, Tools, Experiments*, pages 270–286, 2014.

[SA06] Sen, Koushik and Gul Agha: *CUTE and jCUTE: Concolic unit testing and explicit path model-checking tools*. In *Computer Aided Verification*, pages 419–423, 2006.

Verbesserung der Fehlersuche in inkonsistenten formalen Modellen

(Erweiterte Zusammenfassung)

Nils Przigoda[1] Robert Wille[1,2] Rolf Drechsler[1,2]
[1]Arbeitsgruppe Rechnerarchitektur [2]Cyber Physical Systems
Universität Bremen DFKI GmbH
28359 Bremen 28359 Bremen
E-Mail: {przigoda,rwille,drechsle}@informatik.uni-bremen.de

1. Einleitung, Hintergrund und Motivation

Modellierungssprachen wie die *Unified Modeling Language* (UML) [RJB99] oder die *Systems Modeling Language* (SysML) [MM05, Wei07] rückten in der jüngeren Vergangenheit vermehrt in den Fokus der Forschung. Dies liegt unter anderem an der Möglichkeit, Systeme bereits vor Beginn der Implementierung präzise als formales Modell zu spezifizieren und anschließend zu analysieren. Die Vorabanalyse des Modells wird durch die *Object Constraint Language* (OCL) [WK99] besonders interessant: Sie bietet unter anderem die Möglichkeit, sogenannte *Invarianten* zu definieren, welche z. B. die Anzahl valider Instanziierungen von UML-Klassendiagrammen einschränken und eine präzisere Spezifikation einzelner Komponenten ermöglichen. In dieser Arbeit wird UML in Kombination mit OCL als Modellierungssprache betrachtet.
Bei der Modellierung von Systemen kann die Komplexität der verschiedenen Beschreibungen z. B. für Komponenten und Relationen sowie sonstigen Abhängigkeiten schnell sehr stark ansteigen. Dies kann zu nicht trivialen oder unübersichtlichen Modellen führen, welche Fehler enthalten, die auf den ersten Blick schwer erkennbar sind. Solche Fehler können etwa dazu führen, dass keine gültige Instanziierung eines Modells existiert. Um dies zu überprüfen wurden in der Vergangenheit sogenannte *Konsistenzprüfer* entwickelt [GBR07, WSD12], die entweder einen gültigen Systemzustand erzeugen (und somit die Konsistenz des Modells zeigen) oder nachweisen, dass zu einem Modell keine gültige Instanz existiert (und damit die Inkonsistenz eines Models beweisen). Auf diese Weise können Fehler bereits auf sehr abstrakter Ebene erkannt und behoben werden, was sich als deutlich kosten- und zeitgünstiger erweist als in späteren Entwurfsschritten. Als problematisch erweist sich dabei, dass Konsistenzprüfer nur die bloße Existenz einer Inkonsistenz nachweisen. Für die Ermittlung deren Ursache, d. h. die eigentliche Fehlersuche, wären allerdings auch Hinweise zu den Gründen der Inkonsistenz wünschenswert.
Bisherige Ansätze können dies jedoch nur sehr begrenzt leisten. Im *UML-based Specification Environment* (USE) [GBR07] können zwar mehr oder weniger automatisch Beispiele, welche die Konsistenz eines Modells zeigen, erzeugt werden, jedoch erhält der Modellierer im Falle eines fehlerhaften Modells keine Informationen über die Ursachen der Inkonsistenz. Hinzu kommt, dass USE den Suchraum enumerativ durchläuft, was bereits bei verhältnismäßig kleinen Modellen sehr hohe Laufzeiten zur Folge haben kann.

ISBN 978-3-00-048889-4

In [WSD12] wurde ein auf dem Booleschen Erfüllbarkeitsproblem aufbauender Ansatz vorgestellt, der sogenannte Fehlerkandidaten in der Beschreibung ermittelt. Die Liste der Fehlerkandidaten ist im Allgemeinen allerdings weder vollständig noch enthält sie Informationen über mögliche Zusammenhänge zwischen einzelnen Fehlerkandidaten.
In dieser Arbeit sollen Ideen für einen vollautomatischen Ansatz vorgestellt werden, welcher auf dem zuletzt zitierten Verfahren aufbaut, jedoch die oben genannten Probleme adressiert. Somit erhält der Modellierer am Ende eine Liste von allen sogenannten minimalen Gründen für die Inkonsistenz inklusive der Verbindungen untereinander. Durch diese zusätzlichen Informationen wird er bei der Fehlersuche entscheidend unterstützt.

2. Grundlagen

Damit diese Arbeit in sich geschlossen bleibt, folgen einige Definitionen zur Notation von UML/ OCL-Modellen und den dazugehörigen Systemzuständen.

Definition 1 *Ein* Modell $\mathcal{M} = (\mathcal{C}, \mathcal{R})$ *ist ein Paar von* Klassen $\mathcal{C}$ *und* Relationen $\mathcal{R}$ *(auch* Assoziationen *genannt). Eine* Klasse $c \in \mathcal{C}$ *besteht aus Attributen und Operationen. Eine Relation* $r = (c_1, c_2, (l, u))$ *ist ein Tripel, bestehend aus zwei Klassen* c_1 *und* c_2 *in* $\mathcal{C}$ *sowie einem Paar, welches die sogenannte* untere und obere Schranke *repräsentiert: Jede Instanz der Klasse* c_1 *ist mit mindestens* l *und höchstens* u *Instanzen von* c_2 *verbunden. Diese Schranken werden im Folgenden auch als* UML-Beschränkungen *bezeichnet. Während die untere Schranke eine beliebige natürliche Zahl ist, kann die obere Schranke entweder eine positive natürliche Zahl oder unendlich sein. Darüber hinaus werden* Invarianten *als weiteres Beschreibungsmittel der OCL zugelassen. Die Menge aller Invarianten eines Modells wird mit* $\mathcal{I}$ *bezeichnet. Des Weiteren ist jede Invariante* $i \in \mathcal{I}$ *mit genau einer UML-Klasse assoziiert. Invarianten werden im Folgenden auch* OCL-Beschränkungen *genannt.*

Definition 2 *Eine Instanziierung eines Modells* $\mathcal{M} = (\mathcal{C}, \mathcal{R})$ *wird* Systemzustand *genannt. Ein Systemzustand wiederum wird* zulässig *genannt, wenn in ihm alle UML- und OCL-Beschränkungen gelten. Falls eine Beschränkung in einem Systemzustand nicht erfüllt ist, so wird dieser als* unzulässig *bezeichnet. Außerdem muss jede zulässige Instanziierung von jeder Klasse mindestens ein Objekt enthalten. Ein Model, zu welchem es mindestens einen zulässigen Systemzustand gibt, heißt* konsistentes *Modell. Im Gegensatz dazu wird ein Model, zu welchem es keinen zulässigen Systemzustand gibt,* inkonsistent *genannt.*

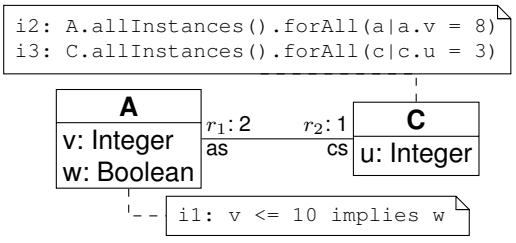
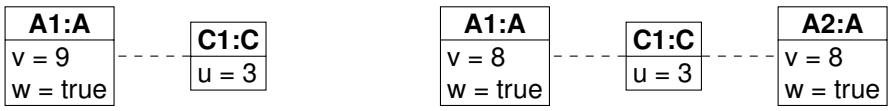


Abbildung 1: Ein einfaches Modell



((a)) Eine unzulässige Instanziierung. **((b))** Eine zulässige Instanziierung.

Abbildung 2: Zwei mögliche Modellinstanziierungen

Beispiel 1 *Man betrachte das in Abbildung 1 dargestellte Modell. Es besteht aus zwei Klassen,* A *und* C*, welche durch zwei Relationen verbunden sind. Die erste Relation r_1 fordert, dass jedes Objekt der Klasse* C *mit genau zwei Objekten der Klasse* A *verbunden ist. Umgekehrt muss jedes Objekt der Klasse* A *mit genau einem Objekt der Klasse* C *verbunden sein. Da die obere und untere Schranke in beiden Relationen gleich sind, wird nur eine Schranke im Klassendiagramm spezifiziert. Des Weiteren gibt es insgesamt drei Attribute sowie drei Invarianten, welche jeweils auf beide Klassen verteilt sind. Die drei Invarianten schränken die zulässigen Werte für die Attribute ein.*

Eine Anwendung von Definition 2 auf die in Abbildung 2(a) dargestellte Instanziierung des Modells führt dazu, dass die Instanziierung als unzulässig *erkannt wird. Dies liegt unter anderem daran, dass die Relation r_1 verletzt ist, denn das Objekt C1 (eine Instanz der Klasse C) sollte mit genau zwei Objekten der Klassen A verbunden sein: Es ist jedoch nur mit dem Objekt A1 verbunden. Wegen dieser Relation ist es offensichtlich auch nicht möglich, eine gültige Instanz mit genau einem Objekt der Klasse A und genau einem Objekt der Klasse B zu finden. Neben der Relation r_1 ist in Abbildung 2(a) auch die Invariante i_2 verletzt, da nicht jedes Attribut v von Objekten der Klasse A mit dem Wert 8 belegt ist.*
Im Gegensatz dazu zeigt Abbildung 2(b) einen zulässigen Systemzustand. Zunächst sind beide Relationen erfüllt, da das Objekt C1, als einzige Instanz der Klasse C, mit genau zwei Objekten der Klasse A verbunden ist. Umgekehrt sind genau diese beiden Objekte, A1 und A2, lediglich mit C1 verbunden. Des Weiteren lässt sich aus der Belegung der Variablen folgern, dass alle drei Invarianten erfüllt sind. Die Invariante i_1 verlangt, dass für jedes Objekt der Klasse A *das Boolesche Attribut w `true` ist, wenn der Wert von Attribut v kleiner oder gleich 10 ist. Da sowohl für A1 als auch für A2 v mit 8 belegt ist und w `true` ist, ist diese Invariante erfüllt. Die Invariant i_2 ist ebenfalls nicht verletzt, da in jedem Objekt der Klasse A das Attribut v mit dem Wert 8 ist. Ebenso ist das Attribut u aller Objekte der Klasse C – es gibt nur das Objekt C1 – mit dem Wert 3 belegt.*

Da mit Abbildung 2(b) ein zulässiger Systemzustand gefunden wurde, ist das Modell aus Abbildung 1 konsistent.

3. Problembeschreibung

Nachdem im vorherigen Abschnitt die grundlegenden Notationen eingeführt und anhand eines Beispiels erläutert wurden, wird in diesem Abschnitt zunächst das Problem der Fehlersuche in inkonsistenten Modellen anhand eines Beispiel erklärt und motiviert. Anschließend folgt eine formale Definition.

ISBN 978-3-00-048889-4

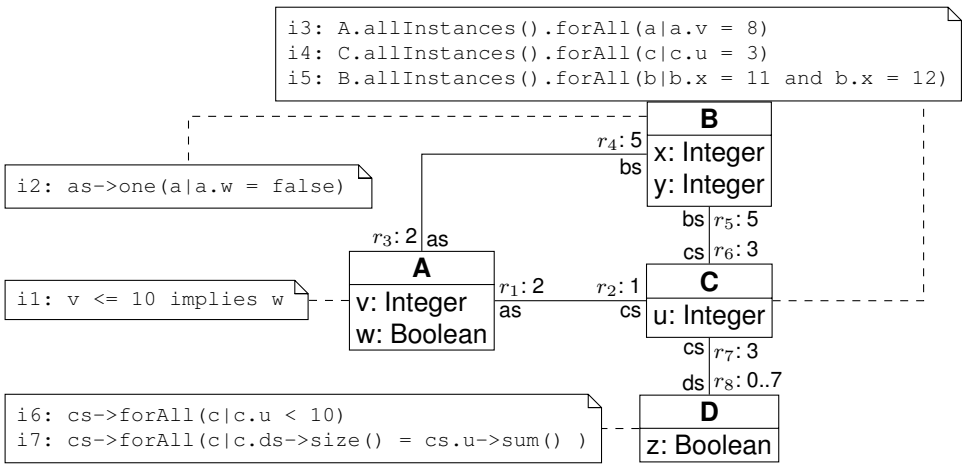


Abbildung 3: Ein inkonsistentes Modell

Beispiel 2 *Gegeben sei das in Abbildung 3 dargestellte Modell*[1]*. Das Modell besteht aus vier Klassen mit ein bis zwei Attributen, vier Relationen untereinander und insgesamt sieben Invarianten. Zusätzlich sei angenommen, dass die Klassen A, B, C und D jeweils 2, 5, 3 und 7 mal instanziiert werden. Auf den ersten Blick ist eventuell nicht offensichtlich, dass es sich hierbei um ein inkonsistentes Modell handelt. Dies ist jedoch der Fall, da das Modell die folgenden Widersprüche enthält:*

- *Die Invariante i_5 kann niemals erfüllt werden.*
- *Die Invariante i_3 fordert, dass für jedes Objekt der Klasse A das Attribut v auf 8 gesetzt ist. Außerdem wird es für einen zulässigen Systemzustand wegen der Invariante i_1 nötig, dass, wenn das Attribut v (eines Objektes der Klasse A) kleiner oder gleich 10 ist, w den Wert `true` annimmt. Andererseits muss für die Gültigkeit von i_2 jedes Objekt von B mit genau einem Objekt der Klasse verbunden, in dem das Attribut w den Wert `false` annimmt. Folglich bilden die drei Invarianten zusammen einen Widerspruch.*
- *Mit der angenommenen Instanziierung ergibt sich außerdem noch ein Widerspruch bezüglich der Relationen r_1 und r_2.*
- *Schlussendlich enthält das Modell auch einen Widerspruch, der aus der Kombination von UML- und OCL-Beschränkungen i_4, i_7 und r_7 besteht.*

Das vorangegangene Beispiel macht deutlich, dass die Fehlersuche für den Modellierer schnell zu einer zeitintensiven Aufgabe werden kann. Dies macht Methoden, die automatisch Gründe für Inkonsistenzen ermitteln und somit den Prozess der Fehlersuche beschleunigen, wünschenswert.
Die möglichen Gründe für die Widersprüche einer Teilmenge der UML/OCL-Beschränkungen lassen sich wie folgt formalisieren:

Definition 3 *Für ein inkonsistentes UML/OCL Modell $\mathcal{M} = (\mathcal{C}, \mathcal{R})$ – wie in Definition 2 eingeführt – wird eine nicht-leere Teilmenge von Beschränkungen $B \subseteq \mathcal{B}$ $(:= \mathcal{R} \cup \mathcal{I})$ als* Grund

[1] Dieses Modell wurde von den Autoren so gewählt, dass es möglichst einfach das Problem skizziert.

bezeichnet, wenn bereits die Konjunktion von allen Beschränkungen $\bigwedge_{b_i \in B} b_i$ *logisch äquivalent zu* 0 *ist, m. a. W. die Beschränkungen bilden bereits einen Widerspruch.*

Beispiel 3 *Wie in Beispiel 2 bereits diskutiert, ist das Modell in Abbildung 3 aus vier Gründen inkonsistent: 1.* $B_1 = \{i_5\}$*, 2.* $B_2 = \{i_1, i_2, i_3\}$*, 3.* $B_3 = \{r_1, r_2\}$ *und 4.* $B_4 = \{i_4, i_7, r_7\}$*.*

4. Vorgeschlagener Lösungansatz

In diesem Abschnitt wird ein Lösungsansatz beschrieben, welcher auf automatische Weise eine minimale Menge von Gründen für inkonsistente Modelle ermittelt. Dabei werden Verfahren zur Lösung Boolescher Erfüllbarkeit eingesetzt, wie sie bereits in [SWK+10] für die Konsistenzprüfung verwendet wurden. Im Folgenden werden die wesentlichen Aspekte des Ansatzes aus [SWK+10] kurz beschrieben. Anschließend wird die hier vorgeschlagene Erweiterung dieser Lösung erläutert. In [SWK+10] wird das Problem der Konsistenzprüfung, d. h. die Ermittlung eines zulässigen Systemzustandes oder der Nachweis, dass ein solcher nicht existiert, auf das Boolesche Erfüllbarkeitsproblem (engl. satisfiability problem; kurz: SAT) abgebildet. Anschließend wird die resultierende SAT-Instanz durch sogenannte SAT-Beweiser gelöst. Für ein gegebenes UML-Modell $\mathcal{M} = (\mathcal{C}, \mathcal{R})$ mit Invarianten $\mathcal{I}$ lässt sich die verwendete SAT-Instanz beschreiben mit

$$f_{\text{con}} = \Phi(\mathcal{M}) \wedge \bigwedge_{r \in \mathcal{R}} [\![r]\!] \wedge \bigwedge_{i \in \mathcal{I}} [\![i]\!], \text{wobei} \tag{1}$$

- $\Phi(\mathcal{M})$ eine logische Teilaussage ist, die alle Informationen zu den UML-Komponenten im Systemzustand wie Objekte, Attribute und Links repräsentiert,
- $[\![r]\!]$ eine logische Teilaussage ist, welche für alle betroffenen Objektinstanzen der entsprechenden Klassen die UML-Beschränkung der Relation $r \in \mathcal{R}$ repräsentiert und
- $[\![i]\!]$ eine logische Teilaussage ist, welche für alle betroffenen Objektinstanzen der Klasse, zu welcher die Invariante i gehört, beschreibt.

Details zu dem Transformationen der einzelnen Teilaussagen können [SWK+10, SWD11] entnommen werden.
Wenn der eingesetzte SAT-Beweiser eine erfüllende Belegung für diese Gleichung ermitteln kann, lässt sich daraus ein zulässiger Systemzustand abgeleiten. Dies zeigt die Konsistenz des Modells. Wenn ein SAT-Beweiser jedoch keine erfüllende Belegung findet, so ist erwiesen, dass das Modell inkonsistent ist und der Modellierer alle Beschränkungen bei der Fehlersuche genauer betrachten muss. Um ihn hierbei zu unterstützen, wird nun vorgeschlagen, die Gleichung (1) so zu erweitern, dass für einzelne Prüfungen verschiedene Beschränkungen durch den SAT-Beweiser deaktiviert werden können. Zu diesem Zweck wird für jede Beschränkung b eine neue freie Variable s_b hinzugefügt und die SAT-Instanz wie folgt erweitert:

$$f'_{\text{con}} = \Phi(\mathcal{M}) \wedge \bigwedge_{r \in \mathcal{R}} (s_r \vee [\![r]\!]) \wedge \bigwedge_{i \in \mathcal{I}} (s_i \vee [\![i]\!]) \tag{2}$$

ISBN 978-3-00-048889-4

Dies ermöglicht, dass während der Suche nach erfüllenden Belegungen (und damit quasi nach gültigen Systemzuständen) bestimmte Bedingungen ignoriert werden können[2]. Ein SAT-Beweiser wird für f'_{con} im Regelfall eine erfüllende Belegung finden. Die jeweiligen Belegungen der s_b-Variablen können anschließend zur Ermittlung der Gründe für eine Inkonsistenz genutzt werden – enthalten sie doch Informationen darüber, welche Beschränkungen der SAT-Beweiser deaktivieren musste, um überhaupt einen gültigen Systemzustand zu generieren.

Eine offensichtliche Lösung wäre zum Beispiel:

$$s_{r_1} = 1,\ s_{r_2} = 1,\ s_{r_3} = 1,\ s_{r_4} = 1,\ s_{r_5} = 1,\ s_{r_6} = 1,\ s_{r_7} = 1,\ s_{r_8} = 0,\ s_{i_1} = 1,\ s_{i_2} = 1,\ s_{i_3} = 1,\ s_{i_4} = 1,\ s_{i_5} = 1,\ s_{i_6} = 1, \text{ und } s_{i_7} = 1.$$

Diese Belegung lässt den Schluss zu, dass die Deaktivierung aller Beschränkungen bis auf r_8 das Modell konsistent machen würde. Leider ist diese Erkenntnis allein nicht wirklich hilfreich. Betrachtet man jedoch *alle* möglichen Kombinationen von Belegungen, die das Modell konsistent werden lassen, lassen sich hieraus alle minimalen Gründe ableiten.
Zur Verdeutlichung sind alle möglichen Kombinationen in Tabelle 1 gelistet. Da es insgesamt 9408 Belegungen gibt, sind Lösungen zusammengefasst worden. Dabei steht eine 1 für die Deaktivierung der jeweiligen Beschränkung, eine 0 besagt, dass die Beschränkung erfüllt wurde. Durch einen − wird ein sogenannter *Don't-Care* dargestellt, d. h. es ist egal, ob die entsprechende Beschränkung deaktiviert oder aktiviert ist. Zusätzlich sind in der letzten Zeile bestimmte Eigenschaften für eine oder mehrere Beschränkungen zusammengefasst, welche im Folgenden erklärt werden sollen.

Die wohl am leichtesten nachvollziehbare Schlussfolgerung aus der Tabelle ist die Selbstwidersprüchlichkeit der Invariante i_5. Sie lässt sich daran erkennen, dass die Invariante in jeder der gefundenen Lösungen deaktiviert sein muss. Dies wurde in der letzten Spalte durch eine **1** vermerkt. Analog hierzu kann man aus den Spalten für die Relationen r_3, r_4, r_5, r_6, r_8 und die Invariante i_6 folgern, dass keine dieser Einschränkungen Teil eines Grundes sein kann, da ihr Wert in jeder Zeile ein Don't-Care ist.
Andererseits kann man den ersten beiden Spalten entnehmen, dass in jeder gefundenen Lösung mindestens eine der beiden Relation r_1 und r_2 deaktiviert ist. Dies legt die Vermutung nahe, dass die beiden Relationen zusammen einen Grund für die Inkonsistenz des Ausgangsmodels bilden. Ähnliche Schlussfolgerungen können sowohl für i_1, i_2 und i_3 als auch für i_4, i_7 und r_7 gezogen werden.
Folglich können der Tabelle folgenden Gründe entnommen: 1. $\{r_1, r_2\}$, und 2. $\{i_1, i_2, i_3\}$, 3. $\{i_5\}$, 4. $\{i_4, i_7, r_7\}$. Dies sind – abgesehen von der Reihenfolge – genau die bereits in Beispiel 3 genannten Gründe.

Die ermittelten Gründe sind auch minimal, da alle Belegungen betrachtet wurden. Wäre einer der Gründe nicht minimal, so müsste es mindestens eine weitere Belegung geben, in welcher alle Beschränkungen dieses Grundes erfüllt sind. Da dies jedoch nicht der Fall ist, d. h. es gibt keine kleineren Teilmengen von $\mathcal{B}$, die ebenfalls Gründe sind, als die oben Benannten, müssen die Gründe minimal sein.

[2] Dies ist ähnlich zu dem Ansatz, wie er in [WSD12] vorgestellt wurde. Hier ist die Anzahl der zu deaktivierenden Beschränkungen jedoch eingeschränkt. Damit wird weder Vollständigkeit noch Minimalität sichergestellt. Die gefundenen Gründe für eine Inkonsistenz sind eine Annäherung.

ISBN 978-3-00-048889-4

Tabelle 1: Alle möglichen Belegungen für Gleichung (2)

r_1	r_2	r_3	r_4	r_5	r_6	i_6	r_8	i_1	i_2	i_3	i_5	i_4	i_7	r_7
-	1	-	-	-	-	-	-	-	1	-	1	-	1	-
-	1	-	-	-	-	-	-	-	0	1	1	-	1	-
-	1	-	-	-	-	-	-	-	-	1	1	1	0	-
-	1	-	-	-	-	-	-	1	-	0	1	1	0	-
-	1	-	-	-	-	-	-	1	0	0	1	-	1	-
1	0	-	-	-	-	-	-	-	1	-	1	-	1	-
-	1	-	-	-	-	-	-	-	1	-	1	0	0	1
-	1	-	-	-	-	-	-	0	1	0	1	1	0	-
-	1	-	-	-	-	-	-	-	0	1	1	0	0	1
-	1	-	-	-	-	-	-	1	0	0	1	0	0	1
1	0	-	-	-	-	-	-	-	1	-	1	1	0	-
1	0	-	-	-	-	-	-	-	1	-	1	0	0	1
1	0	-	-	-	-	-	-	-	0	1	1	-	1	-
1	0	-	-	-	-	-	-	-	0	1	1	-	0	1
1	0	-	-	-	-	-	-	-	0	1	1	1	0	0
1	0	-	-	-	-	-	-	1	0	0	1	-	1	-
1	0	-	-	-	-	-	-	1	0	0	1	-	0	1
1	0	-	-	-	-	-	-	1	0	0	1	1	0	0
mindestens eine 1		–						mindestens eine 1			**1**	mindestens eine 1		

5. Schlussbemerkungen

In dieser erweiterten Zusammenfassung haben wir einen Lösungsansatz vorgestellt, welcher den Modellierer bei der Fehlersuche in inkonsistenten Modellen unterstützt. Mit Hilfe von Verifikationsaufgaben, welche durch SAT- bzw. SMT-Beweiser automatisch und effizient lösbar sind, lassen sich aus allen Belegungen minimale Gründe für Widersprüche in einem Modell, welche die Inkonsistenz erklären, finden.
In weiteren Arbeiten soll der vorgeschlagene Lösungsansatz implementiert und ausführlich evaluiert werden. Weitere Verbesserungsmöglichkeiten bestehen unter anderem darin, dass gar nicht alle Belegungen des SAT-Beweisers berücksichtigt werden müssen, sondern nur eine sehr geringe Teilmenge. Tatsächlich können aus einer erfüllenden Belegungen fast immer weitere erfüllende Belegungen abgeleitet werden. Denn für jede gefundene erfüllende Belegung können die aktivierten bzw. erfüllten Beschränkungen zusätzlich noch deaktiviert werden. Ersetzt man z. B. alle Don't-Cares in Tabelle 1 durch Nullen, so könnte man aus diesen 18 erfüllenden Belegungen alle 9408 möglichen erfüllenden Belegungen ableiten.
Es bleibt jedoch offen, wie man eine solch kleine Anzahl an erfüllenden Belegungen gezielt finden kann, ohne dass die Gültigkeit der vorgestellten Schlussfolgerungen beeinträchtigt wird. In Folgearbeiten sollen genau diese Reduzierungen betrachtet werden.

ISBN 978-3-00-048889-4

Danksagung

Diese Arbeit wurde vom Bundesministerium für Bildung und Forschung (BMBF) im Rahmen des Projektes SPECifIC (Fördernummer 01IW13001), der Deutschen Forschungsgemeinschaft (DFG) im Rahmen eines Reinhart Koselleck Projektes (Fördernummer DR 287/23-1) und eines Forschungsprojektes (Fördernummer WI 3401/5-1) sowie der Siemens AG unterstützt.

Literatur

[GBR07] Gogolla, Martin, Fabian Büttner und Mark Richters: *USE: A UML-based specification environment for validating UML and OCL.* Science of Computer Programming, 69(1-3):27–34, 2007.

[MM05] Martin, Grant und Wolfgang Müller: *UML for SOC Design*. 2005.

[RJB99] Rumbaugh, James, Ivar Jacobson und Grady Booch (Herausgeber): *The Unified Modeling Language reference manual*. Addison-Wesley Longman Ltd., Essex, UK, 1999, ISBN 0-201-30998-X.

[SWD11] Soeken, Mathias, Robert Wille und Rolf Drechsler: *Encoding OCL Data Types for SAT-Based Verification of UML/OCL Models*. In: *Tests and Proof*, 2011.

[SWK+10] Soeken, Mathias, Robert Wille, Mirco Kuhlmann, Martin Gogolla und Rolf Drechsler: *Verifying UML/OCL models using Boolean satisfiability*. In: *Design, Automation and Test in Europe*, 2010.

[Wei07] Weilkiens, Tim: *Systems Engineering with SysML/UML:Modeling, Analysis, Design*. Morgan Kaufmann, 2007.

[WK99] Warmer, Jos und Anneke Kleppe: *The Object Constraint Language: Precise modeling with UML*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999, ISBN 0-201-37940-6.

[WSD12] Wille, Robert, Mathias Soeken und Rolf Drechsler: *Debugging of inconsistent UML/OCL models*. In: *Design, Automation and Test in Europe*, Seiten 1078–1083, 2012.

Deriving AOC C-Models from D&V Languages for Single- or Multi-Threaded Execution Using C or C++

Tobias STRAUCH
R&D EDAptix
Munich, Germany
tobias@edaptix.com

Abstract

The C language is getting more and more popular as a design and verification language (DVL). SystemC, ParC [1] and Cx [2] are based on C. C-models of the design and verification environment can also be generated from new DVLs (e.g. Chisel [3]) or classical DVLs such as VHDL or Verilog. The execution of these models is usually license free and presumably faster than their alternative counterparts (simulators). This paper proposes activity-dependent, ordered, cycle-accurate (AOC) C-models to speed up simulation time. It compares the results with alternative concepts. The paper also examines the execution of the AOC C-model on a multithreaded processor environment.

1. Introduction

C based design and verification languages (DVL) have made an significant impact on the overall design process throughout the last decades. What has been dominated by classical languages like VHDL and Verilog (HDL) is now challenged by a fundamentally different approach. The C language is used to model the design and verification environment. For that the design or the testbench are either written in a syntax that is an extension to C, or the model is automatically translated to C from other languages like VHDL and Verilog.

System level design in C++ is proposed by Verkest et al. in [4]. A language that can be seen as an extension to C is for example SystemC [5]. The code can be directly compiled into an executable for simulation and it can be used for synthesis. Speeding up SystemC simulation is shown by Naguib et al. in [6]. A C-model is also used as an intermediate format in the design and verification flow. Design and testbenches written in languages like Cx [2], Chisel [3], VHDL or Verilog are translated into C, which can then be compiled with standard C compilers. An examples for tools converting Verilog to C is the verilator [7], for converting Verilog into an intermediate format iverilog [8], and for converting VHDL to machine code GHDL [9]. It is also proposed to co-simulate design elements in C and other languages. Bombana et al. demonstrate VHDL and C level cosimulation in [10] and Patel et al. evaluate on cosimulation of Bluespec and C based design elements in [11].

C-models can be cycle or timing accurate representations of the design and test behavior. This is true for most DVLs. In this paper it is assumed, that synthesis does not consider timing relevant aspects (like “delays” for instance) and that the design under test (DUT), which is used for synthesis, is modeled cycle accurately. A cycle (and not timing) accurate description of the DUT can be seen as good design practice, regardless which language is used. A classical example cycle accurate simulation is Hornet, a cycle level multicore simulator proposed by Ren et al. in [12]. Cycle based simulation using decision diagrams (DD) is discussed by Ubar et al. in [13] and based on reduced colored Petri net (RCPN) by Reshadi et al. in [14].

ISBN 978-3-00-048889-4

In this paper an activity-dependent, ordered and cycle-accurate (AOC) C-model of the DUT is proposed. Synthesis techniques are used to convert the RTL design into an elaborated representation. A clock tree analysis enables a cycle accurate simulation of the DUT. The proposed method allows an activity-dependent calculation of different design elements within individual clock domains. The model can also be executed on a multiprocessor system or on a multithreaded processor.

Section 2 describes the translation process of a DUT into a cycle-accurate C-model representation. In section 3 the algorithm is enhanced to support AOC C-models. How the model can be improved to support a multithreaded processor is shown in section 4. Section 5 describes how the AOC C-model can be combined with other verification relevant aspects. The proposed model is then compared to alternative concepts (section 6).

2. C-Model Generation

This section describes the C-model generation process. An algorithm is outlined in Figure 1, which supports the process of translating a design from any common language like Verilog or VHDL into a C-model.

1) Parsing source code
2) Hierarchy generation and parameter passing
3) Function and procedure enrollment
4) Variable unification and ordering
5) Signal and register detection
6) Clock tree detection and dependencies
7) Register and signal dependencies
8) Design graph optimizations
9) C code dumping

Figure 1: Algorithm for RTL to C-model conversion.

After parsing the source code, the design hierarchy is elaborated. During this step, parameter must be passed and generate statements must be considered. Step 3 covers the enrollment of functions, tasks and procedures. For both coding languages (VHDL and Verilog) a variable unification and ordering (step 4) within a single process must be done. After this initial phase, signals and registers need to be identified (step 5). The register detection leads to the step of clock line elaboration for each register. This information is then collected to group registers to individual clock domains and the dependencies of the clock domains itself (e.g. internal generated clocks, step 6). The aspect of using a sensitivity list becomes obsolete. Instead a register and signal ordering based on their dependencies takes place (step 7) and the resulting desing graph is further optimizes (step 8). Finally the design is dumped as C code (step 9).

The conversion algorithm (Figure 1) is common to most HDL-to-C translation tools. After parsing and elaborating the design, the database models the design in a design language independent format. In some alternative design flows, the design is already available in a C-model like fashion and the conversion and mapping steps are less complex. From step 6 onwards, the different language specific aspects of the source code become irrelevant. The mapping of each RTL statement for the Verilog and VHDL languages into C statements is listed in Table 1.

ISBN 978-3-00-048889-4

Table 1. VHDL/Verilog syntax mapping

RTL	VHDL	Verilog	C
if	if the else	… ? … : … / if else	if () {} else {}
case	case (sel) when	case (sel)	if () {} else {}
math	a + b, -, *, …	a + b, -, *, …	+, -, *, ...
comb	not, and, or, …	~, &, |, …	!, &, |, ...
unary		&a, |a, ^a	!, &, |, ...
mux	a(i)	a[i]	a[i]
demux	a(i) <=	a[i] <=	a[i]
shift	shl, shr	>>, <<	>>, <<

It is important for the execution speed, how the design is represented when simulated. Therefore the steps 8 “Design graph optimization” and 9 “C code dumping” have a huge impact on the simulation performance of the C-model. The next section outlines various aspects of the design graph optimization and modeling aspects.

3. AOC C-Model Generation

The activity dependent, ordered, cycle accurate (AOC) C-model generation is discussed in this section. To a certain extend, almost all alternative models are AOC models. Some values (registers) are only calculated at a certain (clock) event (activity dependent), values must be calculated based on an ordered list (otherwise it will get very complicated if not impossible) and almost all models are cycle accurate. Nevertheless, different design representation aspects and different design graph optimization methods can lead to different execution speeds. Numbers will be shown in the result section. Figure 2 lists the various aspects which are discussed in this section.

1) N level signal modeling
2) Multidimensional types and type size
3) Direct computations vs. function calls
4) Design flattening and optimizations
5) Clock and output domain modeling
6) Register ordering
7) Wire ordering
8) Activity dependent signal ordering

Figure. 2: Design Graph Optimization Methods and Design Modeling Aspects

3.1 Definitions

Given is a set of inputs *I*, outputs *O*, sequential elements *R* and a directed graph *G* of combinatorial elements *C* and wires *W*. The simplest form of a combinatorial element $c \in C$ is an assignment (buffer). An *c* input (*ci*) can be an input $i \in I$, register $r \in R$ or wire $w \in W$. A *c* output (*co*) can be an output $o \in O$, register $r \in R$ or wire $w \in W$. All *w* have one driving combinatorial element *c*. All *c* and *w* build a directed graph without functional loops. A signal $s \in S$ can be an *i*, *o*, *r*, or *w* ($\{I, O, R, W\} \in S$). A register *r* can be an event or level sensitive sequential element.

A $cr \in CR$ is a clock root and *CR* a list of all *cr* of the design. A $cd \in CD$ is a set of registers with identical *cr*. *CD* is a list of all *cd* in the design. The register cone input list *rcil(r)* is register specific and is a complete list of register and input signals which drive the directed tree *tr* of combinatorial elements (*c*) with (*r*) at its root. A primary input *pi* can be an input *i* or the output of a sequential element *r*. A primary output *po* can be a register input r or an output *o*. All *po* with the same clock root *cr* are grouped to a primary output domain *pod(cr)*. The *POD* lists all *pod(cr)* of the design.

ISBN 978-3-00-048889-4

3.2 N-level Signal Modeling

For the optimization techniques discussed now, it is assumed that C variables of the standard type "unsigned" generate faster execution models than their comparable representation as a specific class. In the proposed AOC model generation, a 2-value representation of *s* is therefore default, unless specified otherwise. Assuming the signal s0 is an 8 bit wide bus and should only simulate {0, 1}, then s0 can be of type "unsigned". If s0 should simulate more than 2 values {0, 1, X, Z, ...}, then s0 must be represented by a specific signal-class.

3.3 Multidimensional Types and Type Size

The different signal types are elaborated and serialized. Let cw be the bit width of the type "unsigned" of the target architecture for the model execution. A classical cw of a processor architectures is 32 or 64. If (serialized) types have more than cw bits, then the C representation is a two dimensional array of the serialized type. An exception to this rule is a 2-dimensional array type with less than cw bits per dimension. In this case the type is also modeled as a "2-dimensional unsigned array" but not serialized. If the signal must be modeled as a signal-class, then the class can use a serialized or a dynamic representation.

3.4 Direct Computations vs. Function Calls

Each combinatorial element *c* should be modeled as a direct computation "a = b & c;" and not as a function call "a = AND(b, c);". If at lease one signal is a signal-class, then a function call "AND(a, b, c);" is required. Functions must be provided to convert signals of type unsigned to or from a signal-class.

3.5 Design Flattening and Optimizations

The design hierarchy is removed by flattening the design. Signal names are modified to guarantee the uniqueness of the signal. For a better readability (debugging), the hierarchical instantiation names are typically added as a prefix to the signal name (e.g. topi_subsystem1i_cpui_executei_pc) but any other method/prefix to uniquify the signals is applicable.

The design builds a directed graph *G* of combinatorial elements *C* and signals *S*. Constant values (e.g. a signal is driven by a constant value "s <= 1'b0;") are propagated through G and all $c \in C$ and $s \in S$ that become irrelevant are removed. Also direct assignment pairs "s2 <= s1; s1 <= s0;" are simplified "s2 <= s0;" and the irrelevant entries in the design database (*c*, *s*) are removed. Most of these direct assignment pairs result from design flattening.

3.6 Clock and Output Domain Modeling

After the register identification step, the clock (or enable) input of each register (*r*) is traced back to its clock root (*cr*). Clock roots (*cr*) can be inputs (*i*) to the design, outputs of combinatorial logic (*w*) or registers (*r*). The clock roots which are design inputs become independent driver of their individual clock domain (*cd*). Clock roots which are latch or register outputs or outputs of logic cones are drivers of clock domains, which dependent on other clock domains (*cd*).

All outputs $o \in O$ are automatically grouped to the output domain *OD* (= *O*) $\in$ POD. They are independent of any clock (or enable) event and their value must be calculated whenever one of the *pi* of their directed tree *tr*(*o*) has changed its value.

3.7 Register Ordering

The register list of each clock domain (*cd*) must be ordered, based on their interdependency. For that the register cone input list (*rcil(r)*) is generated for each register. Figure 3 shows an example.

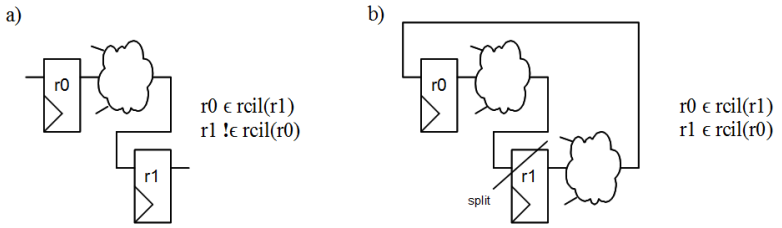


Figure 3: Simple Ordering and Ordering Using Split

The ordering of all *r* of one *cd* is trivial when their *rcil(r)* is used. The logic cones of Figure 3a can easily be ordered and calculated. The registers of Figure 3b depend on each other and no clear order can be found. At least one register must therefore be splitted into its output value and a pre-register value that is calculated first. All relevant *r* then take over their pre-register value as an output value at the end of the modeling task of a clock domain.

3.8 Wire Ordering

The directed tree *tr(po)* for each primary output *po* of a primary output domain *pod(cr)* must be modeled. A *po* can either be a *r* or an *o*. Therefore the modeling of all wires $w \in tr(po)$ must be ordered based on their interdependency. A *w* gets the attribute w^{cal} once it has been added to the list of calculated *w WL*. Then it can be defined, that the value of a *w* can be calculated when the inputs of the associated combinatorial logic element *c* are *r*, *pi*, or w^{cal}. This constraint allows an ordering of all $w \in tr(po)$. The *po* value can be calculated when all $w \in tr(po)$ are w^{cal}. Each w^{cal} holds its attribute until all *po* of a *POD* are calculated. Therefore each relevant *w* of a *pod(cr)* is only calculated once.

3.9 Activity Dependent Signal Ordering

An special modeling technique is the activity dependent signal ordering (ADSO). A combinatorial element *c* changes its output value *co* only when at least one of their inputs *ci* has changed. Classical HDLs like VHDL and Verilog support this fact by using a sensitivity list. Figure 4 shows how ADSO is implemented in an AOC model.

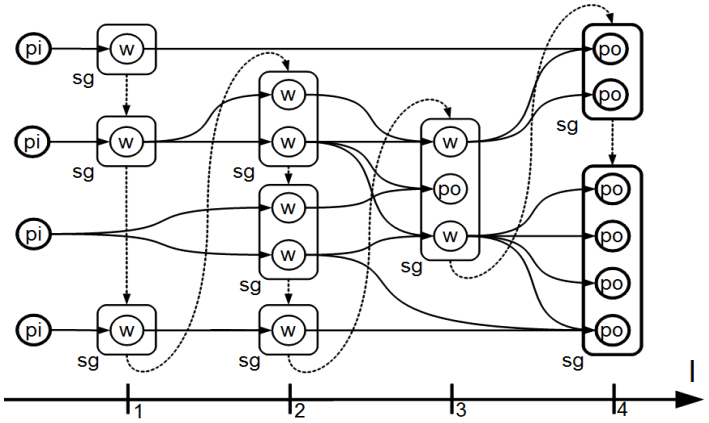


Figure 4: Implementing Activity Dependent Signal Ordering

The ADSO evaluation algorithm is based on two steps. In the first step, all outputs of combinatorial elements *co* (= {*w*, *r*, *po*}) are placed on different levels *l* based on the following rules. **Rule 1** says, that each *co* must be placed on the lowest possible level. **Rule 2** says, that a *co* which depends on a list of *ci* (= {*w*, *pi*}) *lci* must be placed on a higher level than all *w* of *lci*. In the second step, all *co* on a level *l* are grouped to signal groups *sg(l)* based on the following rule. **Rule 3** says, that all *co*

on one *l* which depend on at least one identical wire *w* are grouped to a *sg(l)*. The ordered signal list *OSL* (dotted line in Figure 4) holds all *sg(l)* ordered by their individual level *l* assignment.

The ADSO execution algorithm adds the active attribute sg^{act} to each *sg*. If a *ci* changes its value during execution, then the following is applied. **Rule 4** says, that the sg^{act} is set for the *sg* for which one of the *co* has the *ci*. All *sg* are evaluated based on the *OSL*. All *co* in an *sg* are only calculated, if the sg^{act} attribute is set. After the execution of an *sg*, the sg^{act} attribute is cleared again.

4. Multithreaded Execution of AOC Models

When an AOC model should be executed on a multithreaded processor (or multiprocessor) environment then the AODS algorithm must be enhanced by the following steps. In step 1, the combinatorial elements *c*, wires *w* and output *o* of the output domain *OD* are added to each individual clock domain *cd*, which are connected to the *po* of this *cd*. The resulting clock domain is then called *cdo*. In step 2, the elements of a *cdo* are partitioned to be executed on individual threads *td*. The number of maximal treads *tdmax* must be defined. Each individual *po* of the *cdo* is uniquified and an individual ordered signal list *OSL* is generated. Therefore, *cdo* elements are duplicated if they are elements of individual *OSL*. The *OSL* must be then merged as long as the number of *OSL* is greater than *tdmax* based on the following rule. **Rule 5** says, that these two *OSL* out of all *OSL* are merged, which share the highest number of *cdo* elements. Figure 5 shows an example of 3 *cdo* and 2 *td*/*OSL*.

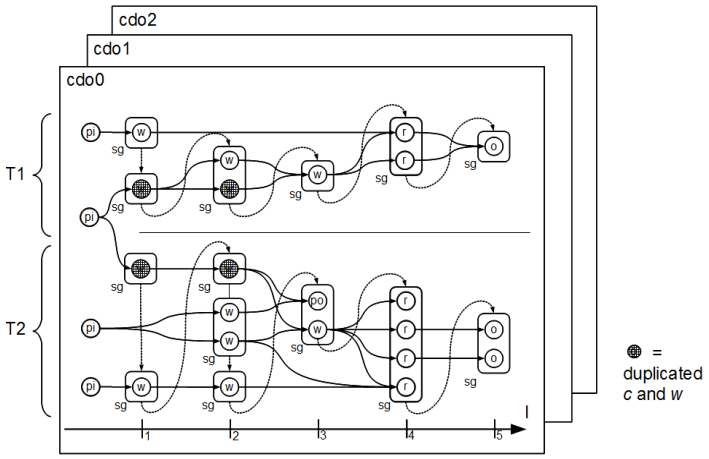


Figure 5: Enhanced Model for Multithreaded Execution

The memory model is critical for the program performance, especially when executed on a multithreaded or multicore system. The "OpenCL Memory Model" [15] is shown in Figure 6. It is used to demonstrate the memory usage of AOC models. Wires *w* do have a very short lifetime and do not need to be shared among multiple threads. They are therefore stored in the "Private Memory" (Figure 6) of a work-item (= thread). Registers *r* are stored throughout the program runtime and most of them must be shared among multiple work-items. They are located in the "Local Memory" (Figure 6) of a workgroup and become primary inputs *pi* in the next cycle. Outputs o are calculated for each cycle and stored in the "Constant Memory"of the "Compute Device" (Figure 6). Inputs *i* are also stored in the "Constant Memory" to be used by the work-items as *pi*.

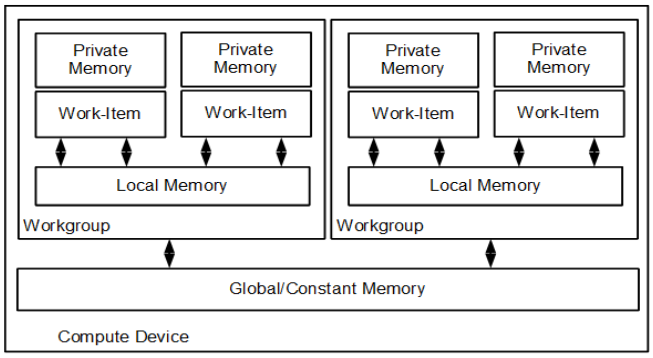


Figure 6: OpenCL Memory Model of the Compute Device

In the proposed *OSL* based modeling technique, wires *w* only have a very short lifetime. Two steps can be made to improve the execution speed. In step 1, the following observations and definitions are made. The order of *w* calculation within one level can be freely selected. It is defined that a wire pair *wp* has a first wire *fw* and a second wire *sw*, whereas the *sw* depends on the *fw*. The *wp* with the *fw* and the *sw* on consecutive levels are added to a level specific wire pair list *WPL(l)*. The *wp* of which the *fw* is only used by the *sw* are listed last in the *WPL(l)*. The following rule is defined to keep the values of *w* in the register file of a processor during execution. **Rule 6** says, that for each *wp* of the ordered *WPL(l)* the *fw* is added at the end and the *sw* is added at the beginning of the *w* calculation of each level. This increases the chances that the compiler avoids memory accesses and keeps the temporary values of *w* in the register file.

In step 2, it is tried to avoid further time consuming memory access by reusing cache entries for multiple wire calculations. To achieve this, a list of placeholders *PHL* and its maximal size (cache size) *phmax* is defined. **Rule 7** says, that a *w* is assigned to a *PHL* entry, if its value is used in the remaining execution of the SGL. If the *w* is not needed anymore, than its *PHL* entry can be used by another *w*. This rule and *phmax* must already be considered when two *OSL* are merged to a single one (rule 5). Rule 7 can help the compiler to store values in a local cache. The parameters *tdmax* and *phmax* are system specific.

Seven rules have been defined throughout the last two sections. They define the transformation process of the device under test (DUT) - which can be seen as a directed graph G defined in section 3.1 - into an activity-dependent, ordered and cycle-accurate (AOC) C-model.

5. Testbenches

This paper discussed so far how synthesizeable HDL code can be transformed into AOC C-models. An extended flow can be used to transfer testbenches (non-synthesize-able code) into timing accurate representations in C++ format. Both models can then be linked to execute the same simulation process as known from HDL simulators.

The additional aspects of this flow are the DUT definition and the sequential process identification. Sequential processes are HDL statements, which are time consuming due to timing related statements (example: “wait for 10 ns;”) or conditional statements (example: “wait on <signal>;”). These processes cannot be converted into cycle accurate models, they need to be modeled timing accurate. The resulting model has an condition checker and an event scheduler. Both reflect the entries of the sequential processes. The flow for this methodology is outlined in Figure 7.

ISBN 978-3-00-048889-4

1) parsing source code
2) parameter passing and hierarchy
3) function and procedure enrollment
4) variable uniquification
5) DUT definition
6) sequential process identification
7) signal and register detection
8) clock tree detection and dependencies
9) register and signal logic cone conversion
10) register and signal dependencies
11) C++ code optimization and dumping
12) condition bag dumping
13) event scheduler dumping

Figure 7: Algorithm for HDL Testbench to Timing Accurate Modeling

A process in VHDL or Verilog is defined as a time-consuming process (TCP) when it is not within the DUT hierarchy and when the keyword "wait" is used within this process. A TCP is partitioned into individual list of assignments, based on the different wait statement. A list of assignments is continuously executed until a "wait" statement is reached. This event is added to the event list in case of a "wait for" statement, and to a the condition list in case of a "wait on" statement. The execution of the assignment list of the TCP is continued, once the simulation time reaches the event in the event list or when the condition in the condition list is true. The conditions in the condition list are checked very simulation step.

6. PSL, Waveform and MatLab

6.1 PSL

The property specification language (PSL) can be used for checking design behavior during verification. The language supports different flavors (VHDL, Verilog) and can be part of the design RTL source code itself. The language is more and more used already on C/SystemC level or higher level of abstraction as proposed by Habibi et al. in [16]. It is therefore important to incorporate the structure into AOC C-models. Obereder et al. describe in [17], how PSL can be converted into synthesizable HDL code. This approach can be used and the resulting synthesizable HDL code can be converted into AOC C-models.

6.2 Waveform

Just like common HDL simulators, AOC C-models can dump simulation waveforms as well. Signals can be defined manually, by script or by HDL syntax (example: Verilog). The AOC C-model then dumps a cycle accurate VCD file during execution, which can then be viewed by a standard waveform viewer.

6.3 Running AOC C-Models in Matlab

AOC C-models can also be executed in a Matlab [18] based environment. For that the C code must be compiled into an S-function. It can then be co-simulated together with other Matlab based simulation components. This is very useful for accelerators or custom DSPs designed in HDL. They can then be simulated and verified in a much more flexible simulation environment than classical HDLs can offer.

ISBN 978-3-00-048889-4

7. Results

This section compares the execution speed of AOC C-models to alternative concepts. Different testcases are used to measure the individual runtimes. The Verilog version of the OpenRISC SoC was taken from [19] and a testcase was added. The verilator [7] and iverilog [8] runtime was then compared to the runtime of the AOC C-model, which was automatically generated from the Verilog source code. A single stage RISCV32IM processor [20] with a lengthy testcase was developed in VHDL and SystemC. A Chisel version was taken from [21]. Their runtime was then compared to the runtime of the AOC C-model, which was automatically generated from the VHDL source code. A license for a standard simulator was not available. This is why a comparison number of verilator vs. VCS was taken from [7] and added to the list as a VCS vs. AOC C-model relative runtime entry, considering the fact, that the AOC C-mode is about 5.09 times faster than the verilator execution. The numbers are based on tests executed on a single processor system. Table 2 and Figure 8 show the results. The AOC C-models are always the 100% reference runtime.

Table 2. Relative runtime compared to AOC C-models.

	verilator	iverilog	GHDL	SystemC	Chisel	VCS
Relative Runtime	5.09	18.39	11.44	7.00	3.91	20.6
AOC C-model	1	1	1	1	1	1

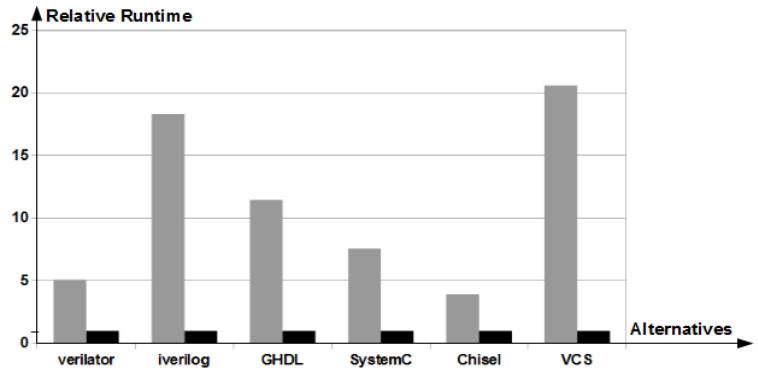


Figure 8: Relative runtime comparison of the AOC C-model and alternatives.

The numbers show that the AOC C-models are always faster than any other known C-model (iverilog, ….) based alternative. The compile time of the individual C-models are almost the same and their comparison can be neglected.

8. Conclusion

This paper introduced activity dependent cycle accurate (AOC) C-models. Multiple improvement steps can be applied to successively decrease the execution time. The main improvements are the activity dependent solving of combinatorial logic equations and their execution based on an extracted signal order. Tests show that they have a faster execution speed than pure cycle accurate C-models or HDL simulators.

This paper also shows that the proposed AOC C-model fits nicely into a multiprocessor system .The current research concentrates on generating AOC OpenCL-models for multiprocessor or multithreading processor systems.

ISBN 978-3-00-048889-4

9. References

[1] ParC, Available Online: http://parallel.cc/cgi-bin/bfx.cgi/index1.html

[2] Cx, Available Online: http://cx-lang.org/

[3] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avizienis, J. Wawrzynek, and K. Asanonic, "Chisel: Constructing Hardware in a Scala Embedded Language", DAC 2012, Design Automation Conference, San Francisco, USA, 3-7 June 2012, pp. 1212-1221

[4] D. Verkest, J. Kunkel, and F. Schirrmeister, "System level design using C++", DATE 2000, Proceedings of the Conference on Design, Automation and Test in Europe, Paris, France, 27-30 March 2000, Pages 74-83

[5] SystemC, Available Online: http://www.accellera.org/community/systemc/

[6] Youssef N. Naguib and Rafik S. Guindi, "Speeding Up SystemC Simulation through Process Splitting", DATE 2007, Proceedings of the Conference on Design, Automation and Test in Europe, Nice, France, 16-20 April 2007, Pages 1-6

[7] verilator, Available Online: http://www.veripool.org/wiki/verilator

[8] iverilog, Available Online: http://iverilog.icarus.com/home

[9] ghdl, Available Online: http://ghdl.free.fr/

[10] M. Bombana, and F. Bruschi, "SystemC-VHDL co-simulation and synthesis in the HW domain", DATE 2003, Proceedings of the Conference on Design, Automation and Test in Europe, Munich, Germany, 3-7 March 2003, Pages 100-105

[11] H. Patel, and S. Shukla, "On Cosimulating Multiple Abstraction-Level System-Level Models", IEEE Transactions on CAD, Vol. 27, No. 2, February 2008, pp. 394-398

[12] P. Ren, M. Lis, M. Cho, K. Shim, and C. Fletcher, "HORNET: A Cycle-Level Multicore Simulator", IEEE Transactions on CAD, Vol. 31, No. 6, June 2012, pp. 890-903

[13] R. Ubar, A. Morawiec, and J Raik, "Cycle-based Simulation with Decision Diagrams", DATE 1999, Proceedings of the Conference on Design, Automation and Test in Europe, Munich, Germany, 9-12 March 1999, Pages 454-458

[14] M. Reshadi, B. Gorjiara, and N. Dutt, "Generic Processor Modeling for Automatically Generating Very Fast Cycle Accurate Simulators", IEEE Transactions on VLSI, Vol. 25, No. 12, December 2006, pp. 2904-2918

[15] OpenCL, "Open Computing Language", Available Online: https://www.khronos.org/opencl/

[16] A. Habibi, and S. Tahar, "Design and Verification of SystemC Transaction-Level Models", IEEE Transactions on VLSI, Vol. 14, No. 1, January 2006, pp. 57-68

[17] H. Obereder, and M. Pfaff, "Behavioral synthesis of property specification language (PSL) assertions", Rapid System Prototyping 2007, International Workshop on Rapid System Prototyping, Porto Alegre, Portugal, 28-30 May 2007, pp. 157-160

[18] MATLAB, Available Online: www.mathworks.de/products/matlab

[19] OpenRISC SoC, Available Online: http://opencores.org/or1k/Main_Page

[20] The RISC-V Instruction Set Architecture, Available Online: http://riscv.org/

[21] The Sodor Processor Collection, Available Online, http://riscv.org/download.html#tab_sodor