Formale Verifikation von Multiplizierern mit Computeralgebra [1]

Daniela Kaufmann[2]

Abstract: Arithmetische Schaltungen werden in Prozessoren zur Implementierung von Boolescher Algebra genutzt. Aufgrund des weitreichenden Einsatzes von Prozessoren ist es äußerst wichtig, die Korrektheit dieser Schaltungen garantieren zu können, um Fehler wie den berühmten Pentium FDIV-Bug zu vermeiden. Mithilfe formaler Verifikation kann festgestellt werden, ob eine Schaltung ihrer gewünschten Spezifikation entspricht. Allerdings stellen arithmetische Schaltungen, insbesondere Integer-Multiplizierer auf Gatterebene, eine Herausforderung für bestehende Verifikationstechniken dar. In dieser Dissertation [Ka20] werden aktuelle Verifikationsmethoden basierend auf Computeralgebra verbessert. Wir zeigen eine rigorose präzise mathematische Formulierung, welche auch die Anwendung der Mathematik in diesem Gebiet erweitert. Außerdem haben wir neue Methoden zur vollautomatischen Verifikation von Integer-Multiplizierern entworfen und implementiert, sowie ein kompaktes Beweisformat entwickelt, um das Ergebnis der Verifikation zertifizieren zu können.

1 Einleitung

Digitale Schaltungen sind ein wesentlicher Bestandteil von Computern und digitalen Systemen. Damit werden mittels binären digitalen Signalen logische Operationen ausgeführt und sie können zahlreiche digitale Komponenten und arithmetische Operationen modellieren. Die Grundfunktion einer digitalen Schaltung ist es aus gegebenen binären Eingangssignalen ein binäres Signal an den Ausgängen für die implementierte logische Funktion zu generieren. Die Hauptkomponenten digitaler Schaltungen sind Logikgatter, welche einfache Boolesche Funktionen repräsentieren. Beispiele für diese Gatter sind Negation (NOT), Konjunktion (UND) und Disjunktion (ODER). Diese logischen Gatter können kombiniert werden, um komplexere Funktionen darzustellen.

Eine Unterkategorie von digitalen Schaltungen sind Schaltnetze. In diesen hängt das Ausgangssignal nur von den gegebenen Eingängen ab, das heißt es gibt keine Rückkopplung der Ausgänge auf die Eingänge. Schaltnetze werden in Computern eingesetzt um Boolesche Algebra zu implementieren. Zum Beispiel ist die arithmetisch-logische Einheit (ALU) in einem Prozessor, die zur Berechnung von mathematischen Operationen eingesetzt wird, aus Schaltnetzen konstruiert. Arithmetische Schaltungen sind spezielle Schaltnetze, die Rechenoperationen wie zum Beispiel Multiplikation durchführen.

Aufgrund des weitreichenden, mitunter sicherheitskritischen, Einsatzes von digitalen Schaltungen ist es äußerst wichtig ihre Korrektheit sicherzustellen. Mittels formaler Verifikation lässt sich die Korrektheit von Software oder Hardware in Bezug auf eine zuvor definierte

[1] Englischer Titel der Dissertation: "Formal Verification of Multiplier Circuits using Computer Algebra"

[2] Johannes Kepler Universität Linz, daniela.kaufmann@jku.at

Spezifikation beweisen. Dazu wird das gegebene System in ein mathematisches Modell übersetzt und automatisierte Beweistechniken werden eingesetzt, um die gewünschten Korrektheitsbeweise zu erzielen. Die verschiedenen Verifikationstechniken unterscheiden sich durch die zugrundeliegenden mathematischen Modellierungen des Systems.

Formale Verifikation von arithmetischen Schaltungen kann helfen Fehler, wie den berühmten FDIV-Bug in frühen Intel Pentium Prozessoren, zu finden und zu vermeiden. Die Gleitkommaeinheit (FPU) dieser Prozessoren enthält einen Hardwarefehler, der bei Division von bestimmten Zahlen zu falschen Ergebnissen führt. Der Fehler wurde erst 1994, rund eineinhalb Jahre nach Markteinführung des Prozessors bekannt [SB94] und kostete Intel ca. 500 Millionen US-Dollar. Eine gleichartige Panne würde heutzutage ernsthafte finanzielle Schwierigkeiten, auch für große Prozessorhersteller, bedeuten. Daher wollen diese Firmen die Korrektheit ihrer Hardware zu hundert Prozent garantieren können.

Seit dem Bekanntwerden des FDIV-Bugs werden formale Verifikationstechniken entwickelt, um die Korrektheit von arithmetischen Schaltungen zu beweisen. Mehr als 25 Jahre später ist dieses Verifikationsproblem jedoch noch immer nicht vollautomatisiert lösbar. Besonders Integer-Multiplizierer, das sind arithmetische Schaltungen die Multiplikationen von ganzen Zahlen ausführen, stellen aufgrund ihres internen Aufbaus der Logikgatter eine Herausforderung für bestehende Verifikationsmethoden dar.

In der Praxis heißt das, dass industrielle Entwickler von arithmetischen Schaltungen momentan entweder aufwändige manuelle Verifikation mittels Theorembeweisern betreiben, oder sich komplett auf Simulationen verlassen. Immer bessere Optimierungen in der Entwicklung erhöhen einerseits die Effizienz einer Schaltung, steigern andererseits aber deren Komplexität. Dadurch sinkt der Prozentsatz der Werte, die simuliert werden können. Daher gelten Simulationen in der Praxis nicht mehr als vertrauenswürdig. Das Fehlen von vollautomatisierten Verifikationsmethoden für arithmetische Schaltungen ist aktuell immer noch ein großer Makel. An diesem Punkt verbessert diese Dissertation den Stand der Technik.

In dieser Dissertation [Ka20] werden formale Verifikationstechniken, welche auf Computeralgebra basieren, untersucht und verbessert. Das Ziel ist es, eine Verifikationsmethode zu erhalten, die für einen gegebenen Integer-Multiplizierer auf Gatterebene vollautomatisiert über dessen Korrektheit entscheidet, ohne dass der Entwickler manuell in den Verifikationsprozess eingreifen muss. Wir zeigen eine präzise mathematische Formalisierung dieses Problems und beweisen die Korrektheit und Vollständigkeit dieser Methode. Neue, in der Dissertation entwickelte, Algorithmen ermöglichen die Verifikation von komplexen Multiplizierer-Architekturen mit Bitbreiten von bis zu 2 048 Bits. Alle entstandenen implementierten Tools und Benchmarks sind als Open-Source verfügbar. Um das Verifikationsresultat des Verifikations-Tools validieren zu können, haben wir ein kompaktes Beweisformat entwickelt, welches erlaubt während der Verifikation einfache Beweiszertifikate zu generieren, die in einem eigenständigen Beweis-Checker validiert werden können. Während meiner Dissertation habe ich als Erstautor zu zehn Publikationen beigetragen und für meine erste Publikation [RBK17] den Best Paper Award bei FMCAD 2017, der wichtigsten Konferenz in Hardwareverifikation, erhalten. Sechs dieser Publikationen sind vollumfänglich in meiner kumulativen Dissertation enthalten.

2 Hintergrund

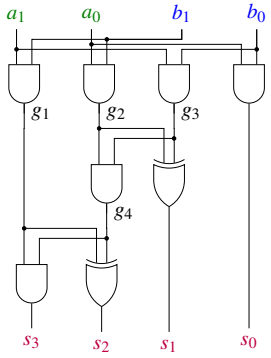
Seit der Entdeckung des FDIV-Bugs werden verschiedenste Verifikationstechniken zur Validierung der Korrektheit von Multiplizierern eingesetzt. Eine gängige Methode ist das Verifikationsproblem als ein Erfüllbarkeitsproblem der Aussagenlogik (SAT) zu kodieren. Bei dieser Methode wird die Schaltung zu einer Formel in konjunktiver Normalform (KNF) übersetzt und ein SAT-Solver eingesetzt, um die Erfüllbarkeit dieser Formel zu überprüfen. Im Jahr 2016 wurde eine größere Menge von solchen Kodierungen für arithmetische Schaltungen als Benchmarks zur jährlichen SAT-Competition eingereicht [Bi16]. Bei diesem Bewerb werden die aktuell besten SAT-Solver anhand einer Menge von Benchmarks ermittelt. Die Ergebnisse dieser Evaluierung zeigen allerdings, dass KNF-Kodierungen von Multiplizierern zu exponentiell großen Beweisen führen, dies deutet auf eine exponentielle Laufzeit von SAT-Solvern hin. Der Grund dafür ist, dass Multiplizierer aus Sequenzen von XOR-Gattern aufgebaut sind und diese mit aktuellen Lösungsmethoden in SAT-Solvern nicht effizient behandelt werden können. In der Theorie konnte die Notwendigkeit der exponentiellen Beweisgröße für simple Multiplizierer-Architekturen bereits widerlegt werden [BL17]. Dieses theoretische Resultat setzt auf strukturelle Kenntnisse über den Multiplizierer und konnte noch nicht praktisch in einem SAT-Solver implementiert werden.

Die erste Technik, mit der es möglich war, den FDIV-Bug zu vermeiden, basiert auf binären Entscheidungsdiagrammen. Genauer gesagt auf Binary Moment Diagrams (BMD) [CB95], da die Knotenanzahl in diesen Diagrammen linear in der Bitbreite bleibt. Jedoch benötigt diese Technik strukturelles Wissen über die Schaltung, da BMDs in einer vorbestimmten Ordnung aufgebaut werden müssen, um die lineare Größe zu garantieren. Daher kann diese Methode nicht vollautomatisiert in komplexen industriellen Designs eingesetzt werden.

Theorembeweiser, wie zum Beispiel ACL2 [Hu17], können in Kombination mit SAT-Solvern industrielle Multiplizierer beweisen. Jedoch ist diese Methode auch nicht vollautomatisiert einsetzbar. Das zu Grunde liegende Beweissystem baut auf einer Menge von problemspezifischen Axiomen und Inferenzregeln auf, und benötigt daher Hintergrundwissen über die Domäne des Problems. Termersetzungssysteme [Va07] benötigen gleichermaßen Hintergrundwissen über den Definitionsbereich und deren Anwendung auf dieses Problem ist auch nicht vollautomatisiert.

Methoden basierend auf Reverse-Engineering [SK04] nutzen arithmetische Darstellungen auf der Bitebene, welche von den gegebenen Multiplizierern auf Gatterebene extrahiert werden. Diese Technik erlaubt es einfache strukturierte Multiplizierer vollautomatisiert zu beweisen, scheitert jedoch an komplexeren industriellen Multiplizierer-Architekturen.

Seit 2016 werden Verifikationstechniken basierend auf Computeralgebra [Yu16, Sa16] als ein vielversprechender Zugang zur automatisierten Verifikation von arithmetischen Schaltungen und insbesondere Multiplizierern gesehen. In dieser Methode werden alle Logikgatter der Schaltung, sowie die Spezifikation als ein Polynom kodiert. Mittels algebraischer Reduktion basierend auf multivariater Polynomdivision kann nun das Problem, ob eine gegebene Schaltung einen korrekten Multiplizierer implementiert, gelöst werden, indem die Normalform des Spezifikationspolynoms bezüglich der Gatterpolynome berechnet wird. Der Multiplizierer ist fehlerfrei genau dann, wenn diese Normalform null ist.



Gatterpolynome	$s_3 = g_1 \wedge g_4$	$-s_3 + g_4 g_1,$
	$s_2 = g_1 \oplus g_4$	$-s_2 - 2g_4 g_1 + g_4 + g_1,$
	$g_4 = g_2 \wedge g_3$	$-g_4 + g_2 g_3,$
	$s_1 = g_2 \oplus g_3$	$-s_1 - 2g_2 g_3 + g_2 + g_3,$
	$g_1 = a_1 \wedge b_1$	$-g_1 + a_1 b_1,$
	$g_2 = a_0 \wedge b_1$	$-g_2 + a_0 b_1,$
	$g_3 = a_1 \wedge b_0$	$-g_3 + a_1 b_0,$
	$s_0 = a_0 \wedge b_0$	$-s_0 + a_0 b_0,$
Binäre Eingänge	$a_1, a_0 \in \mathbb{B}$	$-a_1^2 + a_1, -a_0^2 + a_0,$
	$b_1, b_0 \in \mathbb{B}$	$-b_1^2 + b_1, -b_0^2 + b_0$
Spezifikation	$8s_3 + 4s_2 + 2s_1 + s_0 - 4a_1 b_1 - 2a_1 b_0 - 2a_0 b_1 - a_0 b_0 = 0$	

Abb. 1: 2-Bit Multiplizierer (links) und dessen algebraische Modellierung (rechts).

3 Beitrag der Dissertation

In dieser Dissertation werden formale Verifikation von Multiplizierern auf Gatterebene mittels Computeralgebra untersucht und verbessert, sodass diese Technik auch für komplexe Multiplizierer eingesetzt werden kann. Wir betrachten n-Bit Multiplizierer mit Eingangssignalen $a_0, \ldots, a_{n-1}, b_0, \ldots, b_{n-1} \in \{0,1\}$ und $2n$ Ausgängen $s_0, \ldots, s_{2n-1} \in \{0,1\}$. Eine Schaltung ist ein korrekter Multiplizierer genau dann, wenn für alle möglichen Eingänge $a_0, \ldots, a_{n-1}, b_0, \ldots, b_{n-1} \in \{0,1\}$ und den von der Schaltung berechneten Ausgangssignalen $s_0, \ldots, s_{2n-1} \in \{0,1\}$ die Spezifikation $\sum_{i=0}^{2n-1} 2^i s_i - \left(\sum_{i=0}^{n-1} 2^i a_i\right)\left(\sum_{i=0}^{n-1} 2^i b_i\right) = 0$ gilt. In Worten gefasst bedeutet dies, dass der Ausgangsbitvektor gleich dem Produkt der beiden Eingangsbitvektoren ist. Man beachte, dass die Signale in der Schaltung binär sind, die Spezifikation aber als ein Polynom über den ganzen Zahlen dargestellt wird.

Die Komplexität eines Multiplizierers hängt von seiner Architektur, das heißt, der Anordnung und dem Aufbau der internen Logikgatter, ab. Im Allgemeinen werden Multiplizierer in drei Komponenten unterteilt. In der ersten Komponente werden partielle Produkte $a_i b_j$, für $0 \leq i, j < n$ aus den Eingangssignalen generiert. Dies kann entweder mittels (einer quadratischen Anzahl von) UND-Gattern geschehen oder auch durch eine komplexere Booth-Kodierung. In der zweiten Komponente werden diese partiellen Produkte mit Addierwerken, bestehend aus Voll- und Halbaddierern, akkumuliert und zu zwei Ebenen reduziert. Diese beiden Ebenen werden in einem *abschließenden Addierwerk* aufsummiert, welches das Ausgangssignal des Multiplizierers berechnet. In der Praxis werden die einzelnen Komponenten optimiert, um Multiplizierer platzsparender aufbauen zu können und die Berechnungsdauer der Ausgangssignale zu verringern.

Zu Beginn dieser Dissertation haben wir eine einfache und präzise mathematische Formalisierung für die Verifikation von arithmetischen Schaltungen definiert und umfassende Beweise für die *Korrektheit und Vollständigkeit* dieser Methode gezeigt. Obwohl die allgemeine Idee, arithmetische Schaltungen mit Hilfe von Polynomreduktion zu verifizieren bereits existierte [Yu16, Sa16], wurden diese entscheidenden Eigenschaften der Methode noch nicht belegt.

Die allgemeine Idee algebraischer Verifikation ist es für jedes Logikgatter ein Polynom zu definieren, welches die möglichen Signalkombinationen des Gatters modelliert. Für die Polynomkodierung muss ein zugrundeliegender Polynomring $R[X]$ festlegt werden. Hierfür haben wir zu Beginn dieser Dissertation die Menge der rationalen Zahlen $\mathbb{Q}$ als den Koeffizientenring R gewählt, da diese einen Körper bilden. Abbildung 1 zeigt die vollständige algebraische Kodierung eines 2-Bit Multiplizierers auf Gatterebene. Zum Beispiel impliziert das UND-Gatter $s_3 = g_1 \wedge g_4$ in Abb. 1 die Gleichung $-s_3 + g_4 g_1 = 0$ für die binären Signale s_3, g_1 und g_4. Zusätzlich wird für jedes Eingangssignal a_i (b_j) eine binäre Eingangsbedingung $-a_i^2 + a_i = 0$ ($-b_j^2 + b_j = 0$) generiert, welche definiert, dass diese Signale nur die Werte 0 und 1 annehmen können.

Es soll nun gezeigt werden, dass die Spezifikation aus den Gatterpolynomen und den binären Eingangsbedingungen folgt. Diese Implikation bedeutet algebraisch, dass die Spezifikation im Ideal, welches durch die Gatterpolynome und den binären Eingangsbedingungen erzeugt wird, enthalten ist. Die Idealzugehörigkeit der Spezifikation lässt sich eindeutig mit Hilfe der Theorie von Gröbnerbasen zeigen. Es gelten folgende wichtige Eigenschaften: Für jedes Ideal kann eine Gröbnerbasis ermittelt werden und danach steht einen Reduktionsalgorithmus zur Berechnung der Normalform zur Verfügung, der es uns erlaubt mittels wiederholter Polynomdivision die Frage der Idealzugehörigkeit eindeutig beantworten zu können.

Da wir zunächst als Koeffizientenring den Körper $\mathbb{Q}$ gewählt haben, können wir die Standardtheorie der Gröbnerbasen nutzen. Die Berechnung einer Gröbnerbasis für ein Ideal ist jedoch EXPSPACE-vollständig, und daher für größere Probleme praktisch nicht anwendbar. Wir können allerdings zeigen, dass die Berechnung einer Gröbnerbasis für unsere Anwendung nicht notwendig ist. Werden die Terme in den Polynomen anhand einer lexikographischen Variablenordnung sortiert, sodass die Ausgangsvariable eines Logikgatters immer größer als die Variablen der Eingänge ist, bilden die geordneten Gatterpolynome und binären Eingangsbedingungen automatisch eine Gröbnerbasis.

Auch nach dem Berechnen der Gröbnerbasis bleibt das Problem der Idealzugehörigkeit co-NP-hart [Ka20]. Experimente zeigen, dass bereits für einfache 8-Bit Multiplizierer die Normalform der Spezifikation nicht mehr in gängigen Computeralgebrasystemen ermittelt werden kann. Der Grund dafür ist, dass die Zwischenresultate der einzelnen Polynomdivisionen exponentiell anwachsen, bevor sie zu null reduzieren. Wir untersuchten mögliche Auslöser für diesen Wachstum und erlangten folgende Erkenntnisse. Zum einen wächst das Spezifikationspolynom quadratisch mit der Bitbreite des Multiplizierers. Zum anderen sind Voll- und Halbaddierer wiederkehrende Bausteine in den Multiplizierern, welche genutzt werden um drei bzw. zwei Bits aufzusummieren. Die Funktion dieser Addierer lässt sich mit einem einzelnen linearen Polynom kodieren. In der von der Schaltung induzierten Gröbnerbasis ist diese Polynom jedoch nicht enthalten. Die Gröbnerbasis enthält nur die internen nichtlinearen Gatterpolynome der Addierer, was zu einem exponentiellen Wachstum der Zwischenresultate im Reduktionsalgorithmus führt.

Um das Wachstum der Zwischenresultate zu umgehen, präsentieren wir einen neuen *inkrementellen Verifikationsalgorithmus* [RBK17]. Dieser Algorithmus beruht auf der Erkenntnis, dass die partiellen Produkte in einem Multiplizierer eindeutig in $2n$ Spalten teilbar

sind und jede Spalte maximal eine lineare Anzahl von partiellen Produkten enthält. Wir können dadurch für jede Spalte eine eigene Spezifikation festlegen. Wir erhalten eine Spaltenzerlegung automatisch aus den Einflussbereichen der Ausgangssignalen (und weit einfacher als eine Zeilenzerlegung, die zum Beispiel in BMDs notwendig ist und nicht notwendigerweise existiert). Die Korrektheit des Multiplizierers wird bewiesen, indem jede Spalte inkrementell auf ihre Korrektheit überprüft wird. Der Vorteil dieser Technik ist, dass das Gesamtproblem in mehrere kleinere Teilprobleme aufgeteilt wird und nicht mehr die gesamte quadratische Spezifikation im Reduktionsalgorithmus benötigt wird.

Des Weiteren betreiben wir *Preprocessing*, um eine reduzierte und kompaktere Darstellung [KBK20b] des Multiplizierers zu erhalten. Unsere Preprocessing-Technik fußt auf der Eliminationstheorie von Gröbnerbasen und erlaubt es, Variablen aus der Gröbnerbasis zu entfernen. In unserer ersten Version [KBK20b] suchen wir gezielt nach Voll- und Halbaddierern im gegebenen Multiplizierer und eliminieren deren interne Variablen. Dadurch enthält die Gröbnerbasis nun die gewünschten linearen Polynome, welche die Funktion dieser Addierer beschreiben. Für diese Optimierung führen wir die nötigen theoretischen Grundlagen ein und zeigen, dass in unserer Anwendung lokale Teile der Gröbnerbasis umgeschrieben werden können und die verbliebenen Polynome immer noch eine Gröbnerbasis bilden. Ohne dieses Ergebnis wäre es an dieser Stelle notwendig eine Gröbnerbasis für eine neue Variablenordnung zu berechnen, was uns wieder zu einem EXPSPACE-vollständigen Problem führen würde.

Eine gängige Darstellung von Schaltungen auf Gatterebene sind UND-Invertierer-Graphen (AIG). Wir haben ein Tool implementiert [RBK17, KBK20b], welches Multiplizierer im AIG-Format einliest, die Polynomkodierung übernimmt und die nötigen Preprocessing-Schritte durchführt. Für die eigentliche Berechnung der Normalform benutzen wir die Computeralgebrasysteme Mathematica und Singular. Unsere Experimente zeigen, dass diese Techniken einfache Multiplizierer, die komplett aus Voll- und Halbaddierern aufgebaut sind, verifizieren können, aber bei komplexeren Multiplizierern fehlschlagen.

Wie bereits beschrieben, wird in komplexen Architekturen beispielsweise eine Booth-Kodierung eingesetzt, um die Anzahl der Logikgatter für die Berechnung der partiellen Produkte zu reduzieren. Die Polynomrepräsentation dieser Kodierung hat im algebraischen Verifikationssystem zur Folge, dass in den Zwischenresultaten Terme mit Koeffizienten entstehen, die größer als 2^{2n} sind, welche durch den Ausgangsbitvektor nicht mehr abgedeckt sind. Wir können aufgrund des gewählten Körpers $\mathbb{Q}$ als Koeffiezientenrings diese Terme jedoch nicht kürzen, sondern müssen abwarten, bis sich diese Terme im Reduktionsalgorithmus durch Polynomdivision reduzieren. Darauf basierend erweitern wir unsere bisherigen theoretischen Ergebnisse und Beweise für Polynomringe über allgemeinere Koeffiezientenringe, sodass auch *modulares Kürzen* möglich ist [KBK19]. Modulares Beweisen ermöglicht es nicht nur, dass wir jetzt auch komplexe Integer-Multiplizierer erfolgreich verifizieren können, sondern auch abgeschnittene Multiplizierer behandeln können, bei denen die höchstwertigen Ausgangsbits vernachlässigt werden. Diese Multiplizierer finden zum Beispiel in der Satisfiability Modulo Theory Library (SMT-Lib) Verwendung und entsprechen der Integer-Multiplikation in gängigen Programmiersprachen wie C und Java.

Zusätzlich verallgemeinern wir unsere bestehenden Preprocessing-Techniken, sodass diese unabhängig von syntaktischen Mustern in der Schaltung sind. Einfache Veränderungen in der Gatteranordnung führen zu Fehlschlägen unseres bisherigen Preprocessings. Unsere neue Preprocessing-Methode eliminiert wiederholt Variablen, die nur in exakt einem anderen Polynom vorkommen. Wir zeigen durch einen einfachen Ansatz, dass dadurch unsere erste Preprocessing-Version subsumiert wird.

Nichtsdestotrotz sind die *abschließenden Addierwerke* von komplexen Multiplizierern, im Speziellen Paralleladdierer mit Übertragvorausberechnung schwer mit Computeralgebra zu verifizieren. Diese Art von Paralleladdierern beinhaltet Sequenzen von ODER-Gattern, welche zu einem exponentiellen Wachstum der Zwischenresultate, sowohl im Preprocessing, als auch im Reduktionsalgorithmus führen. Demgegenüber lassen sich diese Addierwerke aber einfach mit Hilfe eines SAT-Solvers verifizieren. Basierend auf dieser Beobachtung entwickeln wir eine *Verbindung von SAT-Solving und Computeralgebra* [KBK19]. Ist das abschließende Addierwerk ein Paralleladdierer, ersetzen wir dieses durch einen Ripple-Carry Addierer, der vollständig aus Volladdierern aufgebaut ist. Die Korrektheit dieser Substitution, im genaueren die Äquivalenz des ursprünglichen Addierwerkes und des Ripple-Carry Addierers wird unter Einsatz eines SAT-Solvers bewiesen. Der vereinfachte Multiplizierer wird mit unseren entwickelten Computeralgebra-Techniken verifiziert.

Als Teil dieser Dissertation haben wir das vollautomatisierte Tool AMULET implementiert, welches einen Multiplizierer im AIG-Format verifiziert. Im ersten Schritt wird in AMULET die Substitution des abschließenden Addierwerkes vorgenommen. Heuristiken erlauben es dieses Addierwerk zu identifizieren und durch einen Ripple-Carry Addierer zu ersetzen. AMULET erstellt eine KNF-Formel um die Korrektheit der Substitution zu beweisen. Der vereinfachte Multiplizierer kann mittels der implementierten Polynombibliothek in AMULET verifiziert werden. Die polynomialen Algorithmen in AMULET sind optimal an unseren Anwendungsfall angepasst und können so die syntaktischen Besonderheiten der Polynome gezielt nutzen, was ein wesentlicher Vorteil gegenüber Computeralgebrasystemen ist, die für allgemeine Anwendungsbereiche konzipiert sind. AMULET ist als Open-Source verfügbar und ermöglicht effiziente Verifikation nun auch von komplexen Multiplizierern. Evaluierungen dazu sind in Kapitel 4 zu finden.

An dieser Stelle müssen wir uns nun Fragen stellen, wie der Verifizierer verifiziert wird. Es kann sein, dass die Verifikationstools Fehler enthalten, welche zu einem falschen Ergebnis führen. Eine mögliche Methode um die Korrektheit der Tools zu garantieren, ist es, die Tools selbst zu verifizieren. Dies ist aber ein aufwändiger manueller Prozess. In der Praxis ist es daher gebräuchlicher, Beweiszertifikate während der Verifikation zu generieren. Diese Zertifikate können dann von einem eigenständigen Tool auf Richtigkeit überprüft werden. Bei der jährlichen SAT-Competition werden beispielsweise schon seit 2013 Beweiszertifikate gefordert. Für algebraisches Beweisen benötigen wir ein Beweissystem, welches über Polynomgleichungen urteilen kann. In dieser Dissertation haben wir das Beweiskalkül *Practical Algebraic Calculus* (PAC) [RBK18] entworfen, der ermöglicht systemnahe algebraische Beweise zu generieren. PAC basiert auf dem Polynomial Calculus, welcher vor allem in der Proof Complexity Community seine Anwendung findet, um über Komplexitätsergebnisse zu argumentieren. Praktische Anwendungen des Polynomial

Calculus kamen bisher nicht vor, da aufgrund seiner Formalisierung generierte Zertifikate nicht effizient überprüft werden konnten.

Wir haben daher PAC als eine Instantiierung des Polynomial Calculus entwickelt. Ähnlich zu einer Rechenprobe mit Rest für Division von Zahlen, modellieren wir in PAC ein Beweiszertifikat für die wiederholte multivariate Polynomdivision aus einer Reihe von Additions- und Multiplikationsschritten. Wir haben den Beweis-Checker PACHECK implementiert, welcher diese Beweiszertifikate auf ihre Richtigkeit überprüft. In der ursprünglichen Fassung von PAC haben wir explizit gefordert, dass alle Beweisschritte aufgelistet werden, was zu großen Beweisdateien führt. In einer erweiterten Version haben wir ein *kompakteres Beweisformat* [KFB20] definiert, welches ermöglicht Polynome mit Hilfe von Indizes anzusprechen. Des Weiteren nutzen wir aus, dass in unserem Anwendungsfall alle Variablen nur binäre Werte annehmen. PACHECK wurde dahingehend adaptiert. Wir generieren diese Beweiszertifikate als ein Nebenprodukt der Verifikation in AMULET und heben uns damit von verwandten Arbeiten auf diesem Gebiet ab, da diese keine Beweiszertifikate erzeugen.

Die Kombination von SAT-Solving und Computeralgebra [KBK19] hat zur Folge, dass Beweiszertifikate in unterschiedlichen Beweissystemen generiert werden. Für SAT wird vom SAT-Solver ein Beweis im sogenannten DRUP Beweissystem generiert. Für Computeralgebra generieren wir Zertifikate in PAC. Diese Beweise werden von unterschiedlichen Tools validiert, was eine Lücke im Beweiszertifikat hinterlässt. Kompositionelles Beweisen mittels eines Theorembeweisers könnte diese Lücke schließen, benötigt jedoch manuelle Unterstützung. Wir zeigen, dass sich *DRUP Beweise vollständig in PAC simulieren* lassen und wir entwickeln einen zugehörigen Übersetzungs-Ablauf [KBK20a]. Daher können wir ein vollständiges PAC Beweiszertifikat für die Verifikation von Multiplizierern generieren.

4 Resultate

Wir evaluieren die algebraische Verifikation von Multiplizierern und betrachten die Entwicklung dieser Technik innerhalb dieser Dissertation [RBK17, KBK19, KBK20b] und vergleichen die Ergebnisse mit aktuellen Tools von verwandten Arbeiten [Ci20, MGD19].

In ersten Experimenten evaluieren wir unsere Technik auf einem breiten Benchmarkset, welches aus 384 unterschiedlichen 64-Bit Multiplizierer-Architekturen besteht. Dieses Experiment ist in der linken Grafik von Abb. 2 zu sehen und zeigt die Anzahl der korrekt verifizierten Multiplizierern (sortiert nach Verifikationszeit) innerhalb eines Prozessorzeitlimits von 300 Sekunden. AMULET 1.0 ist in dieser Dissertation publiziert, AMULET 1.5 ist eine aktualisierte Version mit verbesserten Heuristiken zur Identifikation von Paralleladdierern mit Übertragvorausberechnung. Im zweiten Experiment, zu sehen auf der rechten Seite von Abb. 2, wählen wir einen simplen Multiplizierer, der sich vollständig in Voll- und Halbaddierer zerteilen lässt, mit unterschiedlichen Eingangsbitbreiten n. Das Zeitlimit für dieses Experiment wird auf 86 400 Sekunden (24 Stunden) gesetzt.

Es zeigt sich, dass AMULET und insbesondere dessen aktualisierte Version eine Größenordnung schneller ist als vergleichbare Tools [MGD19] und im Gegensatz zu [Ci20] auch komplexe Multiplizierer effizient verifizieren und zertifizieren kann.

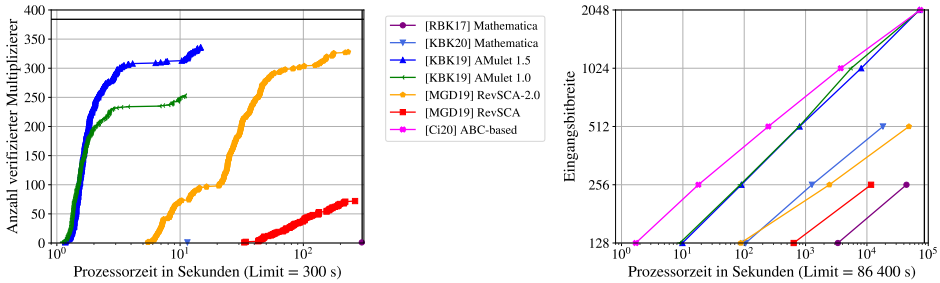


Abb. 2: Verifikation von 384 verschiedenen 64-Bit Multiplizierer-Architekturen (links) und von einfachen Multiplizierern mit großen Bitbreiten (rechts).

5 Schlussfolgerung

Automatisierte formale Verifikation von arithmetischen Schaltungen, insbesondere von Integer-Multiplizierern, wird in der Praxis immer noch als große Herausforderung angesehen. In dieser Dissertation betrachten und verbessern wir Verifikationsmethoden basierend auf Computeralgebra, um effiziente Verifikation von komplexen Multiplizierern zu ermöglichen. Wir zeigen eine umfassende mathematische Formalisierung, welche auf der Theorie der Gröbnerbasen basiert und daher auch die Verwendung von Mathematik in diesem Gebiet erweitert. Eine neue Kombination von logischem und algebraischem Rechnen führt zu innovativen und wirkungsvollen Methoden für die Verifikation von Multiplizierern. Wir präsentieren erstmalig ein algebraisches Beweissystem, welches erlaubt kompakte Zertifikate für die Verifikation zu generieren. Es zeigt sich, dass ein Zusammenspiel der von uns entwickelten Techniken ein effizientes Lösen einer Vielfalt von komplexen Multiplizierer-Architekturen ermöglicht.

Literaturverzeichnis

[Bi16] Biere, Armin: Collection of Combinational Arithmetic Miters Submitted to the SAT Competition 2016. In: SAT Competition 2016. Jgg. B-2016-1 in Dep. of Computer Science - Series of Publications B. University of Helsinki, S. 65–66, 2016.

[BL17] Beame, Paul; Liew, Vincent: Towards Verifying Nonlinear Integer Arithmetic. In: CAV 2017. Jgg. 10427 in LNCS. Springer, S. 238–258, 2017.

[CB95] Chen, Yirng-An; Bryant, Randal E.: Verification of Arithmetic Circuits with Binary Moment Diagrams. In: DAC 1995. ACM, S. 535–541, 1995.

[Ci20] Ciesielski, Maciej J.; Su, Tiankai; Yasin, Atif; Yu, Cunxi: Understanding Algebraic Rewriting for Arithmetic Circuit Verification: a Bit-Flow Model. IEEE TCAD, 39(6):1346–1357, 2020.

[Hu17] Hunt, Jr., Warren A.; Kaufmann, Matt; Strother Moore, J; Slobodova, Anna: Industrial Hardware and Software Verification with ACL2. Philos. Trans. Royal Soc. A, 375(2104):20150399, 2017.

[Ka20] Kaufmann, Daniela: Formal Verification of Multiplier Circuits using Computer Algebra. Dissertation, Informatik, Johannes Kepler University Linz, 2020.

[KBK19] Kaufmann, Daniela; Biere, Armin; Kauers, Manuel: Verifying Large Multipliers by Combining SAT and Computer Algebra. In: FMCAD 2019. IEEE, S. 28–36, 2019.

[KBK20a] Kaufmann, Daniela; Biere, Armin; Kauers, Manuel: From DRUP to PAC and Back. In: DATE 2020. IEEE, S. 654–657, 2020.

[KBK20b] Kaufmann, Daniela; Biere, Armin; Kauers, Manuel: Incremental Column-wise Verification of Arithmetic Circuits using Computer Algebra. FMSD, 56(1):22–54, 2020.

[KFB20] Kaufmann, Daniela; Fleury, Mathias; Biere, Armin: Pacheck and Pastèque, Checking Practical Algebraic Calculus Proofs. In: FMCAD 2020. TU Vienna Academic Press, S. 264–269, 2020.

[MGD19] Mahzoon, Alireza; Große, Daniel; Drechsler, Rolf: RevSCA: Using Reverse Engineering to Bring Light into Backward Rewriting for Big and Dirty Multipliers. In: DAC 2019. ACM, S. 185:1–185:6, 2019.

[RBK17] Ritirc, Daniela; Biere, Armin; Kauers, Manuel: Column-Wise Verification of Multipliers Using Computer Algebra. In: FMCAD 2017. IEEE, S. 23–30, 2017.

[RBK18] Ritirc, Daniela; Biere, Armin; Kauers, Manuel: A Practical Polynomial Calculus for Arithmetic Circuit Verification. In: SC2 Workshop 2018. CEUR-WS, S. 61–76, 2018.

[Sa16] Sayed-Ahmed, Amr; Große, Daniel; Kühne, Ulrich; Soeken, Mathias; Drechsler, Rolf: Formal Verification of Integer Multipliers byCombining Gröbner Basis with Logic Reduction. In: DATE 2016. IEEE, S. 1048–1053, 2016.

[SB94] Sharangpani, H.; Barton, M. L.: Statistical Analysis of Floating Point Flaw in the Pentium Processor. 1994.

[SK04] Stoffel, Dominik; Kunz, Wolfgang: Equivalence Checking of Arithmetic Circuits on the Arithmetic Bit Level. IEEE TCAD, 23(5):586–597, 2004.

[Va07] Vasudevan, Shobha; Viswanath, Vinod; Sumners, Robert W.; Abraham, Jacob A.: Automatic Verification of Arithmetic Circuits in RTL Using Stepwise Refinement of Term Rewriting Systems. IEEE Trans. Comput., 56(10):1401–1414, 2007.

[Yu16] Yu, Cunxi; Brown, Walter; Liu, Duo; Rossi, André; Ciesielski, Maciej J.: Formal Verification of Arithmetic Circuits by Function Extraction. IEEE TCAD, 35(12):2131–2142, 2016.

Daniela Kaufmann, geboren 1991 in Linz/Österreich, studierte von 2011 bis 2016 an der Johannes Kepler Universität Linz das Bachelorstudium technische Mathematik, sowie das Masterstudium Computermathematik. Daniela Kaufmann begann 2016 ihr Doktorratsstudium in Informatik unter der Betreuung von Prof. Armin Biere am Institut für Formale Modelle und Verifikation an der Johannes Kepler Universität Linz. Ihre erste Publikation über formale Verifikation von Multiplizierern wurde mit dem Best Paper Award der Intl. Conference for Formal Models in Computer Aided Design (FMCAD), einer der Top-Tier Konferenzen für Hardwareverifikation, ausgezeichnet. In 2020 wurde ihre Forschung mit dem JKU Young Researcher Award prämiert. Ihre Forschungsinteressen umfassen angewandte formale Methoden, Hardwareverifikation, Beweissysteme, sowie SAT Solving und Computeralgebra.