Divide and Verify: Using a Divide-and-Conquer Strategy for Polynomial Formal Verification of Complex Circuits

Rolf Drechsler[1,2] Alireza Mahzoon[1]
[1]Institute of Computer Science, University of Bremen, Bremen, Germany
[2]Cyber-Physical Systems, DFKI GmbH, Bremen, Germany
{drechsle, mahzoon}@uni-bremen.de

***Abstract*—With the rapid growth in the size and complexity of digital circuits, the possibility of bug occurrence has significantly increased. In order to avoid the enormous financial loss due to the production of buggy circuits, using scalable formal verification methods is essential. The scalability of a verification method for a specific design is proven by showing that the method has polynomial space and time complexities. Unfortunately, not all verification methods have a polynomial complexity, particularly when it comes to the verification of large and complex designs.**

In this paper, we propose a divide-and-conquer strategy for *Polynomial Formal Verification* (PFV) of complex circuits. Instead of using a monolithic proof engine to verify the entire design, we break the verification task down into several problems, which can be solved in polynomial space and time using a hybrid proof engine. As a case study, we investigate the PFV of the ALU in a RISC-V processor using our divide-and-conquer strategy.

I. Introduction

Recently, the verification community has achieved many successes in proving the correctness of a wide variety of digital circuits. Several formal methods based on equivalence checking, model checking, and theorem proving have been proposed to verify both combinational and sequential circuits. However, the main shortcoming of these techniques is unpredictability in performance, leading to several verification problems:

- It cannot be predicted before actually invoking the verification tool whether it will successfully terminate or run for an indefinite amount of time.
- The scalability of these techniques remains unknown, i.e., it is not predictable how much the run-time and the required memory increase when the size of the circuit grows.
- It is not possible to compare the performance of verification methods for a specific design and choose the best one.

In order to resolve the unpredictability of a verification method, its time and space complexities have to be calculated. Knowing the complexity bounds for a verification technique alleviates the three aforementioned verification problems. We are particularly interested in space and time complexities with the smallest possible polynomial order, i.e. $O(n^c)$, where n is a circuit parameter (e.g. the number of input bits) and c is a positive number. The concept of *Polynomial Formal Verification* (PFV) was first introduced in [1], where the author proved that PFV can be applied to three adder architectures using *Binary Decision Diagrams* (BDDs). Shortly, the complexity bounds for the verification of various circuits were calculated and new PFV techniques were proposed [2], [3], [4], [5], [6]. A formal verification method with polynomial complexity bounds (time and space), where the exponent in the polynomial is not too high, is scalable and can be carried out successfully for different circuit sizes [7].

Modern digital circuits consist of several sub-components. For example, a RISC-V processor is made of several sub-components, including an *Arithmetic Logic Unit* (ALU) to carry out arithmetic and logic operations. It is usually the case that a monolithic proof engine cannot ensure the correctness of the entire circuit in polynomial space and time. For example, a word-level proof engine cannot be used for the PFV of the entire RISC-V processor. In this paper, we propose a divide-and-conquer strategy to make the PFV of complex modern systems possible. Our approach takes advantage of a hybrid proof engine that includes both bit- and word-level formal techniques. Thus, the correctness of each block or system task can be ensured in polynomial space and time using a specific verification approach from the environment. We take advantage of the ALU in a RISC-V processor as a case study in order to demonstrate the success of our strategy in PFV of complex circuits. It is an important step toward PFV of highly complex designs, e.g., *Central Processing Units* (CPUs), *Digital Signal Processing* (DSP) blocks, and AI-synthesized architectures.

II. PFV using a Divide-and-Conquer Strategy

In this section, we first introduce our divide-and-conquer strategy for PFV. Then, we present a case study to illustrate the application.

A. Overview

Despite the progress in PFV of various circuits, most of the works are still limited to the polynomial verification of individual components, e.g., adders, and are based on a monolithic proof engine. Thus, the PFV of complex systems, consisting of many different sub-components, is an almost unexplored area. The challenge originates from the fact that a verification method (e.g., equivalence checking using BDDs) might verify a sub-component (e.g., an adder) in polynomial time but have an exponential verification complexity for another sub-component (e.g., a multiplier).

We propose a divide-and-conquer strategy for PFV of complex circuits. Instead of using a monolithic proof engine to verify the entire design, we break the verification task down into several problems, which can be solved in polynomial

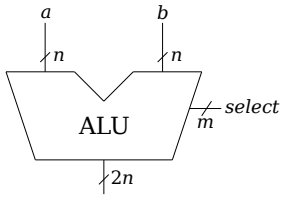

Fig. 1. Symbolic representation of the ALU

TABLE I
LIST OF SUPPORTED OPERATIONS

s_2	s_1	s_0	function
0	0	0	$0 \dots 0$
0	0	1	$b - a$
0	1	0	$a - b$
0	1	1	$a + b$
1	0	0	$a \times b$
1	0	1	$a \oplus b$
1	1	0	$a \vee b$
1	1	1	$a \wedge b$

space and time using a hybrid proof engine. Each sub-component or system task can be verified using a suitable formal approach that ensures PFV. Consequently, PFV can be applied to complex circuits which could not be verified using a single formal method in polynomial space and time. We take advantage of BDDs and *Symbolic Computer Algebra* (SCA) as our bit-level and word-level verification methods in our hybrid proof engine, since their polynomial upper-bounds have been proven for a wide variety of circuits (see e.g., [3], [4], [5], [8]).

B. Case Study: PFV of the ALU in a RISC-V Processor

An ALU is a combinational digital circuit that performs arithmetic and bitwise operations on integer binary numbers. The type and the number of supported operations in an ALU depend on the application. Fig. 1 shows the symbolic representation of an ALU. It receives two n-bit inputs a and b. The operation between the inputs is determined by an m-bit $select$. Finally, the result of the operation is returned as a $2n$-bit output.

Here, we consider a simplified ALU with 8 operations, i.e. the $select$ signal has 3 bits. The complete list of supported operations is depicted in Table I. The ALU can perform three arithmetic operations (i.e., addition, subtractions, and multiplication) as well as three bitwise logic operations (i.e., XOR, OR, and AND). The ALU can be used in a RISC-V processor to carry out logic and arithmetic operations.

We now discuss the results of verifying the ALU using a monolithic proof engine without using the proposed strategy:

- BDD-based verification reports very good results when it comes to ensuring the correctness of various adder architectures. It has been proven in [1] that carry look-ahead adder can be verified in polynomial space and time using BDDs. PFV can be also applied to the subtractor, since it is built by adding XOR gates to the inputs of the adder. However, BDD-based verification runs out of memory when it comes to the verification of multipliers. It has been proven in [9] that the size of output BDDs becomes exponential for a multiplier. As a result, a monolithic proof engine based on BDDs cannot be used for the PFV of the entire ALU.
- SCA-based verification has shown very good results for the verification of structurally simple multipliers. The experimental results demonstrated the efficiency of SCA-based verification in proving the correctness of million-gate multipliers [10]. In addition, it has been shown that the PFV of structurally simple multipliers is possible using SCA [5]. However, SCA-based methods run quickly out of memory when it comes to the verification of adders that are not only made of half-adders and full-adders. The authors of [5] have proven that the size of intermediate polynomials becomes exponential during the verification of a carry look-ahead adder. As a result, a monolithic proof engine based on SCA cannot be used for the PFV of the entire ALU.

We can overcome the limitations of monolithic proof engines in verifying the ALU by using our divide-and-conquer strategy: The verification of logic operations (AND, OR, and XOR) as well as addition and subtraction is performed using BDDs in polynomial space and time. Moreover, the SCA-based method is used for the PFV of the multiplication operation. As a result, the entire ALU can be verified polynomially.

III. CONCLUSION

In this paper, we propose a divide-and-conquer strategy for PFV. Complex digital circuits usually consist of many sub-components, which can be verified in polynomial space and time using a suitable verification technique. However, the PFV cannot be guaranteed using a monolithic proof engine. This problem can be alleviated by breaking down and introducing a hybrid proof engine that integrates bit- and word-level formal methods in one environment. Thus, each sub-component or system task is verified using one of the formal methods in polynomial space and time. We discussed the success of our strategy in the PFV of an ALU.

In the future, we plan to investigate the PFV of other complex digital circuits using the divide-and-conquer strategy.

ACKNOWLEDGEMENT

This work was supported by DFG within the Reinhart Koselleck Project *PolyVer* (DR 287/36-1).

REFERENCES

[1] R. Drechsler, "PolyAdd: Polynomial formal verification of adder circuits," in *DDECS*, 2021, pp. 99–104.
[2] R. Drechsler, A. Mahzoon, and L. Weingarten, "Polynomial formal verification of arithmetic circuits," in *ICCIDE*, 2021, pp. 457–470.
[3] A. Mahzoon and R. Drechsler, "Late breaking results: Polynomial formal verification of fast adders," in *DAC*, 2021, pp. 1376–1377.
[4] A. Mahzoon and R. Drechsler, "Polynomial formal verification of prefix adders," in *ATS*, 2021, pp. 85–90.
[5] R. Drechsler, A. Mahzoon, and M. Goli, "Towards polynomial formal verification of complex arithmetic circuits," in *DDECS*, 2022, pp. 1–6.
[6] J. Kleinekathöfer, A. Mahzoon, and R. Drechsler, "Polynomial formal verification of floating point adders," in *DATE*, 2023.
[7] R. Drechsler and A. Mahzoon, "Polynomial formal verification: Ensuring correctness under resource constraints," in *ICCAD*, 2022.
[8] M. Barhoush, A. Mahzoon, and R. Drechsler, "Polynomial word-level verification of arithmetic circuits," in *MEMOCODE*, 2021, pp. 1–9.
[9] R. E. Bryant, "On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication," *TC*, vol. 40, no. 2, pp. 205–213, 1991.
[10] A. Mahzoon, D. Große, and R. Drechsler, "RevSCA-2.0: SCA-based formal verification of non-trivial multipliers using reverse engineering and local vanishing removal," *TCAD*, pp. 1573–1586, 2022.