One is not Enough: Using Hybrid Proof Engines for Polynomial Formal Verification

Rolf Drechsler
University of Bremen/DFKI
Bremen, Germany
drechsler@uni-bremen.de

Alireza Mahzoon
University of Bremen
Bremen, Germany
mahzoon@uni-bremen.de

Abstract— In this paper, the concept of *Polynomial Formal Verification* (PFV) is reviewed. Then, we introduce a hybrid verification engine to attack the problem of verifying complex modern systems in polynomial space and time. The engine takes advantage of several verification techniques, such as combinational equivalence checking based on bit-level approaches, like SAT and *Binary Decision Diagrams* (BDDs), as well as word-level verification based on e.g. *Symbolic Computer Algebra* (SCA) and *Word-Level Decision Diagrams* (WLDDs). The correctness of each block or system task can be ensured in polynomial time using a specific verification technique from the environment. Thus, we overcome the shortcomings of using only one verification method and pave the way toward polynomial verification of advanced architectures.

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. Particularly, the formal verification of arithmetic circuits has gotten a lot of attention due to the high complexity and big size of these circuits: (a) *Binary Decision Diagram* (BDD) [21] and SAT-based [22] verification methods report very good results for different types of adder architectures, (b) *Multiplicative Binary Moment Diagrams* (*BMDs) [13, 4] are used to verify multipliers, and (c) *Symbolic Computer Algebra* (SCA) [20, 15, 25] is employed to verify multipliers and dividers.

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 [6], 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. 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.

Modern digital circuits consist of several subcomponents. For example, an *Arithmetic Logic Unit* (ALU) is made of several sub-components to carry out logic and arithmetic 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 ALU. In this paper, we propose a hybrid proof engine to make the PFV of complex modern systems possible. The engine takes advantage of both bit- and word-level formal approaches. 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 two case studies, i.e., an ALU and a structurally complex multiplier, in order to demonstrate the success of our hybrid proof engine 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.

A. Related Works

In the last few years, researchers have come up with various PFV methods to resolve the verification unpredictability. It includes 1) proving the polynomial bounds for existing verification methods and 2) improving and extending existing formal methods to obtain polynomial upper-bound complexities [10].

PolyAdd [6] for the first time proved that the formal verification of three adder architectures (i.e., ripple carry adder, conditional sum adder, and carry look-ahead adder) is possible in polynomial time using BDDs. The proof is based on the fact that underlying BDDs remain polynomial during the whole construction process. However, PolyAdd did not provide the upper-bound complexities. The authors of [18] and [19] extended PolyAdd by obtaining the upper-bound time complexities of conditional sum adder and parallel prefix adders (i.e., serial prefix adder, Ladner-Fischer adder, and Kogge-Stine adder). They calculated the time complexities by adding up the computational complexity of *If-Then-Else* (ITE) operation in each step of the symbolic simulation. Formal verification of AI-generated prefix adders in polynomial time was investigated in [9]. The authors of [12] proved that PFV of a simple ALU, consisting of arithmetic and logic operations, is possible. Authors of [23] focused on the PFV of approximate adders. They proved that the upper-bound time complexities of verifying approximate ripple carry adder, conditional sum adder, and carry look-ahead adder, as well as handcrafted approximate adders, are polynomial using BDDs.

The authors of [5, 17] proposed a BDD-based verification technique to ensure the correctness of multipliers. They also proved that the output BDD sizes are polynomial. However, they did not calculate the verification complexity. The work of [16] considered the PFV of a multiplier for the first time. The authors demonstrated that the verification of a Wallace-tree like multiplier can be carried out in polynomial space and time using *BMDs. The proof was extended by [1] to arithmetic circuits consisting of multiplication and addition operations. Moreover, the authors showed that PFV can be also performed using SCA. The authors of [11] proved that SCA-based methods have exponential upper-bound complexities when it comes to verifying structurally complex multipliers. Then, they came up with a hybrid formal method based on SCA and BDDs to achieve polynomial bounds.

In addition to arithmetic circuits, there have been some efforts to make PFV possible for other types of circuits. The authors of [8] and [7] proved that ensuring the correctness of symmetric functions and tree-like circuits is possible in polynomial space and time using BDDs. The work of [24] proposed two methods to generate polynomially verifiable circuits for an approximate function.

In this paper, we highlight the limitations of a monolithic proof engine in PFV of complex digital circuits. Then, we propose a hybrid proof engine that takes advantage of bit- and word-level verification approaches to overcome these limitations.

II. Background

In this section, we first review the bit-level verification methods with a focus on BDDs. Then, we give an overview of word-level methods, particularly SCA.

A. Verification using Bit-Level Techniques

In a bit-level verification method, a circuit is described in the Boolean domain, i.e., the functions receive the intermediate and input signals as individual Boolean variables and return the outputs in the Boolean domain as well. The verification method based on BDDs is one of the examples of bit-level verification. In this section, we focus on BDD-based verification.

We first briefly summarize some basics of BDD:

- **Binary Decision Diagram (BDD):** a directed, acyclic graph whose nodes have two edges associated with the values of the variables 0 and 1. A BDD contains two terminal nodes (leaves) that are associated with the values of the function 0 or 1.
- **Ordered BDD (OBDD):** a BDD, where the variables occur in the same order along each path from the root to a leaf.
- **Reduced OBDD (ROBDD):** an OBDD that contains a minimum number of nodes for a given variable order.

We refer to ROBDD as BDD in the rest of the paper, since it is the canonical representation that is used in the verification of arithmetic circuits.

The ITE operator (If-Then-Else) [2] is used to calculate the results of the logic operations in BDDs:

$$ITE(f, g, h) = (f \wedge g) \vee (\overline{f} \wedge h), \tag{1}$$

The basic binary operations can be presented using the ITE operator:

$$\begin{aligned} f \wedge g &= ITE(f, g, 0), \qquad & f \vee g &= ITE(f, 1, g), \\ f \oplus g &= ITE(f, \overline{g}, g), & \overline{f} &= ITE(f, 0, 1). \end{aligned} \tag{2}$$

In order to formally verify a circuit, we need to have the BDD representation of the outputs. Symbolic simulation helps us to obtain the BDD for each primary output. In a simulation, an input pattern is applied to a circuit, and the resulting output values are observed to see whether they match the expected values. On the other hand, symbolic simulation verifies a set of scalar tests (which usually

covers the whole input space) with a single symbolic test. Symbolic simulation using BDDs is done by generating corresponding BDDs for the input signals. Then, starting from primary inputs, the BDD for the output of a gate (or a building block) is obtained using the ITE operation. This process continues until we reach the primary outputs. Finally, the output BDDs are evaluated to see whether they match the BDDs of the circuit.

B. Verification using Word-Level Techniques

In a word-level verification method, a circuit is described in the integer domain, i.e., the functions receive the intermediate and input signals as individual Boolean variables and return the outputs in the integer domain. The verification method based on SCA is one of the examples of word-level verification.

We now summarize some basics of SCA:

- **Monomial:** power product of the variables, i.e. $M = x_1^{a_1} x_2^{a_2} \dots x_n^{a_n}$, where $a_i \geq 0$.
- **Polynomial:** finite sum of monomials, i.e. $P = c_1 M_1 + \dots + c_j M_j$ with coefficients in field k.
- **Division:** Assuming p is a polynomial and F is a set of polynomials, the division of p by F is denoted by $p \xrightarrow{F} r$, where r is called a remainder.

The goal of SCA-based verification is to formally prove that all signal assignments consistent with the gate-level or *AND Inverter Graph* (AIG) representation evaluate the *Specification Polynomial* (SP) to 0. The SP determines the word-level function of an arithmetic circuit based on its inputs and outputs, e.g. for the half-adder of Fig. 1 $SP = 2C + S - (A + B)$, where $2C + S$ represents the word-level representation of the 2-bit output, and $A + B$ represents the addition of the 1-bit inputs.

Before verification, the gates of the circuit should be modeled as polynomials describing the relation between inputs and outputs. If the circuit is built from basic logic gates (e.g., NOT, AND, OR, and XOR), four different operations might happen in the circuit. Assuming z is the output, and a and b are the inputs of a gate, the polynomials for the basic logic gates are as follows:

$$\begin{aligned} z &= \neg a \Rightarrow p_g := z - 1 + a, \\ z &= a \wedge b \Rightarrow p_g := z - a \cdot b, \\ z &= a \vee b \Rightarrow p_g := z - a - b + a \cdot b, \\ z &= a \oplus b \Rightarrow p_g := z - a - b + 2a \cdot b. \end{aligned} \tag{3}$$

The extracted gate polynomials are in the form $P_g = x - tail(P_g)$, where x is the gate's output, and $tail(P_g)$ is a function based on the gate's inputs. Similarly, the polynomials for the nodes can be extracted in an AIG representation (see [26, 20]).

Based on the Gröbner basis theory, all signal assignments consistent with the AIG evaluate the specification polynomial SP to 0, iff the remainder of dividing SP by the gate polynomials is equal to 0 (see [15] for more details).

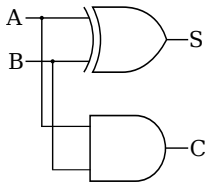

Fig. 1.: Half-adder

The step-wise division of SP by gate polynomials for the half-adder of Fig. 1 is as follows:

$$\begin{aligned} &SP := 2C + S - (A + B), \\ &SP \xrightarrow{P_{AND}} SP_1 = 2AB + S - (A + B), \\ &SP_1 \xrightarrow{P_{XOR}} r = 0. \end{aligned} \tag{4}$$

Since the remainder is zero, the circuit is bug-free. In arithmetic circuits, dividing SP_i by a gate polynomial $P_{g_i} = x_i - tail(P_{g_i})$ is equivalent to substituting x_i with $tail(P_{g_i})$ in SP_i. For example, dividing SP_1 by P_{XOR} in Eq. (4) is equivalent to substituting S with $tail(P_{XOR}) = A + B - 2A \cdot B$ in SP_1. In the results, we always replace powers $x_i^{a_i}$ with $a_i > 1$ by x_i, since x_i can only take values from $\{0, 1\}$. In the theory, this corresponds to adding $x_i^2 - x_i$ to the gate polynomials. The process of step-wise division (substitution) is called *backward rewriting*.

III. PFV using a Hybrid Proof Engine

In this section, we first introduce our hybrid proof engine that uses both bit- and word-level approaches for PFV. Then, we present two case studies to illustrate the applications of our hybrid proof engine.

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 hybrid proof engine that integrates both bit- and word-level approaches in an environment. As a result, the verification is not limited to a single formal method. Each sub-component or system task can be verified using a suitable formal approach that ensures

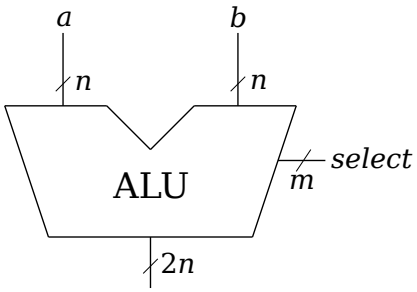

Fig. 2.: Symbolic representation of the ALU

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	$a \wedge b$

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 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., [18, 19, 11, 1]).

B. Case Study I: PFV of an ALU

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. 2 shows the symbolic representation of a general 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.

In this paper, we consider an 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 addition and subtraction are implemented based on the carry look-ahead algorithm. On the other hand, the architectures for the three stages of the multiplier (see Fig. 3) are as follows: simple partial product generator, array, and ripple carry adder. The multiplier is structurally simple, since the second and third stages are only made of half-adders and full-adders.

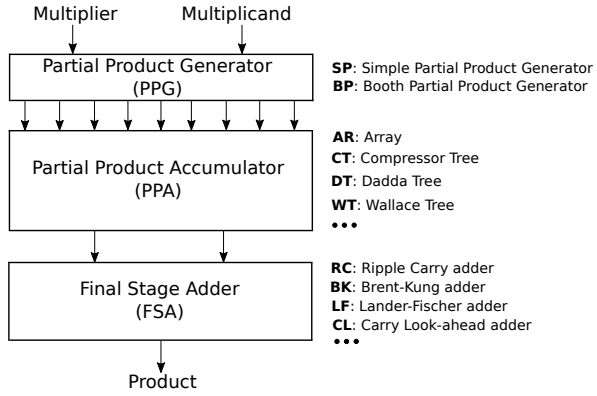

Fig. 3.: Multiplier structure

We now discuss the results of verifying the ALU using a monolithic proof engine based on BDDs and SCA:

- BDD-based verification reports very good results when it comes to ensuring the correctness of various adder architectures. It has been proven in [6] 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 [3] 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 [20]. In addition, it has been shown that the PFV of structurally simple multipliers is possible using SCA [11]. 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 [11] 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 hybrid proof engine. 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 using our hybrid proof engine.

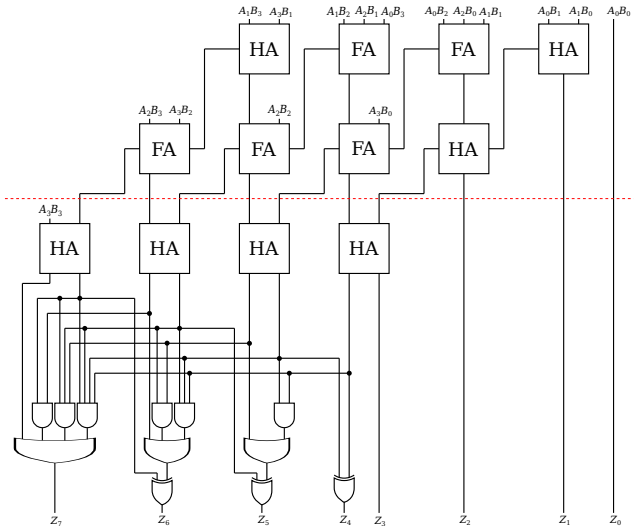

Fig. 4.: 4×4 structurally complex multiplier

C. *Case Study II: PFV of a Structurally Complex Multiplier*

If the second and third stages of a multiplier (see Fig. 3) are not only made of half-adders and full-adders, it is called a structurally complex multiplier. Fig. 4 depicts a 4×4 structurally complex multiplier, where the final stage adder has a carry look-ahead adder architecture. Ensuring the correctness of structurally complex multipliers is a big challenge for the verification community. Several formal verification methods based on SCA have been proposed to overcome the challenges [20, 14]. However, it is not trivial to prove their polynomial complexity for all multiplier architectures due to some heuristics in their flow.

We now discuss the results of verifying the complex multiplier using a monolithic proof engine based on BDDs and SCA:

- Similar to the structurally simple multipliers, the size of output BDDs becomes exponential for structurally complex multipliers. Several techniques have been proposed to make the verification of multipliers possible using BDDs. The work of [5] considers the partial products as new input variables and constructs the output BDDs based on them. As a result, the size of output BDDs becomes polynomial with respect to the input width. However, it is still not clear whether the size of intermediate BDDs is polynomially bounded during the symbolic simulation. Thus, BDD-based verification cannot ensure the PFV of structurally complex multipliers.

- Although SCA-based verification has shown very good results for the verification of structurally simple multipliers, it fails when it comes to ensuring the correctness of structurally complex multipliers. It has been shown experimentally that the size of intermediate polynomials grows drastically during backward rewriting. The authors of [11] proved that the size of intermediate polynomials increases exponentially for structurally complex multipliers. As a result, a monolithic proof engine based on SCA cannot be used for the PFV of a structurally complex multiplier.

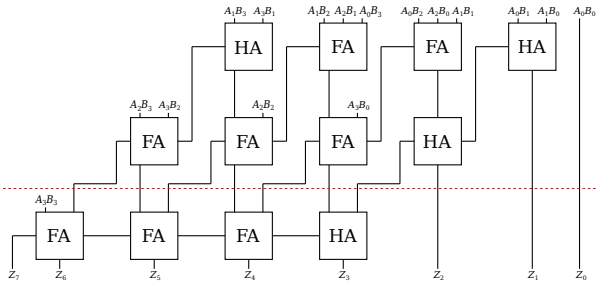

Fig. 5.: 4×4 structurally simple multiplier

If the design hierarchy, including the boundaries between the three stages of the multiplier (i.e. PPG, PPA, and FSA) and the components in each stage are available, we can take advantage of our hybrid proof engine to ensure the correctness of the structurally complex multiplier. Our method consists of three main steps:

1. the final stage of the multiplier, i.e. FSA, is replaced with a ripple carry adder,
2. the new multiplier architecture is verified using SCA,
3. the FSA is verified using BDDs.

If both verification methods ensure correctness, the multiplier is bug-free. Otherwise, it is buggy.

It is now possible to calculate the space and time complexity of SCA and BDD-based methods separately and prove their polynomial upper-bounds with respect to the multiplier size: After replacing the FSA with a ripple carry adder, the new multiplier is structurally simple, since the second and third stages are made of half-adders and full-adders (see Fig. 5). It has been proven in [11] that structurally simple multipliers can be verified in polynomial space and time using SCA. On the other hand, PFV can be applied to the original FSA using BDDs as proven in [6]. As a consequence, PFV of structurally complex multipliers becomes possible.

IV. Conclusions

In this paper, we illustrated the importance of using a hybrid proof engine 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 introducing a hybrid proof engine that integrates bit- and word-level formal methods in an 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 a hybrid proof engine in the PFV of an ALU and a structurally complex multiplier.

In the future, we plan to investigate the PFV of other complex digital circuits such as CPUs and DSP blocks using a hybrid verification engine.

Acknowledgements

Parts of this work have been supported by DFG within the Reinhart Koselleck Project *PolyVer: Polynomial Verification of Electronic Circuits* (DR 287/36-1).

References

[1] Barhoush, M., Mahzoon, A., Drechsler, R.: Polynomial word-level verification of arithmetic circuits. In: MEMOCODE. pp. 1–9 (2021)

[2] Brace, K.S., Rudell, R.L., Bryant, R.E.: Efficient implementation of a BDD package. In: DAC. pp. 40–45 (1990)

[3] Bryant, R.E.: On the complexity of VLSI implementations and graph representations of boolean functions with application to integer multiplication. TC 40(2), 205–213 (1991)

[4] Bryant, R.E., Chen, Y.A.: Verification of arithmetic circuits with binary moment diagrams. In: DAC. pp. 535–541 (1995)

[5] Burch, J.: Using BDDs to verify multipliers. In: DAC. pp. 408–412 (1991)

[6] Drechsler, R.: PolyAdd: Polynomial formal verification of adder circuits. In: DDECS. pp. 99–104 (2021)

[7] Drechsler, R.: Polynomial circuit verification using BDDs. In: ICEECCOT. pp. 466–483 (2021)

[8] Drechsler, R., Dominik, C.: Edge verification: Ensuring correctness under resource constraints. In: SBCCI. pp. 1–6 (2021)

[9] Drechsler, R., Mahzoon, A.: Towards polynomial formal verification of AI-generated arithmetic circuits. In: ISDCS (2021)

[10] Drechsler, R., Mahzoon, A.: Polynomial formal verification: Ensuring correctness under resource constraints. In: ICCAD (2022)

[11] Drechsler, R., Mahzoon, A., Goli, M.: Towards polynomial formal verification of complex arithmetic circuits. In: DDECS. pp. 1–6 (2022)

[12] Drechsler, R., Mahzoon, A., Weingarten, L.: Polynomial formal verification of arithmetic circuits. In: ICCIDE. pp. 457–470 (2021)

[13] Hamaguchi, K., Morita, A., Yajima, S.: Efficient construction of binary moment diagrams for verifying arithmetic circuits. In: ICCAD. pp. 78–82 (1995)

[14] Kaufmann, D., Beame, P., Biere, A., Nordström, J.: Adding dual variables to algebraic reasoning for gate-level multiplier verification. In: DATE. pp. 1431–1436 (2022)

[15] Kaufmann, D., Biere, A., Kauers, M.: Verifying large multipliers by combining SAT and computer algebra. In: FMCAD. pp. 28–36 (2019)

[16] Keim, M., Drechsler, R., Becker, B., Martin, M., Molitor, P.: Polynomial formal verification of multipliers. Formal Methods in System Design: An International Journal 22(1), 39–58 (2003)

[17] Kumar, J., Miyasaka, Y., Srivastava, A., Fujita, M.: Formal verification of integer multiplier circuits using binary decision diagrams. TCAD (2022), early access

[18] Mahzoon, A., Drechsler, R.: Late breaking results: Polynomial formal verification of fast adders. In: DAC. pp. 1376–1377 (2021)

[19] Mahzoon, A., Drechsler, R.: Polynomial formal verification of prefix adders. In: ATS. pp. 85–90 (2021)

[20] Mahzoon, A., Große, D., Drechsler, R.: RevSCA-2.0: SCA-based formal verification of non-trivial multipliers using reverse engineering and local vanishing removal. TCAD pp. 1573–1586 (2022)

[21] Malik, S., Wang, A.R., Brayton, R.K., Sangiovanni-Vincentelli, A.L.: Logic verification using binary decision diagrams in a logic synthesis environment. In: ICCAD. pp. 6–9 (1988)

[22] Mishchenko, A., Chatterjee, S., Brayton, R.K.: DAG-aware AIG rewriting a fresh look at combinational logic synthesis. In: DAC. pp. 532–535 (2006)

[23] Schnieber, M., Fröhlich, S., Drechsler, R.: Polynomial formal verification of approximate adders. In: DSD (2022)

[24] Schnieber, M., Fröhlich, S., Drechsler, R.: Polynomial formal verification of approximate functions. In: ISVLSI (2022)

[25] Yu, C., Brown, W., Liu, D., Rossi, A., Ciesielski, M.: Formal verification of arithmetic circuits by function extraction. TCAD 35(12), 2131–2142 (2016)

[26] Yu, C., Ciesielski, M., Mishchenko, A.: Fast algebraic rewriting based on and-inverter graphs. TCAD 37(9), 1907–1911 (2017)