Polynomial Formal Verification: Ensuring Correctness under Resource Constraints

(Invited Paper)

Rolf Drechsler
University of Bremen
Cyber-Physical Systems, DFKI GmbH
Bremen, Germany
drechsler@uni-bremen.de

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

ABSTRACT

Recently, a lot of effort has been put into developing formal verification approaches by both academic and industrial research. In practice, these techniques often give satisfying results for some types of circuits, while they fail for others. A major challenge in this domain is that the verification techniques suffer from unpredictability in their performance. The only way to overcome this challenge is the calculation of bounds for the space and time complexities. If a verification method has polynomial space and time complexities, scalability can be guaranteed.

In this tutorial paper, we review recent developments in formal verification techniques and give a comprehensive overview of *Polynomial Formal Verification* (PFV). In PFV, polynomial upper bounds for the run-time and memory needed during the entire verification task hold. Thus, correctness under resource constraints can be ensured. We discuss the importance and advantages of PFV in the design flow. Formal methods on the bit-level and the word-level, and their complexities when used to verify different types of circuits, like adders, multipliers, or ALUs are presented. The current status of this new research field and directions for future work are discussed.

CCS CONCEPTS

• **Hardware** → **Functional verification**.

KEYWORDS

polynomial formal verification, complexity, binary decision diagrams, symbolic computer algebra

1 INTRODUCTION

With the invention of the transistor in 1947, the cornerstone for the digital revolution was laid. As a fundamental building block, the transistor triggered the development of digital circuits. The mass production of digital circuits revolutionized the field of electronics, finally leading to computers, embedded systems, and the internet. Hence, the impact of digital hardware on society, as well as the economy, was and is tremendous. Over the last decades, the enormous growth in the complexity of integrated circuits continues as expected. Digital circuits nowadays are far more complex, sometimes even consisting of billions of transistors. Back in 2000, an Intel Pentium 4 processor had 42 million transistors, and it was working with a 1.4 GHz frequency. Thirteen years later, Intel released its Core-i Series processors. They consist of more than 5 billion transistors (i.e. 120× Pentium 4 transistors) and work with clock speeds of up to 4.4 GHz. Moreover, modern digital circuits are usually designed based on sophisticated algorithms, leading to fast but complex architectures.

As modern electronic devices are getting more and more ubiquitous, the fundamental issue of functional correctness becomes more important than ever. This is evidenced by many publicly known examples of electronic failures with disastrous consequences. This includes e.g., the Intel Pentium bug in 1994 [2], the New York blackout in 2003 [41], and a design flaw in Intel's Sandy Bridge chipset in 2011. Such costly mistakes can only be prevented by applying rigorous verification to the circuits before they get to production [10, 11]. Exhaustive simulation (i.e., checking the outputs for each provided test-vector) is not a feasible approach to ensure correctness since it is impossible to cover the whole input space in the case of large digital circuits. As a result, a lot of effort has been put into developing formal verification techniques by both academic and industrial research. Essentially, formal verification aims to formally prove that an implementation is correct with respect to its specification. Formal verification methods take advantage of rigorous mathematical reasoning to prove that a design meets its specification. Nowadays, formal verification is an essential task in each phase of the design flow since it is the only way to ensure the 100% correctness of an implementation.

Several bit-level and word-level formal verification algorithms have been proposed in recent years to prove the correctness of electronic circuits (see e.g. [17, 24–26, 35]). In practice, they might give satisfying results for some types of circuits, but they might also fail due to non-efficient run-time and memory usage if the size of the circuits increases. As a result, these verification algorithms suffer from unpredictability in their performance. The time and space complexities of many formal methods are unknown when it comes to verifying various types of designs. It cannot be predicted before actually invoking the verification tool whether (a) it will successfully terminate or (b) run for an indefinite amount of time. It is a serious challenge in the verification phase and can dramatically affect the time schedule for the implementation and fabrication of a digital circuit. This obstacle can only be overcome by calculating the verification complexity of different types of circuits. We are particularly interested in verification techniques whose space and time complexities are polynomially bounded.

Polynomial Formal Verification (PFV) was first introduced in [12] for adder architectures. Shortly, researchers put a lot of effort into proving the polynomial bounds for the existing methods and proposed new PFV approaches. In general, calculating the space and

ICCAD '22, October 30-November 3, 2022, San Diego, CA, USA

ACM ISBN 978-1-4503-9217-4/22/10.
https://doi.org/10.1145/3508352.3561104

time complexities and proving the polynomial bounds provide us with three main advantages:

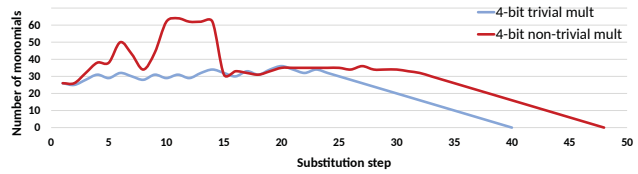
- We can predict before running a verification engine whether it returns the results in a limited period. As a result, we can avoid the verification methods with exponential space and time complexities.
- We can ensure the scalability of a verification method when it comes to proving the correctness of a specific type of circuit. Thus, the verification run-time and memory usage increase polynomially with respect to the size of the circuit. It is particularly important when there is a resource constraint for the verification process.
- We can compare the upper-bound space and time complexities of two verification methods when they are applied to a specific type of circuit. Consequently, we can realize which method performs better in terms of run-time and memory usage.

In this paper, we give **a comprehensive overview of PFV and its importance in the design flow**. We first clarify the advantages of a verification-centric strategy compared to a design-centric strategy in terms of resource management and time-to-market. Then, we review the PFV of various digital circuits. In the last few years, researchers have obtained the complexity bounds for several types of circuits, including various arithmetic circuits. We give an overview of these research works. To clarify the process of complexity calculation, we calculate the time complexity of verifying a carry bypass adder using *Binary Decision Diagrams* (BDDs) [4] for the first time. We demonstrate that the PFV of this adder architecture is possible. Subsequently, we illustrate the role of design information, including functional properties and hierarchical information, in proving polynomial bounds. Finally, we discuss the current trends and directions for future work in the field of PFV.

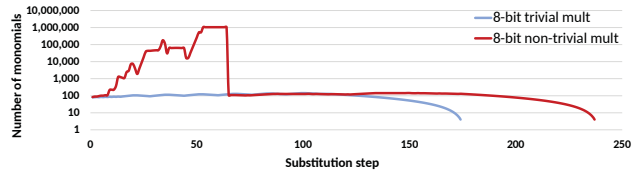
The remainder of this paper is structured as follows: Section 2 highlights the advantages of PFV during the design flow and its role in saving resources and reducing time-to-market. Section 3 reviews the bit-level and word-level verification techniques. An overview of PFV methods for arithmetic circuits, including adders, approximate adders, multipliers, and complex arithmetic circuits are presented in Section 4. Section 5 explains the role of using additional information including functional properties and hierarchical information in PFV. The current trends and future directions of PFV are described in Section 6. Finally, Section 7 concludes the paper.

2 FROM FORMAL VERIFICATION TO PFV

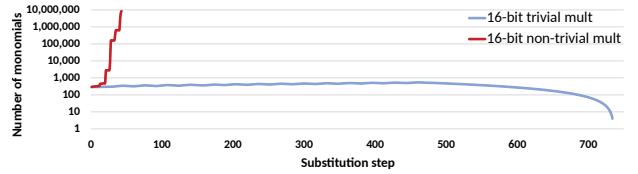
It is usually the case that the verification resources are limited, which applies to available time prior to fabrication (time-to-market) and also the computational resources in terms of computational power and memory. If the space and time complexities of a verification method are unknown, it becomes impossible to predict the required resources before performing the verification task itself. As a result, it is highly possible that we quickly run out of resources during verification. It is a big challenge in the design process that might influence the schedule for the implementation and fabrication of an electronic circuit and causes huge financial losses.



(a) 4-bit multipliers



(b) 8-bit multipliers



(c) 16-bit multipliers

Figure 1: Memory usage at each step of SCA-based verification

As an example of memory consumption, consider the verification of two types of integer multipliers using *Symbolic Computer Algebra* (SCA) (see [34, 35] for more details). The first type is a trivial multiplier that uses a simple partial product generator (i.e., AND gates) in the first stage. Then, it takes advantage of only half-adders and full-adders to reduce partial products and generate the final product. The second type is a non-trivial multiplier. Similarly, it uses a simple partial product generator (i.e., AND gates) for its first stage. However, it also uses a carry look-ahead adder architecture in its final stage; thus, it is not fully made of half-adders and full-adders. Figure 1 reports the memory usage in terms of number of monomials at each step of SCA-based verification for 4×4, 8×8, and 16×16 multipliers. The memory usage during the SCA-based verification of trivial multipliers remains almost constant, and it drops at the final steps. Thus, the trivial multipliers can be easily verified using SCA-based verification with limited resources. However, we face serious challenges in the verification of non-trivial multipliers. While, the memory usage is acceptable for the 4×4 non-trivial multiplier in Figure 1(a), it grows significantly during the verification of an 8×8 multiplier ($1,000\times$ compared to trivial multiplier) in Figure 1(b). The situation becomes even worse for the verification of a 16×16 non-trivial multiplier in Figure 1(c) as we run out of memory.

The above-mentioned example clarifies the importance of PFV in the design flow. The non-trivial multiplier outperforms the trivial multiplier in terms of some design parameters, e.g., speed. However, it cannot be verified in polynomial space and time. As a result, the verification task consumes a lot of resources without successfully

returning the results. It is a common challenge in a design-centric flow, where only design parameters (e.g., speed, area, and power) are taken into account. In order to overcome this challenge, the gap between the design and verification phase has to be filled. So far, the verification phase was a task that has to be carried out when the design phase is finished and designers do not care about the verifiability of the circuit. We can fill this gap by employing a verification-centric strategy that considers verifiability as an important parameter during the design. Hence, we can only use the circuits whose formal verification is possible in polynomial space and time. Following this strategy makes the verification phase much shorter and more predictable.

Adopting a verification-centric strategy is only possible by proving the polynomial bounds of existing verification methods and proposing PFV approaches. We review the recent developments in this domain in the rest of the paper.

3 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.

3.1 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 methods based on BDDs [8, 26] and SAT [9, 24] are the most notable 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 in 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) [3] 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:

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

ITE can be also used recursively in order to compute the results:

$$ITE(f, g, h) = ITE(x_i, ITE(f_{x_i}, g_{x_i}, h_{x_i}), ITE(f_{\overline{x}_i}, g_{\overline{x}_i}, h_{\overline{x}_i})), \tag{3}$$

Algorithm 1 If-Then-Else (ITE)

Input: f, g, h BDDs
Output: ITE BDD
1: **if** terminal case **then**
2: **return** *result*
3: **else if** computed-table has entry $\{f, g, h\}$ **then**
4: **return** *result*
5: **else** ▷ General case
6: v = top variable for f, g, or h
7: $t = ITE(f_{v=1}, g_{v=1}, h_{v=1})$
8: $e = ITE(f_{v=0}, g_{v=0}, h_{v=0})$
9: $r = FindOrAddUniqueTable(v, t, e)$
10: $InsertComputedTable(\{f, g, h\}, r)$
11: **return** R

where f_{x_i} ($f_{\overline{x}_i}$) is the positive (negative) cofactor of f with respect to x_i, i.e., the result of replacing x_i by the value 1 (0).

The algorithm for calculating ITE operations is presented in Algorithm 1. The result is computed recursively based on Eq. (3) in this algorithm. When calculating the results of ITE operations for the BDDs of f, g, h, the arguments for subsequent calls to the ITE subroutine are the sub-diagrams of f, g and h. The algorithm employs two major data structures: a *Unique Table* to guarantee the canonicity of the BDDs (see Line 9), and a *Computed Table* to store results of previous computations and avoid repetition (see Line 10). The number of sub-diagrams in a BDD is equivalent to the number of nodes. For each of the three arguments, the subroutine is called at most once. Assuming that the search in the *Unique Table* is performed at a constant time, the computational complexity of the ITE algorithm, even in the worst-case, does not exceed $O(|f| \cdot |g| \cdot |h|)$, where $|f|$, $|g|$ and $|h|$ denote the size of the BDDs in terms of the number of nodes[1].

In order to formally verify an adder, 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 algorithm. This process continues until we reach the primary outputs. Finally, the output BDDs are evaluated to see whether they match the BDDs of an adder.

3.2 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 methods based on SCA and word-level graphs are the most notable examples of word-level

[1]This bound holds under the assumption of an optimal hashing in $O(1)$. However, in the case of a worst-case behavior of the hashing function, ITE still remains polynomial (see [23]).

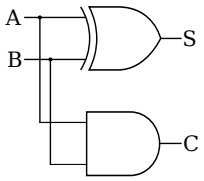


Figure 2: Half-adder

verification. In this section, we first focus on SCA-based verification and then briefly introduce the methods based on *Word-Level Decision Diagrams* (WLDDs).

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 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 Figure 2 $SP = 2C + S - (A + B)$, 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:

$$\begin{aligned} z &= \neg a \Rightarrow p_g := z - 1 + a, \\ z &= a \wedge b \Rightarrow p_g := z - a \cdot b, \\ z &= a \vee b \Rightarrow p_g := z - a - b + a \cdot b, \\ z &= a \oplus b \Rightarrow p_g := z - a - b + 2a \cdot b. \end{aligned} \tag{4}$$

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 [35, 51]).

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 [29] for more details).

The step-wise division of *SP* by gate polynomials for the half-adder of Figure 2 is as follows:

$$\begin{aligned} &SP := 2C + S - (A + B), \\ &SP \xrightarrow{P_{AND}} SP_1 = 2AB + S, \\ &SP_1 \xrightarrow{P_{XOR}} r = 0. \end{aligned} \tag{5}$$

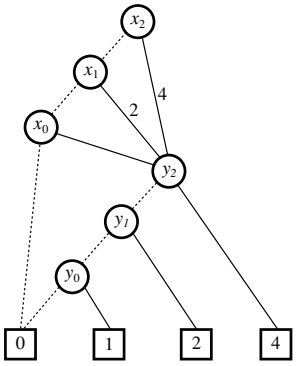


Figure 3: *BMD of a 3-bit multiplication function

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. (5) 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*.

The verification methods based on WLDDs [23] are very similar to the SCA-based method. The only major difference is that they represent polynomials as graphs which usually require less memory. For example, *Binary Moment Diagrams* (BMDs) [15] are constructed using the moment decomposition:

$$f = f_{\overline{x}_i} + x_i \cdot f_{\dot{x}_i} \qquad (f_{\dot{x}_i} = f_{x_i} - f_{\overline{x}_i}), \tag{6}$$

where f_{x_i} and $f_{\overline{x}_i}$ are the functions resulted from the substitution of x_i with 1 and 0, respectively. The BMD representing a function f is constructed recursively so that its root labeled by x is linked via its 0-edge (1-edge) to the root of the BMD representing $f_{\overline{x}_i}$ ($f_{\dot{x}_i}$). The size of a BMD can be reduced using common factors in the constant moment ($f_{\overline{x}_i}$) and linear moment ($f_{\dot{x}_i}$), resulting in a new representation called *Multiplicative Binary Moment Diagrams* (*BMD). Figure 3 shows the *BMD of a 3-bit multiplication function, i.e., $X \times Y = (4x_2 + 2x_1 + x_0) \times (4y_2 + 2y_1 + y_0)$.

4 PFV FOR ARITHMETIC CIRCUITS

In this section, we give an overview of research works dedicated to the PFV of arithmetic circuits.

4.1 Adders

It was known for a long time that BDD-based verification reports very good results when it comes to adder architectures. However, the upper-bound time complexities of this method were not investigated until recently. PolyAdd [12] demonstrated that the BDD-based verification of three adder architectures (i.e., ripple carry adder, conditional sum adder, and carry look-ahead adder) has polynomial upper-bound complexities. The author proved that the underlying BDDs remain polynomial during the whole BDD construction process. This was ensured by proving upper bounds on the BDD sizes for each internal signal. While the BDD sizes for the outputs of the

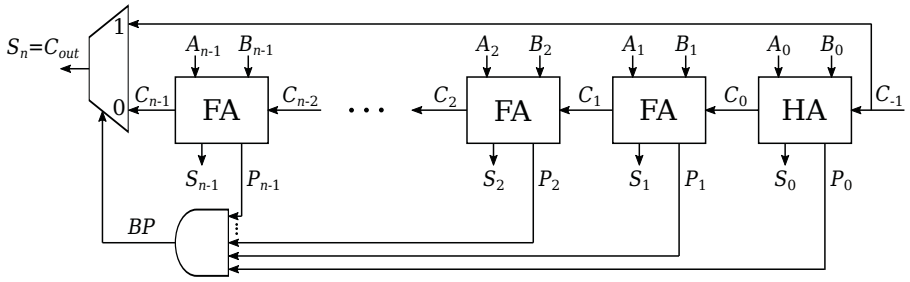


Figure 4: Carry bypass adder

adder functions were known to be polynomially bounded, this was the first time that a polynomial proof process was ensured for efficient adder circuits of logarithmic run time. However, PolyAdd did not provide the exact verification complexity bounds for different adder architectures. Several research works addressed this issue by calculating the complexity bounds.

The authors of [32] calculated the time complexity of BDD-based verification for an n-bit conditional sum adder. They proved that the whole verification process has a quartic (i.e., $O(n^4)$) upper-bound time complexity, and thus PFV is possible for this architecture. The authors of [33] obtained the time complexity of verifying prefix adders using BDDs. They demonstrated that a serial prefix adder has quadratic (i.e., $O(n^2)$) upper bound verification time complexity. On the other hand, the verification complexities of a Ladner-Fischer adder and a Kogge-Stone adder have quartic bounds. The work of [33] was extended by [18] to cover general prefix adders. The authors proved that the worst-case time complexity never exceeds $O(n^4)$ for the verification of an n-bit prefix adder. It is an important result since it is also applicable to the BDD-based verification of AI-generated prefix adders, which have better area and delay compared to the known prefix adder architectures [39, 42, 43].

The time complexity of verifying an adder architecture is obtained by calculating the computational complexity of ITE operation at each step of symbolic simulation. Then, these complexities are added up to get the complexity of the entire verification task. In order to clarify this process, we now calculate the upper-bound time complexity of verifying a carry bypass adder [40] using BDDs.

A carry bypass adder is an adder architecture that improves the worst-case delay of a ripple carry adder by using minimum extra hardware. Figure 4 shows the structure of an n-bit carry bypass adder. Each full-adder generates three outputs, i.e., sum (S_i), carry(C_i), and propagate signal (P_i):

$$\begin{aligned} S_i &= A_i \oplus B_i \oplus C_{i-1}, \\ C_i &= (A_i \wedge B_i) \vee (A_i \wedge C_{i-1}) \vee (B_i \wedge C_{i-1}), \\ P_i &= A_i \oplus B_i, \end{aligned} \tag{7}$$

where A_i and B_i are the i-th inputs. If the result of performing AND operation between all propagate signals P_i generate 1, the output carry equals the input carry. Otherwise, the output carry should be computed through the full-adder chain.

In order to obtain the time complexity of symbolic simulation for an n-bit carry bypass adder, we first calculate the complexity of symbolic simulation for a single full-adder. The sum, carry, and propagate outputs of a full-adder can be expressed in terms of ITE operations as follows:

$$\begin{aligned} S_i &= A_i \oplus B_i \oplus C_{i-1} = ITE(C_{i-1}, A_i \odot B_i, A_i \oplus B_i), \\ A_i \odot B_i &= ITE(A_i, B_i, \overline{B}_i), \\ A_i \oplus B_i &= ITE(A_i, \overline{B}_i, B_i). \end{aligned} \tag{8}$$

$$\begin{aligned} C_i &= (A_i \wedge B_i) \vee (A_i \wedge C_{i-1}) \vee (B_i \wedge C_{i-1}) \\ &= ITE(C_{i-1}, A_i \vee B_i, A_i \wedge B_i), \\ A_i \vee B_i &= ITE(A_i, 1, B_i), \\ A_i \wedge B_i &= ITE(A_i, B_i, 0). \end{aligned} \tag{9}$$

$$P_i = A_i \oplus B_i = ITE(A_i, \overline{B}_i, B_i). \tag{10}$$

The ITE operations are computed by Algorithm 1 to get the BDDs for the S_i, C_i, and P_i signals. Assuming that f, g and h are the inputs of an ITE operator, the computational complexity is obtained by $|f| \cdot |g| \cdot |h|$. Note that the size of the BDD for a single variable ($|x_i|$), AND/OR of two variables ($|x_i \wedge y_i|$ and $|x_i \vee y_i|$), and XOR/XNOR of two variables ($|x_i \oplus y_i|$ and $|x_i \odot y_i|$) equals 3, 4, and 5, respectively. As a result, the complexities of computing S_i, C_i, and P_i are as follows:

$$\begin{aligned} &Cpx(S_i) = Cpx(A_i \odot B_i) + Cpx(A_i \oplus B_i) + |C_{i-1}| \cdot |A_i \odot B_i| \cdot |A_i \oplus B_i| \\ &= |A_i| \cdot |B_i| \cdot |\overline{B}_i| + |A_i| \cdot |\overline{B}_i| \cdot |B_i| + |C_{i-1}| \cdot |A_i \odot B_i| \cdot |A_i \oplus B_i| \\ &= 3 \cdot 3 \cdot 3 + 3 \cdot 3 \cdot 3 + |C_{i-1}| \cdot 5 \cdot 5 = 25 \cdot |C_{i-1}| + 54, \\ &Cpx(C_i) = Cpx(A_i \vee B_i) + Cpx(A_i \wedge B_i) + |C_{i-1}| \cdot |A_i \vee B_i| \cdot |A_i \wedge B_i| \\ &= |A_i| \cdot |B_i| + |A_i| \cdot |B_i| + |C_{i-1}| \cdot |A_i \vee B_i| \cdot |A_i \wedge B_i| \\ &= 3 \cdot 3 + 3 \cdot 3 + |C_{i-1}| \cdot 4 \cdot 4 = 16 \cdot |C_{i-1}| + 18, \\ &Cpx(P_i) = |A_i| \cdot |\overline{B}_i| \cdot |B_i| = 3 \cdot 3 \cdot 3 = 27, \\ &Cpx(FA_i) = Cpx(S_i) + Cpx(C_i) + Cpx(P_i) = 41 \cdot |C_{i-1}| + 99, \end{aligned} \tag{11}$$

where the computational complexities depends on the size of the incoming carry BDD to the full-adder. It has been proven in [50] and [12] that the BDD size of the i-th carry bit (C_i) is bounded above by $3i + 6$. Thus, the complexity of computing sum, carry, and propagate outputs in the full-adder chain is as follows:

$$\begin{aligned} Cpx_{[FAs]} &= \sum_{i=0}^{n-1} (41 \cdot |C_{i-1}| + 99) = \sum_{i=0}^{n-1} (41 \cdot (3(i-1) + 6) + 99) \\ &= \sum_{i=0}^{n-1} (123 \cdot i + 222) = O(n^2). \end{aligned} \tag{12}$$

The BP signal in Figure 4 is computed by n consecutive AND gates. Each AND operation can be expressed in the form of an ITE operation as follows:

$$P_{[i-1,0]} \wedge P_i = ITE(P_{[i-1,0]}, P_i, 0). \quad (13)$$

The BDDs for the propagate signals do not have any common variables; thus, after each AND operation the size of the output BDD is equal to the addition of input BDD sizes, i.e. $P_{[i-1,0]}$ and P_i. With respect to the fact that size of $P_{[i,0]}$ equals $3i + 2$, we can obtain the overall complexity of the AND operations:

$$Cpx_{[AND]} = \sum_{i=1}^{n-1} |P_{[i-1,0]}| \cdot |P_i| = \sum_{i=1}^{n-1} (3i-1) \cdot 3 = O(n^2). \quad (14)$$

Finally, the multiplexer in the final stage can be translated into ITE operation as follows:

$$S_n = ITE(BP, C_{-1}, C_{n-1}). \quad (15)$$

Thus, the complexity of computing S_n is calculated as follows:

$$\begin{aligned} Cpx_{[MUX]} &= |BP| \cdot |C_{-1}| \cdot |C_{n-1}| \\ &= (3(n-1)+2) \cdot 3 \cdot (3(n-1)+6) = O(n^2). \end{aligned} \quad (16)$$

The overall complexity of verifying a carry bypass adder is obtained by adding up the symbolic simulation complexity of the full-adders chain, AND gates, and the multiplexer. As a result, we can conclude that the time complexity of BDD-based verification for an n-bit carry bypass adder is quadratically bounded, i.e., $O(n^2)$.

4.2 Approximate Adders

In recent years, approximate circuits have attracted a lot of attention in both academia and industry. These circuits became a viable alternative to the exact circuit in error resilient applications where energy efficiency is crucial [38]. These applications include image and video processing, which are predominantly implemented using adders and multipliers. Thus, researchers come up with several techniques to generate approximate adders or perform approximations on the existing architectures. Now, an important question arises regarding the verification of approximate adders: While the PFV of exact adders is possible, can we also verify approximate adders in polynomial space and time using BDDs?

The authors of [45] proved that the verification process of several state-of-the-art approximate adders is guaranteed to have polynomial time and space complexities using BDDs. They provided polynomial upper bounds for the BDD sizes during the verification process, as well as for the time complexity. The proofs are presented for several handcrafted approximate adders, which divide the adder into sub-adders. Furthermore, they demonstrated the polynomial verifiability of automatically generated approximate adders, where a set of gates from conventional exact adders (e.g., ripple carry adder, conditional sum adder, and carry look-ahead adder) is deleted or changed.

4.3 Multipliers

Formal verification of multipliers is one of the most challenging problems in the verification community. In the last 30 years, several formal methods have been proposed to speed up the process or support the verification of more architectures.

Despite the success of BDD-based techniques in the verification of adders, they rapidly run out of memory when it comes to the verification of multipliers. It was theoretically proven in [5] that the size of BDD for the outputs of a multiplier grows exponentially with respect to the multiplier size and independent of the input variables ordering. The author of [7] came up with an approach to change the input variables and keep the size of the output BDD polynomial. They considered the partial products (i.e., outputs of the partial product generator) as new inputs and constructed the output BDD based on them. Thus, they achieved an output BDD with a polynomial size for an n-bit multiplier. The authors of [31] extended this approach to support the verification of optimized multipliers. Although the size of output BDD is polynomial in these methods, they did not calculate the size of intermediate BDDs. As a result, the space and time complexities of the verification method are unknown.

The word-level verification methods achieved more success in the verification of multipliers. The authors of [6, 25] used *BMDs to verify trivial multiplier architectures. The method starts from the *BMD of the output and takes advantage of backward construction to create the *BMD of the input function. Then, it checks whether the *BMD of the input function matches the *BMD for the multiplication. The authors of [28, 29, 35–37, 44] took advantage of SCA-based methods to attack the hard problem of verifying non-trivial multipliers. They used the SCA (see Section 3.2) as their core verification engine and improved it with several techniques, including heuristics, to overcome the challenges. The experimental evaluations showed that the proposed SCA-based methods can verify various non-trivial multipliers with millions of gates. Despite the practical success of the aforementioned methods, none of them ensured polynomial bounds.

The space and time complexities of a word-level verification method were first studied in [30]. The authors analyzed *BMD-based verification by backward construction when it is applied to the class of Wallace-tree like multipliers. They formally proved polynomial upper bounds on run-time and space requirements with respect to the input word sizes. They showed that the whole verification process is bounded by $O(n^2)$ with respect to space and $O(n^4)$ with respect to time, where n is the number of input bits. However, the proof in this work is only limited to trivial Wallace-tree like multipliers, and it does not support non-trivial multipliers.

The authors of [21] targeted the PFV of non-trivial multipliers. They introduced a hybrid method consisting of BDD- and SCA-based verification to ensure the correctness of non-trivial multipliers in polynomial space and time. The proposed method replaces the final stage adder of the non-trivial multiplier with a ripple carry adder. As a result, the multiplier is converted into a trivial multiplier whose second and third stages are only made of half-adders and full-adders. Then, the original final stage adder and the trivial multiplier are verified using BDDs and SCA, respectively. The authors proved that the space and time complexities of the proposed hybrid method are bounded by $O(n^2)$ and $O(n^4)$. Therefore, the PFV of a non-trivial multiplier becomes possible. The proposed method requires two pre-conditions to ensure the polynomial bounds: 1) The boundaries between multiplier stages and the components in each stage are available, and 2) the multiplier is not optimized; thus, the boundaries are preserved.

4.4 Complex Arithmetic Circuits

In addition to adders and multipliers, there are complex arithmetic circuits containing several operations. The PFV of these circuits is usually more challenging since they consist of several arithmetic units. As an example, an *Arithmetic Logic Unit* (ALU) performs logic (e.g., AND, OR, and XOR) and arithmetic (e.g., addition, subtraction, and multiplication) operations. As another example, a multiply-add circuit ($Z = A \times B + C$) carries out multiplication and addition operations; thus, it requires a multiplier and an adder in its structure.

The PFV of a complex arithmetic circuit was first investigated in [22]. The authors proposed a BDD-based verification method to ensure the correctness of a simple ALU in polynomial time. The ALU was designed to perform simple logic operations as well as addition and subtraction. The authors first demonstrated that applying BDD-based verification to the whole ALU results in a memory blow-up. To overcome this challenge, they took advantage of a divide and conquer method. The proposed method first applies an opcode to activate a certain ALU function, e.g., addition. Then, it extracts the hardware related to the function. Subsequently, it performs a symbolic simulation for each function and ensures its correctness. The authors proved that for an n-bit ALU, the time complexity of verifying each function does not exceed $O(n^2)$. As a result, the PFV of the entire ALU is possible.

The authors of [1] proved the polynomial bounds for the space and time complexities of a general arithmetic circuit that computes a polynomial, e.g., $Z = A^3 + A \times B + C$. They analyzed the space and time complexities using two word-level methods (i.e., SCA- and BMD-based verification) and demonstrated that an arithmetic circuit can be verified in linear space and quadratic time with respect to the size of the circuit function. For instance, the size of the circuit function that computes $Z = A^3 + A \times B + C$ is n^3, since the biggest monomial degree is related to A^3 and the size of all inputs equals n. Thus, the space and time complexities of verifying the arithmetic circuit are bounded by $O(n^3)$ and $O((n^3)^2) = O(n^6)$, respectively. Please note that the polynomial bounds are only valid for the trivial arithmetic circuits in which the second and third stages of multipliers, as well as adders, are only made of half-adders and full-adders.

5 USING ADDITIONAL INFORMATION FOR PFV

In this section, we illustrate the importance of additional information in PFV. This information is usually provided prior to the verification and it plays an important role in guiding us through the verification process. In general, additional information is categorized into two groups: functional properties and hierarchical information. We give insight into these two groups in the following.

5.1 Functional Properties

It is in general impossible to prove the polynomial bounds for the verification of a digital circuit with an arbitrary function and structure. However, if the circuit has some specific functional and structural properties, its PFV can be ensured. Moreover, these properties can be also considered by designers in a circuit or parts of it in order to implement polynomially verifiable designs.

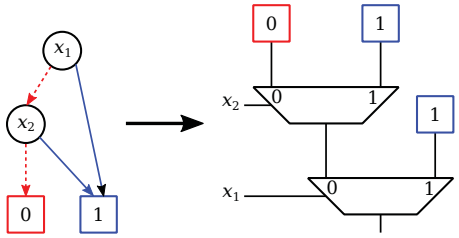


Figure 5: BDD circuit

The author of [13] proved that the PFV of tree-like circuits, i.e., circuits without fanouts, is possible using BDDs if the circuit does not contain any XOR gates. It was proven before in [14] that the output BDDs of a tree-like circuit have polynomial size; however, the size of intermediate BDDs and the time complexity of the construction process were never investigated. The proof was made by analyzing the controlling value (non-controlling value) of each gate and how it can end the recursion in the ITE algorithm (see Algorithm 1). The author also came up with a general theory for PFV: If the BDD size of each internal signal of a circuit has a size polynomial in n, then the complete symbolic simulation of the circuit can be carried out in polynomial space and time. In addition to tree-like circuits, the work of [13] investigated the PFV of BDD circuits, i.e., circuits which are resulted from a BDD by substituting each internal BDD with a MUX (see Figure 5). The author proved that for BDD circuits with MUX using the standard representation the complete symbolic simulation of the circuit can be carried out in polynomial space and time.

The authors of [16] investigated the PFV of totally symmetric functions using BDDs. A Boolean function is called totally symmetric if its output does not depend on the order of the n input variables. Observing the BDD of a symmetric function, it can be concluded that the sub-graph of every node in the BDD again denotes a totally symmetric function [49]. The authors calculated polynomial bounds for each step of the entire formal verification of low depths circuits synthesized by the Ishiura technique [27] to ensure that the verification can be always carried out efficiently. The calculations show that the size of output BDD is always bounded by $O(n^2)$ while considering the intermediate BDD sizes and the construction process, the space and time complexities are bounded by $O(n^6)$ and $O(n^8)$, respectively.

5.2 Hierarchical Information

In many cases, it is impossible to ensure the polynomial bounds for a pure gate-level circuit without any hierarchical information. Complex digital circuits usually consist of several sub-components that cannot be verified with one individual formal method in polynomial space and time. Unfortunately, the boundaries of these sub-components are lost during the synthesis to the gate-level netlist. However, we can overcome the verification challenge by preserving design hierarchy information, including boundaries of sub-components. Thus, PFV becomes possible through the stepwise verification of sub-components and the use of different formal techniques.

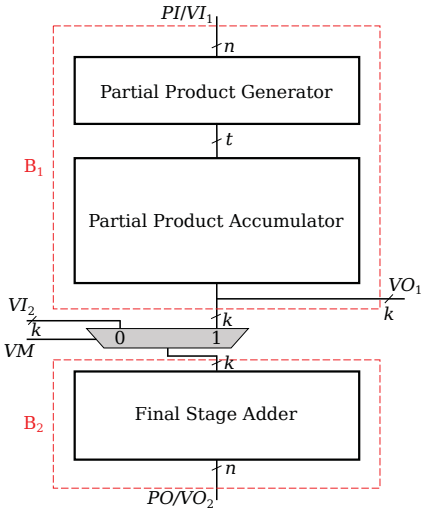


Figure 6: Modified multiplier to preserve hierarchical information

The work of [20] demonstrated how preserving the design hierarchy information and using them during the verification can help to ensure the correctness of complex arithmetic circuits consisting of multipliers and adders. The authors first studied the challenges of verifying a complex arithmetic circuit when no design hierarchy information is available. They showed by experimental evaluations that a memory blow-up happens even for small benchmarks. Then, they proposed a hybrid technique based on SCA and BDDs to verify complex arithmetic circuits in polynomial space and time when the design hierarchy information, including the boundaries between stages and components, is at hand. The upper-bound space and time complexities were calculated for a case study, i.e., $A \times B + C \times D$ arithmetic circuit. The theoretical calculations were confirmed in practice by several experimental evaluations.

The authors of [19] proposed a method to preserve the important hierarchical information by design modification. They modify a complex design in a way that the outputs of components are set as the new outputs of the circuit. These outputs are called *Verification Outputs* (VO) as they are only used during the verification. Furthermore, a multiplexer is added to the inputs of each component. The first input of the multiplexer is connected to the inputs of the component and the second input is connected to new inputs. These new inputs are called *Verification Inputs* (VI) as they are only used during the verification process. When the circuit enters the verification mode, VIs and VOs of each component become available. As a result, each component can be verified in polynomial space and time using the suitable verification engine. The authors applied their design modification method to an integer multiplier to make each stage visible during the verification (see Figure 6). Thus, they achieved polynomial complexity bounds using BDDs and SCA to verify the stages.

6 FUTURE OF PFV

Even though PFV has had significant progress recently still many problems and unexplored areas exist. In this section, a list of possible avenues for the future of PFV is presented. The list is not complete in the sense that all challenges are covered, but many important ones are mentioned. This gives a better understanding of current problems in PFV and shows directions for future research.

Non-Trivial Multipliers: Researchers have shown by experimental evaluations that basic word-level methods (i.e., SCA and WLDDs) run quickly out of memory when it comes to the verification of non-trivial multipliers [34, 35]. However, they never investigated the space and time complexities in theory. Thus, it is necessary to prove the exponential lower-bound complexities of word-level methods for non-trivial multipliers. The proof gives us more insight into the stages or components in a non-trivial multiplier that cause exponential behavior.

Dividers and Other Arithmetic Circuits: In addition to adders and multipliers, the verification of dividers has attracted significant attention, recently. The researchers have taken advantage of SCA to ensure the correctness of restoring and non-restoring dividers [46–48]. Despite this progress, the space and time complexities of these methods are still unknown. Proving the polynomial upper bounds for the existing methods or coming up with PFV approaches for dividers is an important direction for the future of arithmetic circuits PFV. Moreover, the PFV of other integer arithmetic circuits, e.g., square root as well as floating point arithmetic circuits is an untouched area of research.

Functional Properties: In addition to tree-like circuits, BDD circuits, and totally symmetric functions, we are interested in investigating other possible functions and structures whose verification is possible in polynomial space and time. As a result, designers have more options when designing polynomially verifiable circuits.

Design Instructions for PFV: One of the main goals of PFV is to make the verification-centric strategy possible. Thus, designers only design polynomially verifiable circuits. This can happen by introducing some design rules that guarantee PFV. We have to investigate the structures of various circuits, e.g., arithmetic circuits, and their relation with the verification complexity. Then, we can come up with the design instructions that ensure PFV.

7 CONCLUSION

In this paper, we gave a comprehensive overview of PFV and its importance in the design flow. We highlighted the challenges of a design-centric strategy and how they can be overcome by adopting a verification-centric strategy. We illustrated that a verification-centric strategy becomes only possible by proving the polynomial bounds for the verification of various architectures. Then, we presented the recent developments in the PFV of arithmetic circuits, including adders, approximate adders, multipliers, and complex arithmetic circuits. Subsequently, we reviewed the research works that take advantage of additional design information, including functional properties and hierarchical information, to prove the polynomial space and time complexities. We concluded this paper with a brief mention of possible future directions.

ACKNOWLEDGMENTS

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] M. Barhoush, A. Mahzoon, and R. Drechsler. Polynomial word-level verification of arithmetic circuits. In *ACM & IEEE International Conference on Formal Methods and Models for Codesign*, pages 1–9, 2021.

[2] M. Blum and H. Wasserman. Reflections on the Pentium division bug. *IEEE Transactions on Computers*, 45(4):385–393, 1996.

[3] K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient implementation of a BDD package. In *Design Automation Conference*, pages 40–45, 1990.

[4] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

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

[6] R. E. Bryant and Y. A. Chen. Verification of arithmetic circuits with binary moment diagrams. In *Design Automation Conference*, pages 535–541, 1995.

[7] J. Burch. Using BDDs to verify multipliers. In *Design Automation Conference*, pages 408–412, 1991.

[8] G. Cabodi and M. Murciano. BDD-based hardware verification. In M. Bernardo and A. Cimatti, editors, *Formal Methods for Hardware Verification*, pages 78–107, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[9] S. Disch and C. Scholl. Combinational equivalence checking using incremental SAT solving, output ordering, and resets. In *Asia and South Pacific Design Automation Conference*, pages 938–943, 2007.

[10] R. Drechsler. *Advanced Formal Verification*. Kluwer Academic Publishers, 2004.

[11] R. Drechsler. *Formal System Verification: State-of the-Art and Future Trends*. Springer, 2017.

[12] R. Drechsler. PolyAdd: Polynomial formal verification of adder circuits. In *IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, pages 99–104, 2021.

[13] R. Drechsler. Polynomial circuit verification using BDDs. In *International Conference on Electrical, Electronics, Communication, Computer Technologies and Optimization Techniques*, pages 466–483, 2021.

[14] R. Drechsler and B. Becker. *Binary Decision Diagrams – Theory and Implementation*. Kluwer Academic Publishers, 1998.

[15] R. Drechsler and B. Becker. *Binary Decision Diagrams Theory and Implementation*. Springer, 1998.

[16] R. Drechsler and C. Dominik. Edge verification: Ensuring correctness under resource constraints. In *Symposium on Integrated Circuits and System Design*, pages 1–6, 2021.

[17] R. Drechsler and S. Höreth. Manipulation of *BMDs. In *Asia and South Pacific Design Automation Conference*, pages 433–438, 1998.

[18] R. Drechsler and A. Mahzoon. Towards polynomial formal verification of AI-generated arithmetic circuits. In *International Symposium on Devices, Circuits and Systems*, 2021.

[19] R. Drechsler and A. Mahzoon. Design modification for polynomial formal verification. *International Symposium on Electrical, Electronics and Information Engineering*, 2022.

[20] R. Drechsler and A. Mahzoon. Preserving design hierarchy information for polynomial formal verification. *VLSI of System-on-Chip*, 2022.

[21] R. Drechsler, A. Mahzoon, and M. Goli. Towards polynomial formal verification of complex arithmetic circuits. In *IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, pages 1–6, 2022.

[22] R. Drechsler, A. Mahzoon, and L. Weingarten. Polynomial formal verification of arithmetic circuits. In *International Conference on Computational Intelligence and Data Engineering*, pages 457–470, 2021.

[23] R. Drechsler and D. Sieling. Binary decision diagrams in theory and practice. *Int. J. Softw. Tools Technol. Transf.*, 3(2):112–136, 2001.

[24] E. I. Goldberg, M. R. Prasad, and R. K. Brayton. Using SAT for combinational equivalence checking. In *Design, Automation and Test in Europe*, pages 114–121, 2001.

[25] K. Hamaguchi, A. Morita, and S. Yajima. Efficient construction of binary moment diagrams for verifying arithmetic circuits. In *International Conference on Computer-Aided Design*, pages 78–82, 1995.

[26] A. Hu. Formal hardware verification with BDDs: an introduction. In *Pacific Rim Conference on Communications, Computers and Signal Processing*, volume 2, pages 677–68, 1997.

[27] N. Ishiura. Synthesis of multilevel logic circuits from binary decision diagrams (special issue on synthesis and verification of hardware design). *IEICE Transactions on Information and Systems*, 76:1085–1092, 1993.

[28] D. Kaufmann, P. Beame, A. Biere, and J. Nordström. Adding dual variables to algebraic reasoning for gate-level multiplier verification. In *Design, Automation and Test in Europe*, pages 1431–1436, 2022.

[29] D. Kaufmann, A. Biere, and M. Kauers. Verifying large multipliers by combining SAT and computer algebra. In *Formal Methods in Computer-Aided Design*, pages 28–36, 2019.

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

[31] J. Kumar, Y. Miyasaka, A. Srivastava, and M. Fujita. Formal verification of integer multiplier circuits using binary decision diagrams. *IEEE Transactions on Computer Aided Design of Circuits and Systems*, 2022. early access.

[32] A. Mahzoon and R. Drechsler. Late breaking results: Polynomial formal verification of fast adders. In *Design Automation Conference*, pages 1376–1377, 2021.

[33] A. Mahzoon and R. Drechsler. Polynomial formal verification of prefix adders. In *Asian Test Symp.*, pages 85–90, 2021.

[34] A. Mahzoon, D. Große, and R. Drechsler. PolyCleaner: clean your polynomials before backward rewriting to verify million-gate multipliers. In *International Conference on Computer-Aided Design*, pages 129:1–129:8, 2018.

[35] 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. *IEEE Transactions on Computer Aided Design of Circuits and Systems*, pages 1573–1586, 2022.

[36] A. Mahzoon, D. Große, C. Scholl, and R. Drechsler. Towards formal verification of optimized and industrial multipliers. In *Design, Automation and Test in Europe*, pages 544–549, 2020.

[37] A. Mahzoon, D. Große, C. Scholl, A. Konrad, and R. Drechsler. Formal verification of modular multipliers using symbolic computer algebra and boolean satisfiability. In *Design Automation Conference*, 2022.

[38] S. Mittal. A survey of techniques for approximate computing. *ACM Computing Surveys*, 48(4):1–33, 2016.

[39] T. Moto and M. Kaneko. Prefix sequence: Optimization of parallel prefix adders using simulated annealing. In *IEEE International Symposium on Circuits and Systems*, pages 1–5, 2018.

[40] B. Parhami. *Computer arithmetic - algorithms and hardware designs*. Oxford University Press, 2000.

[41] P. Pourbeik, P. S. Kundur, and C. W. Taylor. The anatomy of a power grid blackout-root causes and dynamics of recent major blackouts. *IEEE Power and Energy Magazine*, 4(5):22–29, 2006.

[42] H. Ren, S. Godil, B. Khailany, R. Kirby, H. Liao, S. Nath, J. Raiman, and R. Roy. Optimizing vlsi implementation with reinforcement learning - iccad special session paper. In *International Conference on Computer-Aided Design*, pages 1–6, 2021.

[43] R. Roy, J. Raiman, N. Kant, I. Elkin, R. Kirby, M. Siu, S. Oberman, S. Godil, and B. Catanzaro. PrefixRL: Optimization of parallel prefix circuits using deep reinforcement learning. In *Design Automation Conference*, pages 853–858, 2021.

[44] A. Sayed-Ahmed, D. Große, U. Kühne, M. Soeken, and R. Drechsler. Formal verification of integer multipliers by combining Gröbner basis with logic reduction. In *Design, Automation and Test in Europe*, pages 1048–1053, 2016.

[45] M. Schnieber, S. Fröhlich, and R. Drechsler. Polynomial formal verification of approximate adders. In *EUROMICRO Symposium on Digital System Design*, 2022.

[46] C. Scholl and A. Konrad. Symbolic computer algebra and SAT based information forwarding for fully automatic divider verification. In *Design Automation Conference*, pages 1–6, 2020.

[47] C. Scholl, A. Konrad, A. Mahzoon, D. Große, and R. Drechsler. Verifying dividers using symbolic computer algebra and don't care optimization. In *Design, Automation and Test in Europe*, pages 1110–1115, 2021.

[48] C. Scholl, A. Konrad, A. Mahzoon, D. Große, and R. Drechsler. Divider verification using symbolic computer algebra and delayed don't care optimization. In *Formal Methods in Computer-Aided Design*, 2022.

[49] C. Scholl, D. Moller, P. Molitor, and R. Drechsler. BDD minimization using symmetries. *IEEE Transactions on Computer Aided Design of Circuits and Systems*, 18(2):81–100, 1999.

[50] I. Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM, 2000.

[51] C. Yu, M. Ciesielski, and A. Mishchenko. Fast algebraic rewriting based on and-inverter graphs. *IEEE Transactions on Computer Aided Design of Circuits and Systems*, 37(9):1907–1911, 2017.