Polynomial Formal Verification of Area-efficient and Fast Adders

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

***Abstract*—Despite the recent success of formal verification methods, the computational complexity of most of them is still unknown. It raises serious questions regarding the scalability of the approaches. One of the most successful formal methods to prove the correctness of adders is Binary Decision Diagram (BDD)-based verification. It reports very good results for verification of different adder architectures. However, the computational complexity of BDD-based verification has not been yet fully investigated. In this paper, we calculate the complexity of two adder architectures: ripple carry adder and conditional sum adder. These architectures have the smallest area and delay among adders, respectively. Then, we show that the verification of these architectures is possible in time polynomial in n, where n is the size of the adder (i.e., the number of bits per input). Finally, we confirm our theoretical calculations by experimental results.**

I. Introduction

The importance of arithmetic circuits is rapidly growing due to the demands for complex and extensive computations in modern systems. A wide variety of adders, multipliers, and dividers with different architectures have been proposed to satisfy the community needs for area-efficient, fast, and low-power designs. These architectures are usually complicated; thus, proving the correctness of them before implementation is of the utmost importance to avoid bugs. Several formal verification methods based on *Binary Decision Diagrams* (BDDs) [1], *Binary Moment Diagrams* (BMDs) [2], [3], term-rewriting [4], [5], and *Symbolic Computer Algebra* (SCA) [6], [7], [8], [9] have been proposed to check the correctness of arithmetic circuits. They usually report very good results when it comes to the verification of sophisticated architectures.

Despite the practical success of formal verification methods, the computational complexity of most of them is still unknown. Therefore, two critical problems arise: 1) we cannot show the scalability of the verification method, and 2) we cannot compare the complexity of two verification methods for a particular architecture and choose the best one. A good example of the unknown complexity is the formal verification of integer adders using BDDs. It has been shown in practice that BDDs are very efficient in proving the correctness of adders. However, the computational complexity of many adder architectures has not been yet calculated.

The core idea of the BDD-based verification is based on symbolic simulation. During simulation, an input pattern is applied to a circuit, and the resulting output values are observed to see whether they are the expected values. Symbolic simulation verifies a set of scalar tests in the input space with a single symbolic test. In order to cover all the possible values on each input, symbolic functions are encoded using BDDs. At the end of the simulation, the resulting BDD for each primary output is evaluated. A BDD is a canonical representation; thus, independent of the adder architecture, the outputs' BDDs should be always identical.

To the best of our knowledge, PolyAdd [10] is the only work that focuses on the complexity of adder verification using BDDs. The author proves that the complete formal verification process of some adder architectures can be carried out polynomially. However, PolyAdd does not provide the exact order of the complexity for the adder verification.

Ripple carry adder and conditional sum adder are two important architectures whose verification complexity has not been yet fully investigated. The ripple carry adder has a small area compared to the other adder architectures, making it an excellent option for the small designs. On the other hand, the conditional sum adder has a small delay thanks to its unique structure containing many multiplexers [11]. Therefore, it is the first choice for the designs in which the delay is the most important parameter. In this paper, we calculate the computational complexity of verifying ripple carry and conditional sum adders using BDDs. We also prove that verifying these architectures is possible in time polynomial in n, where n is the size of the adder (i.e., the number of bits per input). Finally, we compare the theoretical calculations with the experimental results to confirm the correctness of the obtained complexities in practice.

The remainder of this work is structured as follows. Section II reviews BDDs and the two adder architectures, i.e., ripple carry and conditional sum adder. The calculations of verification complexities for the two adders are presented in Section III. A caparison between the theoretical complexities and the experimental results is given in Section IV. Finally, Section V concludes the paper.

II. Preliminaries

In this section, first, formal verification using BDDs is reviewed. Then, a brief overview of the ripple carry adder and the conditional sum adder is given.

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
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 $R$
```

A. Binary Decision Diagrams

Definition 1. *A* Binary Decision Diagram *(BDD) is a directed, acyclic graph. Each node of the graph has 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.*

Definition 2. *An* Ordered Binary Decision Diagram *(OBDD) is a BDD, where the variables occur in the same order in each path from the root to a leaf.*

Definition 3. *A* Reduced Ordered Binary Decision Diagram *(ROBDD) is an OBDD that contains a minimum number of nodes for a given variable order. The ROBDD of a Boolean function is always unique.*

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

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

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

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

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

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

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 f, g, h BDDs, the arguments for subsequent calls to the ITE subroutine are the sub-diagrams of f, g and h. The number of sub-diagrams in a BDD is equivalent to the number of nodes. For each of the three arguments, the sub-routine 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 [12].

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. During 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 cover 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.

B. Adder Architectures

We now review the structure of two integer adders, i.e., ripple carry and conditional sum adder.

Ripple Carry Adder: Fig. 1 presents the structure of an n-bit ripple carry adder. In this structure, one *Half-Adder* (HA) and $n-1$ *Full-Adders* (FAs) are cascaded to compute the sum result. The carry will be generated at every FA stage in the circuit. The generated carry output is forwarded to the next FA and applied as a carry input. This process continues up to the last FA stage. The ripple carry adder occupies the least area among the adder architectures. However, it is the slowest adder due to the fact that the carry must propagate through every FA before the addition is complete.

Conditional Sum Adder: Fig. 2 shows the structure of a 4-bit conditional sum adder. In this architecture, two sets of outputs are generated for a given group of k operand bits. Each set contains k sum bits and one outgoing carry. For one of the sets, it is assumed that the eventual incoming carry will be zero, while for the other set it will be one. Once the incoming carry is known, we select the correct set of outputs using multiplexers. The conditional sum adder has a large architecture among the integer adders. On the other hand, it enjoys a small delay thanks to its unique structure containing many multiplexers.

The MUX blocks in Fig. 2 consist of two multiplexers: a multiplexer to select between two k-bit sums and a multiplexer to select between two 1-bit carries. These MUX blocks can be put in different levels based on their inputs. Fig. 3 shows the MUX blocks (boxes) of an 8-bit conditional sum adder in four levels. The number inside each box presents the size of the input sum bits, i.e., k.

III. Computational Complexity

In this section, we calculate the computational complexity of verifying the ripple carry adder and the conditional sum

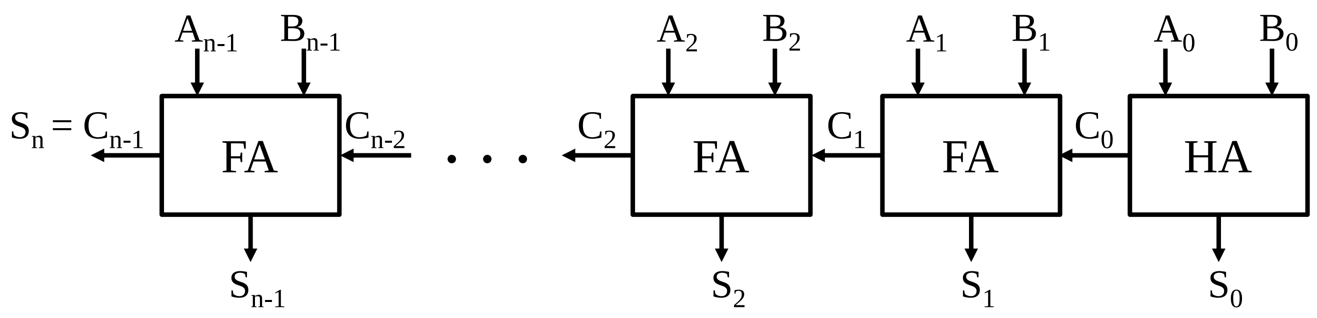


Fig. 1. Ripple carry adder structure

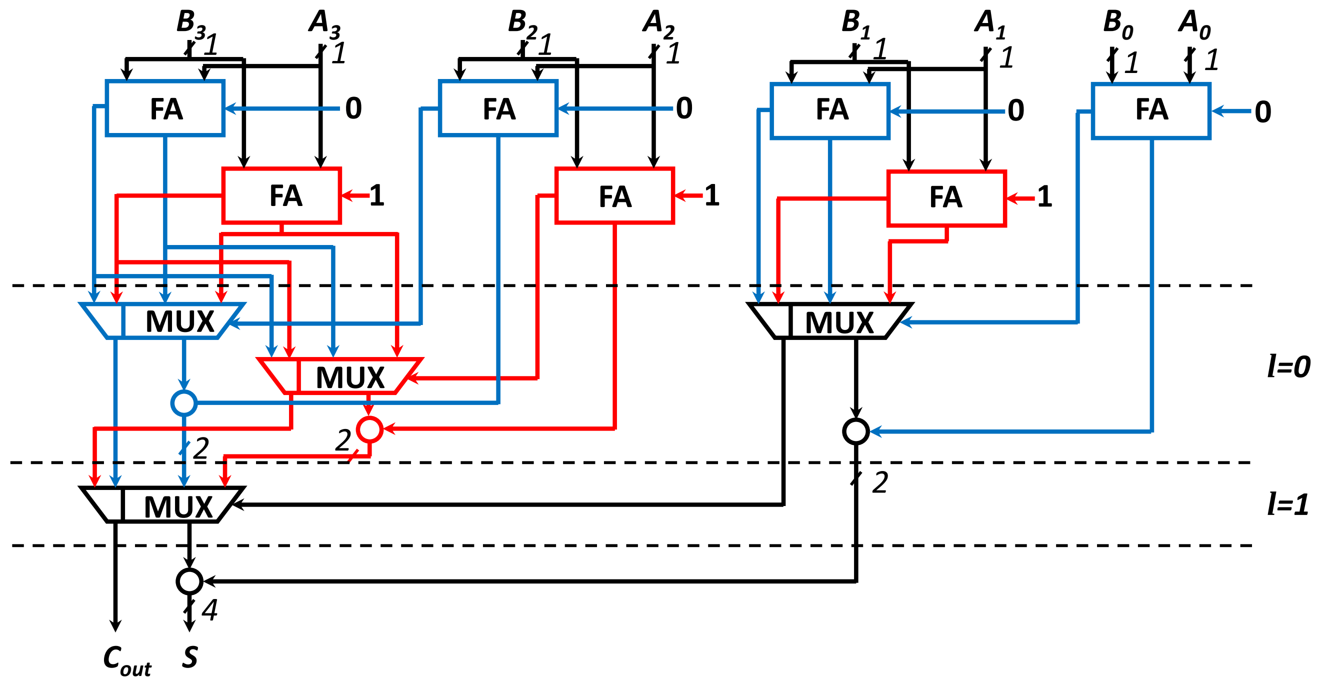


Fig. 2. A 4-bit Conditional sum adder structure

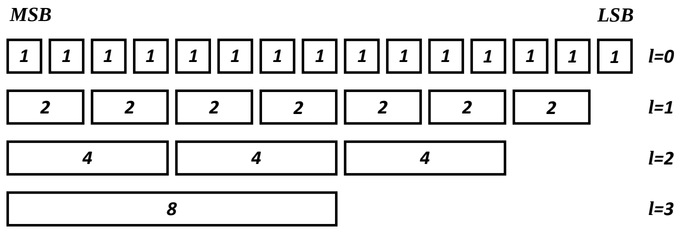


Fig. 3. MUX blocks in an 8-bit Conditional sum adder

adder. We also prove that verification of these two adders is possible in polynomial time.

A. Ripple Carry Adder

In order to obtain the computational complexity of an n-bit ripple carry adder, we first calculate the complexity of a single FA. The sum and carry bits of a FA can be shown by ITE operations:

$$S_i = A_i \oplus B_i \oplus C_{i-1} = ITE(C_{i-1}, A_i \odot B_i, A_i \oplus B_i) = ITE(C_{i-1}, ITE(A_i, B_i, \overline{B}_i), ITE(A_i, \overline{B}_i, B_i)), \quad (4)$$

$$C_i = (A_i \wedge B_i) \vee (A_i \wedge C_{i-1}) \vee (B_i \wedge C_{i-1}) = ITE(C_{i-1}, A_i \vee B_i, A_i \wedge B_i) = ITE(C_{i-1}, ITE(A_i, 1, B_i), ITE(A_i, B_i, 0)) \quad (5)$$

The ITE operations are computed by Algorithm 1 to get the BDDs for the S_i and C_i signals. Assuming that f, g and h are the input arguments of an ITE operator, the computational complexity is computed as $|f| \cdot |g| \cdot |h|$. As a result, the complexity of computing S_i and C_i is as follows:

$$\begin{aligned} \mathit{Complexity}(S_i) &= |C_{i-1}| \cdot |A_i|^2 \cdot |B_i|^2 \cdot |\overline{B}_i|^2 \\ &= 729 \cdot |C_{i-1}| \end{aligned} \quad (6)$$

$$\begin{aligned} \mathit{Complexity}(C_i) &= |C_{i-1}| \cdot |A_i|^2 \cdot |B_i|^2 \\ &= 81 \cdot |C_{i-1}| \end{aligned} \quad (7)$$

where A_i, B_i, and $\overline{B}_i$ BDDs have only one internal node and two terminal nodes; thus, the size of them is the same and equals 3.

It has been proved in [13] that the BDD size of the i^{th} carry bit (C_i) is bounded above by $3(i+1)$. Thus, the overall complexity of verifying a ripple carry adder can be obtained as follows:

$$\begin{aligned} complexity_{[\mathit{RCA}]} &= 810 \cdot \sum_{i=1}^{n-1} |C_{i-1}| = 2430 \cdot \sum_{i=1}^{n-1} i \\ &= 1215n^2 - 1215n \end{aligned} \tag{8}$$

We can conclude that the order of the verification complexity is $O(n^2)$, where n is the number of bits per input (i.e., size of the adder). As a result, proving the correctness of a ripple carry adder has quadratic time complexity.

B. Conditional Sum Adder

An n-bit conditional sum adder is divided into two main stages: 1) $2n-1$ FAs to generate initial sum and carry bits. 2) An array of MUX blocks (see Fig. 3) to select the sum and carry bits. We first calculate the computational complexity of the first stage in verification; then, we focus on the second stage. Finally, we add up the two complexities to obtain the overall verification complexity.

A FA with a '0' input (blue FAs in Fig. 2) is a *Half-Adder* (HA); thus, the output BDDs can be obtained by two ITE operations:

$$\begin{aligned} s_i &= A_i \oplus B_i = ITE(A_i, \overline{B}_i, B_i), \\ c_i &= A_i \wedge B_i = ITE(A_i, B_i, 0) \end{aligned} \tag{9}$$

Similarly, the output BDDs of a FA with an '1' input (red FAs in Fig. 2) can be obtained as follows:

$$\begin{aligned} s'_i &= A_i \odot B_i = ITE(A_i, B_i, \overline{B}_i), \\ c'_i &= A_i \vee B_i = ITE(A_i, 1, B_i) \end{aligned} \tag{10}$$

The computational complexity of these two adders is the same and equals:

$$\begin{aligned} Complexity(s_i) &= Complexity(s'_i) \\ &= |A_i| \cdot |B_i| \cdot |\overline{B}_i| = 3 \cdot 3 \cdot 3 = 27 \\ Complexity(c_i) &= Complexity(c'_i) \\ &= |A_i| \cdot |B_i| = 3 \cdot 3 = 9 \end{aligned} \tag{11}$$

Since there are $2n-1$ FAs in the first stage of the adder, the overall complexity of the first stage is calculated as follows:

$$complexity_{[\mathit{stage1}]} = (2n-1) \cdot (27+9) = 72n - 36 \tag{12}$$

In order to calculate the computational complexity of the second stage, we first need to obtain the complexity of a single MUX block. A MUX block in level l (see Fig. 3) receives two inputs with $2^l + 1$ bits, i.e., $M[2^l : 0]$, and $N[2^l : 0]$. These inputs are the results of adding two 2^l-bit numbers. Then, the MUX selects between these two inputs based on the c signal, which is an output carry resulted from adding two 2^l-bit numbers. Therefore, a MUX block can be translated into $2^l + 1$ ITE operations as follows:

$$\begin{aligned} o_0 &= ITE(c, M_0, N_0), \\ o_1 &= ITE(c, M_1, N_1), \\ &\vdots \\ o_{(2^l)} &= ITE(c, M_{(2^l)}, N_{(2^l)}) \end{aligned} \tag{13}$$

Thus, the overall complexity of a MUX block is calculated as follows:

$$complexity_{[\mathit{MUX}]} = |c| \cdot \sum_{i=0}^{2^l} |M_i| \cdot |N_i| \tag{14}$$

It has been proven in [13] that the BDD size of the i^{th} sum and carry bits are bounded above by $3i+5$ and $3i+3$, respectively. Based on the facts that M_i and N_i are the sum bits, and c is the $(2^l)^{th}$ carry of an addition, we have:

$$complexity_{[\mathit{MUX}]} = (3 \cdot 2^l + 3) \cdot \sum_{i=0}^{2^l} (3i+5)^2 \tag{15}$$

The number of MUXs in each row and the number of rows in a conditional sum adder (see Fig. 3) are calculated as follows:

$$\begin{aligned} \mathit{number\_of\_MUXs\_in\_row} &= 2^{(\log_2 n - l + 1)} - 1, \\ \mathit{number\_of\_rows} &= \log_2 n + 1 \end{aligned} \tag{16}$$

where the equations are exact for all word length being a power of 2 (i.e., $n = 2^m$) [11].

Consequently, the overall computational complexity of the second stage is obtained:

$$\begin{aligned} &complexity_{[\mathit{stage2}]} = \\ &\sum_{l=0}^{\log_2 n} \left((2^{(\log_2 n - l + 1)} - 1) \cdot (3 \cdot 2^l + 3) \cdot \sum_{i=0}^{2^l} (3i+5)^2 \right) = \\ &\frac{384}{35}n^4 + \frac{720}{7}n^3 + 488n^2 + 399n\log_2 n - \frac{795}{7}n - 75\log_2 n + \frac{1601}{35} \end{aligned} \tag{17}$$

Finally, the overall computational complexity of a conditional sum adder is calculated by adding up the complexity of the two stages in Eq. (12) and Eq. (17).

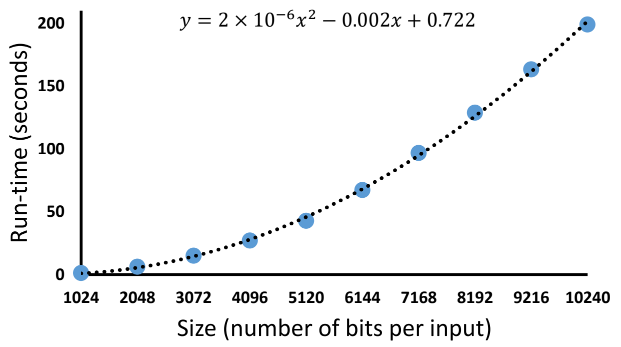
$$\begin{aligned} &complexity_{[\mathit{CSA}]} = \\ &\frac{384}{35}n^4 + \frac{720}{7}n^3 + 488n^2 + 399n\log_2 n - \frac{291}{7}n - 75\log_2 n + \frac{341}{35} \end{aligned} \tag{18}$$

Based on the calculated complexity, we can observe that the order of the verification complexity is $O(n^4)$. Therefore, proving the correctness of a conditional sum adder using BDDs has quartic time complexity.

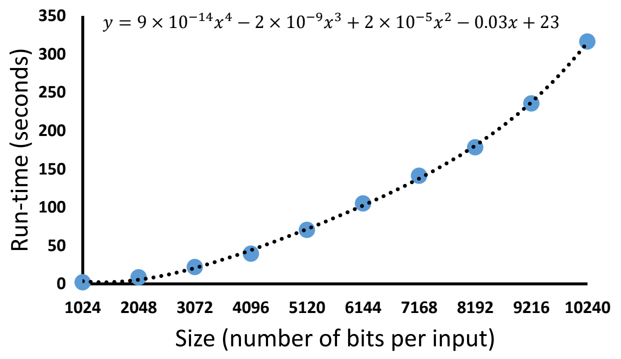
IV. Experimental Results

We have implemented the BDD-based verifier in C++. The tool takes advantage of the symbolic simulation to obtain the BDDs for the primary outputs. Then, the BDDs are evaluated to see whether they match the BDDs for an adder. In order to handle the BDD operations, we used the CUDD library [14]. The benchmarks for the three prefix adders are generated using GenMul [15]. All experiments are performed on an Intel(R) Core(TM) i7-8565U with 1.80 GHz and 24 GByte of main memory.

After calculating the verification complexity bound for a conditional sum adder, we check the correctness of the



(a) Ripple carry adder



(b) Conditional sum adder

Fig. 4. Run-time graphs of the Adders

theoretical results in practice. Fig. 4(a) (Fig. 4(b)) presents the run-times of verifying ripple carry adders (conditional sum adders) with different sizes. We fit a curve (dash lines) to the points with an acceptable error and evaluate the curve function. We can fit a curve with the order of 2 to the verification run-times of ripple carry adders in Fig. 4(a). On the other hand, a curve with the order of 4 can be fitted to the verification run-times of conditional sum adders in Fig. 4(b). It confirms that verification of the ripple carry adder and the conditional sum adder has quadratic and quartic time complexity, respectively.

V. Conclusion

In this paper, we calculated the computational complexity of verifying one of the smallest adders (i.e., ripple carry adder) and one of the fastest adders (i.e., conditional sum adder) using the BDD-based method. Based on the calculations, we proved that verification of the ripple carry adder and the conditional sum adder has quadratic and quartic time complexity, respectively. We also confirmed the correctness of the complexity bounds obtained in our theoretical calculations by experimental results.

Our research is the first step for the polynomial formal verification of arithmetic circuits. In the future, we plan to calculate the complexity bounds for available formal verification methods when they are applied to different architectures, particularly arithmetic circuits.

Acknowledgment

This work was supported by the German Research Foundation (DFG) within the Reinhart Koselleck Project *PolyVer: Polynomial Verification of Electronic Circuits* (DR 287/36-1).

References

[1] R. E. Bryant, “Binary decision diagrams and beyond: enabling technologies for formal verification,” in *International Conference on Computer-Aided Design*, 1995, pp. 236–243.

[2] R. Drechsler, B. Becker, and S. Ruppertz, “The K*BMD: A verification data structure,” *IEEE Design & Test of Computers*, vol. 14, no. 2, pp. 51–59, 1997.

[3] M. Keim, R. Drechsler, B. Becker, M. Martin, and P. Molitor, “Polynomial formal verification of multipliers,” *Formal Methods in Sys. Design*, vol. 22, no. 1, pp. 39–58, 2003.

[4] S. Vasudevan, V. Viswanath, R. W. Sumners, and J. A. Abraham, “Automatic verification of arithmetic circuits in RTL using stepwise refinement of term rewriting systems,” *IEEE Trans. on Comp.*, vol. 56, no. 10, pp. 1401–1414, 2007.

[5] M. Temel, A. Slobodová, and W. A. Hunt, “Automated and scalable verification of integer multipliers,” in *Computer Aided Verification*, 2020, pp. 485–507.

[6] 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*, 2018, pp. 129:1–129:8.

[7] A. Mahzoon, D. Große, and R. Drechsler, “RevSCA: Using reverse engineering to bring light into backward rewriting for big and dirty multipliers,” in *Design Automation Conf.*, 2019, pp. 185:1–185:6.

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

[9] D. Kaufmann, A. Biere, and M. Kauers, “Verifying large multipliers by combining SAT and computer algebra,” in *Int’l Conf. on Formal Methods in CAD*, 2019, in press.

[10] R. Drechsler, “PolyAdd: Polynomial formal verification of adder circuits,” in *IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, 2021, pp. 99–104.

[11] R. Zimmermann, “Binary adder architectures for cell-based VLSI and their synthesis,” Ph.D. dissertation, Swiss Federal Institute of Technology, 1997.

[12] K. S. Brace, R. L. Rudell, and R. E. Bryant, “Efficient implementation of a BDD package,” in *Design Automation Conf.*, 1990, pp. 40–45.

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

[14] F. Somenzi, “CUDD: CU decision diagram package release 2.7.0,” available at https://github.com/ivmai/cudd, 2018.

[15] A. Mahzoon, D. Große, and R. Drechsler, “GenMul: Generating architecturally complex multipliers to challenge formal verification tools,” in *Int’l Workshop on Logic Synth.*, 2019.