Transformation-Aided Verification of MAC Designs using Symbolic Computer Algebra

Lennart Weingarten[𝕌], Kamalika Datta[△,𝕌], Rolf Drechsler[△,𝕌] *Fellow, IEEE*
[△]*German Research Centre for Artificial Intelligence (DFKI), Bremen, Germany*
[𝕌]*Institute of Computer Science, University of Bremen, Germany*
{kdatta, len_wei, drechsler}@uni-bremen.de

***Abstract*—The increasing complexity of modern digital circuits requires robust verification to ensure reliability and prevent costly failures. Among various formal verification methods, *Symbolic Computer Algebra* (SCA) offers a powerful approach by representing circuits using polynomials. However, a significant challenge in SCA verification is the exponential term expansion during substitution, which drastically increases verification time. This paper addresses this challenge by investigating the impact of circuit transformations on SCA verification efficiency. We propose a transformation-aided verification process, showcasing its effectiveness through a case study on *Multiply-and-Accumulate* MAC-based designs. Specifically, we examine the transformation of NAND/NOR-based designs and demonstrate its substantial impact on verification time for certain MAC circuits. Experimental results reveal interesting findings, notably a many-fold performance gain for some benchmarks.**

***Index Terms*—Formal Verification, Symbolic Computer Algebra (SCA), Multiply and Accumulate (MAC)**

I. Introduction

The increasing complexity of modern digital circuits necessitates rigorous verification to ensure functionality and reliability. As designs grow in scale and complexity, the potential for errors increases, leading to costly failures. Therefore, robust verification methodologies are crucial to guarantee the correct operation of these vital systems. For verifying complex functionalities, particularly in arithmetic circuits, a range of formal proof engines are employed, including *Binary Decision Diagrams* (BDDs) [1], *Satisfiability Solvers* (SAT) [2], *Answer Set Programming* (ASP) [3], and *Symbolic Computer Algebra* (SCA) [4].

SCA-based verification involves representing a digital circuit's functionality and structure using polynomials [5], [6]. The process starts by creating a *Specification Polynomial* (SP) that mathematically describes the desired behavior of the circuit in terms of its inputs and outputs. Then, *Gate Polynomials* (GPs) or *Node Polynomials* (NPs) are constructed to represent the behavior of individual logic gates or nodes within the circuit. Verification is achieved through a backward rewriting procedure. This involves systematically substituting the GPs or NPs into the SP, working backward from the circuit's outputs towards its inputs. Finally, the resulting SP is evaluated. If the evaluation yields a zero remainder, it indicates that the circuit's implementation matches its specification, implying a bug-free design. Conversely, a non-zero remainder signifies a discrepancy, revealing a fault or error in the circuit's construction.

A key limitation of SCA-based verification is the term expansion that occurs during the substitution process [7]. While optimization techniques exist to improve the *Circuit Under Verification* (CUV), transformations at the circuit level offer the potential for even greater impact on the overall verification efficiency [8]. The primary challenge is that the number of terms can grow exponentially during the substitution process, leading to increased verification time. Numerous techniques have been explored in the literature to address this issue [9], [10]. A critical aspect is how to mitigate this term explosion. Among other factors we have identified three crucial factors that influence the explosion during substitution, whether using GP or NP rules: (i) the substitution order, (ii) the rule application frequency and (iii) circuit transformation.

Choosing an optimal substitution order is a known optimization problem, and various heuristics have been developed to address it. While rule application frequency is circuit-dependent, circuit transformations can significantly alter the overall frequency. In this work, we have observed that transforming a circuit to a NAND/NOR-based design dramatically impacts verification time for specific circuit classes.

With the growing popularity of AI/ML *Multiply-Accumulate* (MAC) is one of the core designs in such applications, hence the verification of such units are of immense importance. Recent work [11] demonstrates verification of MAC circuits up to 256 bits for structurally simple designs. However, for complex MAC circuits, their method requires around 24 minutes to verify 8-bit circuits, whereas our transformed circuit can be verified within 0.03 seconds, which amounts to a performance increase of 44338 times.

Although NAND/NOR-based transformation yield superior performance for a class of circuits, given the influence of both substitution order and rule application frequency, further research is necessary to determine the most effective circuit transformation for SCA-based verification. Experiments were conducted on behavioral *Multiply-and-Accumulate* (MAC) circuits with complex structures to demonstrate the impact of circuit transformation on SCA-based verification.

The remainder of this paper is structured as follows: Section II introduces the concept of SCA-based verification, Section III elaborates on proposed transformation aided verification process, Section IV presents the experimental results with case study on MAC based designs, and finally Section V concludes the paper.

II. Background

A. SCA-based Verification

SCA verification proceeds by defining the *Specification Polynomial* (SP), followed by *Gate Polynomial* (GP) or *Node Polynomial* (NP) identification, and concluding with backward rewriting process. This verification technique can be applied to circuits represented as either gate-level netlists or *And-Inverter Graphs* (AIGs). Initially, the *Specification Polynomial* (SP) is created, mathematically defining the intended circuit behavior. Subsequently, for each gate or node type encountered, a set of *Gate Polynomials* (GPs) or *Node Polynomials* (NPs) is established. The core of the verification lies in the backward rewriting phase. Here, the SP is progressively modified by replacing gate or node outputs with their respective GPs or NPs, following the circuit's reverse topological order. Upon reaching the circuit's inputs, the final SP is evaluated. A zero remainder from this evaluation confirms the circuit's correctness, while a non-zero remainder indicates the presence of a fault.

For an AIG the following cases for a primary output edge of an AND-gate node can occur, they are summarized as gate polynomial rules:

R1 $z = a$
a primary output edge can act as a buffer or simple wire

R2 $z = 1 - a$
a primary output edge can act as a complemented edge

R3 $z = ab$
a simple AND-gate without any complemented inputs

R4 is split into two sub-rules

R4a $z = b - ab$ (a complemented)
R4b $z = a - ab$ (b complemented)

an AND-gate with one complemented input edge

R5 $z = 1 - a - b + ab$
an AND-gate with both complemented input edges

B. Full Adder Example

Fig. 1 shows the AIG of a full adder. For a full adder, the SP for input a, b and c_{in} and output sum and $carry$ is given below:

$$SP_{FA} = 2carry + sum - a - b - c_{in} = 0$$

To verify the full adder (Fig. 1), the backward rewriting process begins by processing each output node (sum and carry) individually. Node substitution is performed based on the rules provided above. The step-by-step process is shown in Fig. 2. There are a total of 11 substitutions, of which 9 are for the AIG nodes and 2 are for the primary outputs. Of these 11 substitutions, rule R2 is used twice, rule R3 is used four times, rules R4a and R4b are each used once, and rule R5 is used three times. After all the substitutions, the final remainder is 0, indicating that the circuit is bug-free.

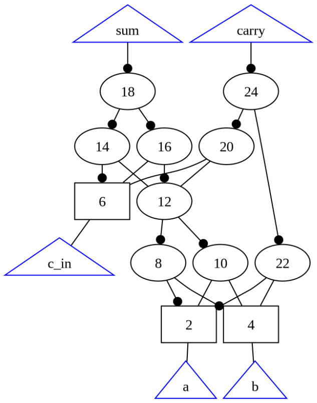

Fig. 1: An AIG of a full adder. The blue triangle nodes represent primary inputs and outputs, the square nodes map a primary input to a node ID, and all the oval nodes model AND-gates and the small black circles realize complemented edges.

$$\begin{aligned}
&SP := 2carry + sum - a - b - c_{in} = 0 \\
\xrightarrow[\text{R2}]{carry}\ &SP_1 = 2(1 - n24) + sum - n2 - n4 - n6 = 0 \\
\xrightarrow[\text{R2}]{sum}\ &SP_2 = 2(1 - n24) + (1 - n18) - n2 - n4 - n6 = 0 \\
&\quad = 3 - 2n24 - n18 - n2 - n4 - n6 = 0 \\
\xrightarrow[\text{R3}]{n24}\ &SP_3 = -2n20n22 - n2 - n4 - n6 - n18 + 1 + 2n20 + 2n22 \\
&\quad \dots \\
&\xrightarrow[\text{R3}]{n22} SP_4 \xrightarrow[\text{R3}]{n20} SP_5 \xrightarrow[\text{R5}]{n18} SP_6 \xrightarrow[\text{R4a}]{n16} SP_7 \xrightarrow[\text{R4b}]{n14} SP_8 \xrightarrow[\text{R5}]{n12} SP_9 \\
&\quad \dots \\
\xrightarrow[\text{R3}]{n10}\ &SP_{10} = -n2 - n4 - n8 + 1 + n2n4 + n2n4n8 \\
\xrightarrow[\text{R5}]{n8}\ &SP_F = -n2 + -n4 - 1(n8) + 1 + n2n4 + n2n4(n8) \\
&\quad = -n2 + -n4 - 1 + n2 + n4 - n2n4 + 1 + n2n4 \\
&\quad\quad + n2n4 - n2n4 - n2n4 + n2n4 \\
&\quad = (-1 + 1) + (-n2 + n2) + (-n4 + n4) \\
&\quad\quad + (-3n2n4 + 3n2n4) \\
&\quad = 0
\end{aligned}$$

Fig. 2: Backward rewriting steps for the full adder AIG

C. Related Works

Over the past decade, SCA has emerged as a powerful tool for verifying complex arithmetic circuits, including multipliers, dividers, MAC units, and DP units, as evidenced by a substantial research [4]–[6], [9], [11]–[19]. Most approaches utilize SCA directly on the AIG representation [10], [11], [19], [20].

Researchers have explored various strategies to mitigate the polynomial explosion problem inherent in SCA. These include techniques for removing redundant terms and applying algebraic simplifications [7], as well as methods that decompose the verification task into manageable sub-problems by analyzing the circuit column-wise [16]. In [10], reverse engineering is employed to extract circuit structure and algebraic relationships, proving especially beneficial for optimized designs, though this process can be computationally intensive. Variable ordering and phase selection are leveraged in [9] to optimize SCA-based verification, offering heuristics that minimize polynomial growth and enhance performance compared to existing methods. Hybrid approaches combining SCA with SAT solvers have also been investigated for multiplier verification [6], [17]. Moreover, the computational and memory limitations of Gröbner basis rewriting are addressed in [18] through parallelization and memory optimization. Recent studies have extended SCA techniques to verify MAC and DP units [11], [19], with [11] specifically proposing an adaptation of RevSCA [10] for MAC designs.

In all above mentioned techniques, as the backward rewriting process progresses, intermediate polynomials become increasingly complex, leading to excessive memory usage and computational time. Specifically for optimized circuits where the circuit structure is not known, the problem becomes extremely challenging, resulting in high verification overhead. In this work, we apply various transformations before performing the verification process. This leads to immense benefits in terms of verification time and memory. The next section discusses the specific transformations and their effect.

III. Transformation Aided Verification for SCA

This section details the transformations that enhance SCA-based verification. The general SCA verification process was outlined in the preceding section. Here, we demonstrate how various transformations can influence verification outcomes. For our case study, we utilized the behavioral model of a *Multiply-Accumulate* (MAC) unit, as defined in Listing 1. This model represents the MAC at a high level of abstraction, with implementation details generated by a synthesis tool. The behavioral model's flexibility allows us to easily generate MAC units of varying bit-widths by simply adjusting a single parameter, *n*.

Listing 1: Behavioral definition of MAC in Verilog

```
module MAC (parameter n = 8) (A,B,S,R);
    input  [n-1:0] A, B;     // multiply
    input  [(2*n)-1:0] S;    // add
    output [2*n:0] R;

    assign R = (A*B) + S;
endmodule
```

The complete verification process is illustrated in Fig. 3. Starting with a Verilog behavioral model of a MAC unit, the code is first synthesized into an AIG. This synthesis involves two key steps: first, using Yosys [21], we perform the initial synthesis, apply basic optimizations, and output the result as a *Berkeley Logic Interchange Format* (BLIF) file. Optionally, the commands abc -g NAND or abc -g NOR can be executed to implement the circuit exclusively with NAND or NOR gates, respectively. Second, the BLIF file is converted into an AIG using ABC [22]. This conversion involves reading the BLIF file, performing structural hashing and refactoring, and then writing the resulting AIG to a file. We opted to generate the AIG with ABC rather than directly within Yosys, as we found it yielded superior synthesized graph results. Finally, the generated AIG is verified using an extension of RevSCA [10], [11].

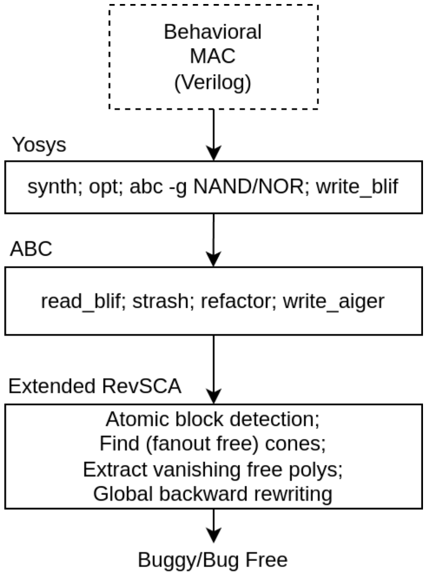
Behavioral
MAC
(Verilog)
Yosys
synth; opt; abc -g NAND/NOR; write_blif
ABC
read_blif; strash; refactor; write_aiger
Extended RevSCA
Atomic block detection;
Find (fanout free) cones;
Extract vanishing free polys;
Global backward rewriting
Buggy/Bug Free

Fig. 3: Overview of the verification process

Using this process, we can create the standard AIG from a behavioral MAC model, as well as two modified versions where the graph is first expressed using only NOT and NAND or NOT and NOR gates. The next subsection illustrates how the graph's structure changes when the circuit is transformed, for example, to a NAND-based representation.

A. NAND/NOR Transformation for a 2-bit MAC

This section analyzes the AIG graph after NAND/NOR transformation. Fig. 4 and Fig. 5 show the graph structure with and without NAND/NOR transformation. We analyze the following parameters [10] for the verification process:

- **Atomic Block Detection**: To enhance verification efficiency, specific AIG nodes are organized into *Atomic Blocks*(ABs), enabling word-level substitution. Half adders and full adders serve as common examples of these ABs.
- **Vanishing Monomials**: During the substitution of SP variables with AIG node polynomials, the monomial count typically rises. However, subsequent substitutions often result in the cancellation of these monomials. These are termed as *Vanishing Monomials* (VMs).
- **Converging Cones**: The cones starting from half-adder outputs and ending in a converging node are called *Converging Node Cones* (CNC). They are known to be the origin of VM. CNC are comprised of extra nodes, which are not part of any atomic block: half-adder or full-adder. Having no CNC in a design or the local removal of

VM in CNCs results in a vanishing-free global backward rewriting process. Which should ideally result in a blow up free verification process.

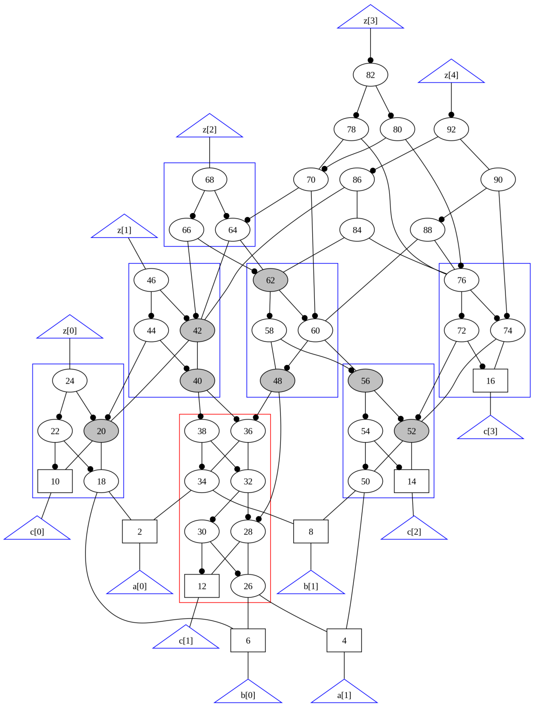

Fig. 4: 2-bit MAC representation (AND)

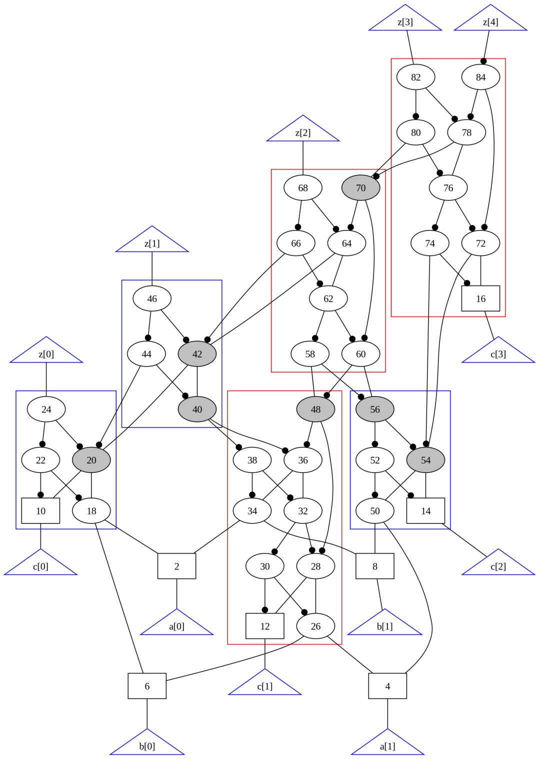

Fig. 5: 2-bit MAC representation (NAND/NOR)

In both Fig. 4 and Fig. 5, the AIG nodes are visually grouped based on their constituent atomic blocks. Nodes belonging to half-adder atomic blocks are enclosed in blue frames, while those within full-adder atomic blocks are framed in red. In particular, certain nodes participate in multiple atomic blocks, often serving as primary inputs or outputs for these blocks. To highlight this shared functionality, these nodes are marked in gray.

Examining the standard AIG synthesized design shown in Fig. 4, we observe a composition of six half-adder atomic blocks and one full-adder atomic block, along with several nodes that do not fall within any defined atomic block. In contrast, the NAND/NOR-transformed AIG in Fig. 5 presents a significantly transformed structure. Here, the number of half-adder atomic blocks is reduced to three, while the number of full-adder atomic blocks increases to three. Furthermore, a key distinction is that every node in the NAND/NOR-transformed AIG is integrated into an atomic block, eliminating any extra nodes. Because there are no extra nodes, vanishing monomials are eliminated, which also minimizes intermediate blowup. This indicates that the NAND transformation has restructured the graph to exclusively consist of atomic blocks, potentially impacting the verification process.

While our observations indicate that NAND or NOR transformations can significantly alter circuit structure, potentially influencing verification outcomes, it is crucial to acknowledge that this effect is not universally applicable across all circuit types. Our experimental findings suggest that the impact of these transformations is highly circuit-specific and, therefore, cannot be generalized. In essence, the effectiveness of NAND or NOR transformations in optimizing verification is dependent upon the inherent characteristics of the circuit being analyzed. Therefore, careful consideration of the circuit's architecture and functionality is essential before applying such transformations, as they may not consistently yield improvements.

IV. Experimental Results

The SCA-based verification approach is implemented using C++. All experiments were performed on a AMD Ryzen 7 PRO 4750U with 40GB main memory and 8GB of Swap. We have used the extended RevSCA [11] tool for our experimentation.

A. *Transformation Comparison*

Table I provides a comparative analysis of verification results for *Multiply-Accumulate* (MAC) designs, ranging from 2 to 12 bits, across three distinct AIG representations: the standard AND-based design, and two transformed variants, NAND-based and NOR-based. The MAC designs are generated from the behavioral verilog and hence there is no structural details known unlike other MAC designs as mentioned in [11]. The columns show, in order from left to right: MAC bit size, *Atomic Blocks* (AB), *Vanishing Monomials* (VM), *Maximum Polynomial* (MP), and *Verification Time* (VT), with results for AND, NAND, and NOR given sequentially for each

MAC bit size. The last two columns present the *Verification Time Improvement* (VTI) and *Maximum Polynomial Improvement* (MPI). Certain experiments were terminated due to a *Memory Time Out* (MTO), and these instances are indicated as MTO in the table. VTI and MTO present the ratio between the old and the new value, for verification time and maximum polynomial respectively. The table is designed to facilitate a clear understanding of the impact of these transformations on verification metrics.

The NAND and NOR transformations consistently reduce the number of VM to zero across all successfully verified cases. These transformations also generally reduce the MP size and VT compared to the standard AND representation. This demonstrates a significant improvement in verification efficiency through these transformations. For MAC bit sizes (2-8 bits), NAND and NOR transformations result in substantial reductions in VT, with some cases showing a significant improvement as shown on the last two columns. For example, at MAC bit size 2, the verification time improved by a factor of 1.33. As shown in the MPI column, the maximum polynomial size is significantly reduced by a factor of 1.46. The standard AND representation encounters MTO for MAC bit sizes (10-12 bits), indicating its inability to handle the increasing complexity. The NAND and NOR representations, while also encountering MTOs for the largest bit sizes (9 and 12), but can successfully verify MAC circuits (10-11 bits) compared to the AND representation. These results suggest that NAND/NOR transformations do not consistently yield performance improvements across all circuit types. Consequently, a transformation does not guarantee an advantage over the standard AND representation in all cases. But in some cases (like 5-8,10 and 11 bits), it improves significantly, by a maximum factor of 44338 for VTI and 17320 for MPI (for 8-bit MAC).

B. Rule Frequency Analysis

Table II presents an analysis of AIG node polynomial rules for different bit sizes (2 to 12 bits) across three circuit representations: AND, NAND, and NOR. The table aims to quantify the complexity of *Node Polynomial* (NP) rules associated with each representation, providing insights into their impact on verification processes. The first column indicates the bit size. The next 12 columns represent the various NP rules (R3, R4 and R4) and the number of negated edges (NE) for all three representations (AND, NAND, and NOR).

The NAND and NOR representations show very similar counts for R3, R4, R5, and NE across all bit sizes. This suggests that the transformation either NAND or NOR results in a similar level of polynomial complexity. In most cases, the AND representation has higher counts for R3, R4, and NE compared to NAND and NOR. This indicates that the NAND and NOR transformations tend to simplify the polynomial representation of AIG nodes, potentially leading to more efficient verification. The counts for R5 are consistently higher than R3 and R4 across all bit sizes and representations. This suggests that R5 rules, are more prevalent in the polynomial representation of AIG nodes. Analyzing the distribution of R3, R4, and R5 rules provide insights into the specific characteristics of polynomial representations and their impact on verification performance.

While the frequency of rule application, as given by the data in Table I, influences verification performance, it is crucial to acknowledge that NAND and NOR transformations do not entirely eliminate the risk of verification failures due to *Memory Time Out* (MTO). Specifically, the 9-bit and 12-bit MAC designs, despite undergoing these transformations, still resulted in MTO. This observation emphasize that, although circuit transformations can significantly impact verification efficiency, they do not guarantee complete mitigation of the intermediate polynomial blowup problem. The inherent complexity of larger circuits can still lead to an unmanageable explosion of intermediate terms, ultimately resulting in time outs. Therefore, a more in-depth, circuit-specific analysis is necessary to draw definitive conclusions about the efficacy and limitations of these transformations in various scenarios.

C. Case study for 8-bit MAC

To demonstrate the effectiveness of transformations aided verification, we present an example using the behavioral 8-bit MAC verification, comparing results with and without transformation. Fig. 6 illustrates the polynomial size over time for both the standard AND and the NAND transformation. The Y-axis represents the *Specification Polynomial* (SP) size on a logarithmic scale, while the X-axis indicates the substitution steps.

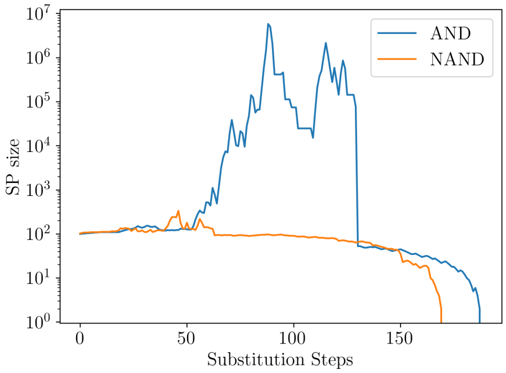

Fig. 6: Polynomial size over the substitution steps for an 8-bit MAC with standard AND and NAND transformation

From the figure, it can be observed that the SP size of the standard AND circuit experiences a rapid increase, or ”blowup,” after approximately one-third of the substitution steps, followed by a rapid decrease after two-thirds. In contrast, the NAND-transformed MAC exhibits a minor peak shortly before the 50th substitution step, but avoids any significant intermediate blowup. Furthermore, the NAND-transformed MAC requires fewer substitution steps for verification compared to the AND circuit. A numerical comparison of AND and NAND is presented in Table III. Table III clearly

TABLE I: Verification Result

	AND [11]				NAND				NOR				Ratio	
MAC	AB	VM	MP	VT	AB	VM	MP	VT	AB	VM	MP	VT	MPI	VTI
2	7	4	19	0.0012	6	0	13	0.0009	6	0	13	0.0009	1.46	1.33
3	13	4	29	0.0016	12	0	23	0.0016	12	0	23	0.0014	1.26	1.14
4	22	18	49	0.0025	20	0	34	0.0022	20	0	34	0.0021	1.44	1.14
5	37	21	4297	0.1150	29	0	50	0.0061	29	0	50	0.0071	85.94	18.85
6	51	24	418960	176.2480	43	0	90	0.0093	43	0	90	0.0056	4655.11	18951.40
7	68	31	732122	274.2510	61	0	146	0.0090	61	0	146	0.0110	5014.53	30472.33
8	89	80	5819510	1445.4100	77	0	336	0.0326	77	0	336	0.0332	17319.97	44337.73
9	112	87	5888912	2472.2900	MTO	MTO	MTO	MTO	MTO	MTO	MTO	MTO	-	-
10	MTO	MTO	MTO	MTO	125	0	1261	0.1277	125	0	1261	0.1335	-	-
11	MTO	MTO	MTO	MTO	147	0	50908	15.6242	147	0	50908	12.9876	-	-
12	MTO	MTO	MTO	MTO	MTO	MTO	MTO	MTO	MTO	MTO	MTO	MTO	-	-

TABLE II: AIG Node Polynomial Rule Analysis

	AND				NAND				NOR			
Bits	R3	R4	R5	NE	R3	R4	R5	NE	R3	R4	R5	NE
2	15	3	20	45	11	4	19	43	11	4	19	43
3	25	14	42	100	22	14	41	97	22	14	41	97
4	46	28	73	177	37	28	71	171	37	28	71	171
5	73	44	120	288	56	46	109	265	56	46	109	265
6	96	68	170	412	85	69	160	391	84	71	159	390
7	128	94	229	557	114	93	223	540	114	93	223	540
8	168	128	300	734	152	119	295	714	148	127	291	710
9	211	161	379	926	183	159	363	886	183	159	363	886
10	256	201	462	1133	232	189	463	1120	231	191	462	1119
11	306	243	557	1366	276	232	551	1339	275	234	550	1338
12	361	293	658	1619	333	273	658	1597	328	283	653	1592

TABLE III: 8-bit MAC example

	AND	NAND
Sub. steps	189	171
VT (s)	1445.41	0.0326
#Node	613	583
#HAs	37	24
#FAs	52	53
#Atomic	89	77
#Cone	100	94
#VM	80	0
#MaxPoly	5819510	336

demonstrates the advantage of the NAND-transformed MAC over the standard AND MAC by comparing key metrics. These metrics include: substitution steps (*Sub.steps*), verification time (VT), the number of nodes in the AIG (#Node), the number of atomic blocks (#HA, #FA, #Atomic), the number of cones (#Cones), the number of vanishing monomials (#VM), and the maximum polynomial size (#MaxPoly) during verification. VT is drastically reduced from 1445.41 seconds (approximately 24 minutes) to just a fraction of a second. The number of vanishing monomials is eliminated, dropping from 80 to zero. Beyond the significant reduction in verification time, the most substantial impact of the NAND transformation is observed in the maximum polynomial size, which decreases from 5.8 million monomials to a mere 336. This transformation contributes significantly to the NAND-transformed MAC being more memory-efficient than its AND counterpart.

V. Conclusion

SCA is a powerful formal verification method for complex digital circuits, but its efficiency is affected by exponential term expansion during substitution. This paper addresses this by investigating the impact of circuit transformations on SCA verification time. A transformation-aided process, converting verilog to NAND/NOR-based designs, is proposed and evaluated on behavioral MAC circuits. Experimental results show that this transformation has the potential to drastically reduce the maximum polynomial size and verification time, while also eliminating vanishing monomials, leading to a many-fold speedup for certain benchmarks (maximum improvement factor of 44338 for VTI and 17320 for MPI (for 8-bit MAC)). Although circuit transformations can significantly impact verification efficiency, they do not guarantee complete mitigation of the intermediate polynomial blowup problem for certain cases. The inherent complexity of complex circuits can still lead to an unmanageable explosion of intermediate terms, ultimately resulting in time outs. This indicates the need for further research to identify optimal transformation strategies for different circuit types and verification scenarios.

Acknowledgment

This work was supported in part by DFG within the Reinhart Koselleck Project PolyVer (DR 287/36-1) and partly by the German Federal Ministry of Research, Technology and Space (BMFTR) within the ECXL project under grant no. 01IW22002.

REFERENCES

[1] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *TC*, vol. 35, no. 8, pp. 677–691, 1986.

[2] M. W. Moskewicz *et al.*, "Chaff: Engineering an efficient SAT solver," in *Proceedings of the 38th annual Design Automation Conference*, 2001, pp. 530–535.

[3] M. Nadeem and R. Drechsler, "Polynomial formal verification of multi-valued logic circuits within constant cutwidth architectures," in *2024 IEEE 54th International Symposium on Multiple-Valued Logic (ISMVL)*, 2024, pp. 149–154.

[4] F. Farahmandi and B. Alizadeh, "Gröbner basis based formal verification of large arithmetic circuits using gaussian elimination and cone-based polynomial extraction," *MICPRO*, vol. 39, no. 2, pp. 83–96, 2015.

[5] 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 *DATE*, 2016, pp. 1048–1053.

[6] D. Kaufmann, A. Biere, and M. Kauers, "Verifying large multipliers by combining SAT and computer algebra," in *FMCAD*, 2019, pp. 28–36.

[7] A. Mahzoon, D. Große, and R. Drechsler, "PolyCleaner: Clean your Polynomials before Backward Rewriting to verify Million-gate Multipliers," in *ICCAD*, 2018, pp. 1–8.

[8] R. Drechsler, "Preserving and Improving Verifiability of Circuits Based on Local Transformations," in *2025 IEEE 26th Latin American Test Symposium (LATS)*, 2025, pp. 1–2.

[9] A. Konrad and C. Scholl, "Symbolic Computer Algebra for Multipliers Revisited-It's All About Orders and Phases," in *FMCAD*, 2024, pp. 261–271.

[10] A. Mahzoon, D. Große, and R. Drechsler, "RevSCA-2.0: SCA-Based Formal Verification of Nontrivial Multipliers Using Reverse Engineering and Local Vanishing Removal," *TCAD*, vol. 41, no. 5, 2022.

[11] L. Weingarten, K. Datta, and R. Drechsler, "Towards Polynomial Formal Verification of Neuromorphic Architectures," in *ISED*. IEEE, 2024.

[12] C. Yu, W. Brown, D. Liu, A. Rossi, and M. Ciesielski, "Formal verification of arithmetic circuits by function extraction," *TCAD*, vol. 35, no. 12, pp. 2131–2142, 2016.

[13] D. Ritirc, A. Biere, and M. Kauers, "Column-wise verification of multipliers using computer algebra," in *FMCAD*, 2017, pp. 23–30.

[14] C. Yu, M. Ciesielski, and A. Mishchenko, "Fast algebraic rewriting based on and-inverter graphs," *TCAD*, vol. 37, no. 9, pp. 1907–1911, 2017.

[15] D. Ritirc, A. Biere, and M. Kauers, "Improving and extending the algebraic approach for verifying gate-level multipliers," in *DATE*, 2018, pp. 1556–1561.

[16] D. Kaufmann, A. Biere, and M. Kauers, "Incremental column-wise verification of arithmetic circuits using computer algebra," *Formal Methods in System Design: An International Journal*, Feb. 2019.

[17] R. Li, L. Li, H. Yu, M. Fujita, W. Jiang, and Y. Ha, "RefSCAT: Formal Verification of Logic-Optimized Multipliers via Automated Reference Multiplier Generation and SCA-SAT Synergy," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2024.

[18] H. Liu *et al.*, "Parallel Gröbner Basis Rewriting and Memory Optimization for Efficient Multiplier Verification," in *DATE*, 2024, pp. 1–6.

[19] L. Weingarten, K. Datta, and R. Drechsler, "Late Breaking Results: Towards Efficient Formal Verification of Dot Product," in *DATE*. IEEE, 2025.

[20] R. Drechsler and A. Mahzoon, "Polynomial formal verification: Ensuring correctness under resource constraints," in *ICCAD*, 2022, pp. 70:1–70:9.

[21] C. Wolf, "Yosys Open SYnthesis Suite," https://yosyshq.net/yosys/, 2024.

[22] "Abc: A system for sequential synthesis and verification," available at https://people.eecs.berkeley.edu/~alanmi/abc/, 2018.