Late Breaking Results: Efficient Formal Verification of Highly Optimized MAC Units

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

***Abstract*—The demand for compute-intensive applications such as AI/ML has led to the development of processors with complex functionalities. The Multiply Accumulate (MAC) unit is a vital component in these processors, but its verification is very challenging due to the highly optimized designs used to implement the MAC operation. In this paper, we show some interesting results for optimized MAC design verification using a formal proof engine, Symbolic Computer Algebra (SCA). For the first time, we exploit the combined benefit of phase and dynamic ordering in verifying MAC circuits, a capability not possible using state-of-the-art SCA proof engines.**

***Index Terms*—Symbolic Computer Algebra (SCA), Multiply-Accumulate (MAC), Circuit verification**

I. Introduction

The proliferation of compute-intensive applications such as AI/ML necessitates processors with complex functionalities. A vital component of these processors, the *Multiply Accumulate* (MAC) unit, is highly challenging to verify due to the complexity of its design implementation.

Formal methods, such as SCA, demonstrate immense potential for verifying large circuits (e.g., 256-bit, 512-bit) such as multipliers, MAC units, and Dot-Products [1]–[8] when the design uses simple adder and multiplier building blocks. However, these methods fail to verify optimized MAC designs which are structurally heterogeneous, even at smaller bit-sizes (e.g., 9-bit, 10-bit) [9]. This failure occurs due to the exponential growth of symbolic expressions caused by intermediate term explosion during the substitution process. While sophisticated SCA tools like RevSCA, which utilizes atomic block detection, cone detection, and vanishing monomials, can verify 512-bit MAC designs, they cannot handle 10-bit optimized MAC designs. In this paper we present the design of our proposed SCA-verifier which exploits phase (polarity) and dynamic ordering for the verification of optimized MAC circuits up to 15-bit size. This capability is, to the best of our knowledge, beyond the reach of existing state-of-the-art tools. Results demonstrate of up to $24537\times$ improvement on maximum polynomial size and up to $19713\times$ improvement on verification time for certain benchmarks.

II. Proposed Design and Verification Methodology

A. Multiply-Accumulate (MAC) Architecture

An n-bit MAC is defined by $R_{2n+1} = (A_n \times B_n) + S_{2n}$, it uses inputs `A`, `B` for the multiplication and the `S` input for the accumulator. For the *Specification Polynomial* (SP) the MAC expression must result in zero, therefore the terms are reordered as follows: $SP_{MAC} := R - (A \times B) - S = 0$.

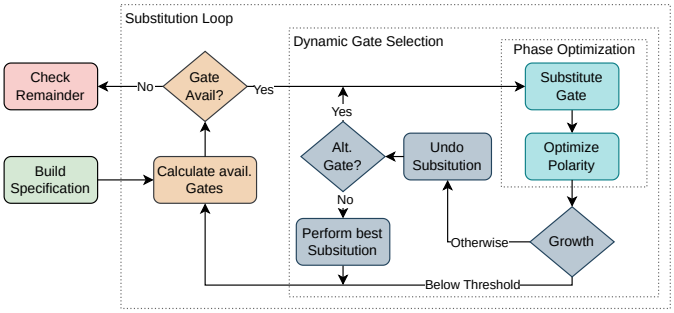

Fig. 1. Verification Algorithm implemented in the Proposed SCA-Verifier

We consider highly optimized MAC operations resulting from the behavioral Verilog representation after the application of Yosys [10] and ABC [11]. The generated circuit is optimized in terms of the number of AIG nodes, negated edges and longest path compared to a structurally simple MAC design, where the underlying architecture is known. Table I shows the number of nodes and the longest path of some of the structurally known MAC architectures, along with the optimized MAC. The MAC designs in the table consist of all the following components: Array (AR) and Dadda Tree (DT) multipliers, and Ripple Carry (RC) and Carry Skip (CS) adders.

TABLE I
8-bit MAC Node and Longest Path Analysis

Design	Nodes	Longest Path
$AR \circ RC \circ RC$	1646	134
$AR \circ CS \circ CS$	1704	140
$DT \circ RC \circ RC$	1646	83
$DT \circ CS \circ CS$	1711	91
Optimized MAC	1324	58

B. Verification Methodology

Our verification algorithm is illustrated in Fig.1. The overall procedure for the proposed look ahead SCA-based verification is as follows: First, the specification polynomial is constructed as described previously. Signals in this polynomial are then iteratively substituted according to the circuit's gates. A gate is considered available when all successors of its output have already been substituted. The process terminates when no further gates are available, and verification succeeds if the remaining polynomial evaluates to zero. We adopt a simple gate-based substitution strategy. Although grouping gates into

larger blocks can simplify processing, detecting such blocks so that they can be substituted in a single step, as described in [4], is often computationally expensive and challenging for circuits with unknown internal structures. To choose among available gates, we employ a dynamic selection algorithm guided by polynomial size, following [12]. For each available gate, we perform a substitution and measure the resulting polynomial size growth. If the growth factor exceeds a predefined threshold (1.3 in our experiments), the substitution is reverted and the next gate is tested. If no gate satisfies the growth constraint, the gate with the smallest growth factor is selected. Although this approach involves frequent substitutions and reversions, it typically reduces intermediate polynomial sizes significantly and improves verification time. To our knowledge, no static ordering achieves comparable performance.

Further polynomial size reduction is achieved through polarity optimization, as introduced in [5]. During SCA substitution, each polynomial represents an output function dependent on a cut set of signals. Some polynomials can grow excessively large; polarity optimization mitigates this by negating certain signals to reduce the number of monomials. For instance, the OR function has exponential size in positive polarity form but becomes linear if all signals are negated:

$$a + b + c - ab - bc - ac + abc \quad (1)$$

$$= 1 - \bar{a}, \bar{b}, \bar{c} \quad (2)$$

Eq.1 shows the OR function for a, b, and c without negation, requiring seven monomials, whereas Eq.2 requires only two monomials when all signals are negated. This behavior is similar to the difference between Positive Polarity Reed-Muller expressions and Fixed Polarity Reed-Muller expressions at the boolean level [13]. To maintain canonical form, each signal must appear with a single polarity. Polarity optimization significantly reduces polynomial size, surpassing state-of-the-art methods and making previously unverifiable designs verifiable. Polarity assignments are determined dynamically: When substituting a signal by its gate function, the look ahead algorithm evaluates whether flipping the signal's polarity decreases the polynomial size for each signal in the gate function. Note, that this heuristic approach does not guarantee to find the best combination of phases as doing so is not feasible, but has immense effect on polynomial size and thereby verifiability.

III. Experiments

All experiments for the C++ implemented SCA-based verifier were conducted on an AMD Ryzen 7 PRO 4750U with 40GB main memory. Optimized MAC designs were studied from 2-to-15 bit. Table II compares the verification results of the proposed look ahead method against the baseline RevSCA [4], [7] for MAC and a RevSCA-based transformation method [9]. The table columns present the MAC bit size (B), maximum polynomial (MP) size and verification time (VT) in seconds. MAC resulting in memory timeout are marked with (MTO). As shown in the table, the proposed SCA-verifier consistently outperforms both RevSCA and the transformation method starting from 9-bits (24537× for MP and 19713× on VT). Furthermore, while the verification time is similar for lower bit-widths, our method outperforms both for larger bit-widths. The proposed method shows great potential to prevent or reduce the intermediate term expansion which is crucial in SCA-based verification. Fig.2 highlights the issue of intermediate term expansion with RevSCA (red) and transformation (blue), whereas the proposed approach results in flattening the substitution curve.

TABLE II
MAC Verification Result

	Proposed		RevSCA [4], [7]		Transform. [9]	
B	MP	VT	MP	VT	MP	VT
2	26	0.0038	19	0.0012	13	0.0009
3	38	0.0073	29	0.0016	23	0.0016
4	62	0.0132	49	0.0025	34	0.0022
5	91	0.0180	4297	0.1150	50	0.0061
6	118	0.0210	418960	176.2480	90	0.0093
7	158	0.0560	732122	274.2510	146	0.0090
8	203	0.0764	5819510	1445.4100	336	0.0326
9	240	0.1254	5888912	2472.2900	MTO	MTO
10	300	0.2219	MTO	MTO	1261	0.1277
11	379	0.2612	MTO	MTO	50908	15.6242
12	1336	0.3291	MTO	MTO	MTO	MTO
13	1562	0.3965	MTO	MTO	MTO	MTO
14	1804	0.7229	MTO	MTO	MTO	MTO
15	6942	10.6026	MTO	MTO	MTO	MTO

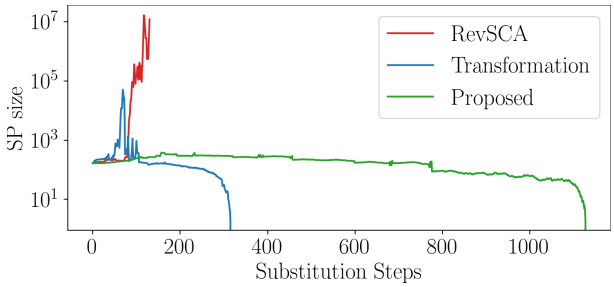

Fig. 2. Comparison of Substitution Curve for 11-bit MAC

We employ a gate-based substitution rather than the block-based substitution used in RevSCA and the transformation work. Although this inherently increases the number of substitution steps, Table II confirms there is no adverse effect on the VT. Consequently, the phase (polarity) and dynamic ordering prove to be far more impactful for SCA-based methods than the characteristics of the atomic block, vanishing monomials, or cones used in [4], [7], [9].

IV. Conclusion

While SCA methods verify some large, MAC designs, they fail against optimized MAC designs due to intermediate term explosion. We addressed this by developing an efficient look ahead SCA-verifier exploiting phase and dynamic ordering, enabling the verification of optimized MAC circuits up to 15-bit size which is beyond current state-of-the-art tools.

Acknowledgment

This work was supported in part by DFG within the Reinhart Koselleck Project PolyVer (DR 287/36-1) and partly by the Federal Ministry of Research, Technology and Space (BMFTR) within the ECXL project under grant no. 01IW22002 and project DI-ReDesign under grant no. 16ME0949. This work was supported by the Data Science Center of the University of Bremen (DSC@UB) funded by the State of Bremen.

REFERENCES

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

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

[3] 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.

[4] 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, pp. 1573–1586, 2022.

[5] A. Konrad and C. Scholl, "Symbolic Computer Algebra for Multipliers Revisited-It's All About Orders and Phases," in *FMCAD*, 2024, pp. 1–11.

[6] H. Liu, P. Liao, J. Huang, H.-L. Zhen, M. Yuan, T.-Y. Ho, and B. Yu, "Parallel Gröbner Basis Rewriting and Memory Optimization for Efficient Multiplier Verification," in *DATE*, 2024, pp. 1–6.

[7] L. Weingarten, K. Datta, and R. Drechsler, "ForMAt: Formal Verification of Scalable Multiply and Accumulate Unit," in *FDL*, 2025, pp. 1–7.

[8] ——, "Late Breaking Results: Towards Efficient Formal Verification of Dot Product Architectures," in *DATE*. IEEE, 2025, pp. 1–2.

[9] ——, "Transformation-Aided Verification of MAC Designs using Symbolic Computer Algebra," in *DVCon Europe*, 2025, pp. 1–7.

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

[11] "ABC: A System for Sequential Synthesis and Verification," available at https://people.eecs.berkeley.edu/~alanmi/abc/, 2018.

[12] A. Mahzoon, D. Große, C. Scholl, and R. Drechsler, "Towards formal verification of optimized and industrial multipliers," in *DATE*, 2020, pp. 544–549.

[13] T. Sasao, *Logic Synthesis and Optimization*. Kluwer Academic Publisher, 1993.