Towards Polynomial Formal Verification of Neuromorphic Architectures

Lennart Weingarten
Institute of Computer Science
University of Bremen
Bremen, Germany
len_wei@uni-bremen.de

Kamalika Datta
Institute of Computer Science
University of Bremen/DFKI
Bremen, Germany
kdatta@uni-bremen.de

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

***Abstract*—With the growing use of Artificial Intelligence various *Neural Processing Units (NPUs)* are becoming increasingly popular. At the core of these processors are the *Multiply and Accumulate (MAC)* operations. As processors with complex functionalities become more relevant in the AI era, it is imperative to have efficient verification strategies to circumvent costly errors. In this context *Polynomial Formal Verification (PFV)* has recently been introduced. As compared to classical formal verification, PFV ensures not only 100% correctness, but also provides an estimate of time and space within polynomial bounds. PFV has successfully been applied for various formal proof engines, like *Binary Decision Diagrams (BDD)*, *Satisfiability (SAT)*, *Answer Set Programming* (ASP), *Symbolic Computer Algebra (SCA)*. BDDs, SAT and ASP have been used to efficiently verify adders whereas SCA was applied to multipliers. The question arises how to exploit the benefits of these polynomially verified sub-components in solving more complex designs. In this paper we investigate how this can be applied to NPUs, particularly we show experimental studies for MAC operations. It is shown that verifiability of the MAC operation heavily depends on the realization of the underlying circuit.**

***Index Terms*—SCA, MAC, PFV, RevSCA, Verification, NPU**

I. Introduction

In recent times with the growing use of AI, *Neural Processing Unit (NPU)* is becoming extremely popular [1]. The most fundamental block in such processors are the *Multiply and Accumulate (MAC)* units. As processors with complex functionalities are becoming more significant in the present AI era, it is imperative to address the verification challenges to prevent costly errors. Various formal proof engines like *Binary Decision Diagram (BDD)*, *Satisfiability Solvers (SAT)*, *Answer Set Programming (ASP)* and *Symbolic Computer Algebra (SCA)* are used for verifying complex functionalities particularly arithmetic circuits. A recent work [2] from industry has targeted to formally verify *Dot Product Accumulate Systolic Units (DPAS)* which are considered as a vital unit and are used in current AI/ML algorithms.

Recently the concept of *Polynomial Formal Verification (PFV)* has been introduced and successfully applied [3]–[9]. Using PFV upper bounds for the time and space complexity for the verification process are proven guaranteeing efficient run times. In this context, various formal proof engines, like BDDs, SAT, ASP, and SCA, have been used especially for verification of arithmetic circuits. E.g. BDDs are found suitable for adder verification, but for multipliers exponential lower bounds exist [10]. Whereas SCA can handle specific class of multipliers within polynomial bounds.

In this work we specifically target the verification of the MAC operation which is the core of many NPUs. We exploit the benefits of polynomially verified sub-components in solving MAC operation. In particular as multipliers can be verified efficiently using SCA [11], we exploit SCA for MAC unit verification. For the construction of the MAC unit we have used an *Array Multiplier (AM)* and a *Ripple Carry Adder (RCA)*. First experiments show that it is possible to verify MAC operations up to 256-bit using SCA-based verification. Here, it can even be expected from the observed experiments that polynomial bounds exist for the entire verification run. To show the effect on verification of the used architecture, we also report experiments on optimzed MAC designs. It turns out that then the formal verification can only be carried out up to 8-bits. Hence, directly transferring the SCA-based method from [11] (which targets multiplier verification) does not work. Further investigation is required to extend the SCA-based verification for MAC. For this, at the end of the paper, we outline some directions for future work resulting from our findings.

The rest of the paper is organized as follows, Section II provides the idea of SCA-based verification, Section III details out the proposed MAC-based verification process. In Section IV we provide the experimental results, and Section V concludes the paper with challenges and open problems.

II. SCA Based Verification

During the last 10 years several works have been proposed for verification of integer multipliers using SCA [12]–[19]. SCA-based verification is applied directly on the gate level [11], [13], [20]. In some cases only specific gate level netlists are used and in some cases *And-Inverter-Graphs* (AIG)s are applied. In the first step, the *Specification Polynomial (SP)* of a function in terms of input and output is defined. In the second step a set of *Gate Polynomials (GP)* is defined for all possible gate types considered for the function. Finally the backward rewriting step is performed starting from the SP. Considering a particular gate order, the backward rewriting process starts from the primary outputs by using reverse topological order. For each gate the applicable GPs are applied and substituted in the SP. After processing the last gate, the remainder of the SP is evaluated. If after all the substitutions the remainder is equal to 0 the circuit is bug free, otherwise it is faulty. The same process can be applied at the AIG node level as well.

A. Specification Polynomial

In this sub-section we provide the basis before going into depth of SCA-based verification. For more details see [11].

Definition 2.1: A monomial is defined as a power product of variables. Also known as a term.

$$M = m_1^{\alpha_1} m_2^{\alpha_2} \cdots m_n^{\alpha_n} \text{ width } \alpha_i \in \mathbb{N}_0$$

Definition 2.2: A polynomial is a finite sum of monomials with coefficients in $\mathbb{Z}$:

$$\mathcal{P} = c_1 M_1 + c_2 M_2 + \cdots + c_j M_j \text{ width } c_j \in \mathbb{Z}$$

A *Specification Polynomial (SP)* captures the mathematical functionality of an arithmetic circuit in the form of a polynomial based on its primary inputs and outputs. The main objective of SCA-based verification is to formally prove that all assignments are inline with the gate-level netlist or AIG and must evaluate the specification polynomial to 0. This is performed using backward rewriting rules. For example the SP for an arithmetic circuit like a *Half Adder (HA)* with primary inputs X, Y and primary outputs C, S and a *Full Adder (FA)* with an additional inputs Z are defined as:

$$SP_{HA} = 2C + S - X - Y = 0$$
$$SP_{FA} = 2C + S - X - Y - Z = 0$$

B. Gate Level Polynomial Evaluation

As mentioned in the previous sub-section, for performing verification using SCA, firstly the SP of the circuit is defined. Thereafter each gate polynomial is evaluated from reverse topological order, i.e. from the outputs to the inputs. This is performed by dividing the SP with each gate polynomial from a set of *Gate Polynomials (GP)*. Ideally substitution is used instead of division. Gate by gate substitution takes place for each gate polynomial until the SP is evaluated to 0 or there exists a remainder. If the SP is 0 then the circuit is bug free else the circuit is buggy.

An example GP set is presented below:

$$\begin{aligned} \text{NOT} \quad & f = 1 - X \\ \text{AND} \quad & f = XY \\ \text{OR} \quad & f = X + Y - XY \\ \text{XOR} \quad & f = X + Y - 2XY \end{aligned}$$

Each logic gate is described by its mathematical function f with inputs X and Y. In the following a backward rewriting example for a FA circuit using SCA is performed. Fig. 1 shows the gate-level netlist of a FA. It has five gates and hence five iterations are required for the backward rewriting. The steps of the backward rewriting procedure are presented in Fig. 2. We begin with the SP and then evaluate each gate from the outputs to the inputs. The notation for the transition from the SP state i to the next state j is represented as $SP_i \xrightarrow[\text{GP}]{gate} SP_j$, with $gate$ describing the current gate and GP the corresponding gate polynomial performed in the current step. Finally after step 5 the remainder is 0, hence the circuit is verified to be correct.

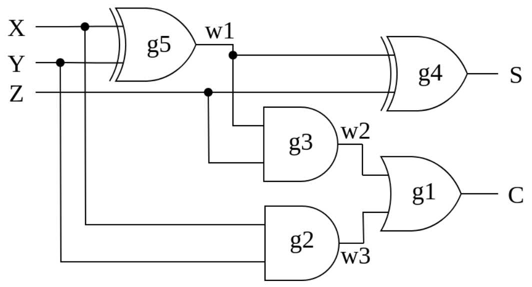


Fig. 1: Full adder gate netlist

$$\begin{aligned}
SP :=& 2C + S - X - Y - Z = 0 \\
SP \xrightarrow[\text{OR}]{g1} SP_1 =& 2w_2 + 2w_3 - 2w_2w_3 + S - X - Y - Z \\
SP_1 \xrightarrow[\text{AND}]{g2} SP_2 =& 2w_2 + 2XY - 2w_2XY + S - X - Y - Z \\
SP_2 \xrightarrow[\text{AND}]{g3} SP_3 =& 2w_1Z + 2XY - 2w_1XYZ + S - X - Y - Z \\
SP_3 \xrightarrow[\text{XOR}]{g4} SP_4 =& 2w_1Z + 2XY - 2w_1XYZ \\
& + (w_1 + Z - 2w_1Z) - X - Y - Z \\
=& + 2XY - 2w_1XYZ + w_1 - X - Y \\
SP_4 \xrightarrow[\text{XOR}]{g5} SP_5 =& +2XY - 2(X + Y - 2XY)XYZ \\
& + (X + Y - 2XY) - X - Y \\
=& (-2X - 2Y + 4XY)XYZ \\
=& - 2X^2YZ - 2XY^2Z + 4X^2Y^2Z \\
=& - 2XYZ - 2XYZ + 4XYZ = 0
\end{aligned}$$

Fig. 2: Backward rewriting steps for the full adder

III. MAC Verification Process

In this section we discuss about the MAC operation and how we can perform verification of the MAC unit using SCA. Fig. 3 shows the MAC unit construction. The MAC unit has three inputs, A, B and S. A and B are n-bit inputs and S is a $2n$-bit input. Initially, the two inputs A and B are fed to the multiplier which generates a $2n$-bit intermediate result. This intermediate result is added with S which is also $2n$-bit to generate the final result R which is $2n+1$ bit wide.

The verification using SCA is performed in two steps: The *Specification Polynomial (SP)* for the MAC operation is first defined. Then the backward rewriting process is performed from the outputs to the inputs. For this purpose atomic blocks at the gate-level representation of the MAC circuit are identified and evaluated, thereafter they are replaced if possible with a compact polynomial. Techniques to minimize so-called *vanishing monomials* (see [11]) are applied

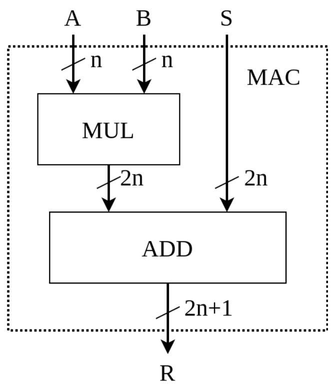


Fig. 3: MAC unit construction

to keep the memory usage low. The verification process is performed using backward rewriting. Finally the remainder polynomial is evaluated. It reduces to a zero polynomial if the MAC operation is correct, otherwise faulty. We have used RevSCA [11] for the verification purpose. RevSCA is extended to accommodate for MAC verification (see Section III-B).

A. Specification Polynomial

In this sub-section we define the SP for n-bit MAC operation. A MAC operation with primary inputs A, B, S and primary output R is defined as follows:

$$R = (A \times B) + S \tag{1}$$

In Eqn. (1) the final result of the operation is denoted by R, the multiplicand and multiplier by A and B and the accumulation input by S. According to this definition the *Specification Polynomial (SP)* for an n-bit MAC is defined in the following Eqn. (2):

$$SP_{MAC} := R_{2n+1} - (A_n \times B_n) - S_{2n} = 0 \tag{2}$$

For example an SP for a 2-bit MAC has the following size:

$$SP_{MAC_2} := R_5 - (A_2 \times B_2) - S_4 = 0$$

The full SP for a 2-bit MAC is presented in Eqn. (3).

$$\begin{aligned} SP_{MAC}(2) &= 16R_4 + 8R_3 + 4R_2 + 2R_1 + R_0 \\ &- \big((2A_1 + A_0) \times (2B_1 + B_0)\big) \\ &- (8S_3 + 4S_2 + 2S_1 + S_0) \end{aligned} \tag{3}$$

B. SCA-Based MAC Verification

For verifying the MAC operation we have extended the functionality of RevSCA-2.0 [11]: As RevSCA is targeted for multipliers we exploited its benefits in verifying MAC. The core of SCA-based verification is dependent on the identification of *Atomic Blocks (AB)*. The ABs presented in [11] are HA, FA and *compressor* (CM). For verifying the circuit, starting from the SP, backward rewriting steps are performed from the outputs to the inputs. The knowledge about the ABs helps in reducing the number of monomials that appear in the backward rewriting steps and also limits the search space for removing vanishing monomials. To incorporate MAC the operation in RevSCA the following changes have been carried out:

1) Firstly, a program mode is added to accommodate MAC and ADD operations apart from multiplication.
2) The SPs modelled for the MAC and ADD operation are appropriately added in the tool by modifying the class *poly*.
3) Unlike multipliers the ADD and MAC operations can have uneven input width, therefore the offset in the SP is adjusted to properly automate the entire process of SP generation.
4) Parameters for memory usage have been incorporated in the tool.

The input to the tool is an *And-Inverted-Graph (AIG)* representation of the MAC. Hence for the backward rewriting process AIG gate polynomials have to be applied. Internally the tool generates the ABs and the backward rewriting steps are performed. In this process the number of monomials are reduced in each substitution steps and finally if the remainder is 0, it is confirm that the circuit is bug free. Experimental result shows that using the extended feature of RevSCA we can verify simple MAC, multipliers and adders within very short time period.

IV. Experimental Evaluation

In this section we provide experimental evaluations for MAC verification using SCA and compare them to BDD-based verification. Run times and memory consumption are reported. All experiments were conducted on a Thinkpad T490 with Intel i7-8565U CPU having 16GB memory and running on a 1.80GHz clock.

We have generated MAC operation using two methods: In the first case we have used genmul [21] to generate a multiplier and a custom made code to prepare a *Ripple Carry Adder (RCA)* to combine them in a Verilog module modeling a MAC operation. The multiplier used in this case is an array multiplier for the *Partial Product Accumulation (PPA)* stage and an RCA is used for the *Final Stage Adder (FSA)*. We have then used the ABC [22] tool to generate the AIG of the MAC. The MAC operation generated using this method is termed as *Linear MAC (LMAC)*. For the second case we have generated the MAC operation using the behavioral structure provided in Listing 1.

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
```

We have used Yosys [23] for synthesizing the netlist and then used ABC to generate the AIG. The generated AIG is fed to our tool for verification. The MAC operation generated using this method is called *Optimized MAC (OMAC)*, since some of the original structure of the multiplier and adder might get removed due to the optimizations carried out by Yosys and ABC.

A. SCA Verification Results

TABLE I: SCA-based Verification Results for LMAC-Operations

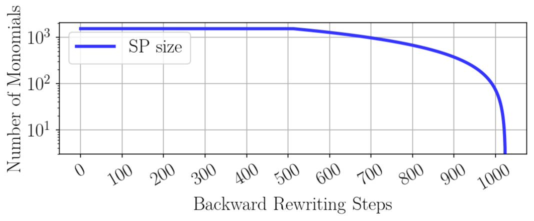
#Mul.	#Add.	#Nodes	MaxPoly	SCA Time (s)	Memory (MB)
2	4	46	13	0.00096	4
4	8	186	33	0.00242	5
8	16	730	97	0.00828	7
16	32	2874	321	0.03765	14
32	64	11386	1153	0.28430	44
64	128	45306	4353	3.00483	163
128	256	180730	16897	84.92770	637
256	512	721914	66561	2333.69000	2532

TABLE II: SCA-based Verification Results for OMAC-Operations

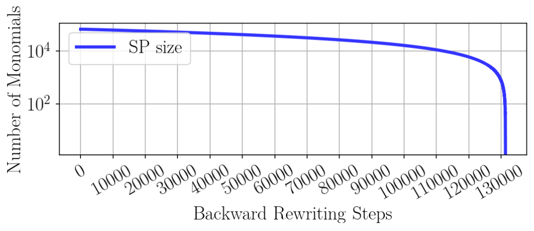
#Mul.	#Add.	#Nodes	MaxPoly	SCA Time (s)	Memory (MB)
2	4	43	19	0.001	4
4	8	156	49	0.002	5
8	16	613	5819510	1394.910	11629
16	32	-	-	-	-

In this sub-section we provide the results for LMAC and OMAC. The results for the LMAC and OMAC are presented in Table I and Table II. The input bit size for the multiplier (#Mul) and the adder (#Add) are given in the first two columns. The third column provides the number of nodes (#Nodes) of the AIG and the maximum polynomial is shown in column four (MaxPoly). MaxPoly is the maximum number of monomials occurring during backward rewriting. The last two columns present the verification time in seconds and memory usage in megabyte, respectively. For the LMAC operation Table I shows that MAC scales very well using SCA-based verification. Results from 2-bit up to 256-bit are generated with limited memory usage. The OMAC results in Table II shows that results could only be provided for 2-, 4- and 8-bit. For 16-bit the machine runs out of memory (16GB of RAM and 16GB of SWAP partition). This clearly shows even for smaller bit size that MAC cannot be handled using SCA if the structural design of the MAC operation is not chosen well. E.g. for 8-bit MAC the OMAC design has maximum polynomial of 5,819,510, therefore an explosion of intermediate monomials could not be prevented. When compared to 256-bit MAC in Table I only a maximum polynomial of 66,048 is generated. This is due to the fact that for LMAC more atomic blocks could be found due to structural benefit which is not possible for OMAC.

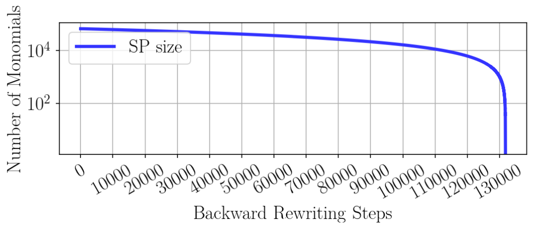
Fig. 4 illustrates the performance of SCA-based verification for LMAC operations. The 512-bit adder shown in Fig. 4a took 0.21 seconds for the verification and uses 25 MBs of memory. The 256 multiplier in Fig. 4b took 2,264 seconds and 2,513



(a) 512-bit adder



(b) 256-bit multiplier



(c) 256-bit MAC

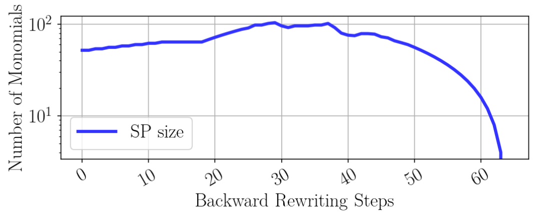
Fig. 4: SCA-based verification for a 256-bit multiplier, 512-bit adder and 256-bit MAC

MBs of memory and Fig. 4c showing the 256-bit MAC which can be verified within 2,333 seconds using 2,532 megabyte of memory. Fig. 4b and Fig. 4c depicts that the MAC operation itself does only take a few more backward rewriting steps for the verification than the multiplication operation alone. From all the sub-figures in Fig. 4 it can be seen for an LMAC-operation the maximum polynomial size is generated at the very beginning of the substitution process at step number zero, after that the polynomial size slowly decreases over the time converging to zero in the end.

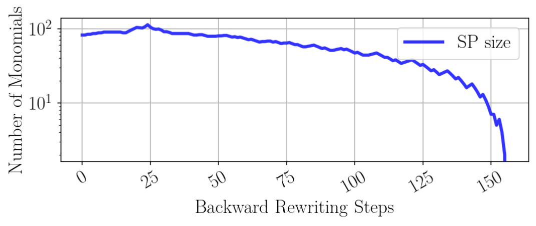
Looking at the memory consumption counted in number of monomials for LMAC (see Fig. 4), it can be seen that it is steadily decreasing and never grows significantly in between. In contrast for OMAC (see Fig. 5) for some intermediate steps the number increases. This can already be seen for the adder and multiplier in Fig. 5a and Fig. 5b, respectively. For the OMAC operation it becomes much worse as depicted in Fig. 5c: the number of intermediate monomails does not only grow, it explodes between backward rewriting steps 50 and 135. This explosion of intermediate monomials does not allow for verification of OMAC of size greater than 8-bit with the given memory limits.

B. Comparison with BDD

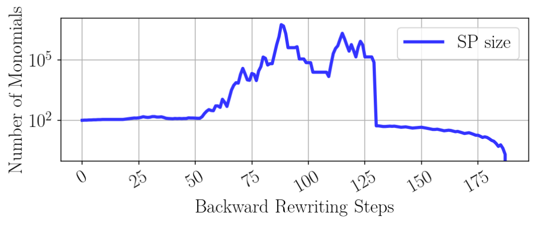
MAC-operations are comprised of multiplication and addition operations. Multipliers can generally be verified using



(a) 16-bit adder



(b) 8-bit multiplier



(c) 8-bit MAC

Fig. 5: SCA-Based verification for optimized 8-bit multiplier, 16-bit adder and 8-bit MAC

SCA-based verification approaches and adders using BDDs. Therefore we analyse the suitability of using BDD-based representation for verification for MAC-operations in this sub-section.

In Fig. 6 we show the performance comparison of BDD generation against the entire SCA-based verification process for LMAC. The SCA-based approach could deliver results up to 256-bits whereas the BDD graph generation could be done for only limited number of bits (up to 11-bits). The memory consumption for the LMAC is shown in Fig. 6a. For 11-bits the construction of the LMAC BDD uses 11.56GB whereas SCA only uses 9MB and even for 256-bits SCA requires only 2.53 GB. Fig. 6b shows the LMAC time required for the BDD and SCA. The maximum time taken by SCA for the entire verification of a 11-bit MAC operation is about 14.10 milliseconds whereas the BDD generation process takes about 903 seconds. From 10 to 11-bits the time for the BDD generation grows from 140 to 903 seconds. This sharp increase of time can also be observed for the SCA-based verification as well, but it happens for much larger bit sizes (i.e. from 128 to 256-bits). These data clearly shows the benefit of using SCA over BDD for MAC verification.

C. Future Direction

From our studies several directions for future research can be derived:

Choice of Architecture Extensive study is required on the influence of various architectures of adders and multipliers for MAC. The main idea is to analyze which designs affects the verification process.

Composition for Verification It is known that multipliers can be efficiently verified using SCA and adders can be verified using BDDs. Further investigation is needed to explore how the composition of these methods can influence the MAC verification process.

Choice of Proof Engines Choice of proof engines plays a vital role in the verification process. An extensive study is required to choose a correct proof engine like either SAT, ASP, BDDs or SCA.

Hybrid Proof-Engines Not only the choice of the proof engine is essential but also some recent work has demonstrated that *hybrid proof engines* can be more beneficial. In this regard further analysis is required to determine what kind of hybrid proof engine can influence verifiability of the MAC unit.

Word-Level Approaches SCA-based verification rely mainly on bit-level evaluation, so far no word-level evaluation, like *SAT Modulo Theory* or *Word-level Decision Diagrams* is used in the context of MAC. Some study shows that the use of higher level of abstraction can influence the overall verification process.

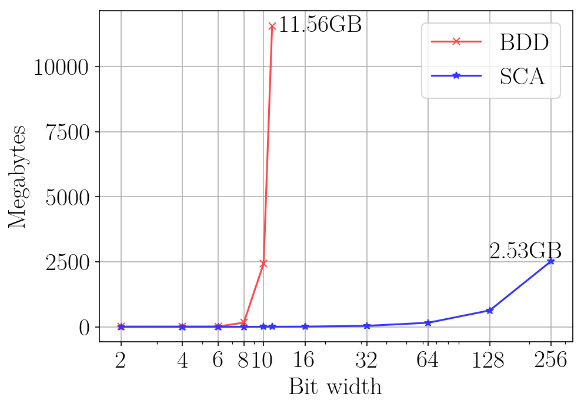
Exploiting Approximation Instead of using a $2n$-bit adder, a single n-bit adder could be placed in the MAC operation construction, i.e. we perform some kind of approximation. As it may be happen that, it is hard to verify the complete circuit but an approximate circuit could be verifiable.

Improved Substitution Rules In SCA-based verification the substitution order from outputs to inputs plays a significant role in intermediate result generation. This in turn might blows up the number of monomials generated during the backward rewriting process. To identify a suitable substitution order further investigation needs to be performed.

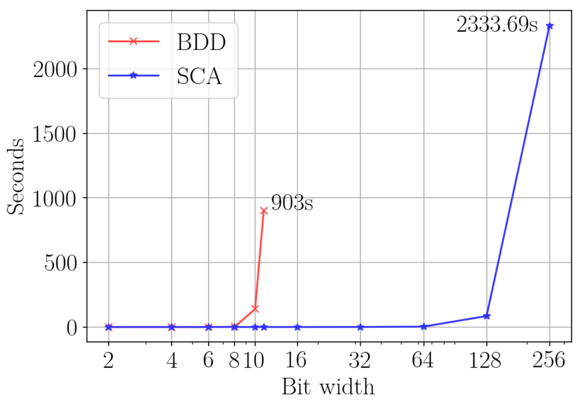
Identification of Atomic Blocks Identification of atomic blocks plays a vital role in SCA-based verification. So far only HA, FA and CM blocks are identified. More atomic block identification needs to be performed. Some domain specific language can be defined for atomic blocks which can aid the verification process.

V. Conclusion

In this paper we show the first results for the MAC unit verification by exploiting *Symbolic Computer Algebra*. We first define the specification polynomial and then perform backward rewriting to verify the circuit. After the substitution process if the remainder is zero then the circuit is bug free else the circuit is buggy. We extend the RevSCA tool for verifying the MAC unit. We have performed experimentation on a range of benchmarks generated by various tools. Experimental results reveal that it is possible to verify large MAC unit consisting of linear multiplier and adder within very short period of time. Although when the structure of the adder or multiplier is modified, even verifying a 16-bit MAC unit is



(a) LMAC Memory



(b) LMAC Time

Fig. 6: BDD-generation time and memory usage compared with SCA-based verification time and memory usage

not possible. Hence the quality in terms of the efficiency of verification process is greatly dependent on the realization of the underlying structure. This reveals that further investigation is required to polynomially verify MAC unit efficiently.

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 Education and Research (BMBF) within the ECXL project under grant no. 01IW22002.

References

[1] N. Jouppi *et al.*, "TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA '23, 2023.

[2] E. Morini, B. Zorn, D. Puri, M. Eranki, and S. Jampana, "Achieving end-to-end formal verification of large floating-point dot product accumulate systolic units," *Design and Verification Conference & Exhibition*, 2024.

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

[4] 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. 1–8.

[5] M. Schnieber, S. Fröhlich, and R. Drechsler, "Polynomial formal verification of approximate adders," in *EUROMICRO Symposium on Digital System Design*, 2022, pp. 761–768.

[6] J. Kleinekathöfer, A. Mahzoon, and R. Drechsler, "Polynomial Formal Verification of Floating Point Adders," in *Design, Automation and Test in Europe*, 2023, pp. 1–2.

[7] M. Nadeem, J. Kleinekathöfer, and R. Drechsler, "Polynomial formal verification of adder circuits using answer set programming," in *In 2023 Reed-Muller Workshop (RM2023)*, 2023.

[8] L. Weingarten, A. Mahzoon, M. Goli, and R. Drechsler, "Polynomial formal verification of processor: A RISC-V case study," in *International Symposium on Quality Electronic Design*. IEEE, 2023, pp. 1–7.

[9] L. Weingarten, K. Datta, A. Kole, and R. Drechsler, "Complete and efficient verification for a risc-v processor using formal verification," in *Design, Automation and Test in Europe*. IEEE, 2024.

[10] R. E. Bryant, "On the complexity of VLSI implementations and graph representations of boolean functions with application to integer multiplication," *IEEE Transactions on Computers*, vol. 40, no. 2, pp. 205–213, 1991.

[11] A. Mahzoon, D. Große, and R. Drechsler, "RevSCA-2.0: SCA-Based Formal Verification of Nontrivial Multipliers Using Reverse Engineering and Local Vanishing Removal," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 5, pp. 1573–1586, 2022.

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

[13] C. Yu, W. Brown, D. Liu, A. Rossi, and M. Ciesielski, "Formal verification of arithmetic circuits by function extraction," *IEEE Transactions on Computer Aided Design of Circuits and Systems*, vol. 35, no. 12, pp. 2131–2142, 2016.

[14] 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*, 2016, pp. 1048–1053.

[15] D. Ritirc, A. Biere, and M. Kauers, "Column-wise verification of multipliers using computer algebra," in *Formal Methods in Computer-Aided Design*, 2017, pp. 23–30.

[16] 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*, vol. 37, no. 9, pp. 1907–1911, 2017.

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

[18] D. Kaufmann, A. Biere, and M. Kauers, "Verifying large multipliers by combining sat and computer algebra," in *2019 Formal Methods in Computer Aided Design (FMCAD)*. IEEE, 2019, pp. 28–36.

[19] ——, "Incremental column-wise verification of arithmetic circuits using computer algebra," *Formal Methods in System Design: An International Journal*, Feb. 2019.

[20] R. Drechsler and A. Mahzoon, "Polynomial formal verification: Ensuring correctness under resource constraints," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2022*, T. Mitra, E. F. Y. Young, and J. Xiong, Eds. ACM, 2022, pp. 70:1–70:9.

[21] A. Mahzoon, D. Große, and R. Drechsler, "GenMul: Generating architecturally complex multipliers to challenge formal verification tools," in *International Workshop on Logic and Synthesis*, 2019.

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

[23] C. Wolf, "Yosys open synthesis suite," https://yosyshq.net/yosys/, 2024.