ForMAt: Formal Verification of Scalable Multiply and Accumulate Units

Lennart Weingarten, Kamalika Datta, Rolf Drechsler
University of Bremen, Department of Computer Science
{len_wei, kdatta, drechsler}@uni-bremen.de

***Abstract*—With the increasing popularity of compute intensive applications like AI, processors with complex functionalities are designed. Multiply and Accumulate (MAC) is one of the essential operations in modern Neural Processor Units (NPUs), but no sound formal verification technique exists that can efficiently ensure correctness. In this paper we analyze almost 200 configurations of MAC instances for various bit-widths starting from 8 up to several hundred bits. On top of the classical area-delay trade-off, we study verifiability as an additional parameter. It is shown that surprisingly the fastest and smallest instances are not the ones that are the hardest to verify. Exploiting *Symbolic Computer Algebra* (SCA) we provide a technique that allows scalable verification for large bit-width and classifies the set of MAC units.**

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

I. Introduction

With the enormous growth of AI during the last few years, *Neural Processing Units* (NPUs) have become increasingly popular and widespread. The fundamental blocks in such NPUs are the *Multiply and Accumulate* (MAC) units. As processors are built with complex functionalities to cater the needs of the AI era, it is of paramount importance to tackle the verification challenges to avoid costly errors. While approaches based on simulation and emulation do not scale for larger designs, only approaches based on formal proof techniques can ensure 100% correctness. While there exist several design approaches for MAC units (see e.g. [1], [2]), none of them provides a sound verification methodology. Recently, in [3] an approach has been presented to formally verify the *Dot Product Accumulate Systolic Unit* (DPA). They use equivalence checking to verify the RTL level code against a golden reference model, that has also to be proven to be complete.

In the following, an approach is presented that does not need a complex reference model, but directly compares the circuit with a high-level specification given in form of a *Specification Polynomial* (SP). Previous studies on verification of arithmetic circuits showed that all techniques based on bit-level representations, like *Binary Decision Diagram* (BDD) [4] or *Boolean Satisfiability Solvers* (SAT) [5] fail for multipliers. This also holds for techniques based on structural hashing, if the descriptions of the multipliers do not have many similarities. Since the MAC unit also includes a multiplier, these findings directly transfer.

The most powerful proof technique for formal verification of multipliers are approaches based on *Symbolic Computer Algebra* (SCA) (see [6]–[14]). Recent studies have also explored verification of MAC and Dot-product operations using SCA [15], [16]. While there are several studies for multipliers, so far the formal verification of MAC units has not been extensively studied. Since the MAC architecture also consists of a multiplier and SCA is well-suited, in this work we perform formal verification of scalable MAC architectures exploiting SCA.

In this paper we present the framework `ForMAt` for formal verification of scalable MAC units. `ForMAt` consists of two main components:

1) `MAC-Gen` allows to generate scalable MAC units.
2) `MAC-Verifier` formally verifies MAC instances based on SCA and gives a prediction based on indicators about the verifiability of scaled designs for larger bit-width.

Thus, the main contributions of the paper are as follows:

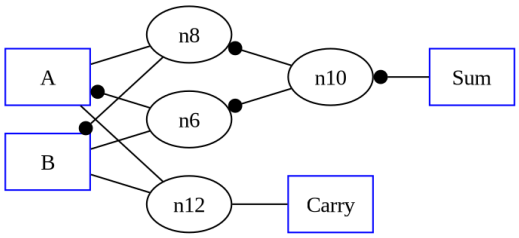
1) Automated generation of MAC architectures
2) Extending SCA-based verification to MAC units
3) Detailed analysis and classification of MAC architectures with respect to verifiability
4) Definition of scalability indicators
5) Adding verifiability as a design parameter

Extensive experimental studies for almost 200 MAC configurations are carried out. It is shown that the indicators for small bit-width (of 8 or 16-bit) allow a precise forecast of which MAC units can be successfully verified when scaled up to several hundred bits. Surprisingly, it can be observed that the circuits with a very good area-delay trade-off can also be fully proven correct using SCA. This is shown for instances with up to 512-bits in our experiments.

The paper is organized as follows, Section II provides the necessary background about the MAC operation and SCA-based verification. Section III describes the MAC generation process using `MAC-Gen`. In Section IV the `MAC-Verifier` is presented along with the definition of the scalability indicators. First experiments show a classification of the MAC units. In-depth studies in Section V present the scalability analysis for MAC architectures, and finally the results are summarized in Section VI.

```
module half-adder (input A,B, output Sum, Carry);
  assign Sum   = A ^ B;
  assign Carry = A & B;
endmodule
```

(a) Half-Adder Verilog Representation



(b) AIG representation of the Half-Adder

Fig. 1: Half-Adder Example

II. Background

To make the paper self-contained, we first briefly give the description of the MAC operation as it is used in the paper. Then we briefly review the core concept of SCA-based verification (for more details see [7], [8], [11]–[14]).

A. MAC Operation

The *Multiply and Accumulate* (MAC) unit is a complex arithmetic circuit comprising of a multiplier and adder (accumulator) component. The word-level description of the MAC-operation is defined as:

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

A and B are the inputs of the multiplier and S is the accumulator input, finally the result is stored in R.

B. SCA-based verification

SCA-based verification is the most well suited verification engine for multipliers. It uses algebraic techniques to prove the correctness of a design using its polynomial representation. In SCA-based verification first a *Specification Polynomial* (SP) has to be defined. An SP is a mathematical description of the circuit representing it only in terms of its inputs and outputs. The verification is performed by iterating over the circuit in reverse topological order and substituting each gate in the SP with its corresponding *Gate Polynomial* (GP). This iterative step is called *backwards rewriting*. The initial SP size is denoted as SP_{init} and the SP size at time step i as SP_i during the computation. After each gate is processed the final SP is evaluated. If it reduces to a zero polynomial, the circuit is bug free, otherwise the circuit is faulty.

Example 1: Consider the Verilog code of the half adder shown and its representation as an

And-Inverter Graph (AIG) [17] in Fig. 1a and Fig. 1b, respectively. For each of the AIG nodes the GP rules are shown in Fig. 2, i.e. for a buffer, a negated output, and for an AND-gate with various complemented or un-complemented edges (see also [11]). The backward rewriting starts at the outputs *Sum* and *Carry*. The SP is then stepwise replaced according

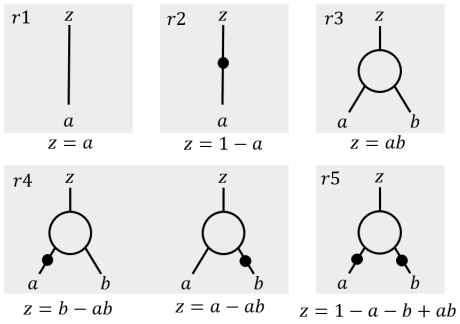


Fig. 2: AIG node Gate Polynomials

$$SP_{HA} := 2C + S - A - B = 0$$

$$SP_0 \xrightarrow[\text{Carry}]{r1} SP_1 = 2n_{12} + S - A - B$$

$$SP_1 \xrightarrow[\text{Sum}]{r2} SP_2 = 2n_{12} + 1 - n_{10} - A - B$$

$$SP_2 \xrightarrow[n_{12}]{r3} SP_3 = 2AB + 1 - n_{10} - A - B$$

$$SP_3 \xrightarrow[n_{10}]{r5} SP_4 = 2AB + 1 - (1 - n_8 - n_6 + n_6 n_8) - A - B$$
$$= 2AB + n_8 + n_6 - n_6 n_8 - A - B$$

$$SP_4 \xrightarrow[n_8]{r4} SP_5 = 2AB + (A - AB) + n_6 - n_6(A - AB) - A - B$$
$$= 2AB + A - AB + n_6 - n_6 A + AB n_6 - A - B$$
$$= AB + n_6 - n_6 A + AB n_6 - B$$

$$SP_5 \xrightarrow[n_6]{r4} SP_6 = AB + (B - AB) - (B - AB)A + AB(B - AB) - B$$
$$= AB + B - AB - AB + A^2 B + AB^2 - (AB)^2 - B$$
$$= AB - AB - AB + AB + AB - AB + B - B$$
$$= 0$$

Fig. 3: Half Adder Substitution Steps

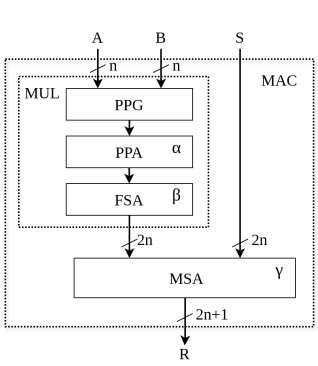
to the AIG netlist by the corresponding GPs. The steps of the substitution are shown in Fig. 3. We start by evaluating rule $r1$ on the *Carry* output and $r2$ on the *Sum* output. Next, $r3$ on node n_{12} is evaluated and so on until all nodes are replaced by their GPs. In the final step, $r4$ is used to substitute node n_6 and the SP size reduces to zero, proving the correctness of the circuit. Boolean algebraic rules are used for the evaluation.

III. MAC Generation

In this section we provide the details of the MAC generation process and of our tool `MAC-Gen` which generates a variety of structurally different MAC architectures. The left-hand side of Fig. 4 shows the general architecture of the MAC unit including the notation of the bit-width: The bit-level description of the MAC is defined as (cf. Subsection II-A):

$$R_{2n+1} = (A_n \times B_n) + S_{2n} \tag{2}$$

The two n-bit inputs of the *multiplier unit* (MUL) are A and B, respectively, and the input S of the *MAC Stage Adder* (MSA) is $2n$-bit wide. The width of the result R is also $2n$ plus an additional carry output of the MSA. MUL consists of three stages:



PPG	
1	[U] Simple Unsigned

PPA	
1	[AR] Array
2	[CWT] Counter based Wallace Tree
3	[DT] Dadda Tree
4	[WT] Wallace Tree

FSA / MSA	
1	[RC] Ripple Carry
2	[CL] Carry Look-Ahead
3	[LF] Ladner-Fischer
4	[KS] Kogge-Stone
5	[BK] Brent-Kung
6	[CS] Carry Skip
7	[SE] Serial Prefix

Fig. 4: MAC-unit Architecture

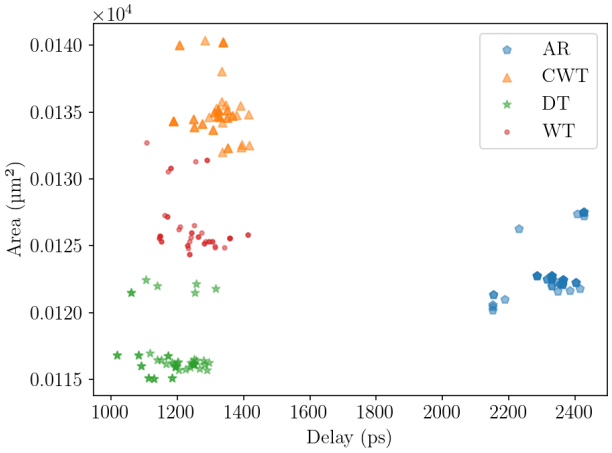


Fig. 5: Delay and area of all MAC architectures for 16-bits

1) *Partial Product Generator* (PPG)
2) *Partial Product Accumulator* (PPA)
3) *Final Stage Adder* (FSA)

The accumulator part of the MAC is comprised of the MSA. On the right side in Fig. 4 we show all the possible options for each component, i.e. for the PPA one out of four designs is chosen and for FSA and MSA one out of seven. Resulting in a total number of $4 \times 7 \times 7 = 196$ different configurations. Since the first stage is identical in all cases, the PPG is not listed in the following. To compactly describe different MAC architectures the three configurations are given as a concatenation:

Example 2: To denote that the PPA consists of a Wallace Tree, the FSA is a Carry Skip adder and the MSA is a Ladner-Fischer adder, we use:

$$WT \bullet CS \bullet LF \tag{3}$$

These three parameters together with the bit-width of the inputs are given to `MAC-Gen`. It then generates a Verilog file that can be converted to an AIG representation using Yosys [18]. The input of the `MAC-Verifier` is the AIG representation of the MAC circuit.

A. Area & Delay Calculation

Before considering the verification of the MAC units in the next section, we first provide some more information on the synthesis process based on `MAC-Gen` and how area and delay is measured. For all our experiments, Cadence Genus Solution [19] utilizing the ASAP 7 [20] 7.5-track standard cell library for the 7-nm technology node is used. Area is presented in μm^2 and delay is given in pico seconds (ps).

Fig. 5 shows the area and delay of all 196 MAC configurations for 16 bits. The MAC configurations are grouped according to the PPA, i.e. four different groups are shown by the colors. E.g. not surprisingly, AR based on the array (shown in blue) has the largest delay.

IV. MAC Verification

In this section we present the SCA-based verification process and show the MAC specific operations. Firstly, the *Specification Polynomial* (SP) is defined, then the verification process is discussed and the two *scalability indicators* are introduced. Finally, we provide the verification results.

A. Specification Polynomial for MAC

For SCA-based verification (as mentioned above), we need to first represent the SP of the circuit. For the MAC block, the functionality can be described by only considering its primary inputs and outputs as mentioned in Section III. From this we can derive the SP as $R-(A\times B)-S=0$. For a fixed bit-width n we obtain:

$$SP := \sum_{i=0}^{2n} 2^i R_i - \left(\sum_{i=0}^{n-1} 2^i A_i\right) \times \left(\sum_{i=0}^{n-1} 2^i B_i\right) - \left(\sum_{i=0}^{2n-1} 2^i S_i\right) \tag{4}$$

Example 3: An SP for a 4-bit MAC has the following form: $SP_{MAC_4} := R_9 - (A_4 \times B_4) - S_8 = 0$ The full bit-level representation of the 4-bit MAC is then given by:

$$\begin{aligned} &SP_{MAC_4} \\ &= 256R_8 + 128R_7 + 64R_6 + 32R_5 \\ &+ 16R_4 + 8R_3 + 4R_2 + 2R_1 + R_0 \\ &- \left((8A_3 + 4A_2 + 2A_1 + A_0) \times (8B_3 + 4B_2 + 2B_1 + B_0)\right) \\ &- (128S_7 + 64S_6 + 32S_5 + 16S_4 + 8S_3 + 4S_2 + 2S_1 + S_0) \end{aligned}$$

B. MAC-Verifier

The input G to the algorithm is an AIG of a MAC[1] The algorithm returns `TRUE` if the SP for the circuit G is the zero polynomial after all the substitution steps are performed; otherwise it returns `FALSE`. Metrics, like the initial and the *Maximum Polynomial* (MaxPoly) size (corresponding to the peak memory needed) of the SP during backward rewriting and the time needed for the verification, are also provided. These will be used in the following for the definition of the scalability indicators.

To estimate the likelihood of a MAC architecture to be verifiable for larger bit sizes, we introduce two *scalability indicators* from the evaluation of the algorithm, where we

[1] `MAC-Verifier` can also handle isolated ADD and MUL circuit. This is supported to verify also MAC components as standalone units.

Algorithm 1: MAC-Verifier

Input : AIG G
Result: TRUE if SP is empty, FALSE otherwise
// preparation
$SP \leftarrow GenerateSP(G)$;
$AB, N \leftarrow AtomicBlockDetection(G)$;
$Cones \leftarrow FindCones(G, AB, N)$;
// global backward rewriting
foreach $cone \in cones$ **do**
  for $candidate \in cone$ **do**
    $SP_{old} \leftarrow SP$;
    $SP \leftarrow Substitute(SP, candidate, AB)$;
    if $SP \leq threshold$ **then**
      break;
    else
      $SP \leftarrow SP_{old}$;
    end
  end
end
// result evaluation
if *size(SP) == 0* **then**
  return TRUE
else
  return FALSE
end

first introduce the indicators by presenting the general idea and describe them in further detail below:

1) Starting from the initial SP size, i.e. SP_{init}, we check for the maximum peak in memory consumption determined by MaxPoly. This is put in relation to the memory needed for the SP at the start of the run:

$$\varphi_{s_1} = \frac{max(|SP_i|)}{|SP_{init}|} \tag{5}$$

2) While φ_{s_1} only consider the maximum peak, also the memory consumption of the entire run can be monitored:

$$\varphi_{s_2} = \frac{\int_{i=0}^{i=ts} |SP_i| - |SP_{init}| \quad [|SP_i| > |SP_{init}|]}{\int_{i=0}^{i=ts} |SP_{init}| + \int_{i=ts+1}^{i=N-1} |SP_i|} \tag{6}$$

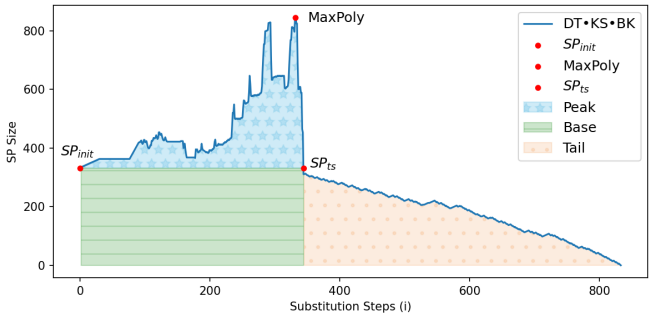


Fig. 6: Trajectory curve of the substitution steps for $DT \bullet KS \bullet BK$ representing the indicators

The memory consumption of one verification run can be shown as follows: consider the trajectory curve of the substitution steps for one example, i.e. $DT \bullet KS \bullet BK$, in Fig. 6. The substitution step i is presented on the x-axis and the corresponding SP size on the y-axis. Key points in the substitution trajectory, like e.g. SP_{init}, MaxPoly and tail start (SP_{ts}), are highlighted.

The φ_{s_1} indicator shows the maximum possible growth of the SP size over time during the verification process. The maximum peak in the SP corresponds to the memory consumption of the `MAC-Verifier`. By this, φ_{s_1} only considers one point in time of the whole verification run, namely when the maximal memory is needed. In contrast φ_{s_2} considers the entire run and gives the ratio of the peak area (cf. Peak), where the *area* denotes the memory usage over time, to the remaining area under the trajectory curve under Base and Tail (see Fig. 6). Here ts is the last point in the substitution, where SP is larger in size than SP_{init} (shown as SP_{ts}). For this example the values of the indicators are $\varphi_{s_1} = 2.55$ and $\varphi_{s_2} = 0.236$, respectively.

TABLE I: MAC Verification Overview

	MSA							
Bits	BK	CS	CL	KS	LF	RC	SE	Total
8-bit	28/28	26/28	28/28	28/28	28/28	28/28	28/28	194/196
16-bit	13/28	0/28	13/28	13/28	12/28	13/28	13/28	77/196
32-bit	12/28	0/28	12/28	12/28	12/28	12/28	12/28	72/196
64-bit	12/28	0/28	12/28	12/28	12/28	11/28	11/28	70/196
128-bit	12/28	0/28	11/28	12/28	12/28	10/28	10/28	67/196

C. MAC Verification Results

The SCA-based `MAC-Verifier` is implemented using C++. All experiments are carried out on an Intel(R) Xeon(R) CPU E5-2630 v3 processor running at 2.4 GHz and 64 GB of memory. Benchmarks are generated using `MAC-Gen` for all possible MAC architectures. In this subsection we evaluate our `MAC-Verifier` tool for all the 196 MAC architectures generated with bit sizes ranging from 8 to 128. A timeout of 30 minutes is considered for all runs.

Table I summarizes the results. The first column presents the bit size of the MAC, the next 7 columns the architecture of the *MAC Stage Adder* (MSA), and the last column the total number of verified MAC architectures. It can be observed that for 8-bit size roughly all MAC architectures are verifiable, but from 16-bits onward only approximately 40% of the circuits can be verified. In the following we have a closer look at the architectures and classify them according to their verifiability.

Table II presents the number of **non-verifiable** 16-bit MAC architectures grouped by the PPA of the multiplier for the MSA. The first column shows the various PPAs for the multiplier and the bit sizes. The next seven columns show the various MSAs and the last column shows how many MACs are non-verifiable. Each cell lists a set of non-verifiable FSA by name for each MSA. If the cell contains **None**, it means that all circuits are verifiable. And when the cell contains **All**, it signifies that none of the circuits could be verified within the given time limit. In all other cases the names of the FSA are provided for which the verification **failed**. The results clearly indicate that multipliers using `AR` and `DT` are

TABLE II: Non-Verifiable MAC Architectures Grouped by PPA and Bit-size

Bits	PPA	MSA BK	CS	CL	KS	LF	RC	SE	Fail/ Total
8	AR	None	None	None	None	None	None	None	0/49
	CWT	None	KS	None	None	None	None	None	1/49
	DT	None	None	None	None	None	None	None	0/49
	WT	None	KS	None	None	None	None	None	1/49
16	AR	None	All	None	None	CS	None	None	8/49
	CWT	All	All	All	All	All	All	All	49/49
	DT	CS	All	CS	CS	CS	CS	CS	13/49
	WT	All	All	All	All	All	All	All	49/49
32	AR	CS	All	CS	CS	CS	CS	CS	13/49
	CWT	All	All	All	All	All	All	All	49/49
	DT	CS	All	CS	CS	CS	CS	CS	13/49
	WT	All	All	All	All	All	All	All	49/49
64	AR	CS	All	CS	CS	CS	CS	CS	13/49
	CWT	All	All	All	All	All	All	All	49/49
	DT	CS	All	CS	CS	CS	CS,KS	CS,KS	15/49
	WT	All	All	All	All	All	All	All	49/49
128	AR	CS	All	CS	CS	CS	CS,KS	CS,KS	15/49
	CWT	All	All	All	All	All	All	All	49/49
	DT	CS	All	CS,KS	CS	CS	CS,KS	CS,KS	16/49
	WT	All	All	All	All	All	All	All	49/49

the best candidates among the PPAs regarding verifiability. All FSAs are suitable for verification except the *Carry Skip Adder* (CS) and for some cases the *Kogge-Stone Adder* (KS). For some specific combinations of multipliers and adders used for MAC, there is an explosion of the intermediate size of the SP. Referring to Table II it can be clearly inferred that most of the MACs using `CS` result in timeout for `AR` and `DT`. Additionally, the `KS` adder is problematic for `RC`, `SE` and `CL` MSA in many cases.

To provide some further insight on the verification runs, the substitution trajectory curves are shown in Fig. 7 from 8 to 128-bits for $AR \bullet SE \bullet RC$ and $DT \bullet KS \bullet RC$. The y-axis shows the memory consumption starting from the SP over the substitution steps along the x-axis. The blue curve shows that starting from the SP_{init} during the backward substitution the memory decreases. Thus the verification can be carried out efficiently. But for the orange curve, the memory increases during the run. This can already be seen in the 8-bit case and becomes worse with increasing bit-width. Beside the curves, also the values for φ_{s_1} and φ_{s_2} are reported. A value close to 1 (0) for φ_{s_1} (φ_{s_2}) is a clear indicator regarding scalability of the verification process. As can already be seen here from the φ-values of smaller bit-width verifiability results can be extrapolated.

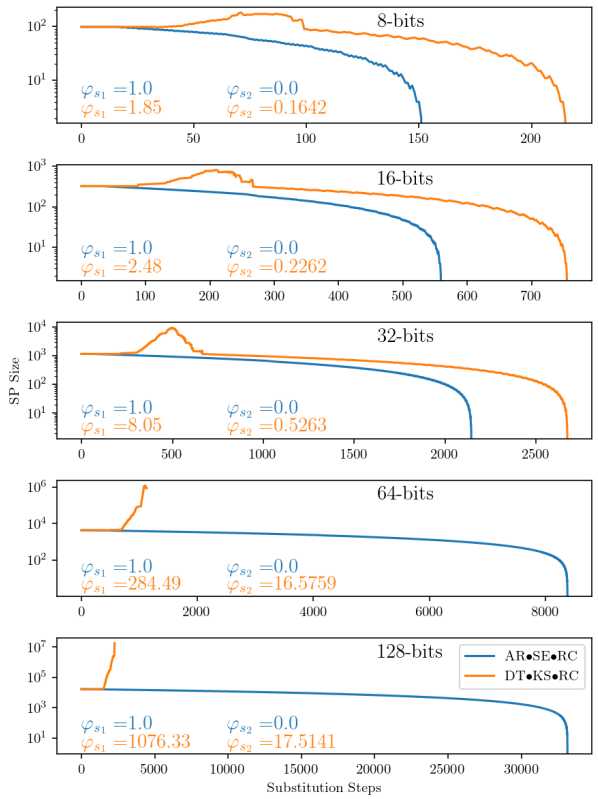


Fig. 7: Substitution Trajectory Curves from 8 to 128-bits for two Examples

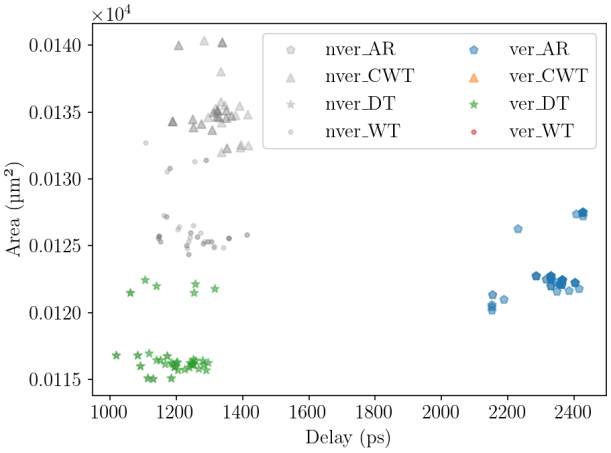


Fig. 8: Verifiable MAC Architectures for 16-bits

V. Scalable Verification for MAC

From the verification results presented in the previous section, now the classification and analysis of scalable MAC architectures can be derived.

In Section III only area and delay have been considered, while verifiability was not addressed (cf. Fig. 5). If the same experiment is re-run, but only the verifiable MAC instances are shown, the results in Fig. 8 are obtained. All non-verifiable (*nver*) instances are shown in gray. From Fig. 8 it can be inferred that there is no locality in terms of area and delay for the verifiability property. The designs that are all verifiable do not necessarily have the largest area or the smallest delay. This is a somehow surprising result, since typically highly parallel circuits of small depth, i.e. fast circuits, tend to be hard to verify. But some particular MAC architectures have lower φ_{s_1} and φ_{s_2}. From the above observations ten optimal MAC architectures in terms of area and delay are derived. These are taken by calculating the Euclidean distance from the origin to each MAC, described in terms of a normalized point (area,

TABLE III: Area and delay minima for 16-bit MAC

MAC Circuit	Area (μm^2)	Delay (ps)	MaxPoly	φ_{s_1}	φ_{s_2}	VT (s)
DT • CS • KS	115.984	1091	4751520	13460.40	215.69	T.O.
DT • RC • KS	115.984	1091	504	1.43	0.11	0.290
DT • CS • BK	115.095	1114	4751520	14311.81	328.02	T.O.
DT • RC • BK	115.095	1114	449	1.35	0.05	0.086
DT • CL • CS	116.815	1018	13404881	21580.73	140.80	T.O.
DT • CL • RC	116.815	1018	351	1.09	0.01	0.183
DT • CS • LF	115.065	1130	4751520	14311.81	333.61	T.O.
DT • RC • LF	115.065	1130	468	1.41	0.07	0.090
DT • KS • CS	116.800	1082	148779	328.25	46.20	T.O.
DT • KS • RC	116.800	1082	797	2.48	0.23	0.348

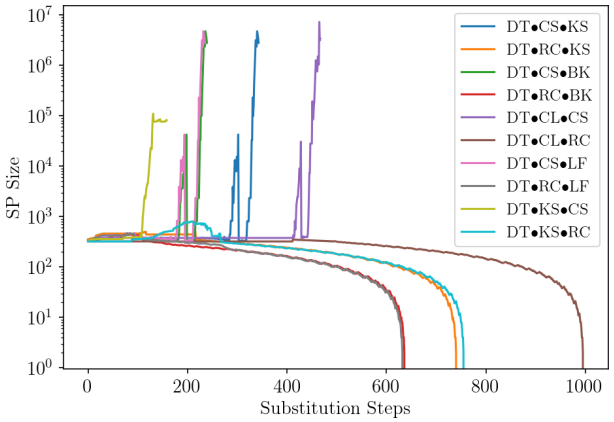


Fig. 9: Substitution Trajectory for 16-bit MAC

delay) for x- and y-axes. All the optimal area-delay MAC designs for 16-bit sizes are presented in Fig. 9. The diagram shows the SP size on the y-axis and the substitution steps on the x-axis. It turns out that five out of the ten best architectures can be verified within the given time limit. From the diagram the five non-verifiable architectures can easily be identified, since all of them have large spikes and result in timeouts.

Table III provides additional details about the *Area* and *Delay* as well as the maximal memory needed during a run (MaxPoly). Furthermore φ_{s_1}, φ_{s_2} and the *Verification Time* (VT) in seconds are reported. It can be seen that the MAC architectures that timed out have significantly higher values of MaxPoly, φ_{s_1}, φ_{s_2}, but the area and delay are comparable with others. Also a clear correspondence between φ_{s_1} andφ_{s_2} with respect to VT can be observed. The higher the VT, the larger are the values of φ_{s_1} and φ_{s_2}. From these findings we can conclude that both φ_{s_1} and φ_{s_2} provide a very good estimate of verifiability for larger designs. Those with the lowest φ_{s_1} and φ_{s_2} provide MAC architectures that have the highest likelihood to be verifiable for larger bit size.

A. Scalability Analysis for higher-bit MAC

Considering the verification results presented earlier, a class of MAC architectures that has the highest potential to scale for larger bit sizes has been identified. From the analysis of smaller MAC designs, now the design strategy for larger designs can be derived. Firstly, the analysis of the relevant parameters are performed and then the scalability indicators φ_{s_1} and φ_{s_2} are determined. Based on these, a classification regarding verifiability is done. Secondly, we combine the verifiable design with the area-delay trade-off. Not only the MAC designs which are area-delay efficient and verifiable are considered, but also the designs which have lower φ_{s_1} and φ_{s_2} values are chosen.

TABLE IV: Result for scalable MAC Design

Bit	MAC	φ_{s_1}	φ_{s_2}	MaxPoly	VT (s)	Commercial (s)
8	$AR \bullet SE \bullet RC$	1.00	0.000	98	0.012	47.38
	$AR \bullet RC \bullet RC$	1.00	0.000	98	0.013	47.36
	$DT \bullet RC \bullet BK$	1.43	0.085	152	0.015	21.84
	$DT \bullet RC \bullet LF$	1.50	0.130	159	0.017	21.91
16	$AR \bullet SE \bullet RC$	1.00	0.000	322	0.051	T.O.
	$AR \bullet RC \bullet RC$	1.00	0.000	322	0.052	T.O.
	$DT \bullet RC \bullet BK$	1.35	0.053	449	0.086	T.O.
	$DT \bullet RC \bullet LF$	1.41	0.066	468	0.090	T.O.
32	$AR \bullet SE \bullet RC$	1.00	0.000	1154	0.393	T.O.
	$AR \bullet RC \bullet RC$	1.00	0.000	1154	0.386	T.O.
	$DT \bullet RC \bullet BK$	1.99	0.049	2305	0.686	T.O.
	$DT \bullet RC \bullet LF$	2.05	0.059	2375	0.773	T.O.
64	$AR \bullet SE \bullet RC$	1.00	0.000	4354	4.329	T.O.
	$AR \bullet RC \bullet RC$	1.00	0.000	4354	4.371	T.O.
	$DT \bullet RC \bullet BK$	1.89	0.027	8225	7.473	T.O.
	$DT \bullet RC \bullet LF$	1.99	0.065	8679	9.013	T.O.
128	$AR \bullet SE \bullet RC$	1.00	0.000	16898	67.889	T.O.
	$AR \bullet RC \bullet RC$	1.00	0.000	16898	67.762	T.O.
	$DT \bullet RC \bullet BK$	1.64	0.021	27715	119.166	T.O.
	$DT \bullet RC \bullet LF$	1.76	0.050	29813	148.031	T.O.
256	$AR \bullet SE \bullet RC$	1.00	0.000	66562	2066.700	T.O.
	$AR \bullet RC \bullet RC$	1.00	0.000	66562	2076.930	T.O.
	$DT \bullet RC \bullet BK$	1.43	0.016	95093	3486.650	T.O.
	$DT \bullet RC \bullet LF$	1.98	0.030	131832	3825.480	T.O.
512	$AR \bullet SE \bullet RC$	1.00	0.000	264194	41661.800	T.O.
	$AR \bullet RC \bullet RC$	1.00	0.000	264194	41429.000	T.O.
	$DT \bullet RC \bullet BK$	1.28	0.004	338227	82098.200	T.O.
	$DT \bullet RC \bullet LF$	1.96	0.015	518515	87809.700	T.O.

The MAC architectures selected for scalability are taken from the set based on fast verifiability, area-delay trade-off and scalability indicators. The list of selected MAC architectures are given below:

Fast verifiable designs
1) $AR \bullet SE \bullet SE$
2) $AR \bullet RC \bullet RC$

Optimal area-delay designs
3) $DT \bullet RC \bullet BK$
4) $DT \bullet RC \bullet LF$

The results for these four identified scalable MAC designs are presented in Table IV, where for this experiments with several hundred bits the runtime limit was increased to 100000 seconds (corresponding to approx. one day). The first column presents the bit size of the MAC and the the second the

MAC architecture. The next two columns are φ_{s_1}, φ_{s_2} and the next two columns present the measured results (MaxPoly) and the *Verification Time* (VT). The last column presents the verification time of a commercial tool (see below). The table presents the results for 8, 16, 32, 64, 128, 256, and 512-bits. This shows that scalable designs of up to several hundred bits can be fully verified. To demonstrate the quality of the approach, we have also used a commercial tool for verifying the MAC architectures. The four considered MAC architectures are compared against a golden MAC. The tool failed to verify any architectures larger than 8-bits.

From Table IV it can be seen, that the lower the value of φ_{s_1} and φ_{s_2}, the lower is the verification time and the smaller is MaxPoly also for scaled designs. Further from the MaxPoly column it can be observed that the two fastest verifiable designs are identical, whereas the two area-delay efficient MACs differ in their MaxPoly size. $DT \bullet RC \bullet LF$ has consistently larger MaxPoly than $DT \bullet RC \bullet BK$ or the other two designs. The verification time shows a similar behavior. $AR \bullet SE \bullet SE$ and $AR \bullet RC \bullet RC$ needs roughly the same time, whereas $DT \bullet RC \bullet LF$ needs consistently more time than $DT \bullet RC \bullet BK$. Among all four designs $DT \bullet RC \bullet LF$ has the highest φ_{s_1} and φ_{s_2} for all bit sizes. Therefore, it is not surprising that the MaxPoly and the verification time are the highest among all.

Our anticipation of the forecast by looking at the smaller design is found to be very effective. From the analysis a class of MACs has been identified that is the fastest, most area efficient and at the same time fully verifiable.

VI. Conclusion

In this paper, we present the first automated formal verification for MAC using SCA. The ForMAt framework consists of the tools `MAC-Gen` and `MAC-Verify` for generation and verification of MAC units, respectively. From our experimental study on lower-bit MAC architectures, two scalability indicators are derived which enable us to extrapolate design information. This helps the designer in choosing the right scalable and verification friendly MAC architecture. While φ_{s_2} monitors the entire verification run more precisely, it has been shown that for MAC verification, also the simpler indicator φ_{s_1} provides very accurate information. By this, the verifiability aspect can be combined with the area-delay trade-off. Finally, MAC designs can be identified which are efficient in terms of area and delay, verifiable and at the same time scalable.

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] Y.-H. Seo and D.-W. Kim, "A New VLSI Architecture of Parallel Multiplier–Accumulator Based on Radix-2 Modified Booth Algorithm," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 2, pp. 201–208, 2010.

[2] K. Neelima and Satyam, "High Performance Variable Precision Multiplier and Accumulator Unit for Digital Filter Applications," in *2021 IEEE International Conference on Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER)*, 2021, pp. 209–212.

[3] 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.

[4] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Transactions on Computers*, vol. 35, no. 8, pp. 677–691, 1986.

[5] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proceedings of the 38th annual Design Automation Conference*, 2001, pp. 530–535.

[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. 1–8.

[7] 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.

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

[9] 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 Conference*, 2019, pp. 185:1–185:6.

[10] C. Scholl and A. Konrad, "Symbolic Computer Algebra and SAT Based Information Forwarding for Fully Automatic Divider Verification," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[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] A. Konrad and C. Scholl, "Symbolic Computer Algebra for Multipliers Revisited-It's All About Orders and Phases," in *2024 Formal Methods in Computer-Aided Design (FMCAD)*, 2024.

[13] 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.

[14] 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 *Design, Automation and Test in Europe*, 2024, pp. 1–6.

[15] L. Weingarten, K. Datta, and R. Drechsler, "Towards Polynomial Formal Verification of Neuromorphic Architectures," in *International Symposium on Electronic System Design*. IEEE, 2024, pp. 1–6.

[16] ——, "Late Breaking Results: Towards Efficient Formal Verification of Dot Product Architectures," in *Design, Automation and Test in Europe*. IEEE, 2025, pp. 1–2.

[17] A. Mishchenko, S. Chatterjee, and R. K. Brayton, "DAG-aware AIG rewriting a fresh look at combinational logic synthesis," in *Proceedings of the 43rd Design Automation Conference, DAC 2006, San Francisco, CA, USA, July 24-28, 2006*, E. Sentovich, Ed. ACM, 2006, pp. 532–535. [Online]. Available: https://doi.org/10.1145/1146909.1147048

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

[19] "Genus(TM) Synthesis Solution - Cadence Design Systems, Inc." https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/synthesis/genus-synthesis-solution.html, 2024.

[20] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "ASAP7: A 7-nm finFET predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S002626921630026X