Formal Verification of Truncated Multipliers using Algebraic Approach and Re-synthesis

Tiankai Su, Cunxi Yu, Atif Yasin, Maciej Ciesielski
ECE Department, University of Massachusetts, Amherst, USA
tiankaisu@umass.edu, ycunxi@umass.edu, ayasin@umass.edu, ciesiel@ecs.umass.edu

Abstract - This paper presents a formal approach to verify multipliers that approximate integer multiplication by output truncation. The method is based on extracting polynomial signature of a truncated multiplier using algebraic rewriting. To efficiently compute the polynomial signature, a multiplier reconstruction approach is used to construct the precise multiplier from the truncated one. The method consists of three basic steps: 1) determine the weights (binary encoding) of the output bits; 2) reconstruct the truncated multiplier using functional merging and re-synthesis; and 3) construct the polynomial signature of the resulting circuit. The method has been tested on multipliers up to 256 bits with three truncation schemes: *Deletion*, *D-truncation*, and *Truncation with Rounding*. Experimental results are compared with the state-of-the-art SAT, SMT, and computer algebraic solvers.

Keywords— Formal Verification; Truncated Multipliers; Approximate Computing; Computer Algebra.

I. Introduction

Multiplication is one of the major operations used in modern digital signal processing (DSP) applications. Performance and energy efficiency of multiplier circuits are critically affected by long carry chain in the adder circuit tree. Both can be significantly improved using more efficient multiplier circuits. To reduce area and power of a multiplier, a broad concept of *truncation* or *approximate computing* is used, which utilizes an inexact computation used in error-tolerant applications. Truncated multipliers discard some of the partial products (PPs) or truncate some of the output bits to reduce the circuit complexity, area, and power at a price of accuracy.

An n-bit precise multiplier has $2n$ output bits. In contrast, truncated multipliers have fewer output bits, with output bits close to the least significant bit (LSB) typically being truncated. Those truncation schemes may result in an unacceptable error rate, hence a correction circuit [1] is usually added at a later stage. *FIR Filter* is one example where truncated multipliers are widely utilized. Although many applications benefit from those truncated multipliers for power efficiency and performance, formal verification problem of those designs has not been adequately addressed.

Verification of precise arithmetic circuits is solved using *arithmetic combinational equivalence checking* (ACEC) [2]. Several approaches have been applied to check an arithmetic circuit against its functional specification, including *canonical diagrams*, Boolean satisfiability (SAT), or satisfiability modulo theories (SMT). However, those methods are not efficient in solving ACEC problems because of "bit-blasting" of gate-level arithmetic circuits [3]. Recently, computer algebraic approaches have been extensively studied for solving ACEC problems in both integer and finite field domain, and have been successfully applied to large arithmetic circuits, such as 512-bit multipliers [2][3][4][5].

However, these techniques are not efficient in verifying truncated arithmetic circuits. During algebraic rewriting the size of intermediate polynomials, which is manageable in conventional arithmetic circuits, can grow exponentially large in truncated circuits, such as multipliers. This is because truncation prevents monomial cancellations that naturally occur during the rewriting of a complete circuit [6]. For example, in a half-adder (HA) block, the polynomials are linear ($2C$+S), whereas the algebraic models of XOR (sum) and AND (carry) gates in the HA are nonlinear. Once the XOR gate of the sum ($a \oplus b = a + b - 2ab$) is removed during the truncation process, the non-linear terms ($2ab$) remain in the polynomial expression, while they are normally canceled in a regular half-adder: $2C+S = 2\cdot(ab)+(a+b-2ab) = a+b$. This causes an exponential increase in the size of intermediate polynomials, resulting in memory explosion for large circuits.

In this paper we propose an algebraic approach with re-synthesis that reconstructs the complete multiplication of the truncated multiplier, followed by algebraic rewriting of the reconstructed multiplier. Specifically, we make the following novel contributions:

- A framework using re-synthesis technique is presented to reconstruct a complete multiplication for arbitrarily truncated multipliers.
- A complete methodology that efficiently determines the correctness of the given circuit is proposed.
- Analysis of arithmetic verification of truncated multiplier using comprehensive truncation schemes is given, including Deletion, D-truncation and Rounding [7].

II. Background

Several approaches have been applied to check an arithmetic circuit against its functional specification, including *canonical diagrams*, *satisfiability* theories, *theorem proving* and others. However, building canonical diagrams, such as BDDs [8], BMDs [9] [10], or TEDs [11], for large and arithmetic circuits is very expensive because of large memory usage.

2159-3477/17 $31.00 © 2017 IEEE
DOI 10.1109/ISVLSI.2017.79

Alternatively, ACEC problems can be solved using Boolean satisfiability (SAT) or satisfiability modulo theories (SMT), using SAT and SMT solvers, such as MiniSAT [12], Lingeling [13], Chaff [14], Boolector [15], and others. These SAT solvers have been widely used in industry because of their capability to solve equivalence checking problem. Other tools have been developed to ease this verification problem [16]. However, even the best SAT solvers cannot solve verification problem of a 16-bit truncated multiplier by just using equivalence checking. High memory usage prevents canonical diagrams from solving the main problem of truncated multipliers verification.

A. Computer Algebraic Approach

Computer algebra method is believed to be the best technique for solving arithmetic verification problems. Using computer algebra methods, the verification problem is typically formulated as a proof that the implementation satisfies the specification [4][5][3][2]. The goal is to prove that the circuit implementation satisfies the specification by performing a series of divisions of the specification polynomial F by the implementation polynomials $B = \{f_1, ..., f_s\}$, that represent components of the implementation circuit. The polynomials $f_1, ..., f_s$ are called the generators of the ideal J. Given a set $f_1, ..., f_s$, a set of all the simultaneous solutions to a system of equations $f_1 = 0, ..., f_s = 0$ is called variety, $V(J)$. Verification problem is then formulated as checking if the specification F vanishes on $V(J)$. If polynomial f contains some term t that is divisible by the leading term $lt(g)$ of polynomial g, then the division of f by g gives a remainder polynomial $r = f - \frac{lt(f)}{lt(g)} \times g$. In this case, we say that f reduces to r modulo g, denoted $f \xrightarrow{g} r$. Techniques based on *Gröbner Basis* demonstrate that this approach can transform the verification problem to *membership testing* of the specification polynomial in the ideals [5]. Some modifications have been addressed in [2] to improve the efficiency of the process. A different approach to arithmetic verification of gate-level circuits called *function extraction* [3] is discussed next.

B. Function Extraction

Function Extraction is an arithmetic verification method originally proposed in [3] for integer arithmetic circuits in $\mathbb{Z}_{2^m}$. It extracts a unique bit-level polynomial function implemented by the circuit directly from its gate-level implementation. Extraction is done by *backward rewriting*, i.e., transforming the polynomial representing encoding of the primary outputs (called the *output signature*) into a polynomial at the primary inputs (the *input signature*). This technique has been successfully applied to large integer arithmetic circuits, such as 512-bit integer multipliers, benefiting from a large number of polynomial reductions obtained during rewriting [6]. A similar approach has also been applied to Galois Field arithmetic circuits GF(2^m), which offer an inherent parallelism that can be exploited in backward rewriting [17]. Although those works showed good performance in solving arithmetic verification problems, they still suffer from potential polynomial explosion problem if the arithmetic function is not complete, i.e., if some output bits have been removed. In the rest of the paper, we will show how to apply function extraction to truncated multipliers with AIG-based functional merging.

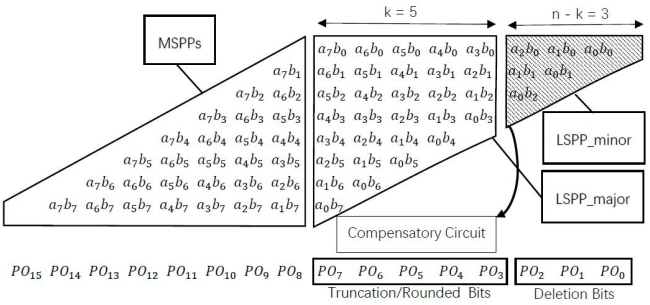

Fig. 1: Partial product array of a 8-bit multiplier.

C. Formal Truncation Schemes

Three formal truncation schemes used to design truncated multipliers are: Deletion, D-truncation and Rounding. Fixed-width multipliers are the prevalently used truncated multipliers. An example of such a multiplier is shown in Fig. 1. Normally, $2n$ output bits are generated in a regular $n \times n$ multiplier, whereas a fixed-width truncated multiplier only computes n most significant bits (MSBs). In this work, we analyze such a fixed-width multiplier. In this example, the inputs $a[n\text{-}1{:}0]$ and $b[n\text{-}1{:}0]$ are assumed to be two integer inputs. The partial products are divided into two subsets:

1) The $most\ significant\ partial\ products$ (MSPPs), corresponding to n MSBs; and,
2) The $least\ significant\ partial\ products$ (LSPPs), further subdivided in LSPP_major and LSPP_minor, corresponding to the k most significant columns of LSPPs and the remaining $(n–k)$ columns respectively.

Deletion: This scheme discards the partial products in LSPP_minor, which have a small impact on the accuracy of the result. Those PPs are dropped to prevent the carry chain from propagating further. This decreases the size of the carry chain, which reduces the delay and power. Even though LSPP_minor have a smaller contribution to the accuracy of the MSBs, the error of just discarding them can be high, in the worst case is $7 * 2^8$ for an 8-bit fixed-width multiplier. To rectify this, a compensatory circuit [1][18] is typically added after deletion. From the verification point of view, such a compensatory circuit can be presented as a separable block, described by a set of polynomial(s) D in terms of the primary inputs. The most crucial thing is to verify partial products and adder tree of a multiplier.

D−truncation: In this scheme some output bits computed by LSPPs are truncated without modifying the entire carry chain. In contrast to the Deletion scheme, which improves performance and saves power, D-truncation only removes those gates that are directly connected to the truncated bits. The purpose of D-truncation is to manage the number of output bits, i.e., maintaining the number of input and output bits to be the same. Although D-truncation will not affect the accuracy

of the MSBs, the error due to losing the value in LSPPs is $(2^8 - 1)$ in the worst case.

Rounding scheme is often introduced to achieve further accuracy and truncation [19]. Instead of truncating all columns in LSPP_major, some of the most significant columns in LSPP_major are kept for the rounding circuit. The value of those columns in LSPP_major is rounded into MSPPs.

III. Implementation

In this section, we describe two our basic techniques, $functional\ merging$ and $re\text{-}synthesis$, used to verify truncated multipliers. We introduce a Partial Product Detector (PPD) which assigns correct weights to the respective output bits. In order to better understand how PPD works, we analyze in section III-A a truncated multiplier designed with only the Deletion scheme. In section III-B, we consider a truncated multiplier designed with only the D-truncation scheme to describe the detail of functional merging and re-synthesis. Lastly, we perform a comprehensive verification analysis where all truncation schemes are combined together. Fig. 3 shows the verification flowchart of our methodology applicable to general truncated multipliers. Two assumptions are made:

- The compensatory circuits for Deletion and Rounding can be extracted from the high-level synthesis netlist, and represented as polynomials in primary inputs.
- The bit position of primary inputs ($a[n\text{-}1:0]$, $b[n\text{-}1:0]$) is known, so that the weights of the partial products are also known.

A. Deletion Scheme Only

In the Deletion scheme, some of the partial products are removed to reduce circuit complexity at the expense of accuracy. For example, in Fig. 1, six of the partial products in LSPP_minor are removed, and the logic columns PO_0, PO_1 and PO_2 in shaded area will disappear. As a result, there is no carry feed into the PO_3 column. Hence, the remaining adder tree still executes complete additions (the remaining HA/FA blocks are complete). In this case, there is no polynomial explosion problem, since all additions preserve polynomial eliminations during the rewriting process. This means that function extraction is sufficient to extract the input signature of the circuit.

In Fig. 1, when the logic cones associated with LSPP_minor columns are removed, PO_3 corresponds to the current LSB. Hence, the weight of the current LSB that was 2^3 in the exact multiplier is shifted to 2^0. The computed input signature will never match the specification until all the output bits are assigned their original weight. Moreover, additional polynomials representing those removed partial products must be added. In this case, PPD perform the following functions (Algorithm 1):

- Assigns correct/original weight to each current output bit.
- Detects the deleted elements in LSPP_minor and generates additional polynomials.

In general, for an $N_a \times N_b$-bit multiplier, Algorithm 1 will search N_a+N_b logic columns. Specifically, for an 8-bit multiplier shown in Fig 1, it searches $2 \times 8 = 16$ logic columns,

Algorithm 1 Weight Determination and Signature Generation

Input: Gate-level netlist of truncated multiplier
Output: Correct weight of each output bit
Output: Additional polynomials representing the removed partial products

$\mathcal{PO}$={$PO_0, PO_1, ..., PO_c$}: current output bits of truncated multiplier
for $i \leftarrow 0$ **to** $2n-1$ **do**
  Create list$_i$ corresponding to logic column i
  for $j \leftarrow 0$ **to** i; $k \leftarrow 0$ **to** i; $j+k \leftarrow 0$ **to** i **do**
    Search partial product $a_j b_k$ in the netlist
    if partial product $a_j b_k$ exists **then**
      Save product $a_j b_k$ into list$_i$
    else Additional Polynomials ← Additional Polynomials + $2^i a_j b_k$
    end if
  end for
  Number_of_Total_PPs ← *Number_of_Total_PPs* + length(list$_i$)
  Rank_of_Logic_Column ← i
  Save Pair$_i$(*Rank_of_Logic_Column*, *Number_of_Total_PPs*)
end for
for $i \leftarrow 0$ **to** c **do**
  Search and count the number of PPs that current output bit PO_i depends on
  Match the count with *Number_of_Total_PPs* in Pairs
  return *Rank_of_Logic_Column*
  Assign correct weight $W_i \leftarrow 2^{Rank\_of\_Logic\_Column}$
end for
return Correct weights $\mathcal{W}$={$W_0, W_1, ..., W_c$} and Additional Polynomials

TABLE I: The relationship between *RLC* and *NTPP*
RLC: Rank of Logic Column, *NTPP* : Number of Total PPs

RLC	*NTPP*	*RLC*	*NTPP*	*RLC*	*NTPP*
0	0	5	15	10	48
1	0	6	22	11	52
2	0	7	30	12	55
3	4	8	37	13	57
4	9	9	43	14,15	58

PO_0 to PO_{15}. The results of applying Algorithm 1 to this example are shown in Table I.

To assign the correct weight for each output bit, we traverse and count all those PPs on which that output bit depends. This information is stored in Table I, which shows for each output bits (rank of logic column, RLC) the number of total PPs ($NTPP$) that this bit depends on. Since all PPs in the LSPP_minor logic were removed, $NTPP$ for $RLC = 0, 1, 2$ is 0. To find the weight of the current LSB of this truncated multiplier, we examine the PPs this bit depends on. In this case, four PPs have been found, namely $a_0b_4, a_1b_3, a_2b_2, a_3b_1, a_4b_0$. We then examine Table I to find which bit (RLC) depends on $NTPP = 4$. The table shows that $RLC = 3$, which means the current LSB of this truncated multiplier is the PO_3 of the complete multiplier. Hence, the correct weight for this bit must be 2^3.

In the same way, we can assign the correct weights to the other output bits. For example, if we find out that one of the output bits depends on 15 PPs, then the correct weight of that output bit is 2^5, since RLC for $NTPP = 15$ is 5. Furthermore, those 15 partial products consist of 6 PPs in logic column PO_5, 5 PPs in PO_4 and 4 PPs in PO_3.

Algorithm 1 explains a general idea of how the PPD works. However, a special case need to be discussed. Notice that the two MSBs PO_{14} and PO_{15} depend on the same number of partial products, that is 58, as shown in Table I. In such a one-to-two mapping problem, PPD will determine the correct weight according to the information of the circuit. That is, the

output bit with the higher rank is assigned the higher weight, that is 2^{15}, and the other output bit as 2^{14}.

Once all the output bits have been assigned their correct weights, we can then apply function extraction to compute input signature. Notice that the boundary between LSPP_minor and LSPP_major does not necessarily have to be straightforward (along the column lines). As long as the correct weight of each output bit is assigned and the LSPP_minor has been detected, the calculated signature will always be correct.

While searching each logic column, the discarded/undetected partial products will be added as additional polynomial at the end of signature calculation, with the corresponding coefficients. In this case, the additional polynomial will be $2^2(a_2b_0 + a_1b_1 + a_0b_2) + 2^1(a_1b_0 + a_0b_1) + 2^0a_0b_0$. The polynomial of the compensatory circuit D also need to be considered, if it exists. The size of the compensatory circuit depends upon the number of partial products removed. The worst case size occurs when $k = 0$, that is, when all the LSPPs are removed. However, the bigger the compensatory circuit, the harder the verification process is. In the worst case scenario, most of the time is spent on computing D. Therefore, if the size of the compensatory circuit is too big, we need to remove it before the verification process.

Finally, we compare **1)** the calculated polynomial of the truncated circuit expanded by additional polynomial, with **2)** the sum of full multiplier specification and polynomial D. This is Because compensatory circuit is included in the truncated circuit. This is trivial when there's no compensatory circuit ($D = 0$). If the two polynomials are equal, the given circuit is proved to be a Deletion-based truncated multiplier.

B. D-truncation scheme only

D-truncation is the process in which some of the output bits are removed without changing the functionality of the remaining output bits. A fixed-width multiplier can be built by simply applying D-truncation scheme to a regular multiplier. For example, in Fig. 1, assuming that $k = 8$, then no partial product is removed and 8 LSBs are truncated. Although the functionality of the remaining 8 MSBs has not changed, the entire structure of adder tree is no longer complete. For example, assume that one output bit is computed by the XOR gate of a HA block. When this bit is truncated, the XOR gate will be automatically removed from the circuit because it is redundant. However, the AND gate will remain to preserve the carry chain.

In this example, the XOR gates directly connected to PO_2 to PO_7 will disappear due to D-truncation. The remaining AND gates make the structure of adder tree incomplete. The nonlinear terms of polynomials, which were supposed to be canceled, will propagate during the backward rewriting. This leads to an exponential increase in memory size, potentially ending up in a memory explosion. In other words, we cannot directly apply function extraction to the circuit. To efficiently apply function extraction, our approach transforms the incomplete function into a complete one. This is achieved by functional merging, followed by re-synthesis of the combined circuit by ABC [20].

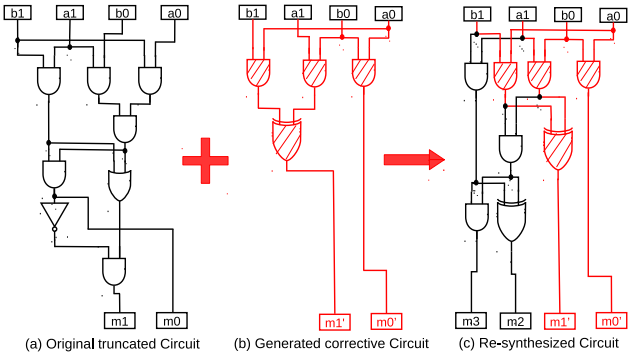

Fig. 2: Functional Merging and Re-synthesis.

To better explain this idea, we use a 2×2 multiplier in Fig. 2 as an example. A 2-bit truncated multiplier (with two LSBs truncated) is presented in Fig. 2(a). Assume that Fig. 2(a) represents a circuit to be verified whether it is a 2-bit fixed-width multiplier. We use Algorithm 1 to determine the correct weights of this circuit. After reading the gate-level netlist, PPD assigns weight 2^3 to $m1$ and 2^2 to $m0$. Then, we generate a regular 2×2 multiplier and remove the output bits with weights 2^2 and 2^3. The generated circuit is shown in Fig. 2(b), whose weight of $m1'$ is 2^1 and $m0'$ is 2^0. Finally, we apply our scheme of functional merging and re-synthesis resulting in the circuit in Fig. 2(c). The red (shaded) part in this figure corresponds to the generated corrective circuit of Fig. 2(b).

If the re-synthesized circuit proves to be a 2×2 multiplier by algebraic polynomial rewriting, this means that the given circuit in Fig. 2(a) is indeed a truncated multiplier. This is because the circuit generated in Fig. 2(b) has a correct function of two LSBs in a regular 2×2 multiplier. If and only if the circuit in Fig. 2(a) has the function of two MSBs in a regular 2×2 multiplier, then the re-synthesized circuit can be a 2×2 multiplier.

In general, for a given circuit X that needs to be verified whether it is the D-truncation-based multiplier or not, we first calculate correct weights for each output bit by PPD. Then, we generate a regular multiplier and truncate those output bits that have the same weight as the ones in circuit X. The generated circuit should have the same number of input bits as circuit X. After functional merging and re-synthesis, the number of output bits in the re-synthesized circuit is equal to the sum of output bits in the given circuit X and the generated circuit, as shown in Fig. 2. Finally we apply function extraction (backward rewriting) to the re-synthesized circuit. If the calculated signature matches the specification of a regular multiplier, the functionality of circuit X as truncated multiplier is confirmed.

The goal of functional merging is to find the maximum functional similarity between the given circuit and the generated corrective circuit. The merging eliminates as much as possible incomplete addition in the given circuit to speed up function extraction. Resynthesis (using ABC) transforms the two circuits into *And-Inverter-Graphs (AIGs)* and performs sweeping, redundancy removal, common logic identification,

etc. If the two circuits (the original one and the reconstructed one) have been built in the same way, resynthesis in effect transforms the structure of the original circuit into the circuit performing a complete arithmetic function.

Despite ABC being the state-of-the-art synthesis and verification tool, it cannot solve the equivalent checking problem between two different 12-bit truncated multipliers. In contrast, our approach is scalable and can be used to verify truncated multipliers at least 256 bits. The success depends upon the local functional similarity between the original and the generated corrective circuit. Such a similarity assumption is acceptable, especially in industry. For example, designer of the fixed-width multiplier has access to the original regular multiplier. If we use the structure of the regular multiplier to build the generated circuit, as in Fig. 2(b), solving the problem will be very fast. This is also true even if the original circuit and the generated circuit are not built using the same algorithm but exhibit enough local functional similarity. The speed of applying function extraction to the re-synthesized circuit will be much higher than directly applying function extraction to the original circuit. This is because more nonlinear terms will be canceled during polynomial rewriting. However, we cannot reconstruct a circuit without considering the algorithm used to design it. For example, functional merging between a Booth-Multiplier and non-Booth multiplier will gain no speedup for function extraction and also cause more additions incomplete that makes the process fail.

Another significant contribution of this approach is that we can verify a multiplier whose arbitrarily output bits are truncated. For example in Fig. 2(a), PPD can assign the correct weight to each of the output bits. We can then generate a regular 2×2 multiplier and truncate those output bits that have the same weight as the one in the given circuit. Therefore, we don't need to know which output bits are truncated. Instead, we can automatically detect arbitrarily truncated bits and follow the same procedure as described earlier.

A special case in weight determination is when either the MSB or (MSB-1)th bit is truncated. Since these two bits depend on all the PPs, PPD cannot tell which one is truncated. In this case, we will generate two circuits: one circuit assuming that MSB is truncated, and the other circuit assuming that (MSB-1)th bit is truncated. In this case, both circuits are reconstructed, and only if either of them turns out to be a regular multiplier, we conclude that the given circuit is a truncated multiplier.

C. Deletion + D-truncation + Rounding

In this section, we discuss the case where a truncated multiplier is designed using all of the three truncation schemes, Deletion, D-truncation, and Rounding. Partial product detector (PPD), functional merging, and re-synthesis can still be used along with the function extraction to solve the verification problem. Fig. 3 shows the verification flow of our methodology applicable to the Deletion only, D-truncation only, and a combination of these two schemes.

An 8-bit fixed-width multiplier is used as an example for the analysis of these truncation schemes. Assume that six of

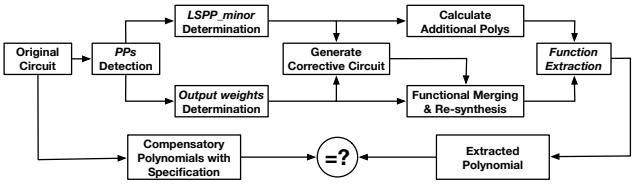

Fig. 3: Verification Flow dealing with all truncation schemes.

the partial products in LSPP_minor are removed, as shown in Fig. 1. Therefore, the output bits associated with PO_0, PO_1, and PO_2 will disappear due to Deletion. Then, five output bits which corresponding to logic columns PO_3 to PO_7 will be removed using D-truncation and Rounding scheme.

In the verification flow, PPD detects the elements in LSPP_minor and assigns a correct weight to each of the output bits. Then, we generate a regular multiplier which has the same number of inputs as the original circuit. An additional step, which is different from analysis in section II-B, is as follows. We discard the undetected PPs in the generated circuit so that the PPs in the generated circuit and the PPs in the original circuit are the same. After that, the generated circuit has 5 output bits, corresponding to PO_3 to PO_7, while the original circuit has 8 output bits, corresponding to PO_8 to PO_{15}. We then apply functional merging and re-synthesis to the combined circuit.

Finally, we calculate the polynomials as shown in Fig. 3, and compare the following two terms: **1)** the computed signature of the re-synthesized circuit with additional polynomials representing the undetected PPs (LSPP_minor); with **2)** the specification of the regular 8-bit multiplier enlarged by the polynomials of the compensatory circuits. Upon being equal, the given circuit is proved to be an 8-bit fixed-width multiplier. Experiments were performed on a Baugh-Wooley multiplier and a CSA array multiplier for up to 256 bits.

IV. Results

We performed our experiments on an Intel® Core™ CPU i5-3470 @ 3.20 GHz × 4 with 15.6 GB memory. The partial product detector (PPD) was implemented in Python. ABC [20] was used to implement the functional merging and re-synthesis. Experiments are performed on a Baugh-Wooley multiplier and a CSA multiplier for up to 256 bits. In particular, the Baugh-Wooley multiplier refers to a modified non-Booth unsigned multiplier using Baugh-Wooley scheme. CSA multiplier implemented in our experiment is an array based CSA multiplier, generated by ABC. Unless stated otherwise, the truncated multipliers in our experiment are obtained by removing 1/4 partial products and truncating up to half of the output bits (LSBs).

As analyzed in section III-B, direct application of function extraction to a truncated multiplier causes a memory explosion during polynomial rewriting. Table II makes a comparison between our scheme and the original function extraction [3] for execution time and memory consumption. Truncated multipliers used here are CSA based multipliers.

Function extraction [3] only succeed for truncated multipliers with the operand size up to 6. For the larger operand sizes,

TABLE II: Results and comparison with *Function Extraction*[3] using truncated CSA multipliers.

**TO* : Time out of 9,000sec, *MO* : Memory out of 10GB.

# bits	Function Extraction [3]		This Work	
	Runtime (sec)	Memory (MB)	Runtime (sec)	Memory (MB)
6	38.94	296.0	0.01	4.2
8	1174.4*	*MO*	0.01	4.8
16	928.2*	*MO*	0.05	7.3
32	796.3*	*MO*	0.25	17.2
64	631.4*	*MO*	1.05	57.3
128	470.9*	*MO*	4.40	220.4
256	286.1*	*MO*	18.94	880.1

TABLE III: Results and comparison with *ABC, SMT, and SAT* solvers using truncated Baugh-Wooley multipliers.

**TO* : Time out of 9,000sec, *MO* : Memory out of 10GB.

# bits	cec-ABC [20] (sec)	Lingeling [21] (sec)	Boolector [15] (sec)	This Work	
				Runtime (sec)	Memory (MB)
6	0.41	0.30	0.16	0.01	4.6
8	16.2	5.92	3.49	0.28	15.4
10	318.1	350.3	261.6	0.50	19.8
12	*TO*	*TO*	8746.4	0.81	24.1
16	*TO*	*TO*	*TO*	1.12	36.8
32	*TO*	*TO*	*TO*	3.87	50.9
64	*TO*	*TO*	*TO*	14.7	268.2
128	*TO*	*TO*	*TO*	68.2	757.6
256	*TO*	*TO*	*TO*	279.3*	*MO*

the system runs out of memory and results in an incomplete calculation. The runtime numbers labeled with * indicate the CPU time up to the moment when memory usage exceeds 10GB. In principle, given enough memory, function extraction might be able to output a correct result. However, such an experiment is not feasible or scalable. Hence, a direct implementation of function extraction to the truncated multiplier is neither efficient nor scalable. In contrast, our approach based on functional merging and re-synthesis is very fast, since we eliminate the incomplete additions in a truncated multiplier.

We also analyzed truncated multipliers designed with Baugh-Wooley scheme with bit-width varying from 6 to 256 bits. Comparison in Table III shows that our approach gives a much better performance than ABC and Lingeling SAT solver. ABC runtime exceeds 9,000 seconds in checking equivalence between a 12-bit truncated Baugh-Wooley multiplier and a CSA truncated multiplier generated by ABC. SAT solver in ABC, Lingeling solver, and SMT solver Boolector all fail for sizes greater than 12 bits for this experiment.

V. Conclusion

We presented an algebraic functional verification technique of truncated multipliers. The method is based on a combination of algebraic rewriting (function extraction) and functional merging using a multiplier reconstruction technique. We demonstrate that this approach can efficiently verify *Baugh-Wooley* and *CSA* truncated smultipliers, up to 256-bit, with arbitrary bits truncated. The experimental results show that our approach surpasses the state-of-the-art *SAT, SMT*, and computer algebraic solvers. This technique can be expanded to other arithmetic circuits.

Acknowledgment: This work has been funded by NSF grants, CCF-1319496 and CCF-1617708.

References

[1] N. Petra and A. G. M. Strollo, "Design of Fixed-Width Multipliers with Linear Compensation Function," *IEEE Trans. Circuits Syst.I: Regular Papers*, vol. 58, no. 5, pp. 947–960, May. 2011.

[2] A. Sayed-Ahmed, D. Große, U. Kühne, M. Soeken, and R. Drechsler, "Formal verification of integer multipliers by combining grobner basis with logic reduction," in *DATE'16*, 2016, pp. 1–6.

[3] M. Ciesielski, C. Yu, W. Brown, D. Liu, and A. Rossi, "Verification of Gate-level Arithmetic Circuits by Function Extraction," in *52nd DAC*. ACM, 2015, pp. 52–57.

[4] E. Pavlenko, M. Wedler, D. Stoffel, W. Kunz, A. Dreyer, F. Seelisch, and G. Greuel, "Stable: A new qf-bv smt solver for hard verification problems combining boolean reasoning with computer algebra," in *DATE*, 2011, pp. 155–160.

[5] J. Lv, P. Kalla, and F. Enescu, "Efficient Grobner Basis Reductions for Formal Verification of Galois Field Arithmatic Circuits," *IEEE Trans. on CAD*, vol. 32, no. 9, pp. 1409–1420, September 2013.

[6] C. Yu, W. Brown, D. Liu, A. Rossi, and M. J. Ciesielski, "Formal verification of arithmetic circuits using function extraction," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 35, no. 12, pp. 2131–2142, 2016.

[7] R. Devarani and C. S. Manikandababu, "Design and Implementation of Truncated Multipliers for Precision Improvement," *Computer Communication and Informatics (ICCCI), 2013 International Conference*.

[8] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. on Computers*, vol. 100, no. 8, pp. 677–691, 1986.

[9] R. E. Bryant and Y.-A. Chen, "Verification of Arithmetic Functions with Binary Moment Diagrams," in *DAC'95*.

[10] Y.-A. Chen and R. Bryant, "*PHDD: An Efficient Graph Representation for Floating Point Circuit Verification," School of Computer Science, Carnegie Mellon University, Tech. Rep. CMU-CS-97-134, 1997.

[11] M. Ciesielski, P. Kalla, and S. Askar, "Taylor Expansion Diagrams: A Canonical Representation for Verification of Data Flow Designs," *IEEE Trans. on Computers*, vol. 55, no. 9, pp. 1188–1201, Sept. 2006.

[12] N. Sorensson and N. Een, "Minisat v1. 13-a sat solver with conflict-clause minimization," *SAT*, vol. 2005, p. 53, 2005.

[13] A. Balint, A. Belov, and M. Heule, "Lingeling, Plingeling and Treengeling Entering the sat Competition 2013," *University of Helsinki*, vol. B-2012-2, pp. 51–52, 2013.

[14] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, 2001, pp. 530–535.

[15] A. Niemetz, M. Preiner, and A. Biere, "Boolector 2.0," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 9, 2015.

[16] A. Mishchenko, N. Een, R. K. Brayton, S. Jang, M. Ciesielski, and T. Daniel, "MAGIC: An Industrial-Strength Logic Optimization, Technology Mapping, and Formal Verification Tool," in *Intl. Workshop on Logic Synthesis*, June 2010, pp. 124–127.

[17] C. Yu and M. J. Ciesielski, "Efficient parallel verification of galois field multipliers," *ASP-DAC'17*, 2017.

[18] M. B. Sullivan and E. E. Swartzlander, "Truncated Error Correction for Flexible Approximate Multiplication," *Signals, Systems and Computers (ASILOMAR)*, vol. ACSSC.2012.6489023, p. 10.1109, Nov 2012.

[19] T. A. Drane, T. M. Rose, and G. A. Constantinides, "On the Systematic Creation of Faithfully Rounded Truncated Multipliers and Arrays," *IEEE Trans. on Computers*, vol. 63, no. 10, pp. 2513–2525, Oct. 2014.

[20] A. Mishchenko *et al.*, "Abc: A system for sequential synthesis and verification," *URL http://www. eecs. berkeley. edu/˜ alanmi/abc*, 2007.

[21] A. Biere, "Lingeling, plingeling and treengeling entering the sat competition 2013," *Proceedings of SAT Competition*, pp. 51–52, 2013.