Approximating Complex Arithmetic Circuits with Formal Error Guarantees: 32-bit Multipliers Accomplished

Milan Češka, Jiří Matyáš, Vojtech Mrazek, Lukas Sekanina, Zdenek Vasicek and Tomas Vojnar
Faculty of Information Technology, Centre of Excellence IT4Innovations
Brno University of Technology, Brno, Czech Republic
{ceskam, imatyas, imrazek, sekanina, vasicek, vojnar}@fit.vutbr.cz

***Abstract*—We present a novel method allowing one to approximate complex arithmetic circuits with formal guarantees on the approximation error. The method integrates in a unique way formal techniques for approximate equivalence checking into a search-based circuit optimisation algorithm. The key idea of our approach is to employ a novel search strategy that drives the search towards promptly verifiable approximate circuits. The method was implemented within the ABC tool and extensively evaluated on functional approximation of multipliers (with up to 32-bit operands) and adders (with up to 128-bit operands). Within a few hours, we constructed a high-quality Pareto set of 32-bit multipliers providing trade-offs between the circuit error and size. This is for the first time when such complex approximate circuits with formal error guarantees have been derived, which demonstrates an outstanding performance and scalability of our approach compared with existing methods that have either been applied to the approximation of multipliers limited to 8-bit operands or statistical testing has been used only. Our approach thus significantly improves capabilities of the existing methods and paves a way towards an automated design process of provably-correct circuit approximations.**

***Index Terms*—approximate computing, logical synthesis, genetic programming, formal methods**

I. Introduction

As many important applications are inherently error resilient, precision of the involved computations can be traded for improved energy efficiency, performance, and/or chip area. Various approaches exploiting this fact have been developed in recent years and presented under the umbrella of the so-called *approximate computing* [1]. These approximations can be conducted at different system levels with circuit approximation being one of the most popular.

Circuit approximation techniques can be classified into two main groups: (1) *Frequency/voltage over-scaling* where timing-induced errors can appear as the circuit is operated on a higher frequency or lower voltage than the nominal value. (2) *Functional approximation* where the original circuit is replaced by a less complex one which exhibits some errors but improves non-functional circuit parameters such as power consumption or chip area. We only deal with the latter approach in this paper. Circuit approximation can be formulated as a multi-objective optimization problem where the error and non-functional circuit parameters are conflicting design objectives. Since the resulting approximate circuits are common circuits, they can be implemented using the standard circuit design flow.

We focus on *approximate arithmetic circuits* (AACs) because they are frequently used in key applications relevant for approximate computing. Prominent examples are signal, image, and video processing circuits (such as filters, discrete transforms, and motion estimation blocks [2]), or the multiply-accumulate-transform structures of artificial neurons in neural networks (consuming about 50% of the total power in neural network accelerators [3]).

Various *error metrics*, such as the worst-case relative error or the mean absolute error, for evaluating approximate circuits have been proposed (cf. Sect. III). A crucial question is then how the error of a given approximation is derived. For that, as discussed in more details in the related work section, methods based on *simulating* the circuit on given inputs are often used. However, such approaches suffer from low scalability (exhaustive simulation), lack of strong guarantees (when simulating the circuit for a random subset of the possible inputs only), and/or specialization to certain circuits only (statistical models). Alternatively, as in our case, the error can be derived using *formal verification*. The main advantages of this approach lie in that (1) formal error bounds can be given as a part of the input and (2) the approach is more scalable than exhaustive circuit simulation.

While formal methods of (exact) equivalence checking have been studied for decades, only a few formal approximate checking methods have been used in circuit approximation tools. Depending on the particular error metric, the error calculation is transformed to a decision problem and solved by means of SAT solving or binary decision diagrams (BDDs). Despite of enormous progress in the area of SAT solvers and BDD libraries, approximation of arithmetic circuits with formal error guarantees was so far limited to circuits no more complex than 16-bit adders and 8-bit multipliers [4], [5], [6].

In this paper, we present a new method for designing complex approximate arithmetic circuits with formal bounds on the approximation error. The method uniquely integrates new formal techniques for approximate equivalence checking into search-based circuit optimization by means of Cartesian genetic programming (CGP). The key idea is to employ a novel search strategy driving the search towards promptly verifiable approximate circuits. We have implemented the strategy within the ABC tool and extended the underlying equivalence checking algorithm to support queries on the worst-case error. This extension builds on a new effective construction of miters, i.e. auxiliary circuits interconnecting the original correct circuit and its approximation such that their approximate equivalence can be checked.

We decided to optimize for the *worst-case error* since its exact value can be important in time-critical and dependable

978-1-5386-3093-8/17/$31.00

systems (e.g., inverse kinematics in robot control [7]) or when complex approximate arithmetic circuits are constructed using less complex approximate building (circuit) blocks. The final error then depends on how the worst case error is propagated from low-level blocks to the result. Moreover, even in not so critical applications such as image processing, low average error but excessive worst-case error can produce unacceptable results [8]. Finally, our results suggest that there is also a high correlation between the worst-case error and the mean absolute error (Sect. V).

While our primary motivation is to automatically approximate complex multipliers, our method is directly applicable to other arithmetic circuits too. The method is capable of providing Pareto fronts showing high-quality compromises between the circuit error and non-functional circuit parameters. Results are presented for approximate multipliers (with up to 32-bit operands) and adders (with up to 128-bit operands) and compared with several approximate circuits available in literature. This is for the first time when such complex approximate arithmetic circuits with formally guaranteed error bounds have been presented.

Contributions: We propose a new miter construction allowing for efficient approximate equivalence checking tailored to search-based approximation of complex arithmetic circuits. We design a novel search strategy for synthesis of approximate circuits with formal error guarantees that integrates Cartesian genetic programming and the proposed approximate equivalence checking. Using a resource-limited verifier, the strategy drives the search towards promptly verifiable candidates and thus provides scalable approximation of complex circuits. We develop an implementation of the miter construction and the search strategy within the ABC tool and perform extensive experimental evaluation of our approach on large circuits including approximation of 128-bit adders and 32-bit multipliers. Within several hours, we are able to construct high-quality Pareto sets of 128-bit adders and 32-bit multipliers that represent the trade-offs between the circuit error and non-functional circuit parameters.

II. Related Work

This section presents a brief survey of the most important approaches developed for functional approximation of multipliers and adders. We restrict our attention to these two arithmetic operations because they represent the key components of more complex circuits and thus their approximation has been intensively studied. Moreover, multipliers—due to their complex structure—represent one of the most difficult arithmetic circuits from the perspective of both approximation as well as verification.

A. Approximation Methods

The approximation process usually starts with a fully functional circuit and a target error. *Circuit-dependent approximation methods* then take the structure of the arithmetic circuit at the input and (manually or algorithmically) introduce modifications to carefully preselected parts of the circuit. In the case of adders, it is possible to approximate elementary 1-bit adders, modify the carry propagation chain, or introduce segments of adders and generate the carry using different methods [9]. In the case of multipliers, generation of partial products, the summation tree, counters, or compressors are approximated [10]. In addition to that, the simple bit-width reduction belongs to this category of methods too.

More complex approximate circuits can be constructed by a smart *composition* of approximate elementary blocks. For example, a 2-bit multiplier was approximated in [11] and then used as a building block of more complex multipliers. This strategy can be improved, e.g., by configurable lossy compression of the partial product rows based on their progressive bit significance [12].

The concept of *quality configurable circuits* uses elementary circuits composed in such a way that their error can be modified online using several configuration bits in order to dynamically reduce the power consumption. The configuration bits can (dis)connect some preselected parts of the circuit. As the source codes of quality configurable adders [13] and multipliers [2] are available online, we compare them with approximate circuits obtained using our approach.

General-purpose methods, such as SALSA [14] or SASIMI [15], aim at automatically approximating circuits independently of their structure. These methods operate with different circuit representations and employ various heuristics to identify circuit parts suitable for approximation. Evolutionary algorithms have been recently applied to accomplish desired approximations in a holistic scenario [16], [17]. A comprehensive library of 8-bit adders and multipliers was built using multi-objective CGP [18].

B. Simulation-Based Error Computation

Conceptually, the simplest approach to obtain precise error bounds of an AAC is to simulate its function on all possible inputs. However, even on state-of-the-art computer architectures, this approach has principal scalability limitations causing that it cannot be used to synthesize approximate circuits with more than 12-bit operands [19].

Due to that, the error is commonly estimated using a subset of input vectors only, e.g. 10^8 inputs were used to evaluate 16-bit adders in [9]. Of course, the main drawback of this approach is that no formal guarantees on the error bound can be provided. Alternatively, the circuit error can be calculated using a statistical model constructed for elementary circuit components and their compositions [20], [21]. However, reliable and general statistical models can only be constructed in some specific situations.

C. Formal Error Computation

Recently, various applications of formal methods have been intensively studied in order to improve the scalability of the design process of correct as well as approximate circuits. For designing correct circuits (where one insists on preserving the original functionality but tries to optimize non-functional parameters), one can consider combinational

equivalence checking based on modern SAT solvers, efficient BDD representations of circuits, or algebraic computation techniques combining polynomial representation of circuits with logic reductions [22], [23]. For designing AACs, a more challenging notion of *relaxed* or *approximate equivalence checking* is needed. This notion requires to quantify the approximation error or, alternatively, prove whether the error is below a certain threshold.

To quantify the approximation error using formal verification techniques, a use of auxiliary circuits, called *miters*, combining the original circuit and the approximate circuit was proposed in [24]. In order to check whether a predefined worst-case error is violated by the candidate approximate circuit, a pseudo-Boolean SAT solver combining a SAT solver with integer linear programming was then employed.

The number of inputs for which an approximate circuit returns an incorrect result can be quantified using *SAT counting methods* (so-called #SAT solvers). However, despite the recent progress in the area of #SAT solvers (see, e.g., [25]), our preliminary experiments indicate that #SAT problems encoding the error quantification are currently beyond the capabilities of state-of-the-art #SAT tools even for 12-bit multipliers.

An efficient BDD-based approach allowing one to guarantee the worst-case and the average-case arithmetic error of approximate adders up to 16-bit operands was proposed in [5]. An alternative approach that uses BDDs representing characteristic functions was employed in [4]. Compared to our approach, this approximation method lags behind in scalability, which is demonstrated by the fact that it has been applied to the approximation of multipliers limited to 8-bit operands and adders limited to 16-bit operands only.

III. Error Metrics for AACs

Various metrics describing the error of AACs have been proposed and shown suitable for different application domains. The most popular error metrics relevant especially to arithmetic circuits are the *worst-case absolute error* (WCAE) and the *mean absolute error* (MAE). For a correct circuit G, further denoted as the *golden circuit*, which computes a function f_G, and its approximation C, computing a function f_C, where $f_G, f_C : \{0,1\}^n \to \{0,1\}^m$, these metrics, relativized by the range of the output, are defined as follows:

$$\mathrm{WCAE}(G,C) = \frac{\max_{x\in\{0,1\}^n} |\mathrm{int}(f_G(x)) - \mathrm{int}(f_C(x))|}{2^m},$$

$$\mathrm{MAE}(G,C) = \frac{\sum_{x\in\{0,1\}^n} |\mathrm{int}(f_G(x)) - \mathrm{int}(f_C(x))|}{2^m},$$

where $\mathrm{int}(x)$ denotes the integer representation of a bit vector x and $|i|$ denotes the absolute value of an integer i.

A. Checking Worst Case Errors

To compute whether the WCAE is violated, we can adopt the concept of approximation miter introduced in [24]. The general configuration of the approximation miter is shown in Fig. 1. The miter consists of the inspected approximate circuit C, the golden circuit G which serves as the specification, a subtractor, and a comparator which checks whether the error introduced by the approximation is greater than a given threshold $\mathcal{T}$. The output of the miter is a single bit which evaluates to 1 if and only if the error is violated, i.e. $\mathrm{WCAE}(G,C) > \mathcal{T}$.

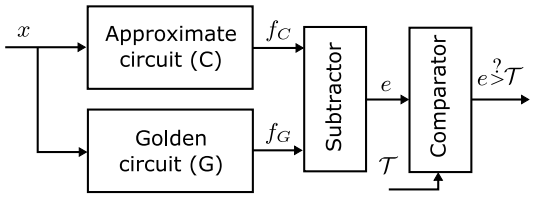

Fig. 1. Approximation miter for the worst-case error analysis, typically $e(x) = |f_G(x) - f_C(x)|$.

For a given input vector x, the subtractor calculates the difference between the output of the golden circuit, i.e. $f_G(x)$, and the output of the approximate circuit, i.e. $f_C(x)$. Let $d = \mathrm{int}(f_G(x)) - \mathrm{int}(f_C(x))$ be the error magnitude. A direct computation of the WCAE according to its definition leads to evaluating the expression $e = |d|$, i.e. the absolute difference of the error magnitude. The absolute difference is typically calculated by means of a common two's complement subtractor (implemented using m full-adders with the first carry-in set to 1 and inverting each bit of the subtrahend) followed by a circuit determining the absolute value (computed using m half-adders and m XOR gates).

B. The Proposed Miter Construction

Miters used in the literature compute the absolute value of the difference between f_G and f_C. The computation is usually performed in two steps. Firstly, a subtractor with a signed output evaluates $f_G - f_C$. Secondly, the absolute value has to be computed. The circuit performing such a task contains XOR chains which are a known cause of poor performance of the state-of-the-art SAT solvers [26]. The main reasons are that unlike AND/OR gates, the Boolean constraint propagation over XOR gates is limited, and the XOR operations cause the CNF form of the formulae to grow rapidly.

In order to avoid long XOR chains at the output of the miter which slowdown the decision process, we propose to employ a different approach. The key idea is to compare the result of the subtractor with both the positive and negative value of the threshold and thus avoid the expensive evaluation of the absolute value. For a given threshold $\mathcal{T}$ on the worst-case absolute error WCAE, it holds that $e > \mathcal{T}$ is satisfied iff d is positive and $d > \mathcal{T}$, or d is negative and $-d > \mathcal{T}$. As we typically deal with numbers in the two's complement, the second condition is equal to $\neg d > (\mathcal{T} - 1)$. Hence, we can use the two's complement representation and examine the positive and negative values separately to avoid usage of the absolute difference of the output.

Since the threshold $\mathcal{T}$ is fixed during the design process, we can easily avoid the standard comparator consisting of a long chain of XOR gates. This helps us to further simplify the miter and improve the performance of the decision procedure.

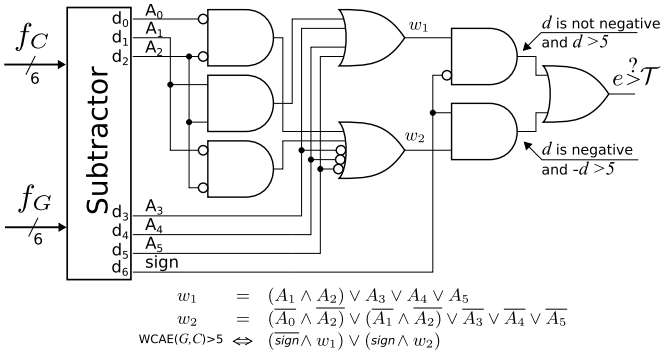

Fig. 2. The proposed approximation miter for the worst-case error analysis: an example for $\mathcal{T} = 5$, $N = 6$.

In particular, we replace the sequential comparison of the particular bits of the operands implemented as

$$A > B \equiv \bigvee_{0 \le i \le N-1} \left(A_i \wedge \neg B_i \bigwedge_{i < j \le N-1} \overline{A_j \oplus B_j} \right),$$

for B being a constant bit vector representing the threshold $\mathcal{T}$, by a simpler procedure implemented as

$$A > B \equiv \bigvee_{0 \le i \le N-1 \ \wedge \ B_i = 0} \left(A_i \bigwedge_{i < j \le N-1 \ \wedge \ B_j = 1} A_j \right).$$

As is evident, the resulting formula does not contain any XOR gate. Note that d is represented as an $m+1$ bit number in the two's complement—hence, A corresponds to the N least significant bits of d where $N = m$. The $(m + 1)$-th bit is reserved for the sign and employed for determining whether d encodes a positive or negative number. The miter for $\mathcal{T} = 5$, f_C and f_G with 6-bit outputs is illustrated in Fig. 2.

The proposed construction, compared to the construction using the absolute value and full comparators, allows us to obtain smaller and structurally less complex miters. Such miters can be efficiently used in the SAT-based CEC procedures, resulting in a significant acceleration of the candidate circuit evaluation. Our experiments show that, in the case of arithmetic circuits having 64 output bits (e.g. 32-bit multipliers), the proposed construction improves the size of the miters (in terms of the number of And-Inverter Graph (AIG) nodes representing the circuit) by about 25–35% depending on the value of $\mathcal{T}$, where $\mathcal{T}$ ranged from 0.0001% to 0.5% of the maximal value at the output (i.e. 2^{64}) in our experiment.

IV. Search-Based Design of AACs

In this section, we present our novel approach to the search-based design of AACs combining principles of CGP with a verifiability-driven search strategy that employs a fitness function based on the approximate equivalence checking.

A. Cartesian Genetic Programming

CGP is a form of genetic programming where the candidate solutions are represented as a string of integers of a fixed length that is mapped to a directed acyclic graph [27]. This integer representation is called a *chromosome*. CGP can efficiently represent common computational structures including mathematical equations, computer programs, neural networks, and digital circuits. The candidate circuits are typically represented in a two-dimensional array of programmable two-input nodes. Every node is encoded by three integers in the chromosome representation where the first two numbers denote the node's inputs, the third represents the node's function (see the illustration in Fig. 3).

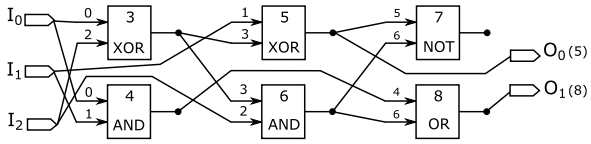

Fig. 3. Full adder represented by CGP. Chromosome: (0, 2, 2) (0, 1, 0) (1, 3, 2) (3, 2, 0) (5, 6, 3) (4, 6, 1) (5, 8), node functions: AND (0), OR (1), XOR (2), NOT (3).

In circuit approximation, the evolution loop starts with a *parent* representing a correctly working circuit. New candidate circuits are obtained from the parent using a *mutation operator* which performs random changes in the candidate's chromosome in order to obtain a new, possibly better candidate solution. In the next step, the algorithm evaluates the quality of each solution using a specified metric, called the *fitness function*. This function assesses important correctness and performance aspects of circuits. The candidate with the best fitness value is chosen as the parent of the next generation, the other solutions are removed and the evolution continues with generating new candidate circuits. The whole loop is repeated until a termination criterion is met. For details of CGP, see [27].

The most critical and time consuming part of the CGP loop is the fitness evaluation, which principally limits the scalability of the search-based design. To alleviate this problem, we propose below a novel search strategy.

B. Verifiability-Driven Search Strategy

The verifiability-driven search strategy can be seen as a general concept improving the scalability of evolutionary design methods. We demonstrate its key idea on the below problem.

Problem: *For a given golden circuit G and a threshold $\mathcal{T}$, our goal is to find a circuit C^* with the minimal size such that the error WCAE$(G, C^*) \le \mathcal{T}$.*

This problem formulation allows us to define the fitness function f in the following way:

$$f(C) = \begin{cases} \text{size}(C) & \text{if WCAE}(G, C) \le \mathcal{T}, \\ \infty & \text{otherwise} \end{cases}$$

where size(C) denotes the size of the circuit C. Since the procedure deciding whether WCAE$(G, C) \le \mathcal{T}$ (further denoted as SAT solver) represents the most time consuming part of the design loop, we avoid calling the procedure as much as possible. Therefore, we only call SAT solver for circuits C satisfying size$(C) <$ size(B) where B is the best solution with an acceptable error (i.e., WCAE$(G, B) \le \mathcal{T}$) that we have found so far. Our experiments show that, during the evolution process, a significant set of candidate designs C does not satisfy the condition size$(C) <$ size(B) and thus their fitness can be easily assessed without SAT solver.

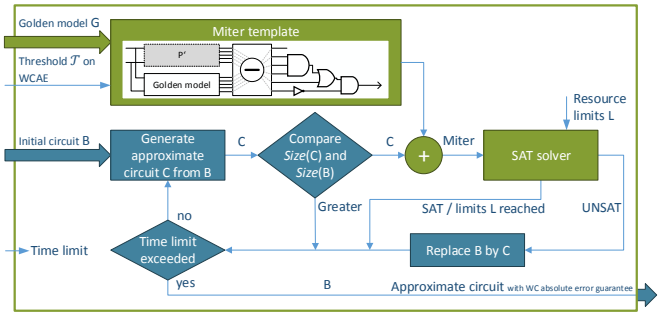

Fig. 4. The main steps of the proposed verifiability-driven search scheme.

Our experiments further indicate that a long sequence of candidate circuits B_i improving the size and having an acceptable error has to be typically explored to obtain a solution that is sufficiently close to C^*. Therefore, both the SAT and the UNSAT queries to SAT solver have to be short. To this end, we use an additional criterion for the evaluation of the circuit C, namely, the ability of SAT solver to prove that $\text{WCAE}(G, C) \leq \mathcal{T}$ with a given limit L on the resources available to the underlying decision procedure. If the procedure fails to prove $\text{WCAE}(G, C) \leq \mathcal{T}$ within the limit L, we set $f(C) = \infty$ and generate a new candidate. The design loop using the verifiability-driven search is illustrated in Fig. 4.

The inputs of the design process include: (1) the golden model G, (2) the threshold on the worst case absolute error $\mathcal{T}$, (3) the initial circuit B having an acceptable error (it can be either the golden model or a suitable approximation we want to start with), and (4) the time limit on the overall design process. The loop exploits the CGP principles; namely, it uses mutations to generate new candidate circuits C from the candidate circuit B representing the best approximation of the circuit C^* that we have found so far. The circuit C is then evaluated using the fitness function f as described above. If the candidate C belongs to the improving sequence (i.e., $\text{size}(C) < \text{size}(B)$ and $\text{WCAE}(G, C) \leq \mathcal{T}$), we replace B by C. The design loop terminates if the time limit is reached and B is returned as the output of the design process.

In our verifiability-driven search scheme, we use the resource limit L (as a parameter of the design loop) to drive the search towards candidates that can be promptly evaluated. We intentionally throw away improving candidates B_i that require greater resources and thus longer, but still feasible, verification time. The reason for this is the fact that by mutating these candidates we would most likely obtain solutions that would require the same or even longer verification times and thus finding the whole improving sequence would become time-infeasible. Instead, we require that every improving candidate B_i has to be verifiable using the resource limit L and thus drive the search towards candidates B_i that, for a given time limit on the overall design process, lead to longer improving sequences. Our experiments indicate that these sequences lead to candidate circuits that are closer to C^*. Since we are able to evaluate a much larger set of candidate circuits, we have a better chance to find a long improving sequence within the given time provided that it exists for the limit L.

The obvious disadvantage is that we possibly cut improving sequences that would lead to good solutions within the given design time. It can also happen that, for the limit L, no improving sequence exists, while it exists for a slightly greater resource limit. Despite of this limitation, our results clearly show that the proposed verifiability-driven search strategy allows us to utilise the given design time in a more efficient way compared to the standard evolution schemes.

C. Integration to the ABC Tool

The proposed approach performs the approximation at the level of the CGP problem representation (i.e., on acyclic oriented graphs with arbitrary two-input logic functions in the nodes). The green part of Fig. 4 shows the position of ABC in our methodology. ABC is primarily used to construct the miter and decide whether the maximal arithmetic error of the candidate circuit is not above $\mathcal{T}$. The proposed miter construction allows us to reduce the problem of approximate equivalence checking to the Boolean satisfiability (SAT) problem. In order to evaluate a candidate circuit, (1) a candidate chromosome is used to construct a corresponding AIG, (2) another AIG, representing the golden circuit, is constructed (just once at the beginning of the evolution), and (3) the miter is built. The state-of-the-art techniques used for CEC in the ABC tool—the `iprove` engine—are then applied to decide the equivalence. An important feature of the mix of techniques used in `iprove` is that one can control the time needed for one query, which is the key feature we exploit in our verifiability-driven search strategy. In particular, the satisfiability checking can be controlled by fine-tuning various resource limits for the different techniques used, such as the number of simulations performed to prove non-equivalence, the number of conflicts in structural hashing, or the number of logic-reduction steps. We so far used solely a limit on the maximal number of conflicts in which a single variable (representing an AIG node) can be involved during the backtracking process. Our experiments show that this resource limit allows us to effectively control the time needed for particular `iprove` queries and thus to drive the search towards promptly verifiable circuits.

A similar approach has recently been used in circuit approximation exploiting the approximate-aware rewriting of an AIG representation of circuits [4]. Principally, our approach differs in the candidate circuit representation (the gate-level CGP encoding), its evaluation, and in using the verifiability-driven evolution instead of a simple greedy algorithm for AIG pruning. The gate-level representation is an important feature of our approach which allows us to efficiently capture XOR-intensive structures existing in arithmetic circuits.

V. Results

To evaluate the proposed method, we primarily focused on complex approximate multipliers as they are the most challenging benchmark problems. Since only 8-bit multipliers with guaranteed error bounds were presented in the literature so far, there are no solutions available for a direct comparison in the case of 16-bit and more complex approximate multipliers.

Hence, (1) we compare the 16-bit approximate multipliers that we generated using our method with 16-bit multipliers (available in the literature) whose error was determined using simulation, and then (2) we present Pareto fronts (the error and key circuit parameters) for 20-bit, 24-bit, 28-bit, and 32-bit approximate multipliers and up to 128-bit approximate adders to demonstrate the scalability of the proposed method.

A. Experimental Setup

We implemented our approach, including the miter construction and verifiability-driven evolution, within the ABC tool [28]. Array multipliers and ripple carry adders composed of 2-input gates were employed as the initial (golden) circuits for CGP. The number of nodes in the CGP's grid is equal to the number of gates of the initial circuit. The set of functions consists of the common two-input logic gates, the buffer, and the inverter. We used 2 circuits in the population and 5 integers were modified by the mutation operator.

For each target WCAE, we performed 30 independent runs of CGP to obtain statistically significant results. Each CGP was executed for 2 hours on an Intel Xeon X5670 2.4 GHz processor using a single core. The individual CGP runs are independent and thus we executed them in parallel using a cluster of these processors to accelerate the design process.

For purposes of the fitness evaluation, the circuit size is estimated as the sum of the relative area of the two-input gates used, where the sizes of each gate are taken from the technology library. At the end of the evolution, the 5 most fitting circuits for each WCAE were synthetized using the Synopsys Design Compiler (high-effort compiling for a better quality of the results) for a 45 nm technology library in order to obtain non-functional parameters like the area and power-delay product (PDP). The accurate implementations were created by means of Verilog $*$ and $+$ operators and synthesized in the same way as approximate circuits.

B. 16-bit Approximate Multipliers

An evaluation of the verifiability-driven search: In the first experiment, we approximated the golden 16-bit multiplier for 9 target values of WCAE from the set {0.1, 0.2, 0.5, 1, 2, 5, 10, 15 and 20%} and evaluated the proposed method with three different settings of the resource limit L controlling the maximal number of conflicts for one AIG node: (1) no limits, i.e., L=∞, (2) L=160K, and (3) L=20K. The limits L=160K and L=20K roughly correspond to the time limit of 120 sec. and 3 sec., respectively, on 16-bit multipliers.

Fig. 5 shows that, for WCAE $\geq$ 2%, the resource limit L has a marginal impact on the PDP and area. However, with a decreasing target WCAE, the limit L=20K provides significantly better results. For example, if WCAE = 0.1% and L=20K, 22,050 SAT calls were produced and 11% of them were terminated on average because of the termination condition. In the case of L=160K, 856 SAT calls were produced only (15% terminated). The average number of SAT calls (across all target errors) that were forced to terminate is 6.28% (for L=160K) and 8.84% (for L=20K). If L=∞, 170 SAT calls were evaluated for WCAE = 0.1% only. Despite the fact that some potentially good candidate circuits are quickly rejected, the aggressive resource limits allowed us to generate and evaluate significantly more candidate circuits and thus to substantially improve the quality of results. Box plots in Fig. 5 also show that independent runs with L=20K lead to circuits having very similar parameters (low inter-quartile distances) and thus this limit is be used in the following experiments.

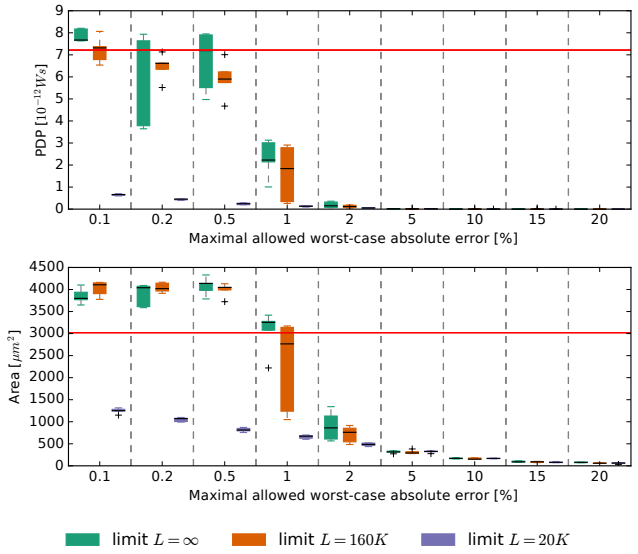

Fig. 5. PDP and area of approximate 16-bit multipliers for 9 target errors obtained using 3 different resource limits L on the SAT solver. The red line shows the PDP and area of the accurate multiplier.

Note that the parameters of some approximate multipliers shown in Fig. 5 are worse than for the accurate multiplier. The reason is that the relative area is the only non-functional circuit parameter optimized by CGP while the PDP and area are computed at the end of the optimization using the Synopsys Design Compiler. We have never observed this discrepancy for the limit L=20K.

A comparison with other multipliers: Next, we generated 16-bit approximate multipliers using the setup described in the previous section and compared them with approximate multipliers available in the literature. In order to perform a fair comparison (the error of the published multipliers was originally estimated using simulation), we modified our method and applied a binary search strategy to determine the WCAE exactly. In addition to WCAE, we also provide MAE obtained using simulation (10^9 vectors).

We considered the following 16-bit approximate multipliers:

M1 Approximate configurable multipliers from the lpACLib library [13], where the multiplication is recursively simplified using two different variants (denoted as *Lit* and *V1*) of an elementary block representing a 2-bit multiplier. The partial results are summed using accurate adders. We implemented 32 different architectures consisting of four 8-bit multipliers where each of these multipliers is configurable as exact/approximate (2^4 configurations) and can be built using either *Lit* (M1Lit) or *V1* (M1V1) blocks.

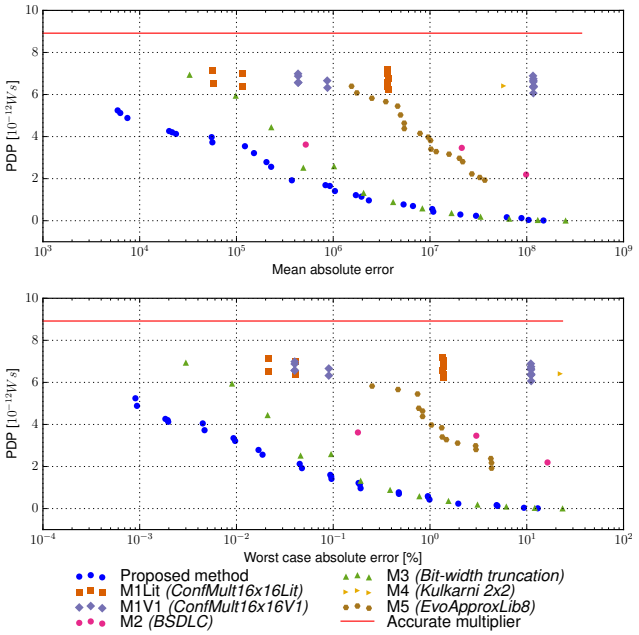

Fig. 6. Parameters of 16-bit approximate multipliers considered in our study.

M2 The approximate multiplier employing the bit-significance-driven logic compression as introduced in [12].

M3 Approximate multipliers obtained from exact multipliers using the bit-width reduction. The reduction replaces 16-bit multipliers by accurate x-bit multipliers (for $x < 16$). It ignores the LSBs of the operands and leaves the LSBs of the result zero.

M4 The approximate multiplier composed of approximate 2-bit multipliers as proposed in [11].

M5 Approximate multipliers composed of 8-bit multipliers that are available in the EvoApproxLib library [18]. The construction principle is taken from [11].

For all considered multipliers, the value of PDP is plotted against WCAE and MAE in Fig. 6 (only Pareto fronts are visualized). While the bit-width reduction provides the same quality of results as our method for large target errors (up to 20% WCAE), it is significantly outperformed by our approach for small target errors. Despite that the existing approximate multipliers typically exhibit good tradeoffs between the error and PDP in specific applications (as demonstrated in the relevant literature), Fig. 6 clearly shows that these multipliers are considerably Pareto-dominated by the multipliers obtained using our approach. These results were, in fact, expected as the proposed method is based on a global holistic optimization approach while the other approximate multipliers were composed of smaller ones and the composition procedure always introduces some overhead. Finally, it is an interesting observation that MAE follows the trend of WCAE. It seems that WCAE can be used as a good indicator of MAE.

C. Complex Multipliers

The aim of our further experiments is to show that the proposed method is scalable and can approximate complex multipliers. We present the results of the approximation process on 12-bit, 16-bit, 20-bit, 24-bit, 28-bit, and 32-bit multipliers. The target WCAEs were adapted accordingly to respect the range of values in the different considered bit widths. We used the same setup as in the previous sections but increased the time of optimization to 4 hours for the 24-bit multiplier and 6 hours for larger multipliers. The reason is that the search space becomes much bigger. While the exact 12-bit multiplier contains 850 two-input gates, the 32-bit exact multiplier requires over 6,300 gates. We obtained (as the result of evolution) over 1190 unique multipliers. Because of this huge number and for the sake of clarity, Fig. 7 shows parameters of approximate multipliers occupying the Pareto fronts only.

In the experiments, we observed that, in the case of 12-bit multipliers, 2.4% of SAT calls were terminated on average due to the resource limit L=20K only. However, this number increased to 36.9% in the case of approximate 32-bit multipliers. For all bit widths, the MAE is around 30% of the worst-case error, which again demonstrates that WCAE is a good indicator of MAE. Fig. 7 also shows that the obtained approximations cover the whole range (up to 100%) of the Area axis. However, this is not the case for PDP. The reason is that we optimize the relative area and PDP is computed after the synthesis.

Since Pareto fronts shown in Fig. 7 follow the trend of the highly competitive fronts for the 16-bit multipliers presented before, we believe that the tradeoffs between the circuit error and size obtained for more complex multipliers are also very good and thus the corresponding circuits represent the cutting edge of approximate multipliers and can serve as a new benchmark set for approximate computing.

D. Approximate Adders

In order to demonstrate that the proposed method is applicable for other complex arithmetic circuits, we constructed Pareto fronts for approximate adders with 20-bit to 128-bit operands. Approximation of adders is much easier than approximation of multipliers since adders are structurally less complicated and the number of outputs is lower. For example, the exact 20-bit adder requires 140 two-input gates and the 128-bit adder consists of 1,000 gates.

The approximate adders were constructed using the same setup as in the previous section. A single CGP run took 2 hours (for all bit widths). Fig. 8 shows parameters of approximate adders occupying the corresponding Pareto fronts. We report 16 to 18 non-dominated implementations of 24-bit, 28-bit, and 32-bit adders in terms of PDP and WCAE. For 64-bit and 128-bit adders, 12 tradeoffs are reported only because we have restricted the number of target error levels. Similarly to the evolved multipliers, the proposed approximate adders are also good candidates for including into a new benchmark suite.

VI. Conclusion

Automated design of approximate circuits with formal error guarantees is a landmark of provably-correct construction of energy-efficient systems. We present a solution to this problem, introducing a novel verifiability-driven search strategy

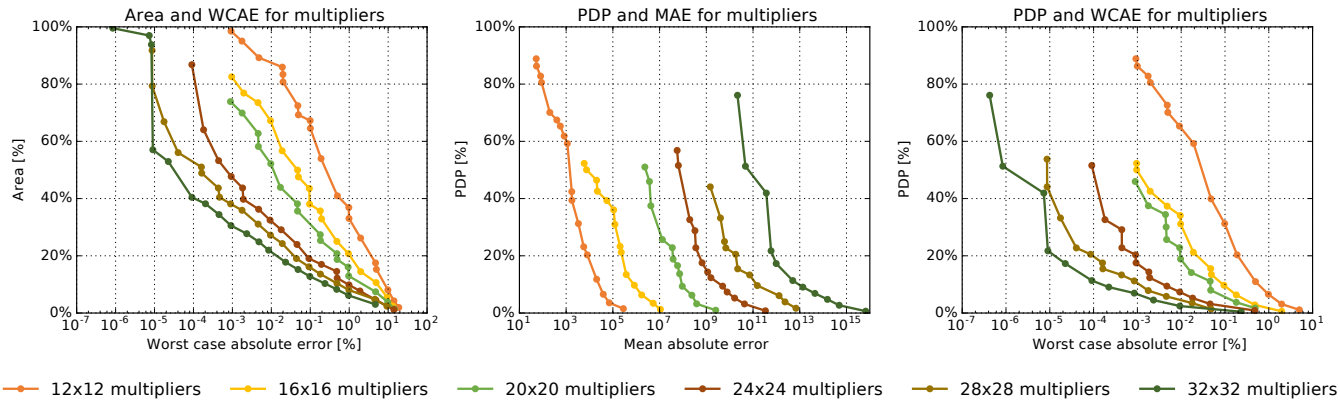

Fig. 7. Pareto fronts showing parameters of evolved approximate multipliers. 100% refers to parameters of the accurate multiplier for a given bit width.

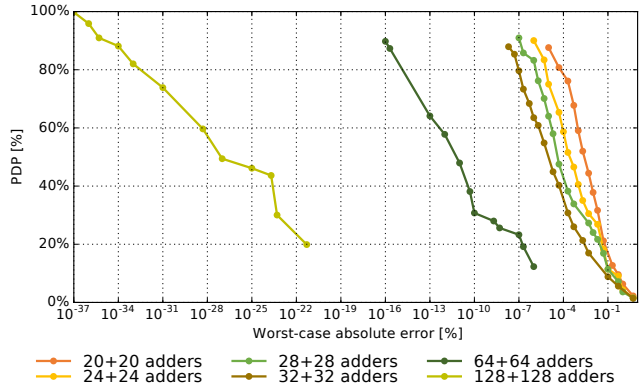

Fig. 8. Pareto fronts showing parameters of evolved approximate adders. 100% refers to parameters of the accurate adder for a given bit width.

that uniquely integrates approximate equivalence checking into a search-based circuit optimisation algorithm. Able to construct high-quality Pareto sets of 32-bit multipliers and 128-bit adders, our method shows excellent scalability and paves the way for design automation of complex approximate circuits.

In the future, we will thoroughly explore relationships between resource limits on the underlying SAT solvers and the structure of the resulting circuits. This will allow us to further improve the performance of our method and thus to go beyond the approximation of 32-bit multipliers. We will also integrate the constructed circuits into real-world energy-aware systems to demonstrate practical impacts of our work.

Acknowledgments: This work has been supported by the Czech Science Foundation grant No. GA16-17538S.

References

[1] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 62:1–62:33, 2016.

[2] M. Shafique, R. Hafiz *et al.*, "Invited: Cross-layer approximate computing: From logic to architectures," in *Proc. of DAC'16*, 2016, pp. 1–6.

[3] P. Judd, J. Albericio *et al.*, "Proteus: Exploiting numerical precision variability in deep neural networks," in *ICS'16*, 2016, pp. 1–12.

[4] A. Chandrasekharan, M. Soeken *et al.*, "Approximation-aware rewriting of AIGs for error tolerant applications," in *Proc. of ICCAD'16*, 2016, pp. 83:1–83:8.

[5] Z. Vasicek, V. Mrazek, and L. Sekanina, "Towards low power approximate DCT architecture for HEVC standard," in *Proc. of DATE'17*, 2017, pp. 1576–1581.

[6] C. Yu and M. Ciesielski, "Analyzing imprecise adders using BDDs – a case study," in *Proc. of ISVLSI'16*, 2016, pp. 152–157.

[7] B. Grigorian and G. Reinman, "Dynamically adaptive and reliable approximate computing using light-weight error analysis," in *Proc. of AHS'14*, 2014, pp. 248–255.

[8] D. S. Khudia, B. Zamirai *et al.*, "Rumba: An online quality management system for approximate computing," in *ISCA'15*, 2015, pp. 554–566.

[9] H. Jiang, J. Han, and F. Lombardi, "A comparative review and evaluation of approximate adders," in *Proc. of GLVLSI'15*, 2015, pp. 343–348.

[10] H. Jiang, C. Liu *et al.*, "A comparative evaluation of approximate multipliers," in *Int. Symp. Nanoscale Architectures*, 2016, pp. 191–196.

[11] P. Kulkarni, P. Gupta, and M. D. Ercegovac, "Trading accuracy for power in a multiplier architecture," *J. Low Power Electronics*, vol. 7, no. 4, pp. 490–501, 2011.

[12] I. Qiqieh, R. Shafik *et al.*, "Energy-efficient approximate multiplier design using bit significance-driven logic compression," in *Proc. of DATE'17*, 2017, pp. 7–12.

[13] M. Shafique, W. Ahmad *et al.*, "A low latency generic accuracy configurable adder," in *Proc. of DAC'15*, 2015, pp. 86:1–86:6.

[14] S. Venkataramani, A. Sabne *et al.*, "SALSA: systematic logic synthesis of approximate circuits," in *Proc. of DAC'12*, 2012, pp. 796–801.

[15] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-simplify: a unified design paradigm for approximate and quality configurable circuits," in *Proc. of DATE'13*, 2013, pp. 1367–1372.

[16] Z. Vasicek and L. Sekanina, "Evolutionary approach to approximate digital circuits design," *IEEE Trans. Evol. Comput.*, vol. 19, no. 3, pp. 432–444, 2015.

[17] K. Nepal, S. Hashemi *et al.*, "Automated high-level generation of low-power approximate computing circuits," *IEEE Trans. Emerg. Topics Comput.*, pp. 1–13, 2017.

[18] V. Mrazek, R. Hrbacek *et al.*, "Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Proc. of DATE'17*, 2017, pp. 258–261.

[19] V. Mrazek, S. S. Sarwar *et al.*, "Design of power-efficient approximate multipliers for approximate artificial neural networks," in *Proc. of ICCAD'16*, 2016, pp. 81:1–81:7.

[20] C. Li, W. Luo *et al.*, "Joint precision optimization and high level synthesis for approximate computing," in *DAC'15*, 2015, pp. 1–6.

[21] S. Mazahir, O. Hasan *et al.*, "Probabilistic error modeling for approximate adders," *IEEE Trans. Comput.*, vol. 66, no. 3, pp. 515–530, 2017.

[22] M. Ciesielski, C. Yu *et al.*, "Verification of gate-level arithmetic circuits by function extraction," in *Proc. of DAC '15*, 2015, pp. 52:1–52:6.

[23] A. Sayed-Ahmed, D. Große *et al.*, "Formal verification of integer multipliers by combining Gröbner basis with logic reduction," in *Proc. of DATE'16*, 2016, pp. 1048–1053.

[24] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, "Macaco: Modeling and analysis of circuits for approximate computing," in *Proc. of ICCAD'11*, 2011, pp. 667–673.

[25] S. Chakraborty, K. S. Meel *et al.*, "Approximate probabilistic inference via word-level counting," in *Proc. of AAAI'16*, 2016, pp. 3218–3224.

[26] C.-S. Han and J.-H. R. Jiang, "When boolean satisfiability meets gaussian elimination in a simplex way," in *CAV'12*, 2012, pp. 410–426.

[27] J. F. Miller, *Cartesian Genetic Programming*, 2011.

[28] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *Proc. of CAV'10*, ser. LNCS, 2010, pp. 24–40.