Inexact Arithmetic Operators*

Lukas Sekanina and Zdenek Vasicek and Vojtech Mrazek

Abstract Approximate implementations of arithmetic circuits have been developed to find the best trade-offs between the key circuit parameters (such as energy, area, and delay) and the error of arithmetic operations. This chapter discusses various methodological aspects of developing approximate arithmetic circuits, including design abstractions, number representation, error analysis methods, and particular design methods. We survey problem-specific methods proposed for the manual design of approximate adders and multipliers. The circuit approximation problem is also formulated as a multi-objective optimization problem that can be solved by a suitable automated circuit design method. A comprehensive open-source library EvoAppoxLib of approximate circuits that was automatically generated by one of the automated methods is introduced. We stress the importance of correct evaluation and comparison of approximate implementations and a proper benchmarking methodology. Finally, two case studies demonstrate some frequently overlooked issues related to the selected error analysis approaches.

* This is a pre-print of a book chapter published in Approximate Computing Techniques, edited by Alberto Bosio, Daniel Ménard, and Olivier Sentieys, Springer, 2022. The final authenticated version is available online at: doi.org/10.1007/978-3-030-94705-7_4

Lukas Sekanina
Brno University of Technology, Faculty of Information Technology, IT4Innovations Centre of Excellence, Czech Republic, e-mail: sekanina@fit.vutbr.cz

Zdenek Vasicek
Brno University of Technology, Faculty of Information Technology, IT4Innovations Centre of Excellence, Czech Republic, e-mail: vasicek@fit.vutbr.cz

Vojtech Mrazek
Brno University of Technology, Faculty of Information Technology, IT4Innovations Centre of Excellence, Czech Republic, e-mail: mrazek@fit.vutbr.cz

1 Introduction

Developing and applying approximate implementations of arithmetic operators is currently one of the most popular approaches to reduce power consumption in compute-intensive signal, image, and video processing applications. The aim of this chapter is to summarize the operation principles of elementary approximate arithmetic circuits and the methods developed for their design. Our focus will be on selected methodological issues and practices related to the approximate circuit design, namely the understanding of the circuit approximation problem as a multi-objective optimization problem, an error analysis methodology, a correct evaluation and comparison of approximate implementations and a fair benchmarking methodology.

We will primarily deal with *approximate adders* and *approximate multipliers* because they are the key circuits of many applications relevant for approximate computing, for example, image, video and speech processing, deep learning, data mining, and nature language processing. No attention will be paid to approximate implementations of subtractors, dividers, and other arithmetic operations because their need in approximate computing systems is rather limited in comparison with adders and multipliers. These circuits are discussed, for example, in [32, 4, 21]. Moreover, we will not deal with common compositions of adders and multipliers in circuits such as 'multiply and accumulate' (MAC) and scalar product. Their approximate implementations can be obtained either by (i) utilizing (independent) approximate adders and approximate multipliers or (ii) designing a single block without any decomposition, where (ii) can lead to much better tradeoffs than (i).

Approximate implementations of digital circuits are often obtained by the so-called *functional approximation*. This method starts with an original (exact) circuit and tries to modify its logic behavior (and the subsequent implementation) in such a way that the best possible tradeoff between the quality of output (the error) and electrical characteristics of the circuit is sought. By electrical characteristics, we mean one or several circuit parameters commonly used to characterize electrical circuits, for example, power consumption, area, and delay. We will not deal with voltage over-scaling and other technology exploiting approximation methods, although they are sometimes combined with functional approximation. The reason is that voltage over-scaling is very technology-dependent and hard to control, making the evaluation and fair comparison of various approximate implementations difficult.

The rest of the chapter is organized as follows. The methodological aspects that are relevant for the design of approximate arithmetic circuits are surveyed in Chapter 2. Some of them are then further elaborated in special chapters. In particular, Chapter 3 deals with error analysis methods for approximate circuits, with a special focus on formal relaxed equivalence checking. Chapter 4 presents the circuit approximation as a multi-objective optimization problem and emphasizes a correct approach enabling to compare approximate circuits under several design metrics and constraints. Problem-specific approximation methods for adders and multipliers are discussed in Chapter 5 while general-purpose automated approximation methods are briefly introduced in Chapter 6. A comprehensive open-source library of approxi-

mate circuits that was automatically generated by one of the automated methods is presented in Chapter 6.2. Chapter 7 includes several case studies that demonstrate some interesting aspects of the circuit approximation methods; for example, it compares the circuit simulation utilizing a subset of input vectors with an exact error analysis. Concluding remarks are given in Chapter 8.

2 Methodological aspects

In this section, we discuss various methodological aspects that a designer has to consider before any approximate (arithmetic) circuit is created. Some of these aspects are solely connected with arithmetic circuits while other aspects are relevant for all approximate circuits. Understanding these aspects is crucial not only for developing efficient circuit approximate methods and high-quality approximate circuits, but also for performing a fair comparison of these design methods and the approximate circuits created by these methods.

2.1 Design abstraction

The original (exact) circuit and its approximate implementation, which we have to devise, are usually described at the same level of abstraction. The approximation of arithmetic circuits is typically conducted at the transistor level (e.g., [11]), gate level (e.g., [55, 21]), register transfer (RT) level (e.g., [44]), behavioral level (e.g., [40]), and look-up table (LUT) level (e.g., [49]) if the target platform is a field programmable gate array (FPGA). Most approximate arithmetic circuits are combinational circuits. We will not deal with iterative or sequential implementations, but this topic is also covered in the literature, e.g., [3].

2.2 Target technology

While most approximation approaches have been developed for application-specific integrated circuits (ASICs), there are some papers dealing with approximate arithmetic circuits for graphic processing units (GPU) [43, 17] and FPGAs [49]. The target technology has to be taken into account by the approximation methodology as an approximate circuit optimized for one technology can show different electrical properties when implemented using a different technology.

2.3 Number representation

Approximation strategies for arithmetic circuits are tightly coupled with the number representation utilized in a given system (Chapter **??**). For the fixed-point (FX) number representation, the designers primarily decide about the number of bits used for the integer and fractional part and whether the circuit will intrinsically process signed numbers, the sign will separately be handled, or only unsigned numbers will be considered. A domain-specific quantization scheme is then used to map the input data range to the code values and vice versa. The quantization scheme should be linear and allow us to represent some important numbers (such as 0.0) exactly.

With the development of GPU-based deep learning, approximate implementations of adders and multipliers operating with the floating-point (FP) number representation have been proposed. The usual scheme for representing FP numbers (known from, e.g., the IEEE 754 standard) is simplified to reduce power consumption. For example, in the *minifloat* representation, any exponent and mantissa bit-width combination is allowed on a given total number of bits; however, to reduce the implementation overhead, the common exception handling, infinity, denormalized values, and alternative rounding modes are not supported [12].

2.4 Circuit approximation methods

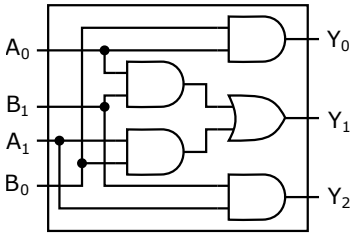
Approximate implementations are usually created by (i) "manual" modifications of exact circuits, (ii) developing new application-specific approximation schemes, or (iii) automated design space exploration algorithms.

The first approach requires a skilled designer who introduces appropriate changes to the original circuit. Very specific approximation techniques were developed for particular types of arithmetic circuits such as multipliers. For example, Fig. 1 shows one of the first approximate multipliers created by a human expert [25]. Its implementation is based on modifying the truth table of the 2-bit multiplier in such a way that the correct results are provided for 15 out of 16 input combinations, the area is reduced to almost 50% and delay is also reduced by one logic level. This approximate multiplier was used as a building block of more complex multipliers. Unfortunately, the manual approximation represents a time-consuming process that is feasible for small circuits only.

The second approach does not start with a common (exact) circuit implementation. It is rather based on a new construction scheme for a given class of problems. For example, a new approximation technique for FP multipliers lies in fitting linear functions with two inputs, referred to as linear planes. The linearization of multiplication allows multiplication operations to be completely replaced with weighted addition [43].

The last method employs fully automated circuit optimization or resynthesis algorithms. We will provide a brief overview of these methods in Chapter 6. In addition to the evaluation of resulting approximate implementations, we are faced with

a new problem – the evaluation of circuit approximation algorithms (in terms of resources and time needed to obtain a solution with desired properties).



B \ A	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	4	6
3	0	3	6	7

Fig. 1 The 2-bit approximate multiplier proposed by Kulkarni et al. in [25] and its specification

2.5 Error metrics and error analysis

Approximate arithmetic circuits are developed with the aim of minimizing one or several error metrics. We will survey commonly used error metrics and error analysis methods in Chapter 3. Here, we will emphasize some important aspects of the error analysis methodology. If an approximate circuit is evaluated under one error metric, then the decision of whether one circuit is better than another circuit is straightforward. If two or more error metrics are evaluated together, this relation becomes more complex and a different concept has to be employed. Chapter 4 will discuss the so-called Pareto dominance relation to handle this situation. Suppose some constraints are imposed on resulting approximate circuits (for example, the worst-case error must always be less than 1% while the mean absolute error is minimized). In that case, all circuits not satisfying these constraints must be excluded from the comparisons.

Most error analysis methods only estimate the exact error as determining the exact error is very time-consuming. The error estimate is obtained by circuit simulation across a reasonably inclusive set of input vectors. The exact error can be obtained by exhaustive simulation, but this approach is not scalable. More scalable approaches are based on formal analysis methods. We will compare the performance of these methods in Chapter 7.

Another issue is that many approximate circuits have been developed without assuming any particular data patterns existing in a given application, i.e. the design method assumes that all input vectors will occur with the same probability. This is a common approach if the approximate circuit should be provided as a reusable component. If the designer knows the input data distribution, the approximate circuit can be designed to reflect this knowledge and thus to provide better tradeoffs between the error(s) and electrical parameters. This can naturally be accomplished by automated approximation methods [54].

Finally, for a few particular implementations of approximate adders and multipliers, detailed probability error analysis methods were published, e.g. [31]. Knowledge of error probabilities becomes very useful if such a circuit is (re)used in a more complex application, and one needs to perform reasoning about the application-level error based on probability models available at the component level. An obvious disadvantage is that a lot of human effort is required to construct reliable probabilistic models for particular circuits.

2.6 Quality configurable circuits

Approximate circuits having some configuration parameters such as the segment size (i.e., the number of subadders working in parallel), the number of fractional bits or the number of active subcircuits are called *quality configurable circuits*. The setting of these configuration parameters is determined either at the design time (before the circuit synthesis is conducted) or online, i.e. during the run time, depending on the requested quality of service. This strategy can also be interpreted as an error compensation support or dynamic approximation. For example, these circuits can be utilized in the signal processing applications that can thus benefit from an *in situ* dynamic adaptation of the quality of processing in response to variable requirements on the quality of result and available resources.

We will demonstrate this idea by extending the 2-bit approximate multiplier from Fig. 1 to support two modes of operation as introduced in [45, 46], see Fig. 2. In the first mode, the quality configurable multiplier (QCM) works exactly as the approximate multiplier from Fig. 1, i.e. it generates an incorrect result (7) when both the inputs are 3. In the second mode, a correction circuit is activated, which modifies the output value of the approximate multiplier if it equals 7. Then, the incorrect value (7) is increased by two. The reconfiguration is implemented using the power gating technique. The parameters of the 2-bit QCM synthesized using Synopsys DC with 45 nm FreePDK are given in Table 1. Compared to the common multiplier implemented using Verilog star operator, the 2-bit QCM exhibits some overhead when we consider area, power, and delay. In the approximate mode, however, the electrical parameters (i.e. power and delay) are significantly improved. The 2-bit QCM in the approximate mode consumes 36% less power compared to the accurate multiplier. This multiplier can be used as an elementary block to build larger 8-bit and 16-bit quality configurable multipliers.

An important outcome of this brief analysis is that one has to be very careful when properties of common approximate circuits and quality configurable circuits are compared because the quality configurable circuits always exhibit some circuit overhead needed to ensure the reconfiguration.

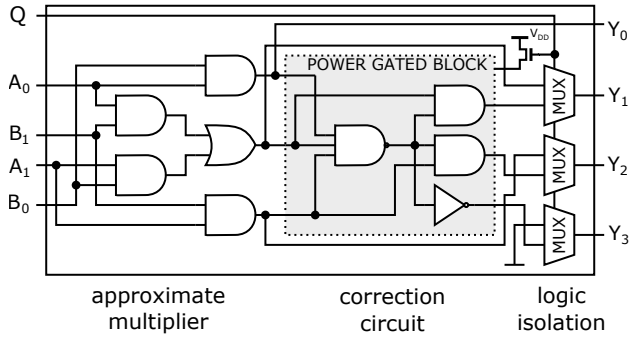


Fig. 2 Quality configurable 2-bit multiplier according to [46]

Table 1 Parameters of 2-bit and 8-bit exact and configurable approximate multipliers according to [45, 46]. e_{mae} and e_{wce} denote the mean absolute error and the worst-case error.

Multiplier	operating mode	Power [μW]	Delay [ns]	Area [μm^2]	e_{mae} [%]	e_{wce} [%]
2-bit exact	–	3.8	0.15	19	0	0
2-bit QCM	approximate	2.4	0.09	12*	13	13
	exact	4.7	0.21	23	0	0
8-bit exact	–	428.3	1.25	727	0	0
8-bit QCM	approximate	483.0	1.51	1197*	1.4	22
	exact	516.4	1.60	1337	0	0

* The value is the area of the approximate subcircuit only.

3 Formal error analysis

Determining the error of an approximate circuit or deciding whether an approximate circuit satisfies a given error constraint represents not only fundamental theoretical problems, but also highly practically relevant problems that must be routinely solved during the design of approximate circuits. This subchapter is focused on the exact error analysis of approximate arithmetic circuits by means of formal methods. But the formal methods can be applied to effectively analyze errors of other combinational circuits as well as sequential systems [3].

Fast and accurate error analysis is especially important in the case automated approximation methods because they usually need to generate and evaluate many candidate designs.

3.1 Error metrics

The functionality of approximate circuits is typically expressed using one or several error metrics. When an arithmetic circuit is approximated, for example, it is nec-

essary to base the error quantification on an arithmetic error metric since the error magnitude could have a significant impact on target application.

Let $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$ be an n-input m-output Boolean function that describes correct functionality (specification) and $\hat{f} : \mathbb{B}^n \rightarrow \mathbb{B}^m$ be an approximation of it, both implemented by two circuits, namely F and $\hat{\text{F}}$. The following paragraphs summarize the error metrics that are relevant for arithmetic circuits.

One of the most popular metrics applied in the context of approximate computing is the *worst-case arithmetic error*, sometimes denoted as *error magnitude* or *error significance*. This metric corresponds with the maximum error the approximation may give and is defined as

$$e_{wce}(f,\hat{f}) = \max_{\forall x \in \mathbb{B}^n} |\,\text{nat}(f(x)) - \text{nat}(\hat{f}(x))| \tag{1}$$

where $\text{nat}(x)$ represents a function $\text{nat} : \mathbb{B}^m \rightarrow \mathbb{Z}$ returning a decimal value of the m-bit binary vector x. Typically, a natural binary representation is considered, i.e. $\text{nat}(x) = \sum_{i=0}^{m-1} 2^i x_i$. The worst-case error represents the fundamental metric which is typically used as a design constraint and helps to guarantee that the approximate output can differ from the correct output by at most ε (i.e. the condition $e_{wce}(f,\hat{f}) \leq \varepsilon$ is always satisfied).

The worst-case arithmetic error is closely related to the *relative worst-case error* defined as

$$e_{wcre}(f,\hat{f}) = \max_{\forall x \in \mathbb{B}^n} \frac{|\,\text{nat}(f(x)) - \text{nat}(\hat{f}(x))|}{\text{nat}(f(x))}. \tag{2}$$

This metric can be used to constrain the approximate circuit to differ from the correct one by at most a certain margin. The maximum error magnitude is considered in relation to the correct output value.

There are also statistically oriented error metrics such as the *average-case arithmetic error* or *average-case relative arithmetic error* describing the mean absolute or relative error magnitude. The average-case arithmetic error is defined as the sum of absolute differences in magnitude between the original and approximate circuits, averaged over all inputs:

$$e_{mae}(f,\hat{f}) = \frac{1}{2^n} \sum_{\forall x \in \mathbb{B}^n} |\,\text{nat}(f(x)) - \text{nat}(\hat{f}(x))| \tag{3}$$

When we replace the expression in the sum by the equation for relative error distance, we can calculate the *mean relative error*:

$$e_{mre}(f,\hat{f}) = \frac{1}{2^n} \sum_{\forall x \in \mathbb{B}^n} \frac{|\,\text{nat}(f(x)) - \text{nat}(\hat{f}(x))|}{\text{nat}(f(x))}. \tag{4}$$

In addition to that, *mean-squared error* corresponding to the average squared error magnitude represents an important metric especially for signal processing applications because it is inversely related to peak signal-to-noise ratio (PSNR). This metric is defined as

$$e_{mse}(f,\hat{f}) = \frac{1}{2^n}\sum_{\forall x\in\mathbb{B}^n}(\text{nat}(f(x)) - \text{nat}(\hat{f}(x)))^2. \quad (5)$$

The error metrics mentioned in the previous paragraphs suppose uniform distribution of the input probabilities. There are, however, cases, where we need to consider skewed input distributions. One example is represented by the approximate multiplications carried out in deep neural networks [54]. To evaluate the error-metric with respect to a given distribution of input probabilities, *weighted mean error distance* can be introduced as an extension of the conventional mean error [54]:

$$e_{wmae}(f,\hat{f}) = \sum_{\forall x\in\mathbb{B}^n} D(x)|\,\text{nat}(f(x)) - \text{nat}(\hat{f}(x))| \quad (6)$$

where X corresponds to a discrete random variable representing data at the inputs and D is a probability mass function of X defined as $D(x) = Pr(X = x)$.

In addition to the arithmetic errors, *error rate* referred to as *error probability* can be investigated. The error rate corresponds to the percentage of inputs vectors for which the output value differs from the original one and is defined as

$$e_{prob}(f,\hat{f}) = \frac{1}{2^n}\sum_{\forall x\in\mathbb{B}^n}[\![f(x)\neq\hat{f}(x)]\!], \quad (7)$$

where $[\![f(x)\neq\hat{f}(x)]\!] = 1$ iff the proposition P is satisfied and $[\![P]\!] = 0$ otherwise.

3.2 Relaxed Equivalence Checking

Formal verification techniques that are widely adopted in the conventional circuit design flow are often based on *equivalence checking*, i.e., checking whether a mathematical model of a circuit under design meets a given specification. Two main approaches have been developed in this direction – techniques based on Reduced Ordered Binary Decision Diagrams (ROBDD) and satisfiability (SAT) solvers [51]. In both cases, an auxiliary circuit, the so-called *miter*, is constructed and then analyzed. Fig. 3(a) shows that the miter instantiates both the candidate circuit $\hat{F}$ (to be checked) and the golden circuit F, and compares their corresponding outputs to detect a difference in their behaviour. In the context of approximate computing, we need to extend this concept to *relaxed equivalence checking*, by stressing the fact that the considered circuits will be checked to be equal up to some bound w.r.t. a suitably chosen distance (error) metric such as the worst case error or the average error. The (approximation) miter always contains an additional component enabling us to determine the error, see Fig. 3(b).

Fig. 4 provides a brief overview of formal error analysis methods for approximate circuits. If the error analysis is performed using ROBDDs, a new ROBDD representing the miter is constructed by a procedure which reads the miter (gate by gate) and adds appropriate nodes to ROBDD. ROBDDs can be directly used for the

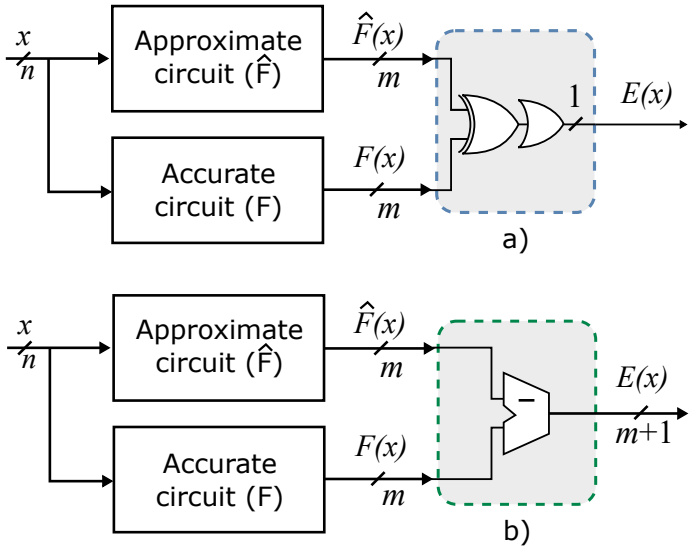


Fig. 3 Miter for equivalence checking (a) and arithmetic error analysis (b). For the equivalence checking, the output E corresponds with $E(x) = [\![f(x) \neq \hat{f}(x)]\!]$. For arithmetic error analysis, the output E equals to the error magnitude $E(x) = \text{nat}(f(x)) - \text{nat}(\hat{f}(x))$.

worst-case as well as the average-case analysis because every library for ROBDD manipulation is equipped with operations enabling us to address questions related to the satisfiability of the miter, namely finding one satisfying assignment and counting the number of satisfying assignments. The first operation provides a single input assignment x from the ON-set of a Boolean function. The second operation computes the size of the ON-set. As ROBDDs are inefficient in representing classes of circuits for which the number of nodes in BDD is growing exponentially with the number of input variables (e.g., multipliers and dividers), their use in relaxed equivalence checking is typically possible for adders and other less structurally complex functions. Anyway, for example, 128 bit adders can be quickly analysed in terms of all relevant error metrics [51].

If the error analysis is based on SAT solving, the miter is represented as a logic formula in Conjunctive Normal Form (CNF) for which SAT solver decides whether is satisfiable or unsatisfiable. The interpretation of this outcome depends on construction of the miter, see Section 3.3. Common SAT solvers are, in principle, applicable to the worst-case analysis only. However, this approach is more scalable than ROBDDs for the error analysis of multipliers [44]. Specialized SAT solvers (#SAT) are capable of counting the number of satisfiable assignments, but their scalability is very limited and thus they are currently less practical for the exact error analysis [51].

Even though ROBDDs offer a more flexible approach than SAT considering the possibilities of computation of approximate error metrics, their application is also limited. Not every error metric can directly be calculated using this technique. ROB-

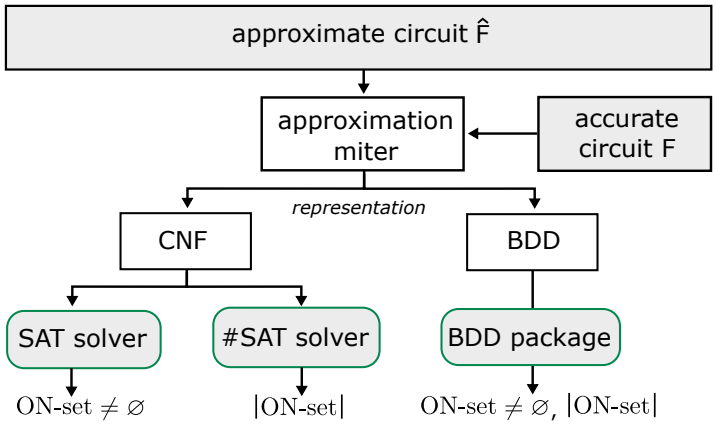


Fig. 4 Overview of formal error analysis approaches

DDs, for example, do not allow incorporating input probabilities [10]. Moreover, it is not easy to evaluate statistical error metrics involving computation of the relative error due to the presence of division. To address both issues, a more advanced approach needs to be introduced. The only approach allowing us to evaluate the error metrics reflecting the distribution of input probabilities is based on the usage of a more advanced representation known as Algebraic Decision Diagrams (ADDs) [10]. The usage of ADDs is naturally connected with a higher computational cost.

3.3 Worst Case Error Analysis

The worst-case error analysis is typically based on an iterative approach in which a variant of binary search is applied.

Algorithm 1: Worst-case absolute error computation

Input: n-input approximation miter with m-bit signed output E in the two's complement
Output: maximum absolute arithmetic error (e_{wce})

```
1 l ← 0; r ← 2^m − 1
2 while l ≤ r do
3     t ← ⌈(l + r)/2⌉
4     if WCEGT(E, t) then
5         l ← t + 1
6     else
7         r ← t − 1
8 return l
```

For computing the worst-case arithmetic error, for example, the miter given in Fig. 3(b) is used. Algorithm 1 illustrates the principle of determining the worst case arithmetic error, i.e. calculating the error magnitude at the m-bit output of the miter denoted as E. The principle of this procedure is to iteratively check whether the error is greater than a given threshold (denoted as t in the algorithm). The search procedure gradually narrows down the interval where the exact error value lies. After a finite number of steps, a single value is determined. As the binary search runs in logarithmic time with respect to the range, at most m comparisons are required. The checking can be ensured by means of the magnitude comparator which is used to form a Boolean function whose output is equal to 1 if and only if a given worst-case error $\mathscr{T}$ is violated by the circuit under analysis.

$$\begin{aligned}\text{WCEGT}(E,\mathscr{T}) &= \exists_{x\in\mathbb{B}^n}|E(x)| > \mathscr{T} \\ &= \text{ON-set}\left([\overline{e_m}\wedge(E>\mathscr{T})]\vee[e_m\wedge(\overline{E}>(\mathscr{T}-1))]\right)\neq\varnothing. \quad (8)\end{aligned}$$

Then, the satisfiability of this function can be investigated. An incremental SAT solver should be employed to mitigate a potential overhead caused by the necessity of constructing a different comparator in each iteration [51].

3.4 Average-case error analysis

Determining the average-case error represents a substantially harder problem because it requires the counting of the number of satisfiable assignments. For computing the average-case arithmetic error, for example, the same miter as in the previous case is used. The mean absolute error can be obtained by determining the error probability per each output bit. The obtained counts are then weighted according to the significance of the output bits and summed up. This is illustrated in Algorithm 2.

Algorithm 2: Mean absolute error computation

Input: n-input approximation miter with m-bit signed output e in the two's complement, i.e. $E = 2^m e_m - \sum_{i=0}^{m-1} 2^i e_i$
Output: mean absolute arithmetic error (e_{mae})

```
ε, c ← |ON-set(e_m)|
for i ∈ {0, 1, ..., m−1} do
    if c > 0 then
        ε ← ε + 2^i |ON-set(e_i ⊕ e_m)|
    else
        ε ← ε + 2^i |ON-set(e_i)|
return 2^−n ε;
```

4 Circuit approximation as a multi-objective optimization problem

The circuit approximation problem can be seen as a *multi-objective optimization problem*, i.e. an optimization problem that involves *multiple objective functions* $g_1(x), g_2(x), \ldots, g_k(x)$, where $g_i : X \to \mathbb{R}$, k is the number of objectives and $x, x \in X$ is a candidate circuit from the set of feasible circuits X [6]. In the context of approximate circuits, multiple objectives are typically defined to minimize one or several error metric(s), power consumption, area, and delay. The set of feasible solutions consists of all candidate circuits that satisfy the constraints imposed on the target circuit. For example, the worst-case error (e_{wce}) has to be less than a given constant, and the power consumption has to be smaller than another constant, as seen in Fig. 5.

In the multi-objective optimization, there does not typically exist one feasible solution that minimizes all objective functions simultaneously because the design objectives are conflicting. Hence, rather than one (optimal) solution, the optimization results in a set of solutions, i.e. the solutions that cannot be improved in any of the objectives without degrading at least one of the other objectives. Formally, a feasible solution $x^{(1)} \in X$ is said to *(Pareto) dominate* another solution $x^{(2)} \in X$, if

- $g_i(x^{(1)}) \leq g_i(x^{(2)})$ for all $i \in \{1, 2, \ldots, k\}$ and
- $g_j(x^{(1)}) < g_j(x^{(2)})$ for at least one index $j \in \{1, 2, \ldots, k\}$

and all g_i have to be minimized. A solution $x^* \in X$ is called a *non-dominated solution*, if there does not exist another solution that dominates it. The set of non-dominated solutions is called the *Pareto front*.

Figure 5 shows an example of Pareto front containing six non-dominated solutions (circles) and many dominated solutions (crosses) for two objectives to be minimized (e_{mae} and power consumption). The original (accurate) circuit is represented using a black diamond. Figure 5 also shows eight infeasible solutions that satisfy the constraints imposed on neither power consumption nor e_{wce}. We say that non-dominated solutions are *Pareto optimal solutions* if all possible candidate solutions are considered during the optimization, and there are no provably better non-dominated solutions in the search space. Claiming that some method finds a Pareto optimal solution without providing a correct proof of it is a clear failure of the author of the method. In practice, we are almost always faced with a situation in which a given method produces suboptimal solutions, i.e., the Pareto front contains the best non-dominated solutions obtained during the experiments conducted with the method. As it is not known "how far" the obtained solutions are from the truly Pareto optimal solutions, a common practice is to introduce a quality metric capable of measuring the distance between two sets of solutions obtained with two multi-objective optimization methods (see, for example, [6]) and compare them under this metric to conclude whether the first method is better than the second method or vice versa.

Another issue is the proper handling of constraints if two approximation methods are compared. For example, solution C4 on Fig. 5 would be on the Pareto front if

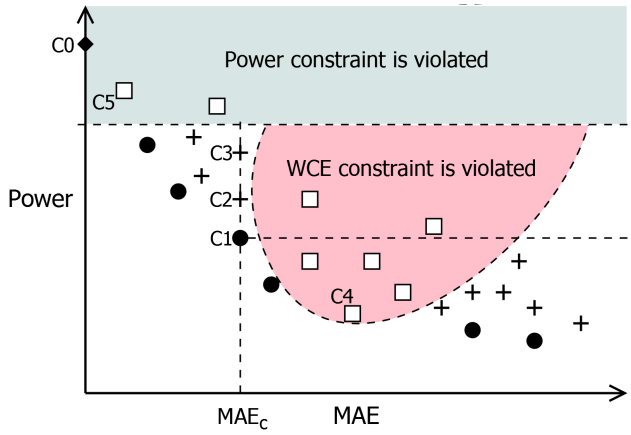


Fig. 5 Example of candidate designs in the objective space (power, MAE). Four types of designs are distinguished: non-dominated solutions (circles), dominated solutions (crosses), infeasible solutions (squares) and original solution (diamond).

no constraint were imposed on e_{wce}. But in our example, this constraint is specified. A direct comparison with a hypothetical approximation method, which does not specify the same constraint on e_{wce} and claims that C4 is a correct solution is then meaningless.

In Fig. 5, non-dominated solution C1 dominates solutions C2 and C3 (and also some infeasible solutions). A common misinterpretation of this situation is that if one is optimizing for selected criterion (e.g., $e_{mea} = MAE_c$). then solutions C1, C2, and C3 are good candidates and one of them can be selected depending on the available power budget. However, it makes no sense to choose C2 or C3 because C1 is always a strictly better solution under our original assumption that only e_{mae} and power consumption are considered.

5 Problem-specific approximation methods

This chapter deals with problem-specific approaches that were developed for obtaining approximate implementations of arithmetic circuits. They are created by experienced engineers who usually start with a common exact implementation. The approximation strategy is dependent on one particular type of circuits (e.g., adders or multipliers). It is not always possible to directly apply these techniques to other types of circuits.

A straightforward approximation technique that can be applied to various arithmetic circuits is *truncation*. In truncation, h-bit (exact) arithmetic circuit is used instead of n-bit circuit $(h < n)$ to reduce area, delay and power consumption. This h bit circuit is employed to process the most significant h bits of the n bit operands and the remaining $n-h$ bits are truncated, see also **??**. Because of its simplicity and

good results, this technique should be used as a baseline implementation for any comparisons. Moreover, the error profile obtained by truncation is well understood and, hence, the error of complex approximate circuits composed of the truncated circuits can naturally be analyzed.

As stated at the beginning of this chapter, we will primarily deal with approximate adders and multipliers because these circuits are essential in many applications and there is a rich body of literature on this topic.

5.1 Approximate Adders

An n-bit (common) adder adds two n-bit operands and produces an $n+1$ bit result. The most straightforward implementation (the so-called ripple-carry adder) is based on employing n one-bit full adders (FA) and propagating the carry from the least significant FA to the most significant FA. Although it requires a low amount of logic, its main disadvantage is that the carry chain introduces a long delay increasing linearly with respect to n. In order to reduce this delay, a carry-lookahead adder (CLA) is often employed, which is capable of predicting the input carry to any of the FAs in constant or log time, depending on available additional logic. Other circuit structures that provide some speedup with respect to the ripple-carry adder are carry-select adders and carry-skip adders, but again, additional logic must be available.

A recent detailed survey of Jiang et al. [21] classified the approximate implementations of n-bit adders into the following classes:

- *Speculative adders*, in which k bits ($k < n$) are used to speculate the carry for each sum bit [57]. This setup leads to a shorter carry chain and thus faster, but inexact addition.
- *Segmented adders*, in which the adder is divided into a number of smaller k-bit sub-adders (segments) operating in parallel. Fast addition is obtained as the carry propagation chain is truncated into shorter segments [22, 34, 41] and no carry is propagated among the sub-adders.
- *Carry-select adders*, in which the adder is also divided into segments, but the carry input for each subadder is selected using different strategies [7, 24, 29, 27, 15, 60, 1, 8].
- *Approximate multi-bit full adders*, in which the least significant bits are implemented by approximate FAs that are typically obtained by simplifying the exact FA at the transistor level [11, 30].

Some of these adders are constructed as accuracy configurable circuits, for example [45, 15, 22].

A detailed analysis conducted in [21] for these approximate adders under several error metrics revealed there is no superior approximation implementation which always provides the best tradeoffs. The user has to carefully choose the most suitable implementation for a particular application.

5.2 Approximate Multipliers

A typical implementation of the exact unsigned combinational n-bit multiplier is based on generating n n-bit partial products and summing them using $n-1$ ripple-carry adders organized in an array. In order to reduce delay, the ripple-carry adders are replaced with carry-save adders (the carry and sum signals generated by the adders in a row are passed on to the adders in the next row of the array) and partial results are summed with a structure called Wallace tree, which requires $\log(n)$ rows of adders. In these optimized multipliers, a-input / b-output important summing subcircuits (called counters and compressors) can be identified as building blocks. Multiplying of signed binary numbers in the two's complement notation is usually performed with Booth's algorithm, which effectively reduces the number of partial products and their bit width.

A recent detailed survey of Jiang et al. [21] classified the approximation methods for multipliers into the following classes:

- Approximation in generating *partial products*. Complex multipliers are composed of simplified elementary multipliers (such as the 2-bit approximate multiplier [25] that we discussed in section 2 or other smaller approximate multipliers [52]), but the accumulation becomes accurate.
- Approximation in the *partial product tree*, in which some adders or their parts are omitted, for example, because of truncation. Examples include broken array multipliers [30], error-tolerant multipliers [23] and static segment multipliers [39].
- Using *approximate designs of adders, counters or compressors* to accumulate the partial products, for example, [28, 35, 20, 13].
- *Approximate Booth multipliers* [26, 48, 58, 5, 9, 19].

Another group of approximation methods does not immediately start with a common multiplier but employs a different approach to obtaining the product. For example, rounding-based approximate (RoBA) multiplier tries to round the operands to the nearest exponent of two to omit the most computationally intensive part of the multiplication [60]. Truncation- and rounding-based scalable approximate multiplier (TOSAM) reduces the number of partial products by truncating each of the input operands based on their leading one-bit position. Hence, the multiplication can be replaced with shift, add, and small fixed-width multiplication operations [50]. In a dynamic range unbiased multiplier (DRUM), an m-bit segment is selected starting from the leading one bit of the input operands and the least significant bit of the truncated values is set to one. The truncated values are multiplied and shifted to the left to generate the final output [14]. Finally, the approximate multiplier can be based on computing an approximate logarithm for both the operands, summing the obtained values and computing antilog [43]. Several schemes for hardware implementation of log and antilog computation exist, but the linear Mitchells' approximation techniques are the most area-efficient [33].

With the development of specialized accelerators for deep learning in which it is useful to employ FP number representation, approximate implementations of FP multipliers have been proposed. Some of them are based on converting multiplica-

tion to the addition of approximate logarithms of the operands [43]. Another approach is to introduce a specific easy-to-compute function capable of approximating the multiplication [18]. Examples of configurable approximate FP multipliers are [17, 16].

A scalable divide-and-conquer strategy was developed for synthesizing a $2n$-bit approximate multiplier from four n-bit multipliers [38]. The operands are divided into four n-bit chunks (each operand has a lower and higher part) that are independently processed using four multipliers whose outputs are reduced using two adders with one n-bit and one $2n$-bit operand each. The key advantage of this method is that if accurate adders are employed and some of the n-bit multipliers are arbitrary chosen approximate multipliers with known e_{wce}, the upper bound of e_{wce} of the $2n$-bit approximate multiplier can be derived. If only one type of approximate multipliers is used, then e_{wce} can be calculated exactly. Moreover, this construction provides superior tradeoffs between the area and error in comparison with many state-of-the-art approximate multipliers [38].

6 Automated Approximation and EvoApprox Library

6.1 Automated methods

Automated functional approximation methods start with a common (exact) circuit implementation and define one or several design objectives and constraints. As discussed in Chapter 4, the circuit approximation problem can be seen as a multi-objective design problem, where the desired output is a set of non-dominated designs from a Pareto front. As this chapter deals with arithmetic circuits, the approximation is typically conducted at the gate level. The initial circuit is modified by an iterative approximation algorithm to produce an approximate implementation satisfying design objectives and other constraints.

The basic algorithmic approximation techniques are pruning (i.e., removing some parts of the circuit), component replacement (i.e., complex subcircuits are replaced with simpler subcircuits), and approximate re-synthesis. If, however, the circuit is provided in a behavioral HDL representation, other more software-oriented techniques (such as loop perforation and memorization, see chapter **??** for more details on these techniques) can be applied. The automated approximation methods select either randomly or heuristically which parts of the circuit have to be removed, re-connected or replaced. Table 2 gives examples of automated approximation methods, benchmark problems used to evaluate them, and the error evaluation approaches.

One of the automated methods – Cartesian genetic programming (CGP) – is briefly introduced in Fig. 6. Based on an original circuit that is supplied by the user, CGP instantiates a population of candidate designs. As CGP is an evolutionary circuit design method, new candidate designs are created by introducing random

Table 2 Selected automated approximation methods, benchmark problems used to evaluate them and the error evaluation approaches.

Method	Ref.	Benchmarks	Error analysis by:
ABACUS	[40]	FIR[b], perceptron, block matcher	simulation
ABM	[47]	6 ISCAS85 benchmark circuits	BDD
ALFANS	[59]	8 bit multipliers, 32 bit adders, MCNC benchmarks	SAT, BDD
ASLAN	[42]	FIR[b], IIR[c], MAC[f], DCT[d], Sobel and 8-input neuron	sequential QCC[a](SAT)
CGP	[52]	2 to 16 bit multipliers, 9-input and 25-input median	simulation
CGP-BDD	[53]	16 circuits from LGSynth, ITC and ISCAS	BDD
CGP-SAT	[2]	8 to 32 bit multipliers, 128 bit adders	SAT
SALSA	[55]	Adders, multipliers, FIR[b], IIR[c], DCT[d]...	QCC[a](SAT)
SASIMI	[56]	ISCAS85 benchmarks, multipliers, adders,...	simulation

[a] Quality Constraint Circuit [b] Finite Impulse Response filter [c] Infinite Impulse Response filter [d] Discrete Cosine Transform [e] Fast Fourier Transform [f] Vector Dot Product

mutations (i.e., modifications) to the circuit netlist. Candidate designs generated by Circuit Generator can be constrained in various ways; for example, only circuits having an acceptable number of gates or showing an error below a given threshold are marked as feasible. Candidate designs are evaluated in terms of error (circuit simulation is combined with formal error analysis methods) and the key electrical parameters are quickly estimated. The best-scored circuits then serve as the parents of the new population. This iterative process is repeated for a predefined number of iterations. The resulting approximate circuits are fully characterized using professional design tools. Details of the method are presented in [52, 2, 44]. CGP was also employed to evolve efficient implementations of quality configurable circuits [37].

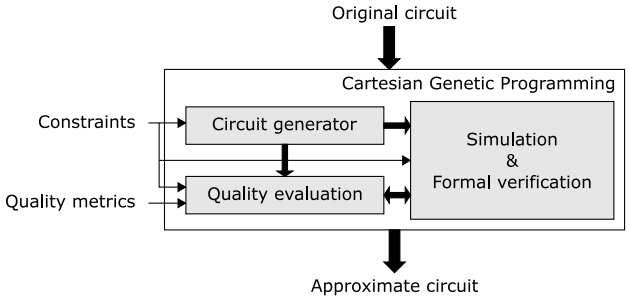


Fig. 6 Employing Cartesian genetic programming for automated design of approximate circuits

6.2 EvoApprox Library

A comprehensive library of approximate arithmetic circuits called EvoApprox8b [36] was introduced in 2017. The idea was to provide well-characterized circuits that can

immediately be used in target applications. All circuits were automatically designed by means of CGP. EvoApprox8b contains hundreds of 8-bit approximate adders and multipliers. All circuits were fully characterized in terms of several error metrics and synthesized with Synopsys Design Compiler (45 nm process, V_{dd} =1V) to obtain their area, delay, and power consumption. By means of a simple web user interface, the user can choose the most suitable circuit based on the criteria she provides.

In 2019, the library was extended by running additional CGP runs for different objectives and bit widths. It now contains thousands of various arithmetic circuits, as shown in Table 3. In order to simplify the selection of the most suitable circuit for a given application, we identified a subset of circuits and composed EvoApprox8b-Lite. The selection follows the principles of Pareto optimality with respect to several objectives in which power consumption is compared against e_{prob}, e_{mae}, e_{wce}, e_{mse} and e_{mre} metrics. For each of the five subsets of components, ten circuits evenly distributed along the power axis were included to EvoApprox8b-Lite.

Table 3 The number of approximate implementations of arithmetic circuits in extended EvoApprox library (December 2019)

Circuit	Bit-width	# approx. implementations
adder	8	6,979
	9	332
	12	4,661
	16	1,437
	32	916
	64	176
	128	196
multiplier	8	29,911
	12	3,495
	16	35,406
	32	349

Power vs. e_{mae} tradeoffs of thousands of 8-bit approximate multipliers are shown in Fig. 7. The black points (corresponding with the EvoApprox8b-Lite) are contrasted with the original circuits of EvoApprox8b (red points) and conventional broken array multipliers (green points) and truncated multipliers (blue points). Note that EvoApprox8b was compared with state of art approximate circuits in a greater detail [36]. Selected approximate circuits and their various parameters can be downloaded from https://ehw.fit.vutbr.cz/evoapproxlib.

The library provides circuit models in Verilog, Matlab, Python, and C. These models enable the user to integrate the approximate circuits to hardware as well as software projects and design tools. All approximate circuits can thus be simulated in order to obtain their other parameters that are not listed on the website (e.g., the errors under different error metrics or power consumption for another fabrication technology).

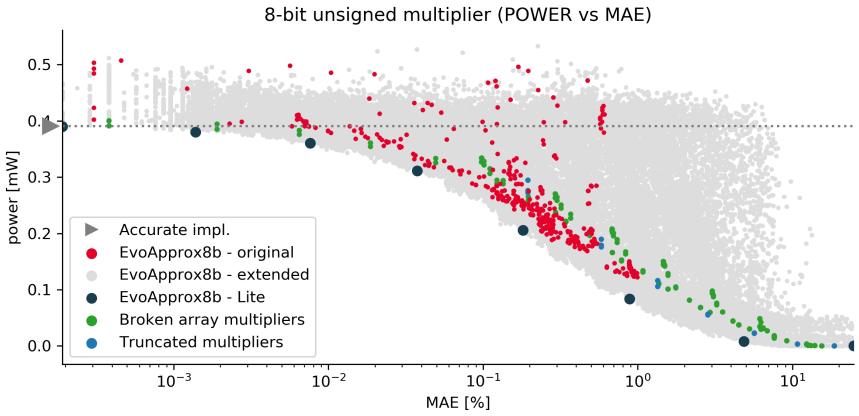


Fig. 7 The 8-bit approximate multipliers (black points) that were selected to EvoApprox8b-Lite from all the evolved approximate multipliers (gray points) and compared to the former version of EvoApprox8b library (red points), broken array multipliers (green points) and truncated multipliers (blue points).

7 Experiments with Error Analysis Methods

This section includes case studies that demonstrate some interesting aspects of the circuit approximation methods, particularly the issues related to the exact error analysis.

7.1 Computational requirements of error analysis methods

Detailed analysis of relaxed equivalence checking algorithms has recently been presented in [51]. The analysis revealed that the computational complexity of the SAT-based methods heavily depends on the actual worst-case error. The computational time increases with a decreasing error, which is noticeable, especially on multipliers. For example, tens of milliseconds are needed to analyze the 12-bit multipliers having an error higher than 2.7%. On the other hand, higher tens of seconds are needed for instances having the error in the range (0.37%, 2.71%], and no result was obtained for multipliers having the worst-case error below 0.05% [51].

Figure 8 shows the computational requirements of the WCEGT procedure (i.e. worst-case error checking) for five different thresholds applied to 8-bit multipliers taken from the EvoApprox library. The worst-case error checking is extremely fast (few milliseconds are required) but only if the actual worst-case error (denoted as wce) is higher than a given threshold $\mathscr{T}$. If this condition is violated, the CPU time may increase by several orders of magnitude. Surprisingly, the difference between the worst case and the best case CPU time increases with decreasing the threshold

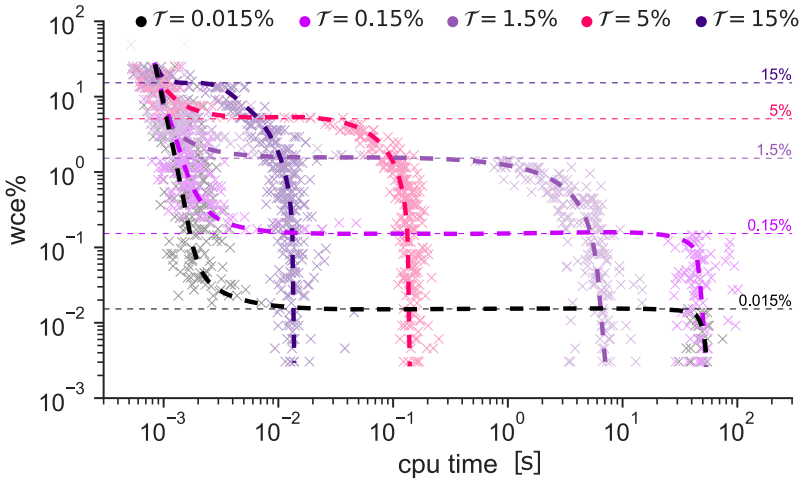


Fig. 8 The computational requirements of the WCEGT procedure proving that $e_{wce} > \mathscr{T}$ of 8-bit approximate multipliers taken from EvoApprox library.

$\mathscr{T}$. Performing WCEGT for thresholds below 1.5% represents the most difficult case. We have to emphasize that the algorithm always terminates for the 8-bit multipliers. Up to 100 seconds are required to analyze the circuit instances whose wce is lower than the chosen threshold.

The same trend was also observed for bigger multipliers. Considering this fact, the design of multiplier-based approximate circuits with low error will be a challenging task because the error analysis will represent a bottleneck of the whole design process.

7.2 The accuracy of circuit simulation

For all 8-bit and 16-bit approximate adders and multipliers available in the EvoApprox library, the error was exactly calculated for all relevant error metrics. Knowing the exact errors, we could perfectly analyze the error of the circuit simulation method, which is conducted with a subset of all input vectors. The objective is to determine the minimum number of test vectors that has to be applied to keep the error of circuit simulation below a given threshold.

Let E_{exact} and E_{est} denote the exact error and the error estimated by circuit simulation. Relative difference (RD) [%] between E_{exact} and E_{est} is defined as

$$RD = 100\frac{E_{est} - E_{exact}}{E_{exact}}[\%]. \tag{9}$$

Boxplots in Figures 9, 10 and 11 show how *RD* depends on the number of input vectors for different circuits. To create one boxplot for e_{wce}, we randomly generated the requested number of input vectors, applied them on 6,275 approximate circuits taken from EvoApprox library and calculated *RD*. No accurate circuit was considered in the evaluation, i.e. E_{exact} is always greater than zero. The same was done for e_{mae}. A clear consequence of this approach which utilizes randomly generated (but not necessarily unique) vectors is that a non-zero *RD* is obtained even if the number of generated vectors is identical with the number of all possible input combinations.

In the case of 8-bit approximate multipliers, it makes no sense to use a subset of input vectors during simulation because *RD* can be higher than 5% even if two-thirds of vectors are used. Moreover, analyzing circuit responses for all $2^{8+8} = 2^{16} = 65,536$ vectors is very fast (few milliseconds on a common CPU [51]). Hence, performing the simulation for all possible input combinations is the best choice.

In the case of 16-bit multipliers, *RD* for e_{wce} can reach over 10% if $144 \cdot 10^6$ vectors are used. Note that 16-bit approximate multipliers are usually analyzed using only $10 \cdot 10^6$ vectors in some studies [21]. On the other hand, we obtained very reliable error estimates for e_{mae} with less than $5 \cdot 10^6$ vectors. Finally, we analyzed 16-bit adders. A very reliable error characterization in terms of e_{wce} as well as e_{mae} requires a considerably lower number of randomly generated vectors, i.e. less than $5 \cdot 10^6$ vectors as seen in Fig. 11.

We can summarize an intuitive fact that estimating e_{mae} with circuit simulation is more reliable than estimating e_{wce} if only a subset of input vectors is used. We encourage the practitioners to provide more statistically relevant error characterizations (e.g., the mean RD and its standard deviation) if the error of approximate circuits is estimated.

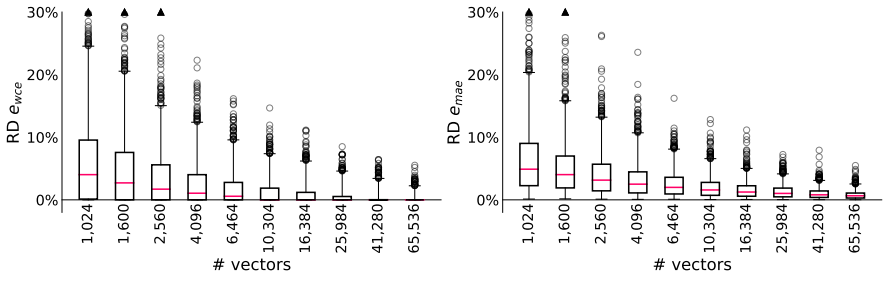


Fig. 9 Relative difference between exact and estimated error for 8 bit approximate multipliers. The whiskers show the 2nd percentile and the 98th percentile. Triangles indicate that there is even higher RD than shown.

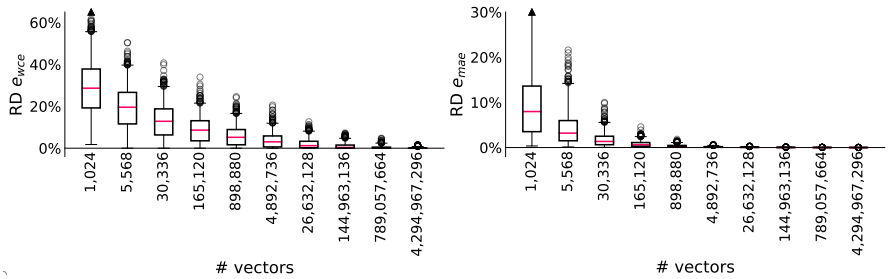


Fig. 10 Relative difference between exact and estimated error for 16 bit approximate multipliers. The whiskers show the 2nd percentile and the 98th percentile. Triangles indicate that there is even higher RD than shown.

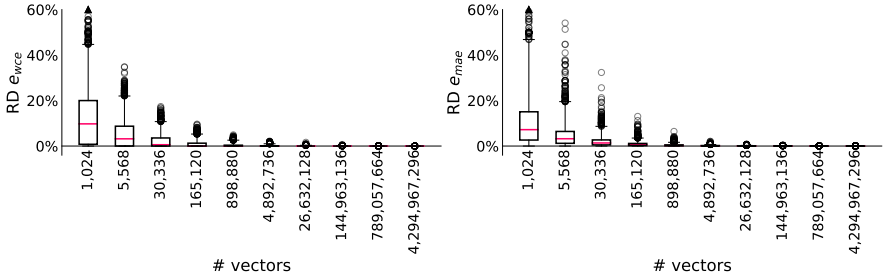


Fig. 11 Relative difference between exact and estimated error for 16 bit approximate adders. The whiskers show the 2nd percentile and the 98th percentile. Triangles indicate that there is even higher RD than shown.

8 Conclusions

In this chapter, we surveyed various methodological aspects that are relevant for the design of approximate arithmetic circuits. Special attention was given to exact error analysis methods and understanding the circuit approximation problem as a multi-objective optimization problem. We briefly presented problem-specific as well as automated approximation methods developed for adders and multipliers. Unfortunately, misunderstanding of the principles of correct evaluation of approximate circuits and correct benchmarking of circuit approximation methods are still visible in the literature. We believe that this chapter can help in establishing a better practice in this emerging area.

Acknowledgements This work was supported by the Czech science foundation project 19-10137S.

References

[1] Camus V, Schlachter J, Enz C (2016) A low-power carry cut-back approximate adder with fixed-point implementation and floating-point precision. In: 2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC), pp 1–6

[2] Ceska M, Matyas J, Mrazek V, Sekanina L, Vasicek Z, Vojnar T (2017) Approximating complex arithmetic circuits with formal error guarantees: 32-bit multipliers accomplished. In: 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp 416–423

[3] Chandrasekharan A, Soeken M, Große D, Drechsler R (2016) Precise error determination of approximated components in sequential circuits with model checking. In: Proc. of DAC'16, ACM, pp 1–6

[4] Chen L, Han J, Liu W, Lombardi F (2017) Algorithm and design of a fully parallel approximate coordinate rotation digital computer (cordic). IEEE Transactions on Multi-Scale Computing Systems 3(3):139–151

[5] Chen Y, Chang T (2012) A high-accuracy adaptive conditional-probability estimator for fixed-width booth multipliers. IEEE Transactions on Circuits and Systems I: Regular Papers 59(3):594–603, DOI 10.1109/TCSI.2011.2167275

[6] Coello Coello CA, Gonzalez Brambila S, Figueroa Gamboa J, Castillo Tapia MG, Hernandez Gomez R (2020) Evolutionary multiobjective optimization: open research areas and some challenges lying ahead. Complex & Intelligent Systems 2020:1–16

[7] Du K, Varman P, Mohanram K (2012) High performance reliable variable latency carry select addition. In: 2012 Design, Automation Test in Europe Conference Exhibition (DATE), pp 1257–1262, DOI 10.1109/DATE.2012.6176685

[8] Ebrahimi-Azandaryani F, Akbari O, Kamal M, Afzali-Kusha A, Pedram M (2019) Block-based carry speculative approximate adder for energy-efficient applications. IEEE Transactions on Circuits and Systems II: Express Briefs pp 1–1, DOI 10.1109/TCSII.2019.2901060

[9] Farshchi F, Abrishami MS, Fakhraie SM (2013) New approximate multiplier for low power digital signal processing. In: The 17th CSI International Symposium on Computer Architecture Digital Systems (CADS 2013), pp 25–30, DOI 10.1109/CADS.2013.6714233

[10] Froehlich S, Große D, Drechsler R (2019) One method - all error-metrics: A three-stage approach for error-metric evaluation in approximate computing. In: 2019 Design, Automation Test in Europe Conference Exhibition (DATE), pp 284–287

[11] Gupta V, Mohapatra D, Raghunathan A, Roy K (2013) Low-power digital signal processing using approximate adders. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 32(1):124–137

[12] Gysel P, Pimentel J, Motamedi M, Ghiasi S (2018) Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks. IEEE Trans Neural Netw Learn Syst 29(11):5784–5789

[13] Ha M, Lee S (2018) Multipliers with approximate 4 – 2 compressors and error recovery modules. IEEE Embedded Systems Letters 10(1):6–9, DOI 10.1109/LES.2017.2746084

[14] Hashemi S, Bahar RI, Reda S (2015) Drum: A dynamic range unbiased multiplier for approximate applications. In: 2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp 418–425, DOI 10.1109/ICCAD.2015.7372600

[15] Hu J, Qian W (2015) A new approximate adder with low relative error and correct sign calculation. In: 2015 Design, Automation Test in Europe Conference Exhibition (DATE), pp 1449–1454

[16] Imani M, Peroni D, Rosing T (2017) Cfpu: Configurable floating point multiplier for energy-efficient computing. In: 2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC), pp 1–6, DOI 10.1145/3061639.3062210

[17] Imani M, Garcia R, Gupta S, Rosing T (2018) Rmac: Runtime configurable floating point multiplier for approximate computing. In: Proceedings of the International Symposium on Low Power Electronics and Design, ACM, New York, NY, USA, ISLPED '18, pp 12:1–12:6, DOI 10.1145/3218603.3218621, URL http://doi.acm.org/10.1145/3218603.3218621

[18] Imani M, Sokolova A, Garcia R, Huang A, Wu F, Aksanli B, Rosing T (2019) Approxlp: Approximate multiplication with linearization and iterative error control. In: 2019 56th ACM/IEEE Design Automation Conference (DAC), pp 1–6

[19] Jiang H, Han J, Qiao F, Lombardi F (2016) Approximate radix-8 booth multipliers for low-power and high-performance operation. IEEE Transactions on Computers 65(8):2638–2644, DOI 10.1109/TC.2015.2493547

[20] Jiang H, Liu C, Lombardi F, Han J (2019) Low-power approximate unsigned multipliers with configurable error recovery. IEEE Transactions on Circuits and Systems I: Regular Papers 66(1):189–202, DOI 10.1109/TCSI.2018.2856245

[21] Jiang H, Liu L, Lombardi F, Han J (2019) Approximate arithmetic circuits: Design and evaluation. In: Reda S, Shafique M (eds) Approximate Circuits, Methodologies and CAD, Springer, pp 67–98, DOI 10.1007/978-3-319-99322-5_4, URL https://doi.org/10.1007/978-3-319-99322-5_4

[22] Kahng AB, Kang S (2012) Accuracy-configurable adder for approximate arithmetic designs. In: DAC Design Automation Conference 2012, pp 820–825, DOI 10.1145/2228360.2228509

[23] Khaing Yin Kyaw, Wang Ling Goh, Kiat Seng Yeo (2010) Low-power high-speed multiplier for error-tolerant application. In: 2010 IEEE International Conference of Electron Devices and Solid-State Circuits (EDSSC), pp 1–4, DOI 10.1109/EDSSC.2010.5713751

[24] Kim Y, Zhang Y, Li P (2013) An energy efficient approximate adder with carry skip for error resilient neuromorphic vlsi systems. In: 2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp 130–137

[25] Kulkarni P, Gupta P, Ercegovac M (2011) Trading accuracy for power with an underdesigned multiplier architecture. In: 2011 24th Internatioal Conference on VLSI Design, pp 346–351, DOI 10.1109/VLSID.2011.51

[26] Kyung-Ju Cho, Kwang-Chul Lee, Jin-Gyun Chung, Parhi KK (2004) Design of low-error fixed-width modified booth multiplier. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 12(5):522–531, DOI 10.1109/TVLSI.2004.825853

[27] Li L, Zhou H (2014) On error modeling and analysis of approximate adders. In: 2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp 511–518

[28] Lin C, Lin I (2013) High accuracy approximate multiplier with error correction. In: 2013 IEEE 31st International Conference on Computer Design (ICCD), pp 33–38, DOI 10.1109/ICCD.2013.6657022

[29] Lin I, Yang Y, Lin C (2015) High-performance low-power carry speculative addition with variable latency. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 23(9):1591–1603, DOI 10.1109/TVLSI.2014.2355217

[30] Mahdiani HR, Ahmadi A, Fakhraie SM, Lucas C (2010) Bio-inspired imprecise computational blocks for efficient vlsi implementation of soft-computing applications. IEEE Transactions on Circuits and Systems I: Regular Papers 57(4):850–862, DOI 10.1109/TCSI.2009.2027626

[31] Mazahir S, Hasan O, Hafiz R, Shafique M, Henkel J (2017) Probabilistic error modeling for approximate adders. IEEE Transactions on Computers 66(3):515–530

[32] Melchert J, Behroozi S, Li J, Kim Y (2019) Saadi-ec: A quality-configurable approximate divider for energy efficiency. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 27(11):2680–2692

[33] Mitchell JN (1962) Computer multiplication and division using binary logarithms. IRE Transactions on Electronic Computers EC-11(4):512—-517

[34] Mohapatra D, Chippa VK, Raghunathan A, Roy K (2011) Design of voltage-scalable meta-functions for approximate computing. In: 2011 Design, Automation Test in Europe, pp 1–6, DOI 10.1109/DATE.2011.5763154

[35] Momeni A, Han J, Montuschi P, Lombardi F (2015) Design and analysis of approximate compressors for multiplication. IEEE Transactions on Computers 64(4):984–994, DOI 10.1109/TC.2014.2308214

[36] Mrazek V, Hrbacek R, Vasicek Z, Sekanina L (2017) Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods. In: Design, Automation Test in Europe Conference Exhibition (DATE), 2017, pp 258–261

[37] Mrazek V, Vasicek Z, Sekanina L (2018) Design of quality-configurable approximate multipliers suitable for dynamic environment. In: Proceedings of the 2018 NASA/ESA Conference on Adaptive Hardware and Systems, IEEE, pp 264–271

[38] Mrazek V, Vasicek Z, Sekanina L, Jiang H, Han J (2018) Scalable construction of approximate multipliers with formally guaranteed worst case error. IEEE

Transactions on Very Large Scale Integration (VLSI) Systems 26(11):2572–2576, DOI 10.1109/TVLSI.2018.2856362

[39] Narayanamoorthy S, Moghaddam HA, Liu Z, Park T, Kim NS (2015) Energy-efficient approximate multiplication for digital signal processing and classification applications. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 23(6):1180–1184, DOI 10.1109/TVLSI.2014.2333366

[40] Nepal K, Hashemi S, Tann H, Bahar RI, Reda S (2019) Automated high-level generation of low-power approximate computing circuits. IEEE Transactions on Emerging Topics in Computing 7(1):18–30

[41] Ning Zhu, Goh WL, Yeo KS (2009) An enhanced low-power high-speed adder for error-tolerant application. In: Proceedings of the 2009 12th International Symposium on Integrated Circuits, pp 69–72

[42] Ranjan A, Raha A, Venkataramani S, Roy K, Raghunathan A (2014) ASLAN: Synthesis of approximate sequential circuits. In: Proceedings of the Conference on Design, Automation and Test in Europe, EDA Consortium, DATE'14, pp 1–6

[43] Saadat H, Bokhari H, Parameswaran S (2018) Minimally biased multipliers for approximate integer and floating-point multiplication. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 37(11):2623–2635

[44] Sekanina L, Vasicek Z, Mrazek V (2019) Automated search-based functional approximation for digital circuits. In: Reda S, Shafique M (eds) Approximate Circuits, Methodologies and CAD, Springer, pp 175–203, DOI 10.1007/978-3-319-99322-5_9, URL https://doi.org/10.1007/978-3-319-99322-5_9

[45] Shafique M, Ahmad W, Hafiz R, Henkel J (2015) A low latency generic accuracy configurable adder. In: 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), pp 1–6

[46] Shafique M, Hafiz R, Rehman S, et al (2016) Invited: Cross-layer approximate computing: From logic to architectures. In: DAC'16

[47] Soeken M, Grosse D, Chandrasekharan A, Drechsler R (2016) BDD minimization for approximate computing. In: 21st Asia and South Pacific Design Automation Conference ASP-DAC 2016, IEEE, pp 1–6

[48] Song MA, Van LD, Kuo SY (2007) Adaptive low-error fixed-width booth multipliers. IEICE Trans Fundam Electron Commun Comput Sci E90-A(6):1180–1187, DOI 10.1093/ietfec/e90-a.6.1180, URL http://dx.doi.org/10.1093/ietfec/e90-a.6.1180

[49] Ullah S, Rehman S, Prabakaran BS, Kriebel F, Hanif MA, Shafique M, Kumar A (2018) Area-optimized low-latency approximate multipliers for fpga-based hardware accelerators. In: 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC), pp 1–6, DOI 10.1109/DAC.2018.8465781

[50] Vahdat S, Kamal M, Afzali-Kusha A, Pedram M (2019) Tosam: An energy-efficient truncation- and rounding-based scalable approximate multiplier. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 27(5):1161–1173

[51] Vasicek Z (2019) Formal methods for exact analysis of approximate circuits. IEEE Access 7(1):177,309–177,331

[52] Vasicek Z, Sekanina L (2015) Evolutionary approach to approximate digital circuits design. IEEE Transactions on Evolutionary Computation 19(3):432–444

[53] Vasicek Z, Sekanina L (2016) Evolutionary design of complex approximate combinational circuits. Genetic Programming and Evolvable Machines 17(2):1–24

[54] Vasicek Z, Mrazek V, Sekanina L (2019) Automated circuit approximation method driven by data distribution. In: Design, Automation and Test in Europe Conference, European Design and Automation Association, pp 96–101

[55] Venkataramani S, Sabne A, Kozhikkottu VJ, Roy K, Raghunathan A (2012) SALSA: systematic logic synthesis of approximate circuits. In: The 49th Design Automation Conference, ACM, pp 796–801, DOI 10.1145/2228360.2228504

[56] Venkataramani S, Roy K, Raghunathan A (2013) Substitute-and-simplify: a unified design paradigm for approximate and quality configurable circuits. In: Design, Automation and Test in Europe, DATE'13, EDA Consortium, pp 1367–1372

[57] Verma AK, Brisk P, Ienne P (2008) Variable latency speculative addition: A new paradigm for arithmetic circuit design. In: 2008 Design, Automation and Test in Europe, pp 1250–1255, DOI 10.1109/DATE.2008.4484850

[58] Wang J, Kuang S, Liang S (2011) High-accuracy fixed-width modified booth multipliers for lossy applications. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 19(1):52–60, DOI 10.1109/TVLSI.2009.2032289

[59] Wu Y, Qian W (2019) Alfans: Multi-level approximate logic synthesis framework by approximate node simplification. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems pp 1–14

[60] Zendegani R, Kamal M, Bahadori M, Afzali-Kusha A, Pedram M (2017) Roba multiplier: A rounding-based approximate multiplier for high-speed yet energy-efficient digital signal processing. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 25(2):393–401, DOI 10.1109/TVLSI.2016.2587696