Relaxed equivalence checking: a new challenge in logic synthesis

Zdenek Vasicek
Brno University of Technology, Faculty of Information Technology, IT4Innovations Centre of Excellence
Bozetechova 2, 612 66 Brno, Czech Republic
Email: vasicek@fit.vutbr.cz

***Abstract*—The functional equivalence has always been the integral part of virtually every logic synthesis tool. The formal equivalence checking represents a key process that helps logic synthesis tool guarantee that two representations of a circuit design exhibit exactly the same behavior. Among others, equivalence checking is routinely applied to prove that a synthesized digital circuit is logically equivalent to the RTL source code. Although formal equivalence checking has matured greatly during the last two decades and designs with millions of gates can be handled and verified in reasonable time, a new challenge has emerged with the recent advent of approaches addressing the problem of synthesis of approximate circuits.**

Approximate circuits are digital circuits which are intentionally designed in such a way that the specification is violated in terms of functionality in order to obtain some improvements in power consumption, performance or area, in comparison with fully functional circuits. The main problem related to the synthesis of approximate circuits is the checking that the synthesized circuits meet the specification. The nature of the approximate circuits involves to replace the strict formal equivalence checking with more advanced methods that enable to perform so called relaxed equivalence checking, i.e. checking that two circuit designs are equal up to some bound. In addition to that, it is crucial to perform the checking as quickly as possible because this procedure is employed in iterative design process. Compared to the formal equivalence checking, only little has been done in this area and the relaxed equivalence checking still represents an open and challenging problem.

The purpose of this paper is to survey and briefly introduce the methods proposed to address the problem of relaxed equivalence checking especially in the context of approximate computing and summarize and discuss the main challenges of this research problem in the context of logic synthesis.

I. Introduction

The logic synthesis tools originally aimed at performance optimization and area minimization. However, the requirement for energy-efficient circuits forced designers to update the available design methods and include power dissipation as the third design parameter. Recently, a new research field was established to investigate how to make computer systems more energy efficient and faster. This field has been coined as *approximate computing*. Approximate design flow extends the established concept of logic synthesis and introduces a fourth design parameter – the error. The key motivation behind approximate computing is the inherent error resilience of many real-world applications. The error resilience means that we are able to accept and synthesize circuits that produce invalid output values. The errors are typically not recognizable in the target application because of limited human perception capabilities or non-existence of a golden model. Among others, multimedia applications, data mining and data learning represent typical examples of error resilient applications where the error can easily be traded-off for power savings. It seems that inherent error resilience of many real-world applications offers a huge potential for power savings. For example, Chippa et al. reported that about more than 83% of runtime is spent in computations that can be approximated [1]. Another motivation for approximate computing can be seen in the recently introduced concept of underdesigned and opportunistic computing which attempts to explore the possibility of constructing machines naturally exploiting various imperfections of underlying hardware [2].

The field of approximate computing is at an early stage of development, but there is a very active research community involved in approximate computing. The current research cover the whole computing stack, integrating thus areas of microelectronics, circuits, components, architectures, networks, operating systems, compilers and applications. Approximations are conducted for embedded systems, ordinary computers, graphics processing units and even field-programmable gate arrays. A good survey of existing techniques and results can be found for example in [3], [4].

Let us briefly discuss the approximations conducted at the circuit level. In hardware, typical modifications of the accurate circuit involve the bit width reduction, intentional disconnecting of subsystems, fault injection and changes in timing and power supply voltage [1], [3]. The common feature of the first three approaches is the usage of inexact circuits. The question is how to design them. Changes in timing and power supply voltage represent a generic approach that can be applied to any digital circuit [5], [6]. In this case, the circuits are designed to be working perfectly under a normal environment. However, their energy consumption is reduced by voltage over-scaling, i.e. using lower power supply voltage in which the circuit is known to occasionally produce erroneous outputs. Conversely, performance can be increased when the circuit is over-clocked. In both cases, timing induced errors appear because some paths in the circuit fail to meet the delay constraints.

Let us further focus on the design of inexact circuits in the rest of the paper. Initially, the authors approximate the key circuit components manually. The paper of Kulkarni et al. [7] is often cited as one of the seminal works in this area. Kulkarni designed an inexact 2-bit multiplier providing a correct output for 15 out of 16 input combinations. Interestingly, he removed the most significant output bit to achieve improvement in performance and power consumption. Despite of that, the obtained multiplier exhibits a relative low error. The multiplication of 3×3 is represented with 7 instead of 9. As a consequence of that, Kulkarni reduced the area by half compared to the exact multiplier and achieved a shorter and faster critical path. It was also demonstrated that arbitrarily large multipliers can be built using the approximate 2-bit multiplier as the building block. By choosing between accurate and inexact versions of the 2×2 block in a large multiplier, a trade-off between error rate and power saving can be achieved.

The manual approximation have recently been replaced by fully automated systematic methods that increase the design productivity as well as the quality and complexity of circuits that can be approximated. The systematic methods (such as SALSA [8], SASIMI [9], ABACUS [10] and various search-based approaches such as [11], [12], [13]) produce Pareto fronts showing various compromise solutions with respect to the optimized accurate implementation. Let us briefly mention some of the most encouraging recent results involving the usage of inexact circuits.

Nepal et al. investigated, for example, how to reduce power dissipation of a block-matching circuit, an essential component of motion estimation circuit [14]. Motion estimation and compensation represent key parts of video compression algorithms and almost all video coding standards use these operations to exploit temporal redundancy. The authors synthesized an approximate version of block matching circuit based on sum of absolute difference. They reported 22% reduction in power dissipation for a very small reduction in accuracy [14]. Additional 23% improvement in power savings was achieved by applying voltage scaling, i.e. reducing supply voltage. The approximate circuits were designed using an improved version of ABACUS tool. ABACUS creates an abstract synthesis tree from the input behavioral description and then applies various operators to the AST using an iterative stochastic greedy algorithm [10].

Mrazek et al. proposed a methodology for the design of well-optimized power-efficient neural networks (NN) with a uniform structure suitable for hardware implementation [12]. Since NN contains hundreds of thousands multiplications, the authors replaced the exact multipliers with approximate multipliers to reduce power consumption. In this work, genetic algorithm was employed to design 8-bit and 12-bit application-specific approximate multipliers. The experiments not only confirmed high error resiliency of NNs but also revealed the capability of the back propagation learning algorithm to adapt with NNs containing the approximate multipliers. The classification accuracy remained unchanged despite the fact that 57% improvement in power saving was achieved by introducing 8-bit approximate multipliers into a convolutional NN trained for the classification of street-view house numbers.

In addition to that, various key components such as small adders, multipliers, simple FIR and IIR filters and DCT and FFT blocks have been approximated [3], [8], [9], [10], [13].

A. Functional approximation

Albeit logic synthesis and optimization represents the research area with more than fifty years of history, the conventional synthesis tools have never been constructed to perform the synthesis of approximate (i.e. erroneous) circuits. The acceptance of partially working solutions during the design process represent only one part of the problem. One of the possibilities how to design energy efficient approximate circuits, for example, is to constrain the number of available gates and let the synthesis tool produce a circuit with minimal error with respect to the accurate circuit. This approach is referred to as the resource-oriented design method [11]. Interestingly, there is no conventional method that could directly solve such an optimization problem and completely new design methods have to be designed to support this design scenario.

Functional approximation represents one of the most popular design methods. The idea of functional approximation is to implement a slightly different function to the original one provided that the error is acceptable and the power consumption or other system parameters are reduced adequately. A functional approximation is typically obtained by a heuristic procedure that modifies the original (i.e. accurate) implementation. This heuristic procedure is repeated iteratively in order to improve the current approximate implementation in the subsequent steps. In each iteration, it is necessary to evaluate to what extent a given approximation satisfies functional and non-functional requirements (area, power consumption, etc.) of the specification. After determining the error, non-functional properties of candidate approximations are also evaluated.

The functionality is expressed using one or several error metrics such as error probability, average-case error, or worst-case error. Unfortunately, the error metrics need to be tailored to a particular application. When an arithmetic circuit is approximated, for example, it is necessary to base the error quantification on an arithmetic error metric since the error magnitude could have a significant impact on target application. For general logic where no additional knowledge is available and where there does not exist a well-accepted error model, Hamming distance or error rate is typically employed. The approximation of general logic is sometimes criticized as the usage of approximate logic could potentially lead to unpredictable or even fatal behavior. However, it has been demonstrated that the approximate circuits can reduce overhead of dependable systems based on triple modular redundancy [15].

Let $f : \mathbb{B}^n \to \mathbb{B}^m$ be a Boolean function that describes correct functionality (specification) and $\hat{f} : \mathbb{B}^n \to \mathbb{B}^m$ an approximation of it, both implemented by two circuits, namely F and $\hat{\mathrm{F}}$. The following paragraphs summarize the error metrics that have been employed in literature to quantify the deviation between the outputs produced by a functionally correct design and an approximate design.

1) Arithmetic error metrics: The *worst-case error*, sometimes denoted as *error magnitude* or *error significance* [16], is defined as

$$e_{wst}(f, \hat{f}) = \max_{\forall x \in \mathbb{B}^n} |\operatorname{nat}(f(x)) - \operatorname{nat}(\hat{f}(x))| \quad (1)$$

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

Similarly, *relative worst-case error* can be employed

$$e_{rel}(f, \hat{f}) = \max_{\forall x \in \mathbb{B}^n} \frac{|\operatorname{nat}(f(x)) - \operatorname{nat}(\hat{f}(x))|}{\operatorname{nat}(f(x))} \quad (2)$$

to constrain the approximate circuit to differ from the correct one by at most a certain margin. Note that a special care must be devoted to the cases for which the output value of the original circuit is equal to zero, i.e. $nat(f(x)) = 0$.

The *average-case error* is defined as the sum of absolute differences in magnitude between the original and approximate circuits, averaged over all inputs:

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

Instead of the absolute error values e_{wst} and e_{avg} depending on the bit width of the original circuits, the corresponding relative error values $e_{wst\%}$ and $e_{avg\%}$ can be utilized. In this case, it is common to express the percentage error ratio as the percentage of the maximum value M ($0 < M \leq 2^m$) that can occur on the output of the correct circuit.

2) General error metrics: In addition to the arithmetic error metrics, there are metrics that are not related to the magnitude of the output of the correct or approximate circuit.

Error rate referred to as *error probability* represents the basic measure that is defined as the percentage of inputs vectors for which the output value differs from the original one:

$$e_{prob}(f, \hat{f}) = \frac{1}{2^n} \sum_{\forall x \in \mathbb{B}^n} [f(x) \neq \hat{f}(x)] \quad (4)$$

In many cases, it is worth to consider also the Hamming distance between $f(x)$ and $\hat{f}(x)$. The *maximum Hamming distance* denoted also as *bit-flip error* [17] is defined as

$$e_{bf}(f, \hat{f}) = \max_{\forall x \in \mathbb{B}^n} \left(\sum_{i=0}^{m-1} f_i(x) \oplus \hat{f}_i(x) \right) \quad (5)$$

and gives the maximum number of output bits that simultaneously outputs a wrong value.

The average number of changed output bits denoted as *average Hamming distance* can be expressed as follows:

$$e_{hd}(f, \hat{f}) = \frac{1}{2^n} \sum_{\forall x \in B^n} \sum_{i=0}^{m-1} f_i(x) \oplus \hat{f}_i(x) \quad (6)$$

Note that $e_{prob}(f, \hat{f}) = e_{hd}(f, \hat{f})$ when applied to single-output functions, i.e. when $m = 1$.

3) Problem-specific error metrics: In some cases, neither the common metric (e.g. error rate) nor the arithmetic metric provide satisfactory assessment of the quality of approximate circuits. Hence, a various problem-specific error metrics have been proposed. For example, *distance error* was proposed to evaluate the quality of approximate median and sorting circuits [18], [19]. The common problem of the previously mentioned metrics is that they are data dependent. To model the error introduced by the approximations, the authors proposed to measure the distance between the rank of the returned element and rank given by the specification. Two additional metrics can be inferred from the distance error: *average distance error* defined as the sum of error distances averaged over all input combinations producing an invalid output value and *worst case distance error* defined as the maximal distance error calculated over all input combinations.

Chandrasekharan et al. [20] analyzed the behavior in sequential circuits that contain approximated combinational components. Although the worst-case can be computed for the approximated component in isolation, the accumulated worst-case in the sequential circuit may differ significantly [20]. The sequence of successive input patterns for the approximated component depends on the sequential logic and composition of the overall circuit. Hence, *accumulated worst-case error* and *accumulated error rate* have been introduced.

II. Relaxed equivalence checking

Many of the authors simplify the problem and evaluate the functionality of approximate circuits by applying a set of input vectors. They perform, for example, Monte Carlo simulation to measure the error of the output vectors with respect to an exact solution [9], [10], [21]. This approach is viable only for small circuits having a reasonable number of primary inputs and it could provide unsatisfactory results when approximating complex circuits and only a negligible error is acceptable. It is clear that when a subset of all possible input vectors is adopted, the error is only estimated and there is no guarantee on the error because the probability of revealing all cases violating the predefined error level is extremely low. If the exact error of the approximation has to be determined, formal relaxed equivalence checking is requested, stressing the fact that the considered systems will be checked to be equal up to some bound with respect to a suitably chosen error metric. This research area is rather unexplored as almost all formal approaches have been developed for exact equivalence checking [22].

Determining whether two Boolean functions are functionally equivalent represents a fundamental problem in formal verification. Although the functional equivalence checking is an NP-complete problem, several approaches have been proposed so far to reduce the computational requirement for practical circuit instances.

State-of-the-art verification tools are based on efficient operations on Boolean formulas. Traditional manipulation techniques are based on Binary Decision Diagrams (BDDs) and SAT solvers. The relation between SAT and BDDs has been studied, for example, in [23].

BDDs have been traditionally used to solve the equivalence checking problem due to their canonical property. The decision procedure is trivial and reduces to pointer comparison However, it is the requirement for canonicity that makes BDDs inefficient in representing certain classes of functions. It is well known fact that the size of BDD is sensitive to the chosen ordering of the variables and the variable ordering should not be chosen randomly [24]. Interestingly, there are functions whose BDD size is always polynomial in the number of input variables (e.g. symmetric functions). On the other hand, there are functions for which the BDD size is always exponential, independent of variable ordering. This holds e.g. for the multiplication function. It has been proven that not only multipliers but also integer division, remainder, square root and reciprocal exhibit exponential memory requirements for any variable ordering [25]. Even the middle bit of the n-bit multiplication cannot be efficiently represented, resulting in the fact that the BDD-based verification of 128-bit multipliers will never be possible [26]. In addition to that, hidden weighted bit function representing a reversible function consisting of a binary counter driving a multiplexer is known to have an exponential size BDD for any variable ordering [24]. Except of these pathological cases of circuits, BDDs are known to be an efficient tool for representation and manipulation with digital circuits.

In the field of digital system design, the use of SAT solvers has been investigated for more than twenty years and many powerful tools utilizing SAT solvers have been developed. Currently, the SAT solver based (or simply SAT-based) equivalence checking represents a method of the first choice. Modern SAT algorithms are extremely effective at coping with large problem instances and large search spaces [27]. The basic principle is to translate the problem of functional equivalence of two combinational circuits to the problem of deciding whether a Boolean formula given in conjunctive normal form (CNF) is satisfiable or not. This can be done using a miter which contains the combinational circuits whose corresponding outputs are connected via XOR gates whose outputs are feed into a single OR gate. To prove functional equivalence, it is necessary to prove that the output of the miter is always false.

Most formal verification approaches that build on testing exact equivalence are not directly extendable for relaxed equivalence checking, however, the ideas behind efficient testing of exact equivalence can serve as a basis for developing efficient methods for checking relaxed equivalence.

A. SAT-based approaches

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 an ILP solver was employed in [28], [20]. The principle of the method is as follows. First, an auxiliary circuit referred to as approximation miter is constructed. This circuit instantiates the candidate approximate circuit and the accurate (reference) circuit and compares their outputs to quantify the error for any given input. The comparison is typically ensured by means of an error computation block followed by a decision circuit. Then, the approximation miter is converted to a CNF formula and the resulting formula is used together with an objective function as input of the SAT solver. The objective function is constructed in such a way that it maximizes the difference at the output. Similar approach was proposed in [29]. The authors introduced so-called quality error circuit that ensures that the maximum allowed error level is not exceeded. However, the quality error circuit has to be constructed by the user similar to a test bench, which is a design problem itself [20].

While violating the worst error can be detected, no practically useful method capable of establishing the average-case error, error rate and total Hamming distance using a SAT solver has been proposed up to now. Chandrasekharan et al. [20] proposed a method for determining accumulate average-case error and accumulate error rate for sequential circuits using bounded model checking. In particular, they determined what is the earliest time that a given approximate circuit exceeds an accumulated worst-case and what is the earliest time that the circuit can reach an accumulated error rate. Unfortunately, the initial experiments with the accumulated average-case error did not conclude on practical designs. The common feature of these metrics is that it is necessary to determine the number of input assignments that evaluates output of a miter to true. This problem generalizes SAT problem and is known as model counting problem or simply #SAT. The model counting represents a challenging problem since it has been demonstrated that #SAT is extremely hard even for some polynomial-time solvable problems [30]. As a consequence of that, the available #SAT solvers are able to handle only small instances. In addition, the mentioned error metrics cannot be typically expressed in terms of Boolean functions efficiently since it requires counting in the solution space. Apart from the exact counting, there have been proposed algorithms for approximate model counting. Some authors proposed approaches that provide fast estimates without any guarantees but they do not offer any significant advantage compared to the Monte Carlo-based simulations. On the other hand, there are promising methods that provide lower or upper bounds with a correctness guarantee, often in a probabilistic or statistical sense [30].

An efficient implementation of lexicographic SAT solver was introduced recently [31]. The lexicographic satisfiability (LEXSAT) is a decision problem similar to the SAT problem. The only difference is that SAT solver typically returns any satisfiable assignment, while LEXSAT returns deterministically the one whose integer value under a given variable order is the minimum (or maximum) among all satisfiable assignments. According to the preliminary results, LEXSAT seems to be a promising tool for worst-case error analysis. The benefit from using LEXSAT compared to BDDs is that usually less than m SAT calls are required to determine the worst-case value for an m-bit output because the solver can learn some bits from the received SAT assignments. In addition to that, the proposed implementation relies on pushing an popping assumptions which significantly improves the performance compared to the worst-case analysis conducted using a common SAT solver based on adding clauses. The removal of clauses is usually hard or even impossible and it might require reinitializing the SAT solver.

B. BDD-based approaches

The binary decision diagrams seem to be the only viable option how to calculate the error metrics, at least for this moment. Binary Decision Diagrams, and especially Reduced Ordered BDDs (ROBDDs) are the most frequently used data structure for representation and manipulation of Boolean functions. On a more abstract level, ROBDDs can be considered as a compact representation of sets or relations. One of the main advantages of ROBDDs is the possibility to efficiently perform many of the operations needed for the manipulation of Boolean functions. Interestingly, the ROBDDs enable a way how to efficiently implement operations for examining the set of satisfying truth assignments which represents a key feature of model counting. In fact, the number of satisfying assignments can be determined in linear time with respect to the number of BDD nodes using SATcount operation.

The Hamming distance computed using BDDs was introduced in the context of approximate synthesis of general logic in [32] and later in [33]. The computation of the average-case Hamming distance is a relative straightforward (see Equation 7). The average-case Hamming distance can be obtained by converting the miter (without final OR gate) to corresponding ROBDD and calling SATcount operation for each output of the XOR gates. Finally, we sum the obtained results and divide them by the total number of input assignments.

$$\begin{aligned} e_{hd}(f,\hat{f}) &= \frac{1}{2^n}\sum_{\forall x\in B^n}\left(\sum_{i=0}^{m-1} f_i(x)\oplus\hat{f}_i(x)\right) \\ &= \frac{1}{2^n}\sum_{i=0}^{m-1}\left(\sum_{\forall x\in B^n} f_i(x)\oplus\hat{f}_i(x)\right) \\ &= \frac{1}{2^n}\sum_{i=0}^{m-1}\mathrm{SATcount}(f_i\oplus\hat{f}_i). \end{aligned} \quad (7)$$

The similar approach can be employed to determine error rate (see Equation 8). The error rate is defined as the percentage of input vectors for which the approximate output differs from the original one. It means that the output is classified as invalid if at least one bit is different. It means that it is sufficient to apply SATcount operation on the output of a common miter as the miter is constructed in such a way that it evaluates to true if and only if a certain input assignment yields an invalid response.

$$\begin{aligned} e_{prob}(f,\hat{f}) &= \frac{1}{2^n}\sum_{\forall x\in\mathbb{B}^n}[f(x)\neq\hat{f}(x)] \\ &= \frac{1}{2^n}\sum_{\forall x\in B^n}\left(\bigvee_{0\le i<m} f_i(x)\oplus\hat{f}_i(x)\right) \\ &= \frac{1}{2^n}\mathrm{SATcount}\left(\bigvee_{0\le i<m} f_i\oplus\hat{f}_i\right) \end{aligned} \quad (8)$$

Recently, a new BDD-based method for arithmetic worst-case and average-case error analysis based on characteristic function was introduced [33]. Independently on that, an alternative approach how to determine average-case arithmetic error was developed in [34]. The main advantage of this method is that it does not involve to construct the characteristic function which may be time consuming. The computation of the average-case arithmetic error using BDDs is derived in Equation 9. To determine the average-case error we can create an auxiliary miter consisting of the combinational circuits whose outputs are feed into subtracter followed by a circuit which computes absolute value. The average-case arithmetic error can be then obtained by several calls of SATcount operation, one per each bit of the circuit producing absolute value. The obtained numbers are weighted by appropriate powers of two and summed up.

$$\begin{aligned} e_{avg}(f,\hat{f}) &= \frac{1}{2^n}\sum_{\forall x\in\mathbb{B}^n} D_{f,\hat{f}}(x) = \frac{1}{2^n}\sum_{\forall x\in\mathbb{B}^n}\left(\sum_{i=0}^{m-1} d_i(x)\cdot 2^i\right) \\ &= \frac{1}{2^n}\sum_{i=0}^{m-1}\left(2^i\sum_{\forall x\in\mathbb{B}^n} d_i(x)\right) \\ &= \sum_{i=0}^{m-1} 2^{i-n}\cdot\mathrm{SATcount}(d_i), \end{aligned} \quad (9)$$

where $D_{f,\hat{f}}(x) = |\,\mathrm{nat}(f(x)) - \mathrm{nat}(\hat{f}(x))|$, $d = \mathrm{nat}^{-1}(D_{f,\hat{f}})$.

In addition to that, various problem-specific error metrics have been proposed. A BDD-based method for analysis of approximate sorting and median circuits was proposed in [18]. To model the error introduced by the approximations more precisely, the authors proposed to measure the distance between the rank of the returned element and rank given by the specification. This metric was denoted distance error [19]. The problem of worst-case error analysis and error distribution analysis was formulated as Pseudo-Boolean Constraint Satisfaction Problem (CSP) that was solved efficiently using BDDs. The BDDs allowed to perform not only the worst-case error analysis but they are able to compute a detailed true error distribution even for large median and sorting circuits.

III. Conclusions

We could observe a lot of work around approximations and formal techniques in the recent five years. Despite of that, many challenges remain. Although the ROBDDs offer an efficient way of representing Boolean functions and provide a tool for solving many practical problems in digital circuit design, it is fair to say that there are situations in which BDDs perform unsatisfactory. For example, multipliers are known for their exponential memory requirements for any variable ordering. As a consequence of that, only small multipliers can be analyzed using BDDs. Unfortunately, multiplier is one of the key arithmetic circuit that is widely used in many applications, especially in digital signal processing and multimedia processing. Similarly, the SAT-based equivalence checking of multipliers is also impractical due to the large runtime requirements. Despite the compact CNF representation, the number of paths traversed by the SAT solver grows exponentially with the increasing number of inputs. Hence, there is currently a clear need to come up with a new approach to the problem of evaluating the quality of approximate complex digital systems.

IV. Acknowledgments

This work was supported by the Czech Science Foundation project 16-17538S (Relaxed equivalence checking for approximate computing).

References

[1] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *The 50th Annual Design Automation Conference 2013, DAC'13*. ACM, 2013, pp. 1–9.

[2] P. Gupta, Y. Agarwal, L. Dolecek, N. Dutt, R. K. Gupta, R. Kumar, S. Mitra, A. Nicolau, T. S. Rosing, M. B. Srivastava, S. Swanson, and D. Sylvester, "Underdesigned and opportunistic computing in presence of hardware variability," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 32, no. 1, pp. 8–23, 2013.

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

[4] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate computing: A survey," *IEEE Design Test*, vol. 33, no. 1, pp. 8–22, 2016.

[5] S. G. Ramasubramanian, S. Venkataramani, A. Parandhaman, and A. Raghunathan, "Relax-and-retime: A methodology for energy-efficient recovery based design," in *50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2013, pp. 1–6.

[6] D. May and W. Stechele, "Voltage over-scaling in sequential circuits for approximate computing," in *2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era*, 2016, pp. 1–6.

[7] 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.

[8] S. Venkataramani, A. Sabne, V. J. Kozhikkottu, K. Roy, and A. Raghunathan, "Salsa: systematic logic synthesis of approximate circuits," in *The 49th Annual Design Automation Conference 2012, DAC '12*. ACM, 2012, pp. 796–801.

[9] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-simplify: a unified design paradigm for approximate and quality configurable circuits," in *Design, Automation and Test in Europe, DATE'13*. EDA Consortium San Jose, CA, USA, 2013, pp. 1–6.

[10] K. Nepal, Y. Li, R. I. Bahar, and S. Reda, "Abacus: A technique for automated behavioral synthesis of approximate computing circuits," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '14. EDA Consortium, 2014, pp. 1–6.

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

[12] V. Mrazek, S. S. Sarwar, L. Sekanina, Z. Vasicek, and K. Roy, "Design of power-efficient approximate multipliers for approximate artificial neural networks," in *Proceedings of the 35th International Conference on Computer-Aided Design*, ser. ICCAD '16. New York, NY, USA: ACM, 2016, pp. 81:1–81:7.

[13] R. Hrbacek, V. Mrazek, and Z. Vasicek, "Automatic design of approximate circuits by means of multi-objective evolutionary algorithms," in *2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era*, 2016, pp. 1–6.

[14] K. Nepal, S. Hashemi, H. Tann, R. I. Bahar, and S. Reda, "Automated high-level generation of low-power approximate computing circuits," *IEEE Transactions on Emerging Topics in Computing*, vol. online first, pp. 1–13, 2016.

[15] A. J. Sanchez-Clemente, L. Entrena, R. Hrbacek, and L. Sekanina, "Error mitigation using approximate logic circuits: A comparison of probabilistic and evolutionary approaches," *IEEE Transactions on Reliability*, vol. online first, pp. 1–13, 2016.

[16] W. T. J. Chan, A. B. Kahng, S. Kang, R. Kumar, and J. Sartori, "Statistical analysis and modeling for error composition in approximate computation circuits," in *31st IEEE International Conference on Computer Design (ICCD)*, 2013, pp. 47–53.

[17] T.-H. Chen, A. Alaghi, and J. P. Hayes, "Behavior of stochastic circuits under severe error conditions," *Information Technology*, vol. 56, pp. 182–191, 2014.

[18] V. Mrazek and Z. Vasicek, "Automatic design of arbitrary-size approximate sorting networks with error guarantee," in *2016 26th International Workshop on Power and Timing Modeling, Optimization and Simulation*. IEEE Computer Society, 2016, pp. 221–228.

[19] Z. Vasicek and V. Mrazek, "Trading between quality and non-functional properties of median filter in embedded systems," *Genetic Programming and Evolvable Machines*, vol. 18, no. 1, pp. 45–82, 2017.

[20] A. Chandrasekharan, M. Soeken, D. Groe, and R. Drechsler, "Precise error determination of approximated components in sequential circuits with model checking," in *53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2016, pp. 1–6.

[21] H. Jiang, C. Liu, N. Maheshwari, F. Lombardi, and J. Han, "A comparative evaluation of approximate multipliers," in *IEEE/ACM International Symposium on Nanoscale Architectures, NANOARCH 2016, Beijing, China, July 18-20, 2016*, 2016, pp. 191–196.

[22] L. Holik, O. Lengal, A. Rogalewicz, L. Sekanina, Z. Vasicek, and T. Vojnar, "Towards formal relaxed equivalence checking in approximate computing methodology," in *2nd Workshop on Approximate Computing (WAPCO 2016)*. HiPEAC, 2016, pp. 1–6.

[23] S. Reda, R. Drechsler, and A. Orailoglu, "On the relation between SAT and BDDs for equivalence checking," in *Proceedings International Symposium on Quality Electronic Design*, 2002, pp. 394–399.

[24] R. Drechsler and B. Becker, *Binary Decision Diagrams: Theory and Implementation*. Springer US, 2013.

[25] J. S. Thathachar, *On the limitations of ordered representations of functions*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 232–243.

[26] P. Woelfel, "Bounds on the obdd-size of integer multiplication via universal hashing," *Journal of Computer and System Sciences*, vol. 71, no. 4, pp. 520 – 534, 2005.

[27] J. Marques-Silva, "Practical applications of boolean satisfiability," in *Workshop on Discrete Event Systems (WODES'08)*. IEEE Press, 2008.

[28] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, "Macaco: Modeling and analysis of circuits for approximate computing," in *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2011, pp. 667–673.

[29] A. Ranjan, A. Raha, S. Venkataramani, K. Roy, and A. Raghunathan, "Aslan: Synthesis of approximate sequential circuits," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '14. EDA Consortium, 2014, pp. 1–6.

[30] C. P. Gomes, A. Sabharwal, and B. Selman, "Model counting," in *Handbook of Satisfiability*, A. Biere, M. Heule, H. Van-Maaren, and T. Walsh, Eds. IOS Press, 2009, ch. 20, pp. 266–290.

[31] A. Petkovska, A. Mishchenko, M. Soeken, G. De Micheli, R. Brayton, and P. Ienne, "Fast generation of lexicographic satisfiable assignments: Enabling canonicity in sat-based applications," in *Proceedings of the 35th International Conference on Computer-Aided Design*, ser. ICCAD '16. New York, NY, USA: ACM, 2016, pp. 4:1–4:8.

[32] Z. Vasicek and L. Sekanina, "Evolutionary design of complex approximate combinational circuits," *Genetic Programming and Evolvable Machines*, vol. 17, no. 2, pp. 169–192, 2016.

[33] M. Soeken, D. Grosse, A. Chandrasekharan, and R. Drechsler, "BDD Minimization for Approximate Computing," in *Proceedings of the 21st Asia and South Pacific Design Automation Conference (ASP-DAC 2016)*. IEEE, 2016, pp. 474–479.

[34] Z. Vasicek, V. Mrazek, and L. Sekanina, "Towards low power approximate DCT architecture for HEVC standard," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '17, 2017, p. to appear.