Introduction to Approximate Computing: Embedded Tutorial

Lukas Sekanina
Brno University of Technology, Faculty of Information Technology, IT4Innovations Centre of Excellence
Bozetechova 2, 61266 Brno, Czech Republic
Email: sekanina@fit.vutbr.cz

***Abstract*—A new design paradigm—approximate computing—was established to investigate how computer systems can be made better—more energy efficient, faster, and less complex—by relaxing the requirement that they are exactly correct. The purpose of this paper is to introduce the principles of approximate computing and survey the research conducted in major subareas of approximate computing which are relevant for design and test of digital circuits.**

I. Introduction

The notion of *approximation* is well established in many fields, for example, in computer science (e.g. approximation algorithms, approximate string matching), mathematics (e.g. function approximation) and engineering (e.g. approximate signal processing in complex video, audio, or communication systems). However, the reasons for introducing the approximations can have completely different motivation.

In recent years, a new design paradigm—*approximate computing*—has been established to investigate how computer systems can be made better—more energy efficient, faster, and less complex—by relaxing the requirement that they are exactly correct. Mittal [1] is highlighting the fact that "*approximate computing exploits the gap between the level of accuracy required by the applications/users and that provided by the computing system, for achieving diverse optimizations*."

This new research effort in approximate computing is primarily caused by a progressive development of two domains:

(1) Low power consumption requirements on electronics. Energy efficiency is definitively one of the major driving forces of current computer industry which is relevant for supercomputers on the one hand as well as small portable personal electronics and sensors on the other hand. In this context, approximate computing exploits the fact that some applications are inherently *error resilient*. Errors are not recognizable because human perception capabilities are limited (e.g., in multimedia applications), no golden solution is available for validation of the results (e.g., in data mining applications), or users are willing to accept some inaccuracies (e.g., when the battery of a mobile phone is almost depleted, but at least some basic functionality is still requested). Therefore, the error (the accuracy of computations) can be used as a design metric and traded for performance or power consumption. Accepting approximate computing as a standard design paradigm requires a deeper understanding of inherent application resilience across a broader range of applications. One of the studies conducted in this direction reported that about 83 % of runtime of the applications (that are potentially suitable for approximations) is spent in computations that can be approximated [2]. This result was obtained using a benchmark suite consisting of 12 widely used recognition, data mining and search applications, along with representative input data.

(2) Specific properties of integrated circuits in the nanoscale era. Integrated circuits developed using recent fabrication technology (45 nm and below) exhibit reliability issues and uncertainties especially when operated at low voltages. Deviations of the transistors' parameters from the average values are no longer small due fabrication variations caused at the nanoscale. Adopting conventional fault tolerance mechanisms (such as introducing redundant components) to increase the circuit reliability requires additional resources and power. By allowing approximations in computing and graceful performance degradation at run-time in the presence of uncertainties or errors, each component could deliver the best possible tradeoff between the quality of result and energy consumption. Hence approximate computing represents a potential solution to implement energy efficient systems on inherently unreliable platforms [3], [4].

The field of approximate computing is at an early stage of development, but a growing number of papers dealing with this topic indicates a very active research community (Fig. 1). The relevant research communities involved in approximate computing cover the whole computer stack, integrating thus the areas of microelectronics, circuits, components, architecture, networks, operating systems, compilers and applications [1], [5]. Approximations are conducted for embedded systems, ordinary computers, graphics processing units (GPUs) and field-programmable gate arrays (FPGAs). Substantial energy consumption reductions are expected for data centers and supercomputers [6], [7].

The purpose of this paper is to introduce the principles of approximate computing and survey the research conducted in major subareas of approximate computing which are relevant for design and test of digital circuits. Approximate computing in software and compiler design will be addressed only marginally. The plan for this paper is as follows:

- The principles of approximate computing (Section II)
- Ad hoc circuit approximations (Section III)
- Design automation methods in approximate computing (Section IV)

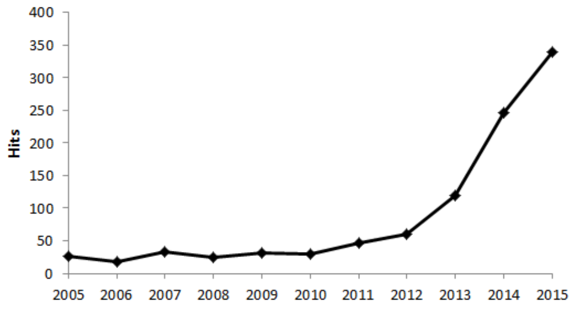

Fig. 1. The number of results for "approximate computing" by Google Scholar (January 2016).

- Software approximation and platforms supporting approximate computing (Section V)
- Evaluation of approximation techniques (Section VI)

II. The principles of approximate computing

In approximate computing, the requirement on functional equivalence of the specification and implementation can be relaxed in order to not only reduce power consumption but also to accelerate computations, minimize the on-chip area or optimize other system parameters. Applications suitable for approximate computing can be broadly classified to four classes [8]:

1) Applications with analog inputs which operate on noisy real-world data.
2) Applications with analog output intended for human perception.
3) Applications with no unique answer such as web search and machine learning.
4) Iterative and convergent applications that iteratively process large amounts of data and the equality of results depends on the number of iterations.

Approximate computing has been developed in different ways and at various levels of the computing stack as discussed in [1], [5]. In this section, we will briefly survey approaches developed to identify approximable parts of applications, error metrics used to evaluate resulting approximate solutions and approximation techniques introduced to perform actual approximations.

A. *Identifying approximable parts of applications*

In order to obtain useful approximate solutions, *sensitivity analysis* has to be performed to identify subsystems suitable for undergoing the approximation. The analysis should reveal how the subsystems and their interactions influence the quality of result, power consumption and other parameters.

In some cases, the selection of components/functions to be approximated is straightforward. In most cases, however, a detailed analysis and profiling have to be performed. In programs, the process includes statistical analysis of the impact of various changes to the accurate code (such as the precision of number representation, data storage strategies, code simplification, relaxed synchronization, unfinished loops and skipped function calls among others) on the quality of result [9]. In [2], the instructions in the program were partitioned into computation kernels which were subsequently marked either as sensitive (whose approximation would lead to crashing the application) or resilient (which are suitable for approximation). The approximation method then operated at the level of the resilient kernels.

In hardware, typical modifications of the accurate circuit involve the bit width reduction, intentional disconnecting of subsystems, changes in timing and power supply voltage and fault injection [2]. For example, a methodology, called MACACO (Modeling and Analysis of Circuits for Approximate Computing) was proposed to systematically analyze how an approximate circuit behaves with reference to a conventional correct implementation [10]. Other system properties of software as well as hardware (such as testability, dependability, security, etc.) have to be monitored by the designer as they can be influenced by the approximations introduced.

B. *Error metrics*

The error introduced by an approximation is typically estimated by measuring the output of the approximate system as a response for a set of data inputs. The resulting values are compared with the outputs produced by an exact solution. It has to be noted that establishing a suitable error measure is highly application dependent.

1) Arithmetic errors: Various error metrics have been developed for this purpose. Let $O_{orig}^{(i)}$ and $O_{approx}^{(i)}$ be the output values of the accurate solution and approximate solution. The common arithmetic errors are defined as follows.

Error magnitude (e_{wst}), sometimes denoted the *worst case error*, is defined as

$$e_{wst} = \max_{\forall i} |O_{orig}^{(i)} - O_{approx}^{(i)}|. \tag{1}$$

If used as the optimization goal (i.e. the following condition $e_{wst} \leq \epsilon$ is satisfied during the whole design process) this metrics guarantees that the approximate output differs from the correct output by at most ϵ.

Maximal relative error is defined as

$$e_{rel} = \max_{\forall i} \frac{|O_{orig}^{(i)} - O_{approx}^{(i)}|}{O_{orig}^{(i)}}. \tag{2}$$

Average error magnitude (e_{avg}) is defined as the sum of absolute differences in magnitude between the original and approximate circuits (i.e. total error e_{tot}), averaged over n test vectors:

$$e_{avg} = \frac{e_{tot}}{n} = \frac{\sum_{\forall i} |O_{orig}^{(i)} - O_{approx}^{(i)}|}{n}. \tag{3}$$

Error probability (*error rate*) defined as the percentage of inputs vectors for which the approximate output differs from the original one represents the last commonly used metrics:

$$e_{prob} = \frac{\sum_{\forall i, O_{orig}^{(i)} \neq O_{approx}^{(i)}} 1}{n}. \tag{4}$$

Let us assume that the circuit under approximation is combinational. If all possible input vectors are applied (i.e. $n = 2^w$, where w is the number of primary inputs) then the resulting error represents a true error of the circuit. If $n < 2^w$ then the error is computed only for a subset of all possible input vectors and nothing is known about the error for test vectors unseen during the evaluation. The latter case is typical for the evaluation of complex circuits.

Specific metrics are used in particular application domains, for example, the peak signal to noise ratio (PSNR) in image processing or the specificity and sensitivity in classification tasks. In Monte-Carlo analysis proposed in [10], circuit simulations were performed to obtain an error distribution for a given operating voltage and frequency.

2) Relaxed equivalence checking: However, in some cases (such as complex arithmetic circuits), it is not sufficient to establish the error on the basis of a data set (i.e., via a circuit simulation using $n < 2^w$ test vectors). 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 w.r.t. a suitably chosen distance metric. This research area is rather unexplored as almost all formal approaches have been developed for (exact) equivalence checking [11]. Checking the worst error can be based on satisfiability (SAT) solving as demonstrated in [10]. However, while violating the worst error can be detected, no efficient method capable of establishing, for example, the average error using a SAT solver has been proposed up to now. In the context of approximate computing, approaches based on binary decision diagrams (BDDs) to determine the average arithmetic error, worst error, error rate [12] and average Hamming distance [13] have been proposed.

The following example demonstrates formal checking of a predefined worst-case error. The principle of the method which combines a SAT solver with an integer linear programming (ILP) solver is shown in Fig. 2 [10]. Firstly, an auxiliary circuit is constructed. This circuit instantiates the candidate approximate circuit and the accurate (reference) circuit. Their outputs are comaperd in miter to quantify the error for any given input. This miter is an integer subtractor followed by a comparison operation. In order to measure the maximum error, the miter subtracts the accurate output (y) from the approximate circuit output (y'): $|y' - y| \leq E$. The auxiliary circuit is converted to a formula in conjunctive normal form and the resulting formula is used together with an objective function as input of the SAT solver. ILP solver is used to determine the maximum error E.

C. Approximation techniques

The sensitivity analysis discussed in Section II-A should reveal the most efficient approach to the actual approximation. Mittal [1] discusses the following approximation strategies: precision scaling, loop perforation, load value approximation, memorization, task dropping/skipping, memory access skipping, data sampling, using different program versions, using inexact or faulty hardware, voltage scaling, refresh rate reducing, inexact read/write, reducing divergence in GPUs, lossy compression and use of neural networks. As this paper primarily deals with digital circuit approximation, we will briefly introduce the principles of the over-scaling and functional approximation which are utilized by many methods available in the literature.

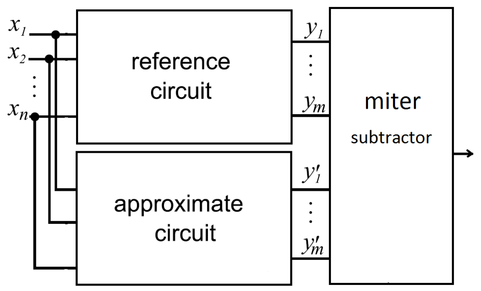

Fig. 2. SAT-based worst-case error analysis

In the case of *over-scaling*, circuits are designed to be working perfectly under a normal environment. However, their energy consumption can be reduced by voltage over-scaling (i.e. using lower power supply voltage in which the circuit is known to occasionally produce erroneous outputs). To deal with errors caused by a large number of near critical paths, methods based on cell sizing [14], logic restructuring [15], and retiming [16] have been proposed to improve the path delay distribution of overscaled circuits. On the other hand, performance can be increased when the circuit is over-clocked. Timing induced errors are due to the fact that some paths in the circuit fail to meet the delay constraints. The combination of scaling the supply voltage and clock frequency is known as dynamic voltage scaling.

The idea of *functional approximation* is to implement a slightly different function to the original (accurate) one provided that the error is acceptable and power consumption or other system parameters are optimized adequately. Various approaches to the functional approximation will be discussed in next sections. The functional approximation is often combined with voltage over-scaling [17].

III. AD HOC CIRCUIT APPROXIMATIONS

By "ad hoc approximations" we mean various approaches introduced to approximate a particular component, assuming that the method is not a general purpose approximation method. A lot of knowledge about a particular system, its typical utilization and quality measurement methodology can be incorporated into the approximation method in order to obtain the best tradeoff for key circuit parameters. For instance, many ad hoc approaches have been developed to approximate adders, see a survey in [5].

Examples of the ad hoc approximation methods given below show that approximations can be introduced at different system levels:

- Transistor level: adders [18], median circuit [19]

- Gate-level: multipliers [20], [21], fault tolerant logic [22]
- RT-level: image filters [20]
- Microarchitecture: pipeline circuits [23]
- Processor: approximate vector processor [24]
- Memory: Multi-Level Cell [25], SRAM [3], memory hierarchy [26], [6]

IV. Design automation methods in approximate circuit design

In this case, the approximations are performed using the same procedure for all problem instances of a given class. Approximate implementations showing different compromises between considered system parameters are generated and presented to the user, whose responsibility is to choose the most suitable approximate solution for a given application.

A. Systematic methods

A functional approximation is typically obtained by a heuristic procedure that iteratively modifies the original, accurate (hardware or software) implementation. Several design automation methods have been proposed to approximate digital circuits. Examples include: SALSA [27], ASLAN [28], SASIMI [17], ABACUS [29], ABM [12] and genetic programming-based methods [30], [13]. Approximate high level synthesis (in particular, allocation and scheduling) enabled to compose complex circuits systems using approximate components [31]. Table I summarizes benchmark problems and error computing approaches used for evaluation of systematic approximation methods. Some of the methods are briefly elaborated below.

The Systematic methodology for Automatic Logic Synthesis of Approximate circuits (SALSA) starts with a description of the accurate circuit and an error constraint that specifies the type of error that can be accepted [27]. SALSA introduces the quality function which takes the outputs from both the original circuit and approximate circuit and decides (by means of SAT solving) if the quality constraints are satisfied. The quality function outputs a single Boolean value. The SALSA algorithm attempts to modify the approximate circuit with the goal of keeping the output of the quality unchanged. The concept of approximation by means of the quality function has been extended to sequential circuits in [28]. The main issues here are how to model and handle errors which can propagate through the combinational logic over multiple cycles of operation.

Another method, SASIMI, tries to identify signal pairs in the circuit that exhibit the same value with a high probability, and substitutes one for the other [17]. These substitutions introduce functional approximations. Unused logic can be eliminated from the circuit which results in area and power savings.

ABACUS creates an abstract synthesis tree (AST) from the input behavioral description and then applies various operators to the AST using an iterative stochastic greedy algorithm [29]. Candidate designs are evaluated in terms of accuracy, power consumption and area in a single objective optimization scenario based on a training data set.

Evolutionary approximation methods, currently based on Cartesian genetic programing (CGP), transform the circuit approximation problem to a multi-objective search problem. Candidate approximations are generated from an exact solution using genetic operators such as mutation and reproduction and evaluated by means of circuit simulation [30] or relaxed equivalence checking based on BDDs [13]. While CGP is capable of delivering high quality approximate solutions showing an excellent trade off among key design objectives, its main weakness is in the scalability and long non-deterministic runs.

B. Quality configurable circuits

Quality configurable circuits allow for a dynamic approximation (and thus dynamic reconfiguration) at the circuit level as response to variable requirements on the quality of result (e.g. in image compression). This concept can be demonstrated using an adder whose input operands are split into several regions, and each region is independently summed up. The partial sums from the regions are conditionally composed together based on the desired quality of addition. Adaptive quality control can be achieved by various design methods, for example, SASIMI metod [17] or subcircuits isolation and their approximation in accordance with the requested quality of results in method [32].

V. Software approximation and platforms supporting approximate computing.

There are many approaches targeting software approximation as documented in the recent survey [1]. One research direction is based on extending common programming languages to support approximate computing. This work includes the disciplined approximate programming paradigm that lets programmers declare which parts of an algorithm can be computed approximately. For example, EnerJ [33] is an extension to Java that adds approximate data types and approximate operations. It assumes that data storages of various reliability and approximate arithmetic units are available in the hardware platform. Approximate data can be processed more cheaply, but less reliably. The system can statically guarantee isolation of the precise program component from the approximate component.

In another approach—Axilog, a set of language annotations was developed that provide the necessary syntax and semantics for approximate hardware design and reuse in Verilog [34]. Axilog enables the designer to relax the accuracy requirements in certain parts of the design, while keeping the critical parts strictly precise

In the Chisel project, reliability- and accuracy-aware optimizations of computational kernels are performed by means of integer linear programming and intended for approximate hardware platforms [35].

However, only in a few cases the software approximations were connected with a hardware platform supporting approximate computing. An integrated HW/SW approach to approximate computing has been developed in [36]. It consists in automatic resilience characterization of the target application

TABLE I
BENCHMARK PROBLEMS AND ERROR COMPUTING APPROACHES USED FOR EVALUATION OF SYSTEMATIC APPROXIMATION METHODS.

Method	Ref.	Benchmarks	Error computing based on:
ABACUS	[29]	FIR,[b] perceptron, block matcher	training data
ABM	[12]	6 ISCAS-85 benchmark circuits	BDD
ASLAN	[28]	FIR,[b] IIR,[c] MAC,[f] DCT,[d] Sobel and 8-input neuron for MPEG encoder and clustering.	sequential QCC[a](SAT)
CGP	[30]	Multipliers, 9-input and 25-input median	training data
CGP-BDD	[13]	16 combinational circuits from LGSynth, ITC and ISCAS	BDD
Logic Isolation	[32]	Adders, multipliers, datapath modules, DCT,[d] FFT,[e] FIR[b]	probability of output
SALSA	[27]	Adders, multipliers, datapath modules, FIR,[b] IIR,[c] DCT[d]	QCC[a](SAT)
SASIMI	[17]	ISCAS85 benchmarks, multipliers, adders, datapath modules	training data

[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

in order to identify those parts of the application that are suitable for approximation. The application is then implemented using a specialized hardware (processor) whose components can be tuned in accordance with the desired output quality to adapt their energy consumption. In addition to the off-line tuning of the application, an automated on-line regulation of the degree of approximation is supported by the hardware. This energy-efficient recognition and mining processor was fabricated demonstrating that approximate computing can lead to 2-20X energy savings with minimal impact on output quality across a range of applications.

Finally, in order to accelerate program execution and reduce power consumption, trained artificial neural networks were proposed to replace an original complex general-purpose code written in an imperative language [37]. Neural networks implemented on a chip were also recognized as a good target for the approximation methods [38].

VI. Evaluation of Approximation techniques

Approximate solutions are always evaluated in comparison with their accurate counterparts. However, also the techniques developed for actual approximations have to be evaluated.

In the case of software approximations, AxBench—a set of representative applications from various domains—was introduced to explore different aspects of approximate computing and compare approximation strategies [37]. Unfortunately, for the evaluation of circuit approximation techniques, suitable benchmark circuits (with known errors and other parameters) are not available.

In order to compare circuit approximation techniques, it would be requested for each technique and each benchmark circuit to provide a Pareto front containing the best tradeoffs achieved for key circuit parameters such as the error, area, delay and power consumption (in a given fabrication technology), assuming that a fixed time budget is available for the approximation procedure. Figure 3 shows a comparison of four approximation methods by means of Pareto fronts obtained using multi-objective CGP (MO) and two-stage single-objective CGP (SO, three different sets of weights for the average error, area and delay considered) in the task of 8-bit multiplier approximation [39]. The axes are normalized with respect to an original accurate multiplier.

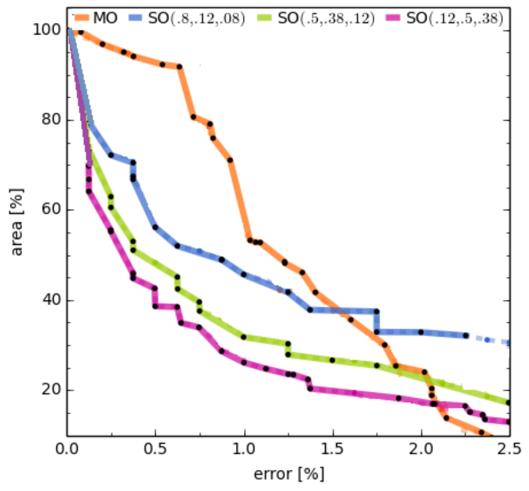

Fig. 3. Pareto fronts obtained for the approximate 8-bit multiplier using four approximation methods when the same time is available for each approximation method.

VII. Conclusions

In this paper, we briefly surveyed the field of approximate computing with a special focus on approximate circuits. We can summarize that approximate computing is currently a rapidly growing multidisciplinary field with a potentially huge impact not only on computer engineering but on the whole society because it tries to address one of the major global issues—energy efficiency. However, approximate computing is not a panacea. Approximations have to be carefully introduced and their results have to be carefully analyzed. In order to be accepted as a standard design paradigm, a lot of research is needed, especially in the areas of quality (error) analysis and computation, automated approximation methodologies and scalable solutions for complex systems.

Acknowledgments

This work was supported by the Czech science foundation project 14-04197S *Advanced Methods for Evolutionary Design of Complex Digital Circuits*.

References

[1] S. Mittal, "A survey of techniques for approximate computing," *ACM Computing Surveys*, pp. 1–34, 2016.

[2] 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.

[3] S. Ganapathy, G. Karakonstantis, A. Teman, and A. Burg, "Mitigating the impact of faults in unreliable memories for error-resilient applications," in *Proc. of the 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. ACM, 2015, pp. 1–6.

[4] 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.

[5] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *Proc. of the 18th IEEE European Test Symposium*. IEEE, 2013, pp. 1–6.

[6] P. Düben, J. Schlachter, Parishkrati, S. Yenugula, J. Augustine, C. Enz, K. Palem, and T. N. Palmer, "Opportunities for energy efficient computing: A study of inexact general purpose processors for high-performance and big-data applications," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, ser. DATE '15. EDA Consortium, 2015, pp. 764–769.

[7] J. Rudi, A. C. I. Malossi, T. Isaac, G. Stadler, M. Gurnis, P. W. J. Staar, Y. Ineichen, C. Bekas, A. Curioni, and O. Ghattas, "An extreme-scale implicit solver for complex pdes: Highly heterogeneous flow in earth's mantle," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15. ACM, 2015, pp. 5:1–5:12.

[8] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," *Commun. ACM*, vol. 58, no. 1, pp. 105–115, 2015.

[9] P. Roy, R. Ray, C. Wang, and W. F. Wong, "Asac: Automatic sensitivity analysis for approximate computing," in *Proceedings of the 2014 SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems*, ser. LCTES '14. ACM, 2014, pp. 95–104.

[10] 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.

[11] 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.

[12] M. Soeken, D. Grosse, A. Chandrasekharan, and R. Drechsler, "BDD minimization for approximate computing," in *21st Asia and South Pacific Design Automation Conference ASP-DAC 2016*. in press, 2016.

[13] Z. Vasicek and L. Sekanina, "Evolutionary design of complex approximate combinational circuits," *Genetic Programming and Evolvable Machines*, vol. 17, no. 2, pp. 1–24, 2016.

[14] A. B. Kahng, S. Kang, R. Kumar, and J. Sartori, "Slack redistribution for graceful degradation under voltage overscaling," in *15th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2010, pp. 825–831.

[15] L. Wan and D. Chen, "Ccp: Common case promotion for improved timing error resilience with energy efficiency," in *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design*, ser. ISLPED '12. ACM, 2012, pp. 135–140.

[16] 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)*. IEEE, 2013, pp. 1–6.

[17] 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. 1367–1372.

[18] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE Trans. on CAD of Integr. Circuits and Systems*, vol. 32, no. 1, pp. 124–137, 2013.

[19] M. Monajati, S. M. Fakhraie, and E. Kabir, "Approximate arithmetic for low-power image median filtering," *Circuits, Systems, and Signal Processing*, vol. 34, no. 10, pp. 3191–3219, 2015.

[20] 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.

[21] C.-H. Lin and I.-C. Lin, "High accuracy approximate multiplier with error correction," in *Computer Design (ICCD), 2013 IEEE 31st International Conference on*. IEEE, 2013, pp. 33–38.

[22] M. Choudhury and K. Mohanram, "Low cost concurrent error masking using approximate logic circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 8, pp. 1163–1176, 2013.

[23] S.-L. Lu, "Speeding up processing with approximation circuits," *IEEE Computer*, vol. 37, no. 3, pp. 67–73, 2004.

[24] S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Quality programmable vector processors for approximate computing," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2013, pp. 1–12.

[25] A. Sampson, J. Nelson, K. Strauss, and L. Ceze, "Approximate storage in solid-state memories," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2013, pp. 25–36.

[26] M. Shoushtari, A. BanaiyanMofrad, and N. Dutt, "Exploiting partially-forgetful memories for approximate computing," *IEEE Embedded Systems Letters*, vol. 7, no. 1, pp. 19–22, 2015.

[27] 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.

[28] 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.

[29] 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.

[30] Z. Vasicek and L. Sekanina, "Evolutionary approach to approximate digital circuits design," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 3, pp. 432–444, 2015.

[31] C. Li, W. Luo, S. S. Sapatnekar, and J. Hu, "Joint precision optimization and high level synthesis for approximate computing," in *Proc. of the 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. ACM, 2015.

[32] S. V. Shubham Jaina and A. Raghunathanc, "Approximation through logic isolation for the design of quality configurable circuits," in *Proc. of the 2016 Design, Automation & Test in Europe Conference and Exhibition (DATE)*. EDA Consortium, 2016, pp. 1–6.

[33] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "EnerJ: Approximate data types for safe and general low-power computation," in *Proc. of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 2011, pp. 164–174.

[34] A. Yazdanbakhsh, D. Mahajan, B. Thwaites, J. Park, A. Nagendrakumar, S. Sethuraman, K. Ramkrishnan, N. Ravindran, R. Jariwala, A. Rahimi, H. Esmailzadeh, and K. Bazargan, "Axilog: Language support for approximate hardware design," in *Design, Automation and Test in Europe, DATE'15*. EDA Consortium, 2015, pp. 1–6.

[35] S. Misailovic, M. Carbin, S. Achour, Z. Qi, and M. C. Rinard, "Chisel: Reliability- and accuracy-aware optimization of approximate computational kernels," in *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*. ACM, 2014, pp. 309–328.

[36] V. Chippa, S. Venkataramani, S. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing: An integrated hardware approach," in *2013 Asilomar Conference on Signals, Systems and Computers*. IEEE, 2013, pp. 111–117.

[37] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," in *Proc. of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2012, pp. 449–460.

[38] S. S. Sarwar, S. Venkataramani, A. Raghunathan, and K. Roy, "Multiplier-less artificial neurons exploiting error resiliency for energy-efficient neural computing," in *Proc. of the 2016 Design, Automation & Test in Europe Conference and Exhibition (DATE)*. EDA Consortium, 2016, pp. 1–6.

[39] Z. Vasicek and L. Sekanina, "Circuit approximation using single- and multi-objective cartesian GP," in *Genetic Programming*, ser. LNCS 9025. Springer, 2015, pp. 217–229.