On Evolutionary Approximation of Logic Circuits

Lukas Sekanina(✉) and Zdenek Vasicek

IT4Innovations Centre of Excellence, Faculty of Information Technology,
Brno University of Technology, Bozetechova 2, 61266 Brno, Czech Republic
{sekanina,vasicek}@fit.vutbr.cz

Abstract. The concept of approximation has intensively been studied, developed and applied not only in computer science, but also in mathematics and engineering disciplines. The never ending requirement for low power consumption led to making approximate circuits and computer systems even in the areas in which only accurately working solutions have traditionally been accepted. Approximate circuits are the circuits relaxing the requirement on the functional equivalence between the specification and implementation in order to reduce the area on a chip, delay or energy consumption. Approximate computing machines further exploit and apply this idea at all system levels. This paper introduces the field of approximate computing and shows how evolutionary design methods can automate the design process of approximate computing systems, in particular, approximate logic circuits.

1 Introduction

The notion of *approximation* is well established in computer science, mathematics and engineering [1]. However, the reasons for approximations can be different.

In computer science, *approximation algorithms* are algorithms used to find approximate solutions to NP-hard optimization problems. As it is intractable to find an optimal solution, the goal is to find polynomial-time exact algorithms and guarantee provable solution quality in provable run-time bounds. An interesting discovery is that, in spite of the isomorphism between NP-complete problems, good approximation algorithms can be surprisingly different for particular problem classes. A detailed overview of the theory of approximation algorithms can be found in [2].

One of the classic utilizations of the concept of approximation is approximate string matching. Finding strings that match a pattern approximately rather than exactly is crucial for spell checking, bioinformatics, spam filtering and other applications.

In bio-inspired artificial intelligence, models of computation, such as artificial neural networks, have been developed which inherently exploit the concept of approximation. For example, a feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions [3].

In mathematics, it is investigated how certain (usually complex) functions can be approximated by means of basic functions that are inexpensive or suitable

© Springer International Publishing Switzerland 2014
C.S. Calude et al. (Eds.): Gruska Festschrift, LNCS 8808, pp. 367–378, 2014.
DOI: 10.1007/978-3-319-13350-8_27

according to a given purpose. In order to do so, traditional approaches (such as Taylor series or Newton's method) utilize only elementary operations: addition, subtraction and multiplication. But multiplication can still be very expensive. As a hardware multiplier is a relatively complex and slow component, multiplierless methods have been discovered in computer engineering to inexpensively and quickly approximate mathematical functions. The most prominent example is the CORDIC algorithm (COordinate Rotation DIgital Computer), developed by Jack E. Volder, which is capable of calculating hyperbolic and trigonometric functions using addition, subtraction, bit shift and table lookup [4]. We will see later that even the adders are approximated in modern computing devices.

All approaches to approximations suppose that an *error measure* is defined in order to quantify how far a given approximation is from an optimal (or known) solution. The aforementioned example dealing with hardware resources shows that the error measure is not the only measure in engineering applications. For example, speed of processing and area on a chip are fundamental measures used for hardware components. Moreover, if a sequence of successive approximations can be constructed, the rate of convergence and stability are other important measures of the approximation method.

It can be seen that the concept of approximation is thus relevant for both algorithms as well as solutions produced by algorithms. One has to carefully distinguish *algorithms* (i.e. problem solving mechanisms, such as the quicksort) from *solutions* to particular problems (e.g. a sorted sequence of integers), because an algorithm can often be a solution produced by another algorithm (e.g. in genetic programming).

In the recent five years, we could observe a lot of work around approximations in a different context, mainly in a connection with *energy consumption*. A new research direction – *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. Approximate computing exploits the fact that the requirement of perfect functional behavior (i.e. accuracy) can be relaxed because some applications are inherently *error resilient* [5]. The errors are not recognizable because human perception capabilities are limited (e.g. in multimedia applications), no golden solution is available for validation of results (e.g. in data mining applications), or users are willing to accept some inaccuracies (e.g. when battery of a mobile phone is almost depleted, but at least a basic functionality is still requested). Therefore, the accuracy can be used as a design metric, traded for area on a chip, delay, throughput, or power consumption.

In approximate computing systems, approximations can be introduced at all design levels, starting from the circuit via the architecture and operating system to programming language. Taking approximate computing closer to mainstream adoption requires a deeper understanding of inherent application resilience across a broader range of applications, which has partially been investigated, e.g. in [6]. As a manual re-design of fully functional (exact) systems is not an efficient design method, several *automated approaches* have been proposed to particular problem classes.

The goal of this paper is to introduce the nascent field of approximate computing and show how *evolutionary design* methods can automate the design process of approximate computing systems, in particular, approximate digital circuits. Note that no support for the design of approximate circuits is available in common circuit design and optimization tools [7,8]. Because of the nature of approximate circuits (in fact, partially working circuits are sought) and principles of evolutionary circuit design (evolutionary-based improving of partially working circuits), evolutionary computing seems to be a promising design method.

The rest of the paper is organized as follows. Section 2 briefly surveys the field of approximate computing and approximate circuit design. In Section 3, evolutionary computing is introduced as a method for approximate circuit design. Section 4 specifically deals with a multi-objective approach to approximate circuit design. Concluding remarks are given in Section 5.

2 Approximate Computing

In the introduction, we have shown that approximate computing is a much wider concept than approximation algorithms and numerical approximation. It deals with new approaches to circuits, components, microarchitectures, operating systems, programming languages, compilers and their interactions.

Approximate computing should not also be confused with stochastic computing and probabilistic computing [5,9]. In stochastic computing, values are represented by streams of random bits. On the other hand, probabilistic computing utilizes random behavior of circuit elements under presence of thermal noise.

The number of papers dealing with approximate computing is rapidly increasing and the field is now very active. Approximate solutions have been applied at various levels of computer systems, including:

– elementary circuits (e.g. adders [10], multipliers [11]);
– high-level processing blocks (e.g. image compression [11], discrete cosine transform, finite and infinite impulse response filters [12]);
– computer architecture (approximate pipelines in microprocessors [13]);
– general purpose approximate computing machines [14];
– programming languages [15].

Considering the energy efficiency as the main driving factor for introducing inaccurate solutions, approximate computing is thus primarily relevant to physical design of circuits and development of software. In the case of software, a key challenge is how to isolate parts of the program that must be precise from those that can be approximated so that a program execution is correct even if quality of the output degrades [15].

Approximate computing is definitely a promising way in computer engineering as small imperfections in functionality can be tolerated in many domains and obtained benefits (especially in terms of power consumption) are impressive. At the same time, future fabrication technologies operating with atomic-scale elements will inherently lead to imperfectly working circuits and thus majority of circuits will have to be considered as approximate circuits. However, there is not still a well-established methodology for automated construction of approximate systems and circuits which could provide a good trade-off among key parameters. A recent comprehensive survey [16] clearly states in its "Implications for Circuits and Architectures" section that

> Much research needs to be done to functionally or parametrically underdesign large general class of circuits automatically. Mechanisms to pass application intent to physical implementation flow (especially to logic synthesis in case of functional underdesign) need to be developed.

2.1 Approximate Circuits

Before a *digital circuit* is implemented using gates and transistors, it is initially represented at the logic level. There is a huge number of possibilities to map the logic behavior onto available gates. In past decades, various optimization techniques were proposed to find the most suitable mapping according to a preselected *metric*, typically reflecting the area on a chip, power consumption and delay [7,8]. The circuits which are intentionally designed in such a way that the specification is not met in terms of functionality and some savings are expected in terms of energy, performance or area are called *approximate circuits*.

Power consumption reduction methods have been developed for decades [17]. However, new technology-level optimizations such as downsizing of gates (i.e. creating smaller than normally sized gates to reduce power consumption, in exchange for increased delay) on critical paths and voltage over-scaling (i.e. using deliberately lower power supply voltage for which the circuit is known to occasionally produce erroneous outputs) enabled additional power savings.

Another technique which is, to some extent, technology-independent is *functional approximation*. An accurate computing circuit is modified so that it does not fully implement the logic behavior given by the specification. A natural way is eliminating the least significant bits in the case of arithmetic circuits. However, this technique leads to insignificant area (and so power consumption) savings for some key circuits such as multipliers. Hence more drastic changes have to be introduced into the circuit structure.

The problem of *circuit approximation* can be formulated as a multiobjective optimization problem: Let C be a combinational circuit (a feed-forward network composed of elementary logic gates) implementing a multiple-output logic function $\{0,1\}^m \rightarrow \{0,1\}^n$ which satisfies the specification S (given by, e.g., truth table, algebraic expressions, netlist etc.), where m and n is the number of inputs and outputs. The goal is to generate a circuit $C^{'}$, implementing S with errors never exceeding predefined threshold error values ϵ_i according to a set of chosen error (constraint) functions E_i and minimizing a set of objective functions F_j.

The role of fabrication technology has to be emphasized in this task. The relation between the area and power consumption can be highly non-linear. A simple assumption that a small circuit will have low power consumption does not always apply. As circuit power consumption in static and dynamic mode depends on a particular technology, detailed simulations of power consumption have to always be performed for the chosen technology in order to get trustworthy results from the circuit approximation process.

2.2 Systematic Design Methods

The design of approximate circuits is typically based on manual modifications of fully functional circuits [11]. Only a few research groups have worked on an automated approach for approximate circuit synthesis.

The Systematic methodology for Automatic Logic Synthesis of Approximate circuits (SALSA) starts with a description of the exact version of the circuit and an error constraint that specifies the type and amount of error that the implementation can exhibit [12]. The methodology introduces the so-called Q-function which takes the outputs from both the original circuit and approximate circuit and decides if the quality constraints are satisfied. The Q-function outputs a single Boolean value. The SALSA algorithm attempts to modify the approximate circuit with the goal of keeping the output of the Q-function unchanged. The execution times of SALSA (on a server with an AMD Opteron 6176, 2.29 GHz processor) ranged from 4 minutes to 2.5 hours for circuits such as multipliers, filters and discrete cosine transform blocks [12].

Another systematic approach, Substitute-And-SIMplIfy (SASIMI), tries to identify signal pairs in the circuit that exhibit the same value with a high probability, and substitutes one for the other [18]. These substitutions introduce functional approximations. Unused logic can be eliminated from the circuit which results in area and power savings.

In both cases, the design process is controlled by a predefined acceptable error, and hence we call the approaches *error-oriented*.

3 Evolutionary Approach to Approximate Circuits

The *evolutionary circuit design*, which has been developed in the framework of modern bio-inspired artificial intelligence, is the use of bio-inspired search algorithms for automated synthesis and optimization of circuit designs. The method has been utilized for digital as well as analogue circuits [19].

3.1 Evolutionary Circuit Design

Electronic circuits encoded as strings of symbols are constructed and optimized by the *evolutionary algorithm* (EA) in order to obtain a circuit implementation satisfying the specification. In order to evaluate a candidate circuit, a reconfigurable circuit (or its simulator if evolution is performed using a circuit simulator) is reconfigured using a new configuration created on the basis of the chromosome content. The configured device is then evaluated and its behavior is compared with the desired behavior. The fitness score is calculated which reflects to what extent the candidate circuit satisfies the specification.

Among various branches of EAs, *multiobjective* EAs (MOEA) have been recognized as a very valuable method in systems design as they naturally provide a set of candidate solutions showing various trade-offs among conflicting design objectives. The circuit design problem is thus transformed into the search problem.

The main reason why evolutionary circuit design has been studied and developed is its ability to (i) provide novel designs hardly reachable by means of conventional methods; (ii) deliver good solutions for problems where the specification is inherently incomplete and any golden solution does not exist; and (iii) achieve adaptation/fault tolerance directly at the hardware level. John Koza, the influential proponent of genetic programming, surveyed dozens of human-competitive designs produced by EA [20].

The main challenge is to overcome the *scalability issues* emerging in real-world applications of evolutionary circuit design, which primarily means developing EA-based methods capable of evolving complex circuits. Another disadvantage is that EA-based methods do not guarantee obtaining a solution with a predefined quality.

3.2 Cartesian Genetic Programming

Cartesian genetic programming (CGP) is one of the most suitable and popular methods for evolutionary circuit design [21].

A candidate circuit is modeled by means of a directed acyclic graph whose nodes (gates) are organized in c columns and r rows. The circuit utilizes m primary inputs and n primary outputs. Primary inputs and processing node outputs are labeled $0, 1, \ldots, m-1$ and $m, m+1, \ldots, m+c \cdot r-1$, respectively. Each node input can be connected either to the output of a node placed in previous l columns or to one of the primary circuit inputs, where l is one of CGP parameters. Figure 1 shows the CGP grid of nodes.

A candidate solution consisting of two-input nodes is represented in the chromosome by $r \cdot c$ triplets (x_1, x_2, ψ) determining for each processing node its function ψ ($\psi \in \Gamma$), and addresses of nodes x_1 and x_2 which its inputs are connected to. The last part of the chromosome contains n integers specifying either the nodes where the primary outputs are connected to or logic constants ('0' and '1') which can directly be connected to the primary output. While the chromosome size is constant for a given product $r \cdot c$, the phenotype size is variable and measured as the number of used nodes (gates).

The initial population of CGP is created either randomly or by means of existing circuits. Calculating the fitness value is a two-phase process. Firstly, the circuit functionality is determined, e.g. by computing responses for all possible assignments to the inputs. After reaching a satisfactory accuracy in the course of evolution or when CGP is seeded by fully functional designs, the second phase is initiated in which the circuit size (or other objectives) can be optimized. CGP employs a $(1 + \lambda)$ evolution strategy whose pseudo-code is given in Algorithm 1. This search method is based on a point mutation operator which modifies h randomly selected genes (integers) of the parent circuit. The role of mutation is

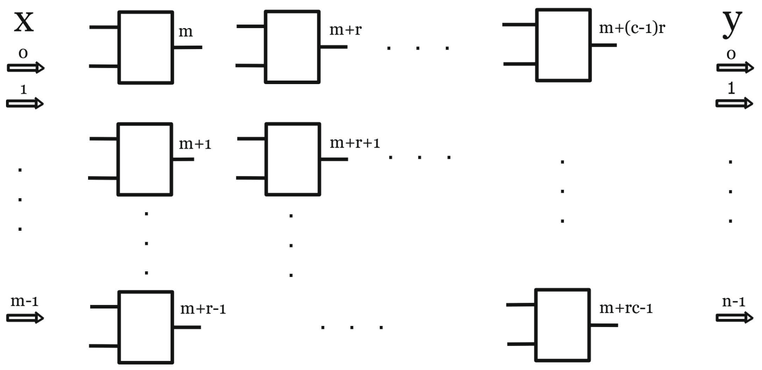

Fig. 1. An array of $r \cdot c$ 2-input nodes used in CGP. The number of inputs and outputs is m and n.

Algorithm 1. CGP

```
Input: CGP parameters, fitness function
Output: The highest scored individual $p$ and its fitness
1 $P \leftarrow$ randomly generate parent $p$ and its $\lambda$ offspring;
2 EvaluatePopulation($P$);
3 while ⟨terminating condition not satisfied⟩ do
4     $\alpha \leftarrow$ highest-scored-individual($P$);
5     if $fitness(\alpha) \geq fitness(p)$ then
6         $p \leftarrow \alpha$;
7     $P \leftarrow$ create $\lambda$ offspring of $p$ using mutation;
8     EvaluatePopulation($P$);
9 return $p$, fitness($p$);
```

substantial because even a single modified gene (integer) can significantly change the phenotype.

3.3 Evolution of Approximate Circuits

The CGP-based design methods were introduced for the design of approximate circuits because it was expected that they can provide much better solutions (i.e. approximate circuits) for a larger class of circuits and multiple conflicting objectives than existing design methods. We will briefly introduce our previous work, in which we proposed two approaches to the evolutionary design of approximate circuits by means of CGP.

In paper [22], we exploited the facts that power consumption is often highly correlated with occupied resources and the evolutionary design is capable of constructing partially working solutions even if sufficient resources (required for finding a fully functional solution) are not available. Let z be the (minimum) number of gates required for obtaining an accurate function. CGP is employed to minimize the error providing that only $z-1$ gates are available. The process can be repeated for $z-2$, $z-3$ etc. gates. The user thus obtains a set of approximate combinational circuits, each of which typically exhibits different trade-off between the functionality and the number of gates. This approach can be considered as an *area-oriented* method because the user can control the used area (and so power consumption) more comfortably than by means of the error-oriented methods.

In paper [23], we proposed a complementary design approach. The user is supposed to define a required error level e_{max} (e.g. the average error magnitude). CGP, which is seeded by a conventional fully functional implementation, is utilized to modify the seed in order to obtain a circuit with predefined e_{max}. After obtaining that circuit, CGP can minimize the mean error, the number of gates or other criteria providing that e_{max} is left unchanged.

Because the utilized power estimation algorithm is very time consuming, it has not been included into the fitness function directly. Power consumption was calculated at the end of evolution for the best evolved approximate circuits. In both cases we demonstrated that for the cost of runtime the proposed methods provide better trade-offs for elementary arithmetic circuits (such as adders and multipliers) than conventional methods. The error-oriented approach tends to be less computationally demanding.

4 Multiobjective Approximate Circuit Evolution

Both aforementioned approaches are the single-objective optimization methods. In this section, we will demonstrate how truly multiobjective evolutionary optimization algorithms can be employed to approximate circuits design.

4.1 Multiobjective Optimization

Multiobjective evolutionary algorithms are utilized if multiple conflicting objective functions are formulated. Contrasted to the single-objective EAs, they internally sort individuals according to the dominance relation, build archives of so-called non-dominating solutions, and ensure population diversity to avoid converging to a single solution. In order to compare two solutions, the *dominance relation* is defined as follows: Solution $\boldsymbol{x}$ dominates another solution $\boldsymbol{y}$ if two conditions are satisfied:

1. The solution $\boldsymbol{x}$ is no worse than $\boldsymbol{y}$ in all objectives.
2. The solution $\boldsymbol{x}$ is strictly better than $\boldsymbol{y}$ in at least one objective.

In the set of solutions P, the non-dominated subset of solutions P' contains those solutions that are not dominated by any member of P. The non-dominated subset of all possible solutions is called the Pareto-optimal set. The ultimate goal of a multiobjective optimization is to find all Pareto-optimal solutions in a single run of MOEA.

Instead of evolving for every possible number of gates or error (as we have seen in Section 3.3), various trade-offs can be obtained in a single run of a suitable MOEA. Hence we combined CGP encoding with a typical multiobjective evolutionary algorithm NSGA-II [24]. The proposed MOEA will be seeded by a fully functional circuit. The goal is to simultaneously minimize the error and the number of gates, providing that solutions showing the error higher than E_{max} are infeasible.

4.2 Case Study

The proposed MOEA is evaluated in the task of a 4-bit multiplier approximation. The conventional 4-bit multiplier considered in this study consists of 59 gates and calculates an 8-bit product from two unsigned 4-bit operands. The error criterion is a mean absolute error between the produced outputs and correct outputs for all possible assignments to the inputs (2^8 vectors). The CGP parameters are initialized as follows: $r = 1$, $c = 59$, $l = c$, $\lambda = 4$, $h = 5\%$, $E_{max} = 5000$. The set of available gates is $\Gamma = \{$NOT, AND, OR, XOR, NAND, NOR, XNOR$\}$. The evolutionary algorithm operates with a 50 member population and stops when $g_{max} = 32 \cdot 10^6$ generations are spent. This corresponds with a four hour run on an Intel Xeon processor running at 3 GHz. The setting of these values is based on our previous experiments.

Fig. 2 shows all the trade-offs obtained from 100 independent runs of the proposed MOEA. It can be seen that several solutions have been discovered for

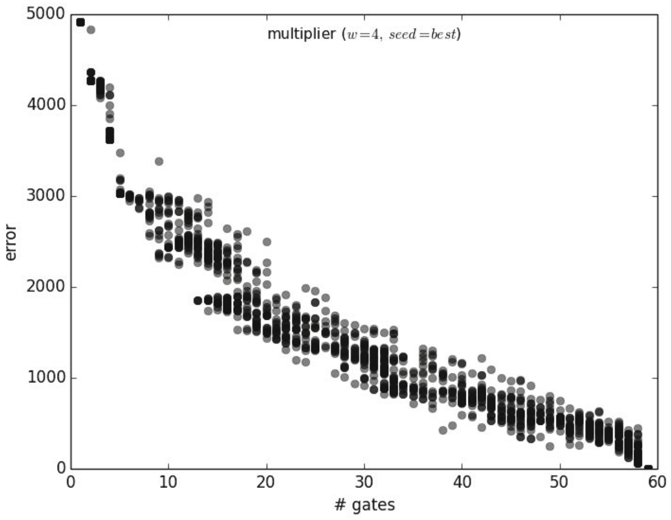

Fig. 2. Trade-offs between the number of gates and error for the 4-bit approximate multiplier (59 gates corresponds to the perfect functionality).

every possible number of gates. A single run led to 16.97 different solutions on average. A detailed analysis of the best evolved approximate circuits revealed that a circuit containing k gates can exhibit a higher error than a circuit containing $k-1$ gates (see, e.g., the error for 24 and 25 gates). It is, however, assumed that a smaller error is obtained if more gates are allowed in the circuit. Hence the current version and setting of the method seems to be inefficient. On the other hand, the method is capable of producing many useful trade-offs much faster than multiple runs of single-objective EAs from [22,23].

5 Conclusions

The field of approximate computing in general and approximate circuits in particular is in an early stage of development. Only a few systematic design methodologies have been proposed for the approximate circuit design so far. Despite the interesting preliminary results obtained by EAs, it is a well-known problem that EAs utilize very time consuming algorithms, the scalability of resulting solutions is limited and the whole process is too non-deterministic for the community of "conventional" designers. On the other hand, even conventional approaches have to employ heuristics and time-consuming procedures in order to approximate circuit designs.

In the context of approximate circuit design methodologies, the future research should mainly deal with the following issues:

– More efficient and accurate multiobjective EAs, employing more scalable circuit representations and efficient genetic operators should be introduced specifically for the task of approximation.
– Discovering time-efficient algorithms for checking to what extent a complex approximate circuit corresponds with the exact specification is a challenging task. Current approaches based on testing circuit's responses for all possible combinations of inputs are not scalable. The desired algorithm must be fast, because it will be called to evaluate millions of candidate circuits produced by MOEA.
– NP-hard problems vary greatly in their approximability. Key circuit classes (such as adders, multipliers and other arithmetic circuits) should be analyzed with respect to their approximability under various error measures and constraints. Complexity measures of approximate Boolean functions, similar to those used for conventional Boolean functions [25], should be developed and exploited.
– Instead of heuristic methods (such as EAs), a more rigorous concept, similar to approximation algorithms in computer science, should be developed to guarantee a provable solution quality in provable run-time bounds.
– Resulting approximate circuit design methods should be integrated to standard circuit design and optimization tools.
– Automated methodologies allowing designers to identify those system's components that can be replaced by their approximate counterparts should be developed.

All these issues have to be seen in the context of hardware, because very good circuit approximations obtained for a given fabrication technology can become useless when another fabrication technology is considered.

Approximate computing is a promising emerging paradigm which is quite important for future low power and resources-efficient computers. However, a lot of work has to be done in order to be widely accepted.

Acknowledgments. This work was supported by the Czech science foundation project 14-04197S.

References

1. Gruska, J.: Foundations of Computing. Int. Thomson Publishing Computer Press (1997)
2. Vazirani, V.V.: Approximation Algorithms. Springer Verlag (2001)
3. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals and Systems **2**(4), 303–314 (1989)
4. Volder, J.E.: The birth of cordic. Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology **25**(2), 101–105 (2000)
5. Han, J., Orshansky, M.: Approximate computing: An emerging paradigm for energy-efficient design. In: Proc. of the 18th IEEE European Test Symposium, pp. 1–6. IEEE (2013)
6. Chippa, V.K., Chakradhar, S.T., Roy, K., Raghunathan, A.: Analysis and characterization of inherent application resilience for approximate computing. In: The 50th Annual Design Automation Conference, DAC 2013, pp. 1–9. ACM (2013)
7. Mishchenko, A.: ABC: A system for sequential synthesis and verification, Berkley logic synthesis and verification group (2012)
8. Kahng, A.B., Lienig, J., Markov, I.L., Hu, J.: VLSI Physical Design: From Graph Partitioning to Timing Closure. Springer-Verlag (2011)
9. Shanbhag, N.R., Abdallah, R.A., Kumar, R., Jones, D.L.: Stochastic computation. In: The 47th Annual Design Automation Conference, DAC 2010, pp. 859–867. ACM (2010)
10. Gupta, V., Mohapatra, D., Raghunathan, A., Roy, K.: Low-power digital signal processing using approximate adders. IEEE Trans. on CAD of Integrated Circuits and Systems **32**(1), 124–137 (2013)
11. Kulkarni, P., Gupta, P., Ercegovac, M.D.: Trading accuracy for power in a multiplier architecture. J. Low Power Electronics **7**(4), 490–501 (2011)
12. Venkataramani, S., Sabne, A., Kozhikkottu, V.J., Roy, K., Raghunathan, A.: Salsa: systematic logic synthesis of approximate circuits. In: The 49th Annual Design Automation Conference, DAC 2012, pp. 796–801. ACM (2012)
13. Lu, S.L.: Speeding up processing with approximation circuits. IEEE Computer **37**(3), 67–73 (2004)
14. Esmaeilzadeh, H., Sampson, A., Ceze, L., Burger, D.: Neural acceleration for general-purpose approximate programs. In: Proc. of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 449–460. IEEE Computer Society (2012)
15. Sampson, A., Dietl, W., Fortuna, E., Gnanapragasam, D., Ceze, L., Grossman, D.: 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, pp. 164–174. ACM (2011)

16. Gupta, P., Agarwal, Y., Dolecek, L., Dutt, N., Gupta, R.K., Kumar, R., Mitra, S., Nicolau, A., Rosing, T.S., Srivastava, M.B., Swanson, S., Sylvester, D.: Underdesigned and opportunistic computing in presence of hardware variability. IEEE Trans. on CAD of Integrated Circuits and Systems **32**(1), 8–23 (2013)
17. Venkatachalam, V., Franz, M.: Power reduction techniques for microprocessor systems. ACM Computing Surveys **37**(3), 195–237 (2005)
18. Venkataramani, S., Roy, K., Raghunathan, A.: Substitute-and-simplify: A unified design paradigm for approximate and quality configurable circuits. In: Design, Automation and Test in Europe, DATE 2013, EDA Consortium San Jose, CA, USA, pp. 1–6 (2013)
19. Lohn, J.D., Hornby, G.S.: Evolvable hardware: Using evolutionary computation to design and optimize hardware systems. IEEE Computational Intelligence Magazine **1**(1), 19–27 (2006)
20. Koza, J.R.: Human-competitive results produced by genetic programming. Genetic Programming and Evolvable Machines **11**(3–4), 251–284 (2010)
21. Miller, J.F.: Cartesian Genetic Programming. Springer-Verlag (2011)
22. Sekanina, L., Vasicek, Z.: Approximate circuits by means of evolvable hardware. In: 2013 IEEE International Conference on Evolvable Systems, Proceedings of the 2013 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 21–28. IEEE CIS (2013)
23. Vasicek, Z., Sekanina, L.: Evolutionary design of approximate multipliers under different error metrics. In: IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems 2013, pp. 135–140. IEEE (2014)
24. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-II. IEEE Transactions on Evolutionary Computation **6**(2), 182–197 (2002)
25. Buhrman, H., de Wolf, R.: Complexity measures and decision tree complexity: A survey. Theoretical Computer Science **288**(1), 21–43 (2002)