IET Computers & Digital Techniques

Special Section: Bio-inspired Hardware and Evolvable Systems

IET Journals
The Institution of Engineering and Technology

Role of circuit representation in evolutionary design of energy-efficient approximate circuits

ISSN 1751-8601
Received on 13th September 2017
Revised 20th March 2018
Accepted on 24th April 2018
E-First on 6th June 2018
doi: 10.1049/iet-cdt.2017.0188
www.ietdl.org

Vojtech Mrazek[1] ✉, Zdenek Vasicek[1], Radek Hrbacek[1]

[1]*Faculty of Information Technology, Brno University of Technology, Centre of Excellence IT4Innovations, Czech Republic*

✉ *E-mail: imrazek@fit.vutbr.cz*

Abstract: Circuit approximation has been introduced in recent years as a viable method for constructing energy-efficient electronic systems. An open problem is how to effectively obtain approximate circuits showing good compromises between key circuit parameters – the error, power consumption, area and delay. The use of evolutionary algorithms in the task of circuit approximation has led to promising results. Unfortunately, only relatively small circuit instances have been tackled because of the scalability problems of the evolutionary design method. This study demonstrates how to push the limits of the evolutionary design by choosing a more suitable representation on the one hand and a more efficient fitness function on the other hand. In particular, the authors show that employing full adders as building blocks leads to more efficient approximate circuits. The authors focused on the approximation of key arithmetic circuits such as adders and multipliers. While the evolutionary design of adders represents a rather easy benchmark problem, the design of multipliers is known to be one of the hardest problems. The authors evolved a comprehensive library of energy-efficient 12-bit multipliers with a guaranteed worst-case error. The library consists of 65 Pareto dominant solutions considering power, delay, area and error as design objectives.

1 Introduction

In recent years, a new research field was established to investigate how computer systems can be made more energy efficient, faster and less complex by relaxing the requirement that they are correct. This field, denoted as *approximate computing*, exploits the fact that many applications are error resilient and the errors in computing are thus either invisible or acceptable [1]. The concept of approximation has intensively been studied, developed and applied not only in computer science, but also in mathematics and engineering disciplines. However, it has never been applied in the areas in which only accurate implementations have traditionally been accepted. Nowadays, the designers intentionally introduce errors into computation to satisfy the never-ending requirement for lowering of power consumption.

As one of the most promising energy-efficient computing paradigms that is able to cope with current challenges of computer engineering, approximate computing has gained a lot of research attention in the past few years. We can identify two main directions in approximate computing: energy-efficient computing with unreliable components and approximation of systems implemented on common platforms [1]. In the first case, the problem is that the exact computation utilising nanometre transistors provided by recent technology nodes is extremely expensive in terms of energy requirements and reliable behaviour. An open question is how to effectively and reliably compute with a huge number of unreliable components. The second research direction is motivated by the fact that many applications (typically in the areas of multimedia, graphics, data mining and big data processing) are inherently *error resilient*. This resilience can be exploited in such a way that the error is exchanged for improvements in power consumption, throughput or implementation cost. After analysing many applications, Chippa *et al.* [2] reported that about 83% of the runtime is spent in computations that can be approximated.

Various approximation techniques have been proposed recently. A good survey of the proposed approaches can be found, for instance, in [1, 3]. According to the level of the computer stack where the approximations are conducted, the approaches could be roughly divided into software level and hardware level. At the software level, for example, we could selectively ignore certain computations and/or memory accesses that are not critical for obtaining the desired quality of the result. At the hardware layer, we could either use a less accurate yet more energy-efficient circuit for computation or purposely reduce the supply voltage for certain hardware components to trade-off energy and accuracy.

As the complexity of today's computer systems grows, the manual approximation is not an efficient design method. Hence, several automated approximate design methods have been introduced. The design of approximate circuits is typically based on modifying fully functional circuits. Venkataramani *et al.* [4], for example, uses a quality function which decides whether a predefined quality constraint is met or not. The algorithm is allowed to modify the circuit as long as the quality constraint is not violated. Among others, 32-bit adders, 8-bit multipliers, finite impulse filter (FIR) filters and discrete cosine transform (DCT) blocks were approximated. Another approach looks for signal pairs having similar values with a high probability. By substituting one signal for the other, a part of the circuit can be removed resulting in area and power savings at the cost of an error introduced to the output [5]. Unlike the aforementioned methods, Nepal *et al.* [6] proposed an approach operating directly on the behavioural descriptions of circuits. His method generates approximate circuits from input behavioural descriptions by performing global transformations on an abstract synthesis tree created from the behavioural description. The outcome approximate circuits are still expressed in behavioural code and can be synthesised by means of standard synthesis tools. The main weakness of these design methods, however, is that they are typically able to produce only a few design points.

Several papers dealing with the evolutionary design of approximate circuits have been published. One of the seminal work on this topic is the paper of Sekanina *et al.* who addressed the problem of evolutionary design of small approximate circuits consisting of elementary gates [7]. Later, Vasicek and Sekanina [8] introduced heuristic seeding and demonstrated how to improve the scalability of the evolutionary design of approximate circuits. The proposed method was applied to the evolution of 4-bit multipliers and 25-input median circuits. Recently, Mrazek *et al.* [9] utilised evolutionary approach to evolve energy-efficient 8-bit approximate multipliers optimised for the usage in artificial neural networks. In addition to that, multi-objective design of 8-bit approximate multipliers was addressed by Hrbacek *et al.* [10]. Apart from the

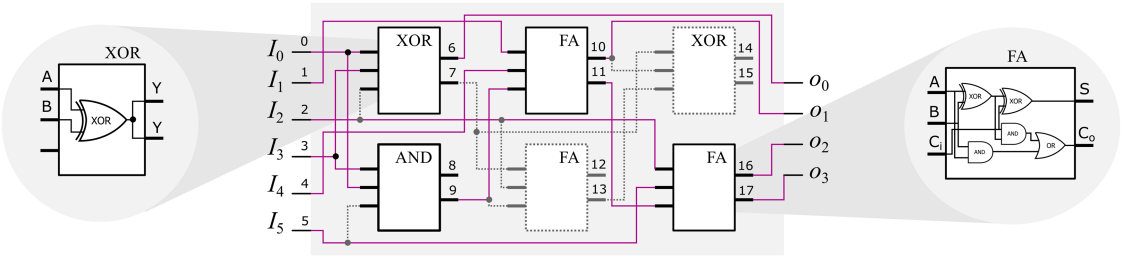

Fig. 1 *Example of a circuit (3-bit ripple-carry adder) encoded using CGP with parameters:* $n_i = 6$, $n_o = 4$, $n_a = 3$, $n_b = 2$, $n_c = 3$, $n_r = 2$, $\Gamma = \{0^{and}, 1^{or}, 2^{xor}, 3^{full-adder}\}$. *Given these parameters, the considered circuit can be represented using a chromosome consisting of the following sequence of 28 integers: 0, 3, 2, 2; 3, 0, 5, 0; 1, 4, 9, 3; 7, 2, 9, 3; 7, 10, 13, 2; 2, 5, 11, 3; 6, 10, 16, 17. Two nodes with outputs 12, 13, 14 and 15 are not used (top-right XOR gate and full adder). These redundant and inactive nodes as well as the redundant wires are greyed out. The common two-input logic gates are implemented as shown on the left – only two inputs are employed, the third input is ignored, both outputs implement the same logic function. More complex blocks (full adder in this example) may utilise all inputs and may capture more Boolean functions (two functions are implemented in case of the full adder). The principle of the CGP encoding is as follows. The first 24 integers are divided into six quadruples that define input connections and function of each of six CGP nodes. The first quadruple (0, 3, 2, 2) is associated with the top-left node. The first three numbers of each quadruple determine indexes where the node's inputs are connected. The last (underlined) number defines the function of a node. Considering the first quadruple, the node represents XOR gate connected to* I_0, I_3 *and* I_2. *However, the third connection, i.e.* I_2, *is in fact redundant because XOR is a two-input logic gate. The last four numbers of the chromosome, i.e. the numbers 6, 10, 16 and 17, define the connection of the primary outputs. The first primary output* O_0 *associated with the first number is connected to the first output of the top-left node because this node output has index 6*

approximate design of median circuits and work of Hrbacek, the authors represent the circuits by means of basic logic gates. There is no work that investigates whether there is a better representation that may improve the performance of the evolutionary algorithm or the quality of the obtained approximate circuits. In addition to that, the power consumption is typically optimised indirectly as the number of gates or the area on the chip (see e.g. [8, 9]).

In this paper, we present a comprehensive analysis which compares two different representations – the mainstream *gate-level representation* where we represent the circuits using common logic gates, and *cell-based representation* that utilises more complex building blocks such as full adders. Interestingly, the gate-level representation represents a routinely adopted approach since the late 1990s [11]. We hypothesise that evolution at the level of more complex cells could produce solutions of higher quality because the standard cells available in every technology library exhibit substantially better design parameters (area, power, delay) compared to the equivalent circuits implemented using standard gates. In order to confirm the validity of this claim, we applied evolutionary methods to the design of key arithmetic circuits such as adders and multipliers. In particular, 8-bit and 12-bit approximate circuits were considered. In order to support the evolution of 12-bit circuits, we implemented a state-of-the-art circuit simulator operating on 256 bits in parallel. The simulator, first introduced in [10], is employed to determine the quality of approximate circuits. In addition to that the switching activity is simultaneously calculated. A robust power estimation engine based on known switching activity represents a key idea how to ensure that the evolutionary approach produces energy-efficient solutions.

The contributions of this paper are as follows. This is the first time a detailed analysis of various representations of digital circuits is evaluated and discussed. We analyse also the impact of the chosen representation on results produced by a synthesis tool utilised to obtain a physical implementation of a given circuit. Finally, this is the first paper that presents an automatic approach that is able to produce high-quality approximate 12-bit multipliers with guaranteed error parameters. We obtained >60 Pareto dominant implementations that are available for download (http://www.fit.vutbr.cz/research/groups/ehw/approxlib). This result is encouraging for evolutionary computation community on the one hand and practically useful for hardware community on the other hand. The 8-bit and 12-bit approximate multipliers can be employed directly to improve the power efficiency of deep neural networks [1, 9] or as building blocks of complex circuits. Four 8-bit multipliers, for example, can be employed to construct a 16-bit approximate multiplier using the common approach of constructing larger multipliers from smaller ones [12].

The rest of this paper is organised as follows. The proposed method is introduced in Section 2. The design methodology followed by the analysis of obtained results is presented in Section 3. Finally, the conclusions are given in Section 4.

2 Proposed method

In order to approximate digital circuits, various approaches have been proposed [4–7]. In this work, we employ Cartesian genetic programming (CGP) [11]. CGP can easily handle constraints given on candidate circuits, the method is naturally multi-objective and high-quality approximate circuits have already been obtained with CGP [8, 10].

This section introduces the overall idea of the proposed method, the utilised evolutionary algorithm and the construction of the fitness function for the design of energy-efficient approximate circuits.

2.1 Representation of digital circuits

Standard CGP is a branch of genetic programming which represents candidate designs using directed acyclic graphs [11]. A candidate circuit is modelled using a two-dimensional (2D) array of programmable nodes with n_c columns and n_r rows. Originally, CGP with single-output programmable nodes was introduced by Miller in 1998 [13]. Simple nodes with two inputs and single output were considered in the evolution of digital circuits [13]. This approach, however, can be generalised to support nodes with arbitrary number of inputs and outputs. In this work, we use the extended version of CGP that supports nodes with n_a inputs and outputs [10]. This arrangement allows one to have a node evaluating up to n_b Boolean functions defined over n_a variables. The function of a node, however, cannot be arbitrary. Each node can implement one of n_G functions defined by Γ. The node parameters (i.e. n_a, n_b) are fixed during the evolution. In order to fully specify the behaviour of each node, we use the following principle. In case that a node implements a Boolean function, which utilises less than n_a operands, the redundant input connections are ignored. In case that a node implements less than n_b Boolean functions, the unused outputs are internally connected to the output of the last Boolean function. Let us suppose, for example, that a single output Boolean function is chosen from Γ. Then, all n_b outputs produce the same value (see e.g. schematics of XOR gate shown in Fig. 1 implemented using three-input two-output CGP node). This mechanism helps us to avoid the necessity to use chromosome validation and repair mechanisms.

The circuit utilises n_i primary inputs and n_o primary outputs. Feedback connections are not enabled. The primary inputs are labelled $0, 1, \ldots, (n_i - 1)$. The first output of the first CGP node is assigned index n_i, the second output $n_i + 1$ and the last output of

this node is associated with $n_i + n_b - 1$. The remaining outputs of the CGP nodes are successively labelled $(n_i + n_b), (n_i + n_b + 1), \ldots, (n_c \cdot n_r \cdot n_b + n_i - 1)$. All the labels are considered as addresses where the node inputs and primary outputs can be connected to. A candidate solution is represented by means of the so-called *chromosome* (which is, in fact, a netlist) by $n_r \cdot n_c$ tuples consisting of $n_a + 1$ items $(x_1, x_2, \ldots, x_{n_a}, \psi)$ determining for each node its function ψ ($\psi \in \Gamma$) and input connections x_i ($0 \leq x_i < n_c \cdot n_r \cdot n_b + n_i$). The last part of the chromosome contains n_o integers specifying the nodes where the primary outputs are connected to. While the chromosome size s is constant $s = n_c n_r (n_a + n_b) + n_o$, the circuit size is variable and measured as the number of active (i.e. used) nodes. The set of valid chromosomes (netlists) represents the whole search space.

The encoding used in CGP is highly redundant as many nodes can be disconnected and deactivated during evolutionary optimisation. In order to deactivate a node, it is sufficient to reconnect all active references to that node. Moreover, there are usually many ways to implement a given logic function in each CGP instance. This redundancy together with a relatively powerful mutation operator is considered as a key feature of CGP allowing for an efficient circuit evolution [11].

An example of a circuit (common 3-bit ripple carry adder) represented using CGP is shown in Fig. 1. Despite the fact that the adder can be represented using three three-input/two-output CGP nodes (three full adder cells), we employ six three-input/two-output CGP nodes arranged into three columns and two rows. This arrangement helps us to demonstrate the redundancy of CGP representation (only four out of six nodes are active) and the flexibility of the proposed encoding (a half adder is implemented using two logic gates, full adders are implemented as standard cells). In addition to that, the example shows that the chromosome captures a valid netlist even though the second output of the AND node (output with label 9) is connected to another node. As stated earlier, all the outputs of a node representing a single-output logic gate are equivalent. The corresponding chromosome is shown in Fig. 1.

2.2 Search strategy

CGP employs a simple search method. In our case, the initial population P of CGP contains one of various implementations of the accurate circuit (e.g. multiplier) and a few circuits generated using mutation of the accurate circuit. Creating the accurate multiplier required by the initial population is trivial as there is a one-to-one mapping between circuit netlists and CGP chromosomes. The next step consists in the evaluation of candidate circuits using the fitness function. Each member of P then receives the so-called fitness score and the highest-scored individual becomes the new parent of the next population. From this parent, λ candidate solutions are generated using mutation. The termination criterion is given by the number of iterations or maximum acceptable runtime.

Despite several attempts to propose a suitable crossover operator to CGP [14, 15], the mutation is still used as the crucial genetic operator. The mutation operator modifies up to h randomly chosen genes (integers) of the chromosome. Their new values are generated randomly, but it is checked whether the new values are valid. One mutation can affect either the node function, node input connection, or primary output connection. As the mutation operator is able to disconnect gates (by changing either primary output connection, node input connection or node function), it can be employed to reduce redundancy of the initial circuit.

2.3 Fitness function

Since the goal is to design energy-efficient approximate circuits, it is necessary to integrate this requirement into the fitness. The power consumption of the candidate circuits can be optimised directly or indirectly. In the context of evolutionary computation, only the latter approach has been applied in the literature because it does not require to implement complex simulation engines or employ time-demanding analogue simulations. In [8], for example, the authors demonstrated that it is sufficient to reduce the number of gates because the power consumption of arithmetic circuits is highly correlated with the area. For recent technology nodes, however, this simplification may lead to unsatisfactory results especially for circuits consisting of few gates exhibiting high switching activity. Hence we propose to optimise the power directly.

The power consumption of digital circuits can be divided into the dynamic (P_{dynamic}) and the static (P_{static}) power components. The first one occurs every time the output of a gate changes its logic value. Static power consumption is caused mainly by the leakage current which exists even when the circuit is in a stable state, i.e. not switching. Even though the static power component has always been present, it has gained importance in sub-micrometre and nanometre devices [16]. As a consequence of that, the total power consumption has to be optimised by reducing static as well as dynamic part of the power consumption.

Since the static part of the power consumption depends only on a function of a logic gate, the total static power consumption can be obtained by summing static leakage P_{leak} for all gates of the candidate circuits, i.e. $P_{\text{static}} = \sum_{\forall i} P_{\text{leak}}^{(i)}$. The situation gets complicated when we want to precisely determine dynamic power. In order to simplify this process and avoid running a costly analogue simulator, we propose to exploit the knowledge of the switching activity of each gate. The dynamic power consumption of a single gate P_{dyn} can be defined as follows:

$$P_{\text{dyn}} = \frac{1}{2} \times C_{\text{load}} \times V_{\text{dd}}^2 \cdot f \cdot E(\text{transitions}), \tag{1}$$

where C_{load} is the total load capacitance of the output (i.e. the sum of all input capacitances of the connected gates defined in the liberty file), V_{dd} is the supply voltage, f is the target frequency and E(transitions) is the expected value of the output transitions per global clock cycle (switching activity) [17]. The total dynamic power is equal to $P_{\text{dynamic}} = \sum_{\forall i} P_{\text{dyn}}^{(i)}$ and the total power consumption of a candidate circuit is calculated as

$$P_{\text{total}} = P_{\text{static}} + P_{\text{dynamic}}. \tag{2}$$

To simplify the problem, glitches are not typically considered. This decision enables to use the zero-delay model. As a consequence of that, the switching activity can be obtained using common circuit simulator which evaluates the response for all (or some for complex problems) input vectors. Total switching activity of a gate is calculated as follows:

$$E(\text{transitions}) = 2 \cdot (p_0 \cdot p_1) = 2 \cdot p_1 \cdot (1 - p_1), \tag{3}$$

where p_0 is the probability that the output of a considered gate is equal to logical zero, similarly p_1 is the probability that the output is equal to logical one. There are more ways to determine the transition probabilities. The simplest approach is to use the simulation and count the number of cases for which the output value was equal to 1. The advantage of this approach is that this calculation can be done during the function verification which represents an inevitable step of the fitness evaluation.

Various error criteria can be utilised to evaluate the quality of approximate arithmetic circuits. The average-case and worst-case arithmetic errors represent the most common metrics considered in the context of design of approximate arithmetic circuits [10, 18]. The worst-case error $e_{\text{wst}}(C)$ is employed in this paper. This metric is defined as the maximum absolute difference in magnitude between the original and approximate circuit computed over all inputs:

$$e_{\text{wst}}(C) = \max_{\forall \boldsymbol{i}} |O(C_{\text{orig}}, \boldsymbol{i}) - O(C, \boldsymbol{i})|, \tag{4}$$

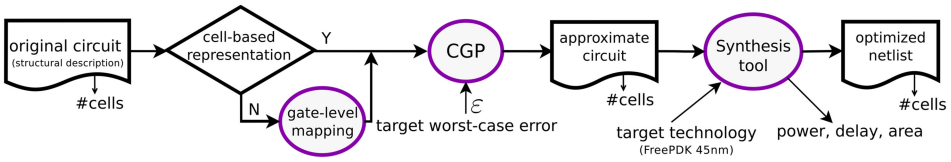

Fig. 2 *Proposed experimental setup for evolutionary design of approximate circuits represented using either standard gates or standard cells*

where $O(C_{\text{orig}}, \boldsymbol{i})$ denotes the output value of the fully functional circuit for the input vector $\boldsymbol{i}$ and $O(C, \boldsymbol{i})$ denotes the output value of approximate circuit C. For a circuit having n_{i} inputs, the input vectors are the numbers $0 \leq \boldsymbol{i} < 2^{n_{\text{i}}}$.

Let ε be the maximum acceptable worst-case error. Then, the fitness value of a candidate circuit C is calculated as follows:

$$\text{fitness}(C) = -\begin{cases} P_{\text{norm}}(C) & \text{if } e_{\text{wst}}(C) < \varepsilon \\ \infty & \text{otherwise,} \end{cases} \tag{5}$$

where $P_{\text{norm}}(C) = P_{\text{total}}(C)/P_{\text{total}}(C_{\text{orig}})$ is the normalised power consumption of C and $P_{\text{total}}(C_{\text{orig}})$ is the energy consumption of the original reference circuit C_{orig} and represents a constant value. The goal of the evolutionary strategy is to maximise the fitness and thus minimise the cost metric P_{norm}. It means that solutions with the error greater than ε are infeasible. Depending on design objectives, CGP has to be executed multiple times with different target errors ε_{i} if a Pareto front is requested.

3 Experimental results

The proposed method was evaluated in the task of evolutionary design of approximate adders and multipliers. These arithmetic circuits were chosen because they represent key components of many real-world applications in signal processing and machine learning [1]. In addition to that, the evolution of multipliers is considered as a very difficult benchmark in the evolutionary community. Apart from common 8-bit instances typically addressed in the literature (see e.g. a survey paper [19] which lists and compares different 8-bit approximate multipliers), 12-bit instances were chosen to validate our hypothesis. In case of 12-bit multipliers, we have to deal with complex netlists consisting of hundreds of gates. As our goal is to evolve approximate circuits with known worst-case error parameters, we need to evaluate the response for all input vectors for every candidate solution. It means that 2^{24} input combinations have to be evaluated for a 12-bit multiplier. This is the main reason, why we did not consider 16-bit instances. As it is infeasible to evaluate 2^{32} responses in a reasonable time, it would be necessary to employ a random simulation. Using a subset of all possible input combinations, however, may lead to a bias in evaluation since we cannot guarantee whether the worst acceptable error is met or not. Due to the limited space, only results for 12-bit multipliers will be presented in detail.

The experimental setup depicted in Fig. 2 is as follows. The goal of the evolution is to design an approximate multiplier showing the lowest possible power consumption for a given worst-case error. The power consumption is estimated as described in Section 2.3 and represents the only criterion reflected in the fitness function. The evolutionary algorithm starts with a common conventional multiplier whose fitness (i.e. power) is gradually optimised while keeping the worst-case error within the required bound. The worst-case error is used as a constraint ε_{i}. For each circuit, 11 error levels ranging from $\varepsilon_1 = 0.02\,\%$ to $\varepsilon_{11} = 20\,\%$ were considered. This range covers the values that are typically employed in the literature (see e.g. [9, 18, 19]). The CGP parameters were initialised as follows. We employed $\lambda = 24$ individuals in the population, mutation rate was $h = 5\,\%$, number of rows $n_{\text{r}} = 1$. The number of columns equals to the number of components of the initial conventional multiplier used to seed the evolution. The setting of the CGP parameters is based on the experiments conducted in our previous research [8]. Two different sets of experiments were executed. In the first scenario, only common two-input gates were considered. Each CGP node had two inputs and one output (i.e. $n_{\text{a}} = 2$, $n_{\text{b}} = 1$) and could implement one of the following eight functions: $\Gamma_1 = \{$BUF (buffer), INV (inverter), AND, OR, XOR, NAND, NOR, XNOR$\}$. In the second scenario, the set of functions was extended to 15 functions that correspond with common standard cells available in the chosen target technology: $\Gamma_2 = \Gamma_1 \cup \{$NAND3 (3-input NAND), NOR3 (3-input NOR), MUX2 (2-to-1 multiplexer), AOI21 (3-input AND/NOR), OAI21 (3-input OR/NAND), FA (full adder), HA (half adder)$\}$. The advantage of complex cells is that they are highly optimised for each target technology considering the area on a chip and performance. The full adder, for example, available as a technology cell occupies typically lower area, consumes less power and has lower delay compared to a full adder implemented using common gates. In order to support these functions, a CGP node with three inputs and two outputs, which corresponds with $n_{\text{a}} = 3$ and $n_{\text{b}} = 2$, was employed. Due to practical reasons, runtime was chosen as the only terminating criterion. This decision helps us to easily predict the end of evolution.

The evaluation of the fitness consists of two steps. In the beginning, active nodes are identified. Then, only active nodes are evaluated by means of a circuit simulator for all input combinations. In both scenarios, we utilised exactly the same circuit simulator based on a machine-code translation introduced in [20] and further extended in [21]. During the detection of the active nodes, each complex block such as full adder is replaced by an equivalent circuit consisting of common two-input gates. This simple preprocessing helps one to improve the performance of the simulator. Since there are only basic operations, SIMD instructions from AVX extension allowing processing 256-bit vectors can directly be exploited. As a consequence of that, we can simulate the response for 256 input vectors in parallel.

In order to avoid a possible bias preventing discovering of some implementations caused by seeding, we will seed the evolution with several conventional architectures of multipliers. The multipliers include ripple-carry array multiplier (denoted as RCAM), two carry-save array multipliers (CSAM1 and CSAM2) and three Wallace tree architectures (WTM1, WTM2 and WTM3). The array multiplier (RCAM) offers the lowest speed but occupies the smallest area on a chip compared to the other variants and especially the most expensive Wallace tree multiplier. The carry-save array multiplier is implemented as a set of 1-bit full adders without any carry-chaining that is finally reduced using a single n-bit adder. This arrangement helps one to slightly improve the delay compared to RCAM without introducing a noticeable area overhead. Two variants of carry-save multiplier are considered depending on the adder employed in the final stage – CSAM1 utilising ripple-carry adder (RCA) and CSAM2 with carry-save adder (CSA). Wallace-tree adder multiplier is the fastest known architecture which sums the partial products using multiple levels of CSA. Similarly to the carry-save array multiplier, the products are finally summed up using a single n-bit adder. In our case, three adders are considered in the final stage – RCA (denoted as WTM1), CSA (WTM2) and carry-look-ahead adder (WTM3). The latter variant offers the lowest delay but occupies a substantial area on a chip.

In total, six different multiplier architectures were described in Verilog language and synthesised using Synopsys DC and 45 nm technology. For each architecture, two Verilog netlists were generated. In the first case, the gate-level description was employed where all high-level building blocks such as half adders, full adders and multiplexers were implemented using generic two-input gates. In the second case, the structural description was

utilised. In this case, all the cells available in the chosen technology could be utilised in the description of the multipliers.

The parameters of the synthesised multipliers such as the power consumption, area on a chip or delay are summarised in Table 1. In addition to that, parameters of the initial netlists (i.e. specification) such as the number of cells (i.e. the number of components, the verilog netlist consists of), the number of full adders (column FAs), estimated power consumption P_{total} calculated using (2) and normalised power consumption P_{norm} are provided.

The results of the synthesis suggest that the Synopsys DC integrates a powerful optimisation engine that is able to significantly improve the initial design parameters. This is noticeable especially if we compare the estimated power consumption with the power consumption of the final implementation. In all cases, the power consumption was substantially improved. Despite a visible difference between the estimated and real power consumption and considering the fact that the inaccuracy of the power estimation tool is believed to be around 10%, the power consumption increases proportionally with the increasing fitness value which is important for our evolutionary optimisation process. Let us notice that 12-bit CSAM1 occupies a smaller area on a chip than RCAM at 45 nm which is quite surprising since RCAM is generally known as the most compact multiplier architecture.

Interestingly, the synthesis tool is quite capable in discovering and recovering the full adders from the gate-level netlists. The number of full adders in the implemented circuits is nearly the same independently on the chosen representation. This suggests that it does not matter whether we conduct the evolution at the level of gates or at the level of more complex building blocks such as full and half adders as the synthesis tool produces quite similar results in both cases.

3.1 Impact of the chosen representation on the efficiency of evolutionary design process

As evident from parameters in Table 1, the netlists containing only standard gates require a substantially higher number of components compared to the netlists composed of cells. For example, 768 gates are required to represent RCAM by means of standard gates. The same design can be implemented using 276 components only provided that half and full adders can be employed. Since the netlists are used to create the initial population, the higher number of components implies long chromosomes and more complex search spaces. In this particular case, the chromosome size increased by >64% when going down to the gate-level representation.

The chromosome length does not impact only the size of the search space but also the process of evaluation and consequently the scalability of the evaluation. The problem is that more nodes may be active because more nodes are available. As a consequence of that, more time is needed to evaluate fitness and quality of a candidate circuit.

In order to evaluate the impact of the chosen representation to the efficiency of the evolutionary design process, we will investigate two different parameters – the average size of a candidate solution and the convergence rate. The average number of active nodes impacts the speed of the fitness evaluation. In addition to that and when investigated at the end of a long-term evolutionary run, it also reflects the quality of discovered approximations because it is natural to expect that compact solutions typically result in a better power consumption.

The average size of a candidate solution can be measured directly as the number of active nodes or indirectly as the average time required to evaluate a single candidate solution. The problem of the first approach is that the average number of active CGP nodes does not reflect varying simulation complexity of each node. Hence, we adopted the latter approach that enables not only to fairly compare both representations but also to estimate practical limits of the evolutionary design process. In fact, it means that we consider the average size of candidate circuits expressed in terms of two-input equivalent gates. This is caused by the construction of the circuit simulator that is used to determine the fitness (the complex cells are replaced with two-input logic gates).

The average time required to evaluate a candidate solution is shown in Fig. 3. Since the size of a candidate solution varies with the error level (the approximate circuits with higher error level typically consist of a lower number of components), the results are reported for each error level separately. Sixty independent evolutionary runs (10 per each seed) were executed for each error level and each representation. The results of these runs are summarised by means of a single boxplot item. Each run was executed for 16,200 s to guarantee statistical significance. As evident, the results confirm our expectation related to the size of candidate solutions and error level. The average size of candidate solutions decreases nearly linearly with the increasing error level independently on the used representation. This trend is noticeable especially in the middle of the range because the error levels are on a logarithmic scale. On average, 224 ms (167 ms) is required to evaluate a candidate solution represented using standard gates (cells). The evaluation time decreased >2.3 times (1.9 for standard cells) when we increased the allowed error from 0.02 to 20%. It suggests that solutions consisting of approximately half of the equivalent gates were produced for 20% error.

If we compare the results for both representations, we can see that the representation based on standard cells leads to a more efficient design process. The time of the evaluation was reduced noticeably. As a consequence of that, more generations can be executed, and more compact solutions can be discovered. Considering the average results, more than four (standard gates) and nearly six (standard cells) candidate solutions can be evaluated per second. It means that we can improve the scalability by a factor ranging from 1.1 (higher errors) to 1.5 (lower errors) just by

Table 1 Parameters of six different 12-bit multipliers described at the level of gates (upper part) and standard cells (bottom part). The estimated power P_{total} as well as power of the implemented circuit is given in mW, area on a chip in μm^2 and delay in ns

Multiplier arch.	Specification					Implemented circuit				
	Repr.	No. instances	No. FAs	P_{total}	P_{norm}	No. cells	No. FAs	Area	Delay	Power
RCAM	gates	768	0	1.60	1.00	433	110	1679	3.22	1.23
CSAM1	gates	768	0	1.61	1.01	369	118	1595	2.28	1.18
CSAM2	gates	831	0	1.73	1.08	377	118	1608	2.33	1.19
WTM1	gates	809	0	1.66	1.04	394	115	1655	2.07	1.22
WTM2	gates	966	0	1.90	1.19	424	113	1708	2.39	1.23
WTM3	gates	1269	0	1.88	1.17	562	103	1961	1.68	1.39
RCAM	cells	276	120	1.27	1.00	424	110	1666	3.40	1.20
CSAM1	cells	276	120	1.28	1.01	374	118	1605	2.28	1.16
CSAM2	cells	296	123	1.37	1.08	393	117	1639	2.25	1.19
WTM1	cells	313	119	1.31	1.04	414	113	1687	2.12	1.21
WTM2	cells	362	129	1.48	1.17	463	111	1767	2.06	1.28
WTM3	cells	824	102	1.52	1.20	532	104	1889	1.77	1.33

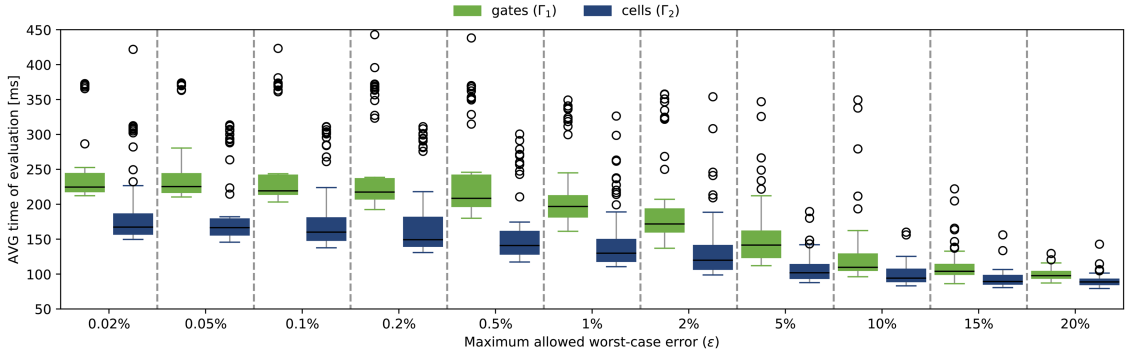

Fig. 3 *Average time required to evaluate a candidate solution representing an approximate 12-bit multiplier*

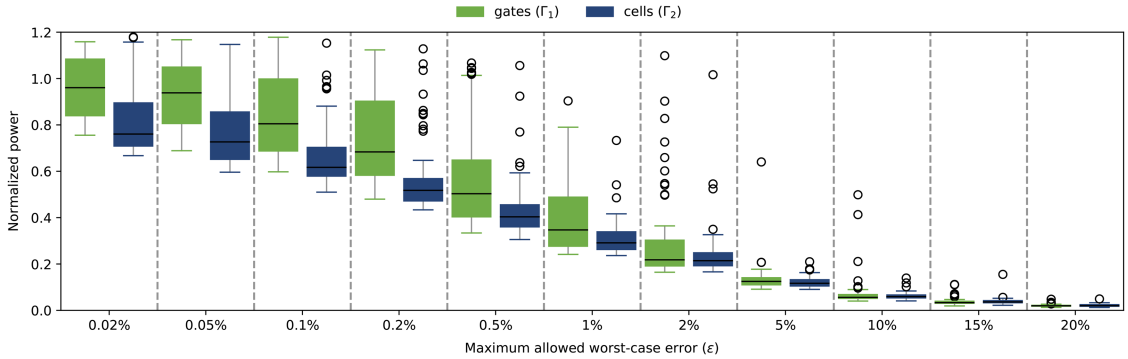

Fig. 4 *Normalised power consumption P_{norm} (the lower value the better result) of discovered 12-bit approximate multipliers*

changing the representation to standard cells. Looking and the boxplots, we can see that there is only negligible difference in the spread between both representations.

The representation impacts the speed of the fitness evaluation but does not necessarily mean that the evolution will produce better (i.e. more power efficient) solutions. Hence we analysed the power consumption of the discovered solutions. The results are shown in Fig. 4 where we plot the boxplots for normalised power (fitness value). We can observe the same trend as in Fig. 3. The power consumption decreases with the increasing error level independently on the chosen representation. In addition to that, a large improvement in the power efficiency can be seen in case of standard cell representation. This is noticeable especially for lower error levels. The cell-based representation is able to discover more efficient implementations. For example in the case of 0.2% error, the average power of a multiplier represented by standard cells equal to the power of best multiplier represented by standard gates. Starting at 5% error, there is no significant difference between parameters of the discovered results. This is caused mainly by the fact that the discovered approximate multipliers consist of an extremely small number of components. For example, approximate multipliers exhibiting 5% error consist of 6–14 times fewer components on average compared to original implementation. More than 64% of the components constitute simple gates. If we compare the boxplots, we can see that not only the extreme but even the spread is improved in the case of the cell-based representation.

The positive effect of the cell-based representation on the evolutionary design process is also evident on the convergence curves that are shown in Fig. 5. The evolutionary design process converges faster to the desired solutions not only from the point of view of absolute time but also when the number of generations is considered. The difference in convergence rate is largest for 0.2% error and gradually disappears with increasing error. This behaviour corresponds with our conclusions stated in the previous paragraph. For 2% error, the cell-based representation reaches the inflection point around a 1000th generation. When we employ a standard gate-level representation, we require more than three times higher number of generations to achieve the same results. This number corresponds to more than four times longer runtime. Nearly, the same convergence is achieved when we increase the error by one order. On contrary, when we decrease error to 0.2%, the difference is substantial. Not only that the gate-level representation converges slowly but also the time of evaluation is larger. Expressed in terms of runtime it means that after >270 min of evolution we obtained solutions whose quality is comparable with candidate solutions represented using cells that were generated in 20th minute of evolution.

A more detailed analysis is provided in Table 2 where we calculated the computational effort required to design an approximate multiplier exhibiting a required power reduction. The table shows the mean number of generations that have to be evaluated to obtain a multiplier satisfying the required criteria. In addition to that, the percentage of the evolutionary runs discovering the required circuit is given. The averages are determined from 120 independent evolutionary runs (60 for each representation). Note that not all combinations are viable as the maximum possible power reduction decreases with decreasing error. The power reduction is determined using P_{norm}.

As evident, the cell-based representation leads to much lower computational complexity than the gate-level representation. In average, less than half generations are required to obtain the same results. To give one example, we need to evaluate at least 455 generations in average to obtain an approximate multiplier with error below 0.2% that consumes 35% less power compared to the accurate RCAM multiplier. For this particular case, 50 out of 60 evolutionary runs successfully discovered such a multiplier. This corresponds to success rate 83%. For error levels above 10%, we can see that practically all runs were able to discover a multiplier having power consumption reduced by at least 95%. The mean number of generations for 95% power reduction suggests that evolution of the multipliers with a high error rate is a relatively easy task. Even the evolution of sub-optimal multipliers is easy. Every evolutionary run was able to discover an approximate multiplier exhibiting at least 20% power reduction for $\varepsilon \geq 1\%$. Less than one hundred generations were required in average. On the other hand, we can see that the computational complexity increases as the required power reduction is approaching the maximum possible reduction for a given error. More than one thousand (2000 for gate-level representation) generations are typically required to achieve the best possible power reduction.

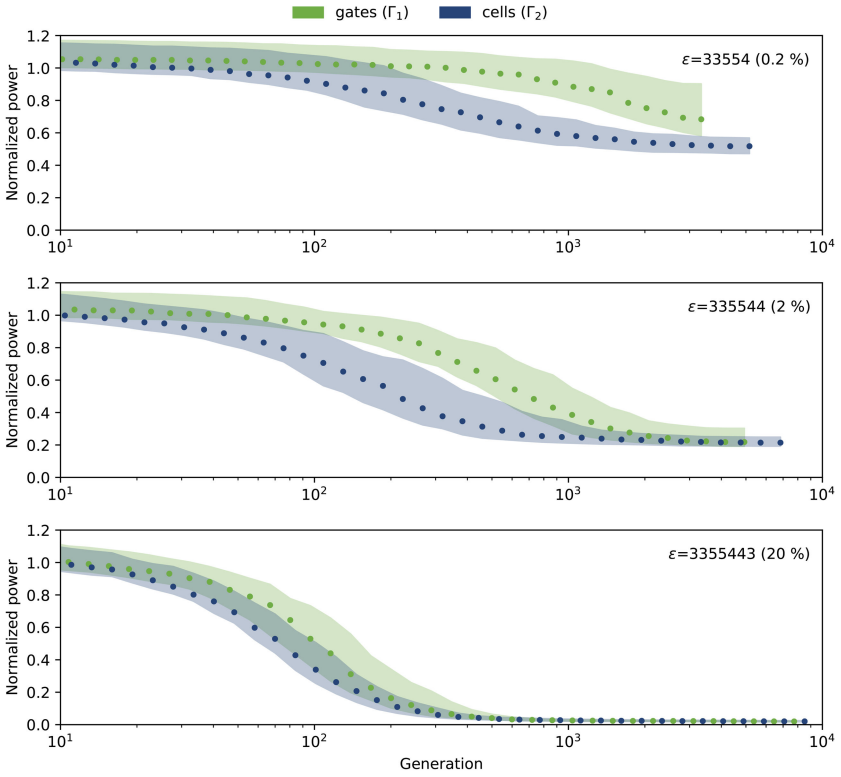

Fig. 5 *Convergence curves for evolutionary approximation of 12-bit multipliers for three error levels and both representations. For each combination, the median value (dotted line), the lower bound and upper bound of the interquartile range are calculated using 60 evolutionary runs*

Table 2 Performance of gate-level and cell-based representation expressed as the mean number of generations that have to be evaluated to obtain a 12-bit approximate multiplier exhibiting a required power reduction. The percentage of evolutionary runs that discovered a circuit satisfying the given parameters is shown after slash symbol (100% means that all 60 runs executed for each target error and each representation successfully produced such a circuit). The better results are printed in bold. In each generation 24 candidate solutions were generated and evaluated

ε	Repr.	Required power reduction					
		20%	35%	50%	65%	80%	95%
0.02	gates	2516/17	—	—	—	—	—
	cells	**935/78**	—	—	—	—	—
0.05	gates	2232/38	—	—	—	—	—
	cells	**688/75**	**2079/35**	—	—	—	—
0.1	gates	1576/58	2697/13	—	—	—	—
	cells	**533/92**	**1073/75**	—	—	—	—
0.2	gates	1045/72	1934/53	2948/8	—	—	—
	cells	**340/95**	**455/83**	**1566/62**	—	—	—
0.5	gates	622/82	1159/78	1932/62	2902/7	—	—
	cells	**207/98**	**382/97**	**669/88**	**1881/27**	—	—
1	gates	509/100	884/97	1283/87	1803/60	—	—
	cells	**138/100**	**271/100**	**403/98**	**809/93**	—	—
2	gates	346/98	552/95	743/90	1107/83	2046/43	—
	cells	**107/98**	**194/98**	**336/98**	**501/95**	**1443/52**	—
5	gates	156/100	273/100	353/98	513/98	890/98	—
	cells	**52/100**	**93/100**	**143/100**	**216/100**	**371/100**	—
10	gates	110/100	185/100	259/100	288/97	447/97	**2810/38**
	cells	**42/100**	**75/100**	**111/100**	**163/100**	**271/100**	2875/25
15	gates	87/100	142/100	206/100	282/100	398/100	824/93
	cells	**36/100**	**63/100**	**93/100**	**134/100**	**210/100**	**768/97**
20	gates	51/100	85/100	117/100	162/100	231/100	536/100
	cells	**42/100**	**73/100**	**103/100**	**139/100**	**202/100**	**473/100**

3.2 Results of synthesis

For each evolved solution, we created a Verilog netlist consisting of the same components as encoded in the corresponding chromosome. The netlists were then implemented in 45 nm technology (https://www.eda.ncsu.edu/) using Synopsys Design Compiler Ultra. The synthesis effort was set to high. Apart from an optimised netlist, the synthesis tool reports energy consumption, delay and area on a chip under iso-speed conditions.

The power consumption of the evolved approximate circuits synthesised using synthesis tool is shown in Fig. 6. When we compare the results with the estimated power consumption shown

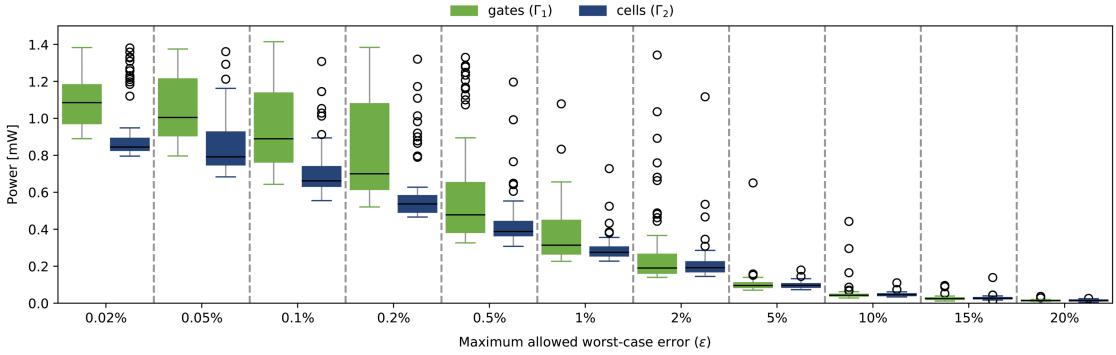

Fig. 6 *Power consumption of the synthesised 12-bit evolved approximate multipliers*

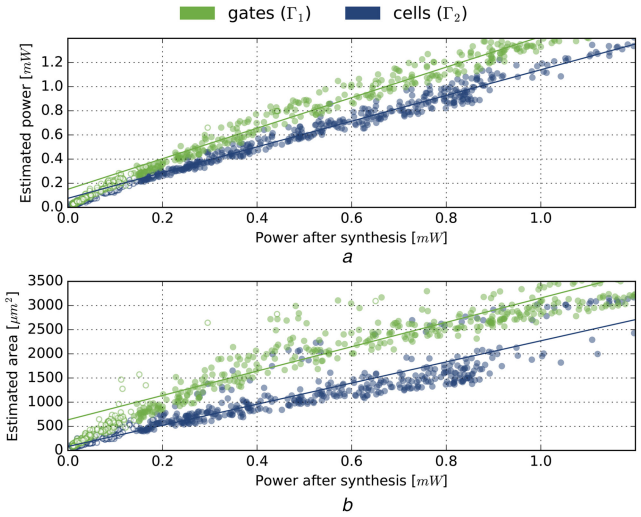

Fig. 7 *Estimated power consumption and area of 12-bit approximate multipliers represented in evolution using*
(a) Standards gates and ***(b)*** Standard cells vs. power after synthesis

in Fig. 4, we can see that both boxplots contain very similar data at least when we consider relative relations and spread. Solutions represented using the standard cells exhibit better energy efficiency compared to the solutions described using standard gates. The absolute values are different, but this was expected as discussed earlier because the discovered netlists may be modified during synthesis by the optimisation engine. In fact, the power consumption of the synthesised and implemented multipliers is typically lower than the power consumption before synthesis and the difference increases with the increasing circuit complexity. This is evident from the plots shown in Fig. 7 where we analysed the degree of correlation between estimated power and power after synthesis. There is a nearly linear relationship. In order to measure the linear relationship, we employed the Pearson correlation coefficient whose value varies between −1 and +1. The correlations of −1 or +1 imply an exact linear relationship, while zero implies no correlation. The correlation coefficient is slightly better than 0.99, in particular it equals 0.994 for standard cells and 0.992 for standard gates. The high positive value implies that as estimated power increases, so does the power of the synthesised multiplier. If we look at the output of simple linear regressions, we can see that the points concentrate around two trend lines having their slope slightly larger than 1. It means that the synthesis tool was able to improve the power in all cases and we actually overestimated the final value. For standard cells, the power of the multipliers was reduced during synthesis by a factor of 1.1 on average. This factor increases to 1.4 when standard gates are employed.

It is necessary to realise, however, that the ability to quantify the relative dependencies of the design power consumption is much more important than the ability to capture the absolute values since the search is driven by the comparison of parental and candidate solution's fitnesses. For comparison of the values it is only needed to achieve a high-fidelity value which can be calculated as follows. Let $R = \{R_0, \ldots, R_n\}$ be a set of n reference values and $E = \{E_0, \ldots, E_n\}$ be a set of n estimated values. Let T be a set of n circuits that were used to calculate R_i and E_i. The fidelity describing the quality of the estimation with respect to its ability to quantify relative dependencies of the tuples reference/estimation values is defined as

$$\text{Fidelity}_{\%} = 100\frac{2}{n(n-1)}\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\mu_{ij}, \tag{6}$$

where μ_{ij} is determined as

$$\mu_{ij} = \begin{cases} 1 & \text{if } R_i > R_j \wedge E_i > E_j \text{ or } R_i = R_j \wedge E_i = E_j \\ & \quad \text{or } R_i < R_j \wedge E_i < E_j \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

According to the analysis, the fidelity calculated using the set of 660 evolved approximate multipliers represented using standard cells is 97.21%. For the second set of 660 multipliers encoded using standard gates, the fidelity is nearly similar – 97.25%. It means that the estimated values correlate with the results after synthesis in 422,814 (422,960 for the second set) out of 434,940 inspected cases.

The second plot in Fig. 7 shows the dependency between estimated area and power of the synthesised multipliers. The obtained results suggest that worse results will be obtained by considering the area in the fitness function only and demonstrates the superiority of the fitness function based on power estimation. The fidelity dropped to 93.7% (95.3% for the second set), the Pearson correlation coefficient decreased to 0.95 (0.97) and even the slope and distance of the regression lines increased.

In order to better understand the behaviour of both investigated representations for errors higher than 2%, we separately analysed the evolved netlists and corresponding netlists after synthesis and determined the number of cells and the corresponding area on a chip. We divided the cells utilised in the netlists to seven categories – buffers (BUF), inverters (INV), common two-input gates (F2X1), more complex cells with three and four inputs (F3X1, F4X1), half adders (HA) and full adders (FA). The area on a chip occupied by these cells before and after synthesis is shown in Fig. 8. For each error level, we plot the area of ten multipliers exhibiting the best power as reported by the synthesis tool.

Let us discuss the results for the gate-level representation. The evolved netlists consist mainly of two-input gates. Few buffers and inverters are employed only. The evolution is not directly forced to use the buffers and inverters but we assume that the evolution introduced them to improve the power since they exhibit a better output capacity and we consider the capacity of each gate in (2). If we look at the netlists optimised by the chosen synthesis tool we can see that the synthesis tool is able to identify a large number of complex cells such as full adders and three-input cells. Few instances of half adders and four-input cells are utilised only. Interestingly, the number of inverters increased substantially after synthesis. We inspected the netlists and identified that buffers and inverters are used mainly to improve the timing.

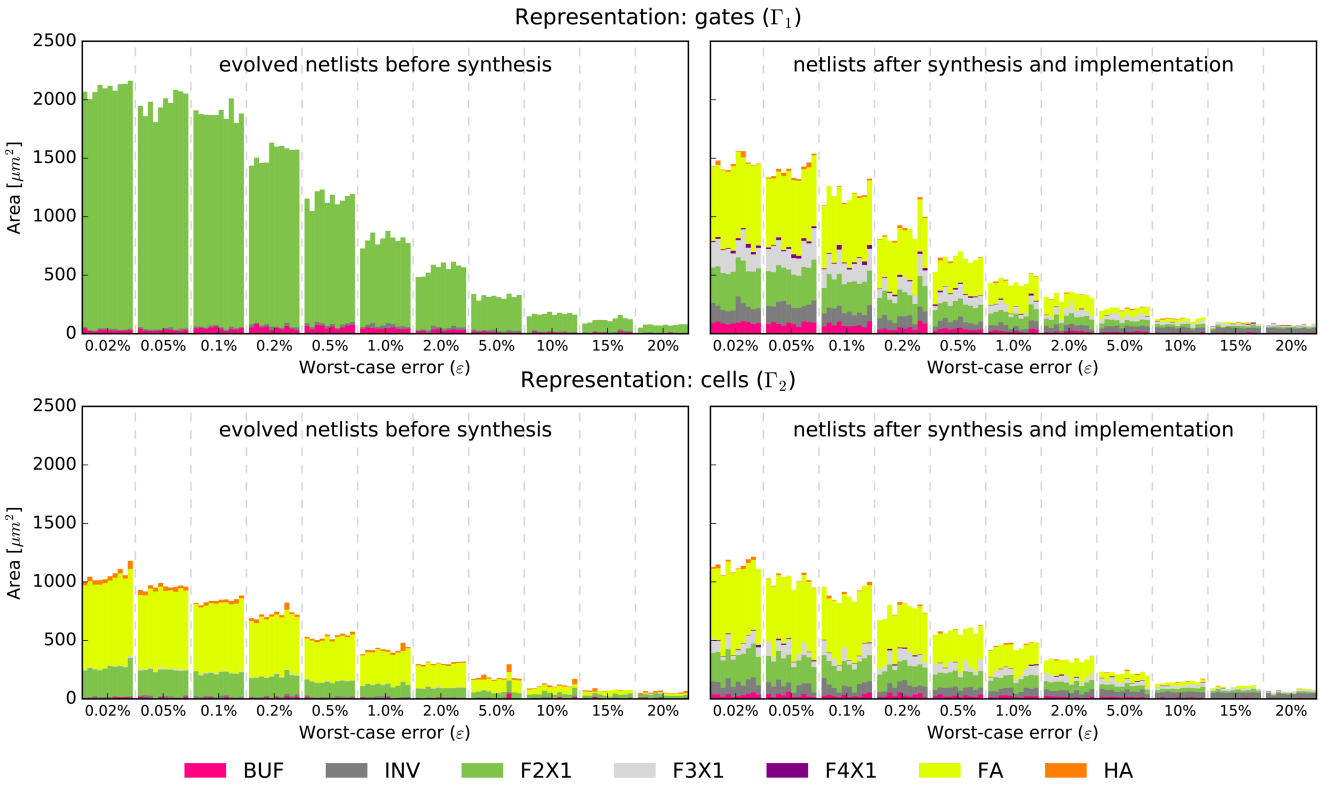

Fig. 8 *Area on a chip of evolved 12-bit approximate multipliers occupied by various types of standard cells before and after synthesis. For each error level, area of ten discovered fittest solutions is analysed separately for gate-level (top row) and cell-based (bottom row) representation*

As evident, full adder represents a key component for the construction of power-efficient multipliers. On the other hand, we can see that the number of full adders decreases with increasing error. Only one or two instances are utilised in approximate multipliers with an error higher than 5%. This observation is valid even for cell-based representation which explains why evolution at the level of gates and cells provide the same results.

As we already discussed, the synthesis tool was able to improve the energy consumption of the initial netlists. In addition to that, we can see that even area on a chip was reduced substantially. The area of approximate multipliers exhibiting 0.02% error, for example, was reduced from >2000 to 1500 μm^2. On the average, the area was improved by 32%.

In the case of the cell-based representation, the area on a chip after synthesis remains practically the same as before synthesis. We can also observe that the number of full adders in the evolved multipliers remains preserved. It suggests that the evolution produces highly optimised solutions that are hard to improve by the synthesis. This is also evident if we compare the results after synthesis for both representations. Multipliers produced from the cell-based netlists are more compact.

There are only a few half adders in the evolved solutions. Surprisingly, they practically disappeared after the synthesis. In comparison with the gate-level evolution, the buffers and inverters are utilised only rarely. This leads to a conclusion that it does not offer any advantage to consider complex three-input and four-input building blocks during the evolution because they are not utilised at all. Even the half adder seems to be not important. The essential thing is to include a full adder in the set of possible functions.

3.3 Evaluation of the proposed multipliers

The approximate circuit design problem is naturally a multi-objective optimisation problem in which the accuracy and other circuit parameters are conflicting design objectives. In our approach, only the power consumption was intentionally considered in the fitness function. It is thus fair to evaluate also the other circuit parameters. Hence, we took all synthesised approximate multipliers, determined the circuit parameters (worst-case error, power, area, delay) and identified the points on the Pareto front. In order to determine the Pareto dominant solutions, the Pareto-dominance relation was employed [22]. The resulting Pareto set projected into three 2D plots is shown in the upper part of Fig. 9. The bottom part of Fig. 9 shows the parameters of the chosen solutions as a function of mean error distance which was not directly optimised. As evident, the cell-based representation lead to substantially better solutions compared to the gate-level representation. A rich library of various implementations was obtained. When we ignored the solutions that exhibit nearly similar parameters, we obtained 17 Pareto-dominant solutions for the gate-level approach and 27 solutions for the cell-based approach. Note that two parameters were considered similar when their difference was <0.1 for the error (in log scale), power consumption and delay, and <100 for the area.

Fig. 9 includes also a comparison with the state-of-the-art approaches. For this purpose, we implemented two approximate architectures that are believed to provide the best results according to the latest review [19]. In particular, we implemented the truncated CSA array multiplier and the broken-array multiplier. Truncation, sometimes also referred to as a bit-width reduction, represents a straightforward approach to perform approximation. The key idea is to remove the least significant bits of the input operands and use a smaller multiplier instead of an accurate one. Similarly to the truncated multiplier, the broken-array multiplier removes some of the carry-save adders in an array multiplier, however the removed adders are determined by two parameters. In addition to that, we reimplemented one of the first approximate multipliers published by Kulkarni *et al.* in 2011 [23]. Kulkarni's multiplier employs a common modular approach and constructs multipliers of higher bit-widths using small manually designed 2-bit approximate multiplier. For more details related to the implemented architectures, we kindly refer the reader to the review. Surprisingly, the multipliers constructed using our method provide the best results if we consider cell-based representation only. Despite the fact that only the worst-case error and power were considered during the evolution, the obtained multipliers perform very well even under the mean error distance. There are only some minor discrepancies in the area– e_{wst} and area– e_{mae} plots. On the other hand, Kulkarni's modular approach results in an extremely inefficient 12-bit approximate multiplier. More than eight times better power consumption can be achieved for the same mean error.

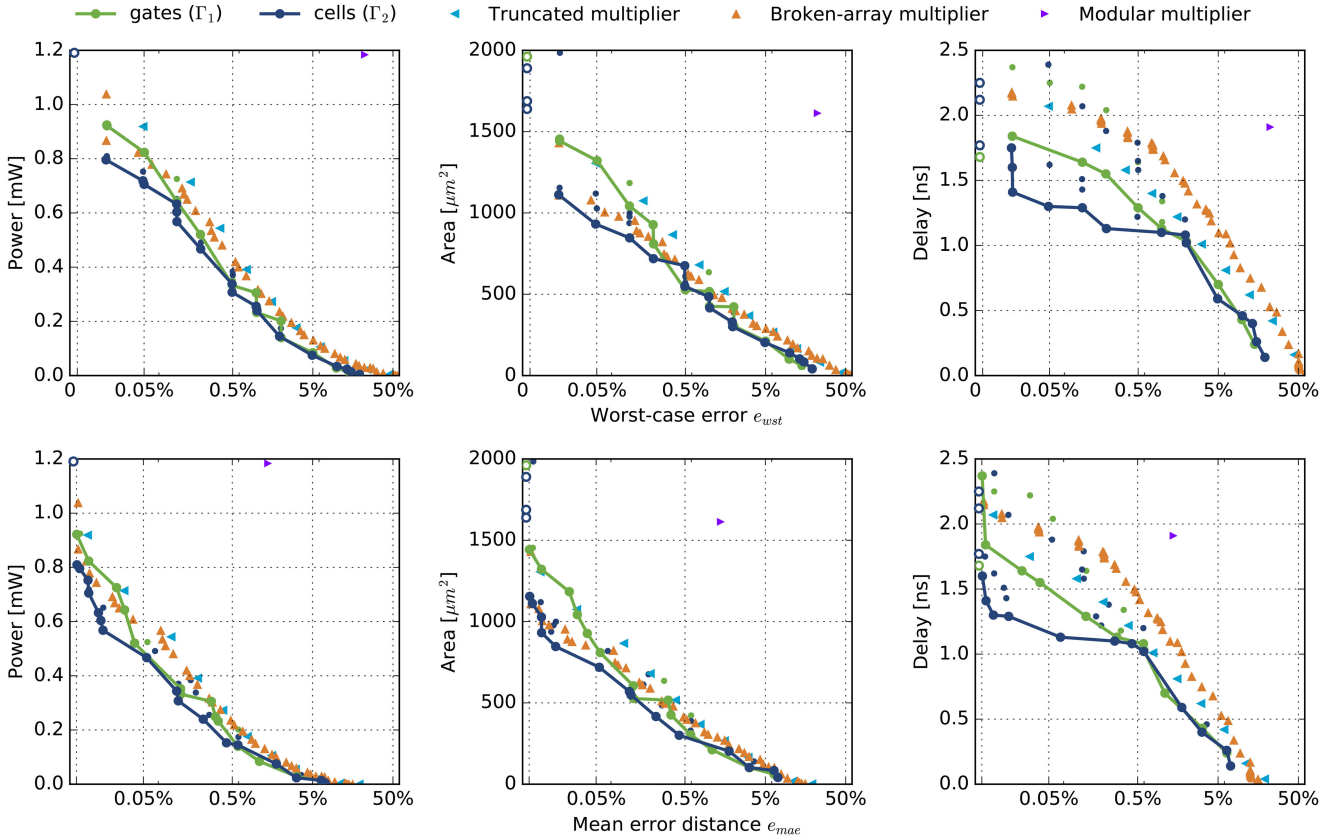

Fig. 9 *Parameters of the evolved approximate 12-bit multipliers occupying the global Pareto front (worst-case error, power, area, and delay considered) as well as the state-of-the-art approximate multipliers. Parameters of the accurate multipliers are shown using empty circles. Note that the x-axis has logarithmic scale. Solid line shows local Pareto fronts in which the remaining two objectives are ignored*

The situation is even worse for the worst-case error. Kulkarni's multiplier exhibits >50 times worse power consumption compared to the other considered multipliers having the same worst-case error.

4 Conclusion

In this paper, we introduced and evaluated a CGP-based evolutionary method for the design of energy-efficient approximate circuits. In contrast to evolutionary circuit design approaches available in the literature, we proposed to conduct the evolution at a higher level. In particular, we employed standard cells routinely used in design automation as building blocks for implementing digital circuits. This decision required to extend common CGP to support nodes with multiple outputs. In addition to that, a more powerful 256-bit simulator and power-estimation engine was employed in the fitness function. These changes gave rise to a more efficient design method which was able to handle more complex problem instances.

It was demonstrated that the proposed cell-based representation enabled not only to reduce the time of the evaluation but also to improve the convergence of the evolutionary design process. Interestingly, the detailed analysis suggests that including of a full adder in the set of possible functions is the key feature. The experiment related to the evolutionary design of 12-bit approximate multipliers revealed that the candidate circuits produced in the 20th minute of the evolution exhibit practically the same quality as the candidate solutions generated in the 270th minute of the evolution conducted at the level of common gates. Considering the fact that the results were achieved on a common 2.4 GHz Xeon CPU using a single thread application, the proposed method can be easily integrated to a standard design flow. As today's CPUs typically contain many cores, we can run several independent evolutionary runs and obtain more different results at the same time.

According to the literature, 8-bit multiplier represents the most complex instance with precisely determined error parameters that have been evolutionarily approximated. In this paper, we were able to push the limits and discover several non-dominated 12-bit multipliers exhibiting different circuit parameters. As shown, the discovered multipliers outperform the state-of-the-art multipliers available in the literature.

Despite this optimism, it is clear that exhaustive simulation cannot be applicable for 16-bit multipliers, where 2^{32} input test vectors have to be evaluated to guarantee the worst-case error. Similarly to the methods originating from the hardware community, however, we can cope with this issue by relaxing the requirement for strict worst-case error guarantee. It means that only a subset of all possible input combinations is used to determine the error parameters. By relaxing this requirement, we can approximate even larger problem instances. Even though worst-case error was employed only, the proposed approach can be applied for any error criteria.

5 Acknowledgments

This work was supported by the Czech Science Foundation grant no. GA16-17538S and by the Brno University of Technology project FIT/FSI-J-17-4294.

6 References

[1] Mittal, S.: 'A survey of techniques for approximate computing', *ACM Comput. Surv.*, 2016, **48**, (4), pp. 62:1–62:33

[2] Chippa, V.K., Chakradhar, S.T., Roy, K., *et al.*: 'Analysis and characterization of inherent application resilience for approximate computing'. The 50th Annual Design Automation Conf., (DAC'13, 2013), Austin, TX, USA, 2013, pp. 1–9

[3] Xu, Q., Mytkowicz, T., Kim, N.S.: 'Approximate computing: a survey', *IEEE Des. Test*, 2016, **33**, (1), pp. 8–22

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

[5] Venkataramani, S., Roy, K., Raghunathan, A.: 'Substitute-and-simplify: a unified design paradigm for approximate and quality configurable circuits'. Proc. of DATE'13 (European Design Automation Association), Grenoble, France, 2013, pp. 1367–1372

[6] Nepal, K., Li, Y., Bahar, R.I., *et al.*: 'Abacus: a technique for automated behavioral synthesis of approximate computing circuits'. Proc. of the Conf.

on Design, Automation and Test in Europe (DATE '14), Dresden, Germany, 2014, pp. 1–6
[7] Sekanina, L., Vasicek, Z.: 'Approximate circuit design by means of evolvable hardware'. IEEE Int. Conf. on Evolvable Systems (ICES), Singapore, 2013, pp. 21–28
[8] Vasicek, Z., Sekanina, L.: 'Evolutionary approach to approximate digital circuits design', *IEEE Trans. Evol. Comput.*, 2015, **19**, (3), pp. 432–444
[9] Mrazek, V., Sarwar, S.S., Sekanina, L., *et al.*: 'Design of power-efficient approximate multipliers for approximate artificial neural networks'. Proc. of ICCAD'16, Austin, TX, USA, 2016, pp. 81:1–81:7
[10] Hrbacek, R., Mrazek, V., Vasicek, Z.: 'Automatic design of approximate circuits by means of multi-objective evolutionary algorithms'. Proc. of DTIS'16, Istanbul, Turkey, 2016, pp. 239–244
[11] Miller, J.F.: '*Cartesian genetic programming*' (Springer-Verlag, New York, NY, USA, 2011)
[12] Weste, N.H., Harris, D.: '*CMOS VLSI design: a circuits and systems perspective*' (Addison-Wesley, Boston, USA, 2005, 3rd edn.)
[13] Miller, J.F., Thomson, P., Fogarty, T.: '*Designing electronic circuits using evolutionary algorithms. Arithmetic circuits: a case study*' (Wiley, 1998), pp. 105–131
[14] Clegg, J., Walker, J.A., Miller, J.F.: 'A new crossover technique for Cartesian genetic programming'. Proc. of The Genetic and Evolutionary Computation Conf. (GECCO), London, 2007
[15] Slany, K., Sekanina, L.: 'Fitness landscape analysis and image filter evolution using functional-level CGP'. Proc. of European Conf. on Genetic Programming, Valencia, Spain, 2007 (LNCS, **4445**), pp. 311–320
[16] Wiltgen, A., Escobar, K., Reis, A., *et al.*: 'Power consumption analysis in static CMOS gates'. 2013 26th Symp. on Integrated Circuits and Systems Design (SBCCI), Curitiba, Brazil, 2013, pp. 1–6
[17] Monteiro, J., Devadas, S., Ghosh, A., *et al.*: 'Estimation of average switching activity in combinational logic circuits using symbolic simulation', *IEEE Trans. Computer-Aided Des. Integr. Circuits Syst.*, 1997, **16**, (1), pp. 121–127
[18] Jiang, H., Han, J., Lombardi, F.: 'A comparative review and evaluation of approximate adders'. Proc. of GLVLSI'15, Pittsburgh, PA, USA, 2015, pp. 343–348
[19] Jiang, H., Liu, C., Maheshwari, N., *et al.*: 'A comparative evaluation of approximate multipliers'. Int. Symp. Nanoscale Architectures, Beijing, China, 2016, pp. 191–196
[20] Vasicek, Z., Slany, K.: 'Efficient phenotype evaluation in cartesian genetic programming'. Proc. of the 15th European Conf. on Genetic Programming, Malaga, Spain, 2012 (LNCS, **7244**), pp. 266–278
[21] Hrbacek, R.: 'Parallel multi-objective evolutionary design of approximate circuits'. GECCO '15 Proc. of the 2015 Conf. on Genetic and Evolutionary Computation, Madrid, Spain, 2015, pp. 687–694
[22] Deb, K.: '*Multi-objective optimization using evolutionary algorithms*' (Wiley, New York, NY, USA, 2001)
[23] Kulkarni, P., Gupta, P., Ercegovac, M.: 'Trading accuracy for power with an underdesigned multiplier architecture'. 2011 24th Int. Conf. on Very-Large-Scale Integration (VLSI) Design, Chennai, India, 2011, pp. 346–351