Evolutionary Functional Approximation of Circuits Implemented into FPGAs

Zdenek Vasicek, Vojtech Mrazek and Lukas Sekanina
Brno University of Technology, Faculty of Information Technology, IT4Innovations Centre of Excellence
Brno, Czech Republic
Email: vasicek@fit.vutbr.cz, imrazek@fit.vutbr.cz, sekanina@fit.vutbr.cz

***Abstract*—In many applications it is acceptable to allow a small error in the result if significant improvements are obtained in terms of performance, area or energy efficiency. Exploiting this principle is particularly important for FPGA-based solutions that are inherently subject to many resources-oriented constraints. This paper devises an automated method that enables to approximate circuit components which are often implemented in multiple instances in FPGA-based accelerators. The approximation process starts with a fully functional gate-level circuit, which is approximated by means of Cartesian Genetic Programming reflecting the error metric and constraints formulated by the user. The evolved circuits are then implemented for a particular FPGA by common FPGA synthesis and optimization tools. It is shown using five different FPGA tools, that the approximations obtained by CGP working at the gate level are preserved at the level look-up tables of FPGAs. The proposed method is evaluated in the task of 8-bit adder, 8-bit multiplier, 9-input median and 25-input median approximation.**

I. Introduction

Thompson's evolutionary circuit design conducted in a field programmable gate array (FPGA) in the middle nineties proved that it is possible to evolve digital circuits directly at the level of configuration bit stream in a reconfigurable chip [1]. This approach has been adopted for other reconfigurable digital well as analogue reconfigurable platforms (such as [2], [3]). Thompson's approach can be classified as the intrinsic and unconstrained evolution. By the *intrinsic evolution* we mean that all candidate circuits are evaluated directly in a chip. The *unconstrained evolution* means that no restrictions are posed on the parts of the chip where the reconfiguration is carried out. In the case that only very specific parts of the reconfigurable chip (such as look-up table (LUT) contents of the FPGA) can be modified, the approach is referred to as *constrained evolution*. However, most of the research in the evolvable hardware field is performed at the level of *extrinsic evolution*, where candidate circuits are evaluated using a circuit simulator. The most innovative and efficient designs that truly exploit properties of the reconfigurable platform and that are optimized for a given environment have been evolved in the intrinsic evolution scenario. These designs, however, often show unwanted properties such as various reliability and robustness issues. In order to eliminate them, some constraints are always introduced in practice.

It is assumed in this paper that a mainstream FPGA chip is the target platform. The objective is to create well-optimized approximate circuit implementations intended for small energy-efficient FPGA-based embedded systems. Approximate circuits, which have been developed in the field of approximate computing [4], are characterized by significantly improved parameters (such as power consumption, area and delay) for the cost of some application-specific acceptable error with respect to their fully functional versions. Evolutionary approximation is one of the methods developed for an efficient circuit approximation [5].

As performing the unconstrained evolution directly at the level of configuration bit stream is currently unsafe and actually almost impossible (because format of the configuration bit stream is not documented), one can utilize the constrained evolution in the FPGA, which consists in pre-synthesizing some parts of the approximate circuit (typically the routing of programmable components) and evolving the remaining parts (typically the LUTs contents), see, for example, the on-chip evolution of image filters using approximate elementary components [6].

In this paper, we propose to evolve approximate circuits extrinsically (i.e. candidate circuits are simulated using a software tool) and then utilize a common FPGA synthesis tool chain to safely obtain desired (i.e. approximate) circuits for a particular FPGA. It has to be noted that common FPGA synthesis tools do not directly support approximate circuit design. The approximate circuits are evolved using Cartesian genetic programming, following the method of the evolutionary functional approximation that we have developed for gate-level circuit approximation in our previous work [5], [7]. The approximation process starts with a fully functional circuit implementation, which is approximated by means of CGP reflecting the error metric and constraints formulated by the user. It is shown in the paper that circuit parameters (particularly the area reduction) obtained by CGP working at the gate level are almost perfectly preserved by common FPGA synthesis tools producing circuits consisting of 6-input LUTs. This paper extends our study devoted to the evolutionary approximation of general logic circuits [8]. In this paper we discuss approximation of 8-bit adders, 8-bit multipliers, and 9-input and 25-input median circuit. In addition to that, evolved approximate circuits are employed and evaluated in three common image processing components – image filters and edge detectors.

The rest of the paper is organized as follows. Section II briefly surveys the related research in evolutionary design

and approximate computing. Section III presents the proposed method. Results are reported in Section IV and discussed in SectionV. Conclusions are given in Section VI.

II. Related Work

A. Circuit Design and Approximation

Conventional circuit synthesis and optimization tools (such as ABC [9]) are typically constructed as deterministic systems. In order to improve their results, non-deterministic heuristic methods are introduced. For example, these heuristics can be based on local resynthesis [10], simulated annealing [11] or evolutionary algorithms [12]. Improvements in the quality of optimization have been reported, but for the cost of runtime and non-deterministic behavior of the optimization procedure. Recently developed circuit approximation tools, however, relies on non-deterministic heuristics [13], [14], [15].

This paper falls into the area of *functional approximation* which is one of the methods allowing designers to approximate circuits at the level of logic behavior. The idea is to implement a slightly different Boolean function to the original one providing that the error is acceptable and the area, power consumption and other parameters are reduced adequately. The approximations are obtained by a heuristic procedure which modifies the original, accurate circuit. Examples of systematic approximation methods that were evaluated for ASIC designs are SASIMI [13], SALSA [14] and ABACUS [15].

In the context of FPGAs, circuit approximation has been introduced and evaluated by means of the GRATER tool [16]. It uses a genetic algorithm to determine the precision of variables within an OpenCL kernel. By selectively reducing the precision, the number of parallel approximate kernels that can be mapped in the fixed area budget of an FPGA can be increased with respect to the original kernel implementations.

B. Evolutionary Approximation

Evolutionary circuit optimization and evolutionary circuit approximation are in principle identical methods, which, in fact, differ only in setting of the optimization objective and constraints. In the former case, the error of the resulting circuit is requested to be strictly smaller or equal to the error dictated by the specification. In the latter case, increasing the error can be exchanged for reducing the area, latency or power consumption. In evolutionary algorithms, the quality of candidate solutions is measured using a fitness function, which can involve one or more objectives, for example, the error, area and delay in the case of circuit approximation.

Evolutionary circuit optimization and approximation methods usually employ CGP, which is a form of genetic programming [17], [5]. The circuit error, which is a crucial optimization objective, is calculated according to the type of circuit and user requirements. In approximate circuit design methods, the arithmetic errors (such as the mean absolute error or RMSE) are often employed. In order to obtain the error for a given circuit, circuit responses are typically computed for a training data set (i.e. a subset of all possible input vectors) and compared with the requested values. However, the resulting value is not the exact arithmetic error. For less complex circuits, it is possible to apply all possible test vectors (2^n vectors for an n-input circuit) to get the exact arithmetic error. In the case of complex circuits, the exact error can be in some cases obtained by advanced formal methods [7].

As many candidate circuits are typically generated and evaluated during the evolution, the circuit simulation has to be very fast. A common solution for the gate-level CGP executed on a processor is a bit-level parallel simulation, in which several test vectors are encoded into w-bit operands and executed using bit-wise logic instructions in parallel [17]. The obtained speedup is w on a w-bit processor (assuming $2^n \geq w$). However, this approach is hard to apply for circuits based on 6-input LUTs.

It is worth to mention that a sort of approximate computing was performed by the evolvable hardware community before the current "era" of approximate computing. Thompson evolved very efficient tone discriminators in the FPGA, but they showed various reliability issues [1]. This result inspired Miller and his collaborators to introduce the concept of "evolution in materio" which enabled to design very energy efficient solutions directly in a suitable programmable material. In 1999, Miller introduced a CGP-based method for finite impulse response (FIR) filter design [18] that would be called functional gate-level approximation nowadays. Kneiper et al. traded the robustness of evolved classifiers (i.e. the classification accuracy) for the area on a chip [19].

III. Proposed Method

The objective is to minimize the area of circuits that have to be implemented into an FPGA, assuming that approximations are allowed. The proposed CGP-based approximation method, which is employed at the level of original circuits, before a common FPGA tool is executed, is described in next sections.

A. Circuit Representation

The evolution could operate either at the level of gates (components) used in the original circuits or at the level of LUTs available in the FPGA. However, there are only a few papers dealing with evolutionary circuit design at the level of LUTs (e.g. [20]). Most CGP-based approaches deal with the gate or component level utilizing two-input gates. The main reason is that the bitwise parallel simulation is not directly applicable for circuits consisting of 4- or 6-input LUTs and the circuit evaluation is then one or two orders of magnitude slower than a well-optimized gate-level simulator. Moreover, employing CGP with 6-input LUTs (each of them encoded using 64 bits) would lead to difficult search spaces and very inefficient search procedures. Hence, the proposed method operates with two-input nodes (gates) because it primarily leads to the fastest evaluation of candidate solutions and well-know types of search spaces.

A gate-level n_i-input/n_o-output circuit is represented using a directed acyclic graph which is encoded in a 1D array consisting of n_c gates. This array is internally stored using a string of integers, the so-called chromosome. The set of available

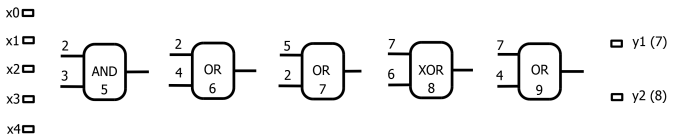

Fig. 1. Example of a circuit in CGP with parameters: $n_i = 5, n_o = 2$, $n_c = 5$, $\Gamma = \{0^{\text{and}}, 1^{\text{or}}, 2^{\text{xor}}\}$. Chromosome: 2, 3, 0; 2, 4, 1; 5, 2, 1; 7, 6, 2; 7, 4, 1; 7, 8. Gate 9 is not used. Its logic behavior is: $y_1 = (x_2$ and $x_3)$ or x_2; $y_2 = y_1$ xor $(x_2$ or $x_4)$

logic functions is denoted Γ. The primary inputs are labeled $0 \ldots n_i - 1$ and the gates are labeled $n_i, n_i+1, \ldots, n_i+n_c-1$. For each gate, three integers are included in the chromosome – two labels specifying indices where the gate inputs are connected to and a code of function in Γ. The last part of the chromosome contains n_o 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 outputs. Example is given in Fig. 1. The chromosome size is $3n_c + n_o$ genes (integers) if two-input gates are used. The main feature of this encoding is that while the size of the chromosome is constant (for a given n_i, n_o and n_c), the size of circuits represented by this chromosome is variable as some gates can remain disconnected.

B. Search method

The search method presented as Algorithm 1 follows the standard CGP approach [17]. The initial population P is seeded by the accurate circuit (p) and λ offspring circuits created by a point mutation operator modifying h genes of the parent individual p (h randomly chosen integers are replaced by randomly generated integers). In order to generate a new population, λ offspring individuals are again created by a point mutation operator. The parent is either the accurate circuit (in the first generation) or the best circuit of the previous generation (in remaining generations).

One mutation can affect either the gate function, gate input connection, or primary output connection. A mutation is called neutral if it does not affect the circuit's fitness. If a mutation hits a non-used part of the chromosome, it is detected and the circuit is not evaluated in terms of functionality because it has the same fitness (i.e. quality) as its parent. Otherwise, the error is calculated. For further details about CGP and its parameters setting, the reader is recommended to consult standard references [17].

There are two design objectives for CGP: minimizing the functionality (error) and the number of gates.

Algorithm 1: CGP

Input: CGP parameters, fitness function
Output: The highest scored individual p and its fitness

```
P ← the accurate circuit p and its λ offspring created by mutation;
EvaluatePopulation(P);
while ⟨terminating condition not satisfied⟩ do
    α ← highest-scored-individual(P);
    if fitness(α) ≥ fitness(p) then
        p ← α;
    P ← create λ offspring of p using mutation;
    EvaluatePopulation(P);
return p, fitness(p);
```

C. Fitness function

For different types of circuits, specific fitness functions have to be constructed. In the case of arithmetic circuits, it is natural to minimize the arithmetic error, for example the mean absolute error, between candidate approximations and the specification for all possible input combinations.

Generating and evaluating all possible input combinations is tractable only for small circuits because the number of all possible input vectors as well as required run-time grows exponentially with the increasing number of input signals. To overcome this problem, two approaches are used in practice. The arithmetic error can efficiently be calculated either using Binary decision diagrams (BDDs) or estimated using a subset of all possible input combinations. The latter approach is usually employed for circuits where the construction of BDDs is intractable.

In our case, we will deal with relative small arithmetic circuits with bit-widths not exceeding 8 bits. Hence, it is more efficient to employ parallel simulation and calculate the error using all possible input vectors (2^{16} for the considered 8-bit arithmetic circuits). Similarly to the BDD-based approach, the simulation-based method keeps the resulting fitness accurate, however, it is much faster. The main reason for that is the presence of instructions available in the contemporary off-the-shelf CPUs that enable to process up to 256 input vectors in parallel. When employed in fitness function, tens of microseconds (depending on the number of gates of a circuit) are required to obtain response for all 2^{16} input vectors [21], [22].

The same error metrics (i.e. the mean absolute error) can be applied to measure the quality of non-linear signal processing circuits. In this case, however, it seems to be sufficient to generate a subset of all possible input vectors that is then used to evaluate candidate approximation. It has to be emphasized, however, that evolved circuit has to be evaluated using a test data set after finishing the evolution in order to determine its behavior for unseen input vectors.

Let A be a candidate circuit represented using CGP and S an original circuit (specification). Let $\mathcal{F}_A, \mathcal{F}_S : B^{n_i} \rightarrow B^{n_o}$ be Boolean functions computed by A and S, respectively. Let $\mathbf{T}$ be a set (a subset in the case of signal processing circuits) of all possible input combinations. The mean absolute error $Error(S, A)$ between S and A can be defined as follows:

$$Error(S, A) = \frac{1}{|\mathbf{T}|} \sum_{\vec{t} \in \mathbf{T}} |N(\mathcal{F}_A(\vec{t})) - N(\mathcal{F}_S(\vec{t}))|, \quad (1)$$

where $N(\vec{x})$ represents a function $N : B^{n_i} \rightarrow \mathbb{Z}$ returning a decimal value of a binary vector $\vec{x}$. In this paper, we consider

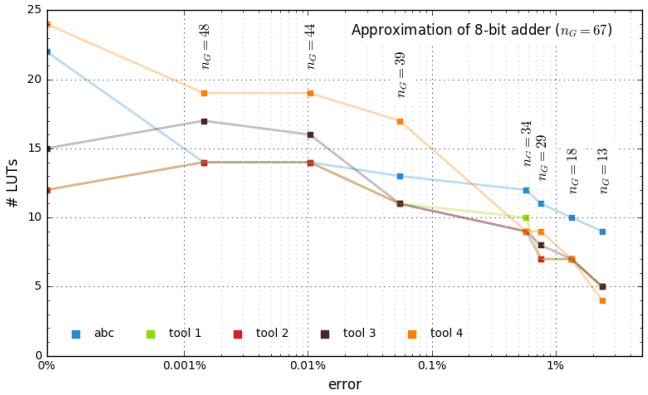

Fig. 2. The number of LUTs and number of gates (n_G) for various approximate 8-bit adders synthesized using common synthesis tools.

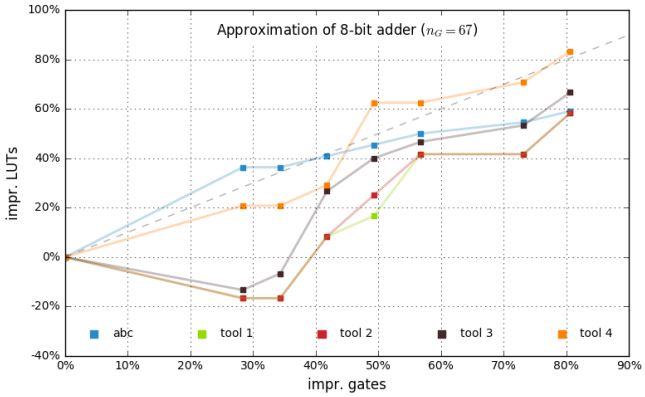

Fig. 3. The relation between improvement in the number of LUTs and in the number of gates for approximate adder

Boolean functions manipulating binary numbers encoded using the two's complement. Hence, N is a natural conversion from two's complement.

D. Evolutionary Approximation

In order to evolve circuits showing different compromises between the error and size, a single-objective CGP is executed multiple times with different parameters. Resulting solutions are displayed using a Pareto front. Two approaches have been proposed in literature.

In [7], a two-stage procedure was employed for a given error e_i. In the first stage, a given accurate circuit (S) is gradually modified by CGP to exhibit error e_i providing that a 5% difference is tolerated with respect to e_i (tolerating a small error is acceptable; otherwise the search could easily be stuck in a local extreme). In the second stage, the number of gates is minimized, assuming that the error remains within the required range.

Another way to obtain a Pareto front is to constrain the number of components or gates to g_i ($g_i < n_G$, where n_G is the number of gates needed to implement the accurate circuit) that can be used for circuit implementation. CGP is then used to minimize the error for a given g_i [5].

In this paper, the two-stage approach which can easily be embedded into the Algorithm 1 is employed. The following fitness function is utilized in the first stage:

$$fitness(A) = \begin{cases} Error(S,A) & \text{if } Error(S,A) \leq e_i \\ -1 & \text{otherwise,} \end{cases}$$

In the second stage, the circuit size is optimized. Hence, the fitness function is constructed as follows:

$$fitness(A) = -\begin{cases} |A| & \text{if } |Error(S,A) - e_i| \leq 0.05e_i \\ \infty & \text{otherwise,} \end{cases},$$

where $|A|$ denotes the number of gates employed in a candidate circuit A.

IV. Results

In order to assess the impact of CGP-based optimization and approximation on the synthesis results, two classes of circuits which differ in the size are considered: small arithmetic circuits (8-bit adders and 8-bit multipliers) and non-linear signal filtering circuits (over ten thousand gates).

The following experimental methodology was utilized. First, we generated conventional gate-level implementations of the considered circuits. In order to avoid a bias, the circuits were highly optimized by ABC. Then, the optimized original circuits were approximated using CGP for various error e_i. Finally, the gate-level circuits were converted to Verilog netlists (one gate is represented by one logic expression) and synthesized. The goal of synthesis is to minimize the area, i.e. the number of (up to 6-input) LUTs. Results are presented for ABC and four commercial tools: Precision RTL 2015.1.6, ISE 14.7, Vivado 2015.2, and Quartus 14.1. Note that all FPGA synthesis tools start with the same result of CGP in a particular experiment. The circuit size and delay are extracted from the resulting technology netlists. In the case of ABC, the synthesis and optimization is performed by 15 iterations of `resyn2` script, followed by mapping `if -K 6 -a`. In the case of Xilinx and Precision, the 6-LUT FPGA chip under label Virtex7 XC7VX330 was chosen for the implementation. The FPGA chip EP4S40G of Altera's StratixIV family was taken in Quartus. Only single-output LUTs are considered (i.e. LUT-combining is not permitted) to provide fair conditions for all tools. The implicit setup of CGP parameters follows the recommendations given in [17]: $\lambda = 4$, $h = 5$, $n_c = n_G$, $g_{max} = 10^4$. Γ includes all 2-input gates. The results are presented in the form of graphs based on the best obtained circuit out of 20 independent runs of CGP. The experiments were conducted on a 64-bit Linux machine running on Intel Xeon X5670 CPU (2.93 GHz, 12 MB cache) equipped with a 32 GB RAM.

A. Arithmetic circuits

The 8-bit Kogge-Stone CLA adder is the simplest circuit in our benchmark set. CGP started with $n_G = 67$ (fully

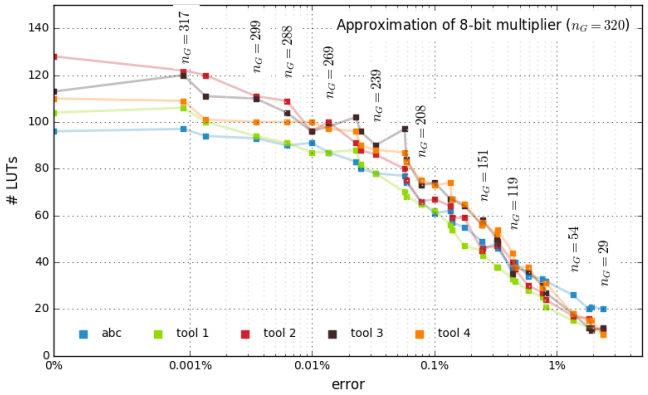

Fig. 4. The number of LUTs and number of gates (n_G) for various approximate 8-bit multipliers synthesized using common synthesis tools.

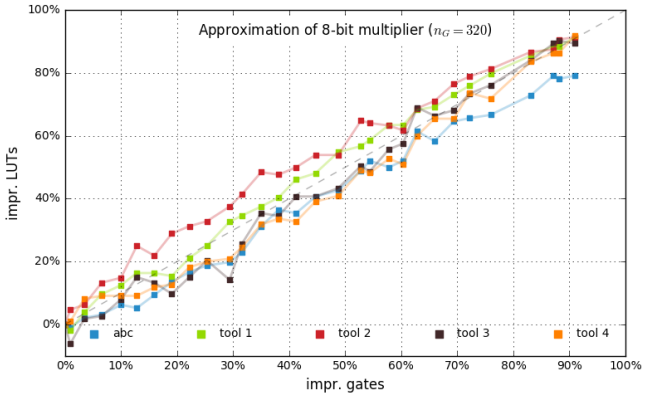

Fig. 5. The relation between improvement in the number of LUTs and in the number of gates for approximate multiplier

functional adder) and then minimized the number of gates for several target mean absolute errors, $e_i = \{0.001\% \ldots 2.5\%\}$ of E_{max}, where E_{max} is the maximum absolute arithmetic error $E_{max} = 2^w - 1 + 2^w - 1$. For 8-bit adder (i.e. $w = 8$ and $n_i = 2w = 16$), $E_{max} = 510$. An approximate adder exhibiting $e_i = 1\%$, for example, produces output values with difference equal to 5.1 in average.

Figure 2 shows that the circuit size (in LUTs) was reduced even for such a small circuit. Note that the x axis is in the logarithmic scale. Xilinx ISE and Precision RTL provide the most compact implementations of not only accurate but also approximate adders. There is only one case ($e_i = 2.5\%$), where Quartus was able to outperform results of Xilinx ISE and Precision RTL. On the other hand, the accurate adder is implemented using half of the LUTs compared to the same accurate adder synthesized using Quartus.

Performance of synthesis tools is roughly similar except the case of a fully functional adder, where ABC and Quartus generate the most resources consuming implementations. As evident, it makes no sense to introduce the approximations for really small target errors as the resulting circuits can be more complex than the fully functional ones.

A detailed analysis of the small circuits revealed that after reducing the original implementation by less than 40% gates, the number of LUTs is increasing for some tools. This is evident in Figure 3 that shows the relation between improvement in the number of LUTs and in the number of gates. Except of ABC and Quartus that have a lot of space for improving bad implementation quality of the accurate adder, the remaining synthesis tools were unable to exploit the reduction in the number of gates. This observation corresponds with Xilinx's approach used for estimating the total capacity of FPGA which counts from 6 to 24 two-input gates for one LUT depending on the number of inputs used. The gate-level optimization has to take this fact into account.

Figure 4 shows various approximations of the 8-bit Carry Save Adder Multiplier. The fully functional multiplier is five times more complex than the accurate adder from the previous experiment. The maximum absolute arithmetic error is $E_{max} = (2^w - 1) \cdot (2^w - 1)$. For 8-bit multiplier, $E_{max} = 65025$. It means that a multiplier with $e_i = 1\%$ produces outputs whose average difference is more than $127\times$ higher compared to the adder. The accurate multiplier with $n_G = 320$ gates led to 95 LUTs (with ABC) and 130 LUTs (with ISE). In approximate scenario and for errors smaller than 0.01%, ABC is the best performing tool. For higher error rates, Precision is the best tool.

As the multiplier is more complex than the adder, the improvement in the area obtained at the gate level is preserved at the LUT level almost perfectly. Hence Figure 5 presents almost linear mapping.

In summary, the gate-level approximation of arithmetic circuits that have to be implemented using LUTs in FPGAs has to be conducted with caution especially in the case of small desired errors where introducing of small errors usually yield none or only neglible improvement.

B. Non-linear signal processing circuits

The 9-input (i.e. 3×3 filtering window) and 25-input (i.e. 5×5 filtering window) median filters were chosen as an example of typical circuit of image processing applications. In order to approximate the median circuits, we used standard CGP with $\Gamma = \{min, max\}$ (operating over 8 bits). The number of generations was restricted to $g_{max} = 3 \cdot 10^6$ for the 9-median and $g_{max} = 300 \cdot 10^3$ for the 25-median which corresponds to 3 hour CGP runs. For purposes of the fitness evaluation, 10^4 training vectors were randomly generated for the 9-median and 10^5 vectors for the 25-median. The accurate 9-median (25-median, respectively) constructed using bitonic-sorting algorithm requires 38 min/max components (220 components, respectively). The aforementioned CGP-based process was repeated with constrained resources, leading to approximate median circuits with the mean error $e_i = \{0 \ldots 10\%\}$. In order to obtain gate-level representations, the min and max operations were synthesized using ABC (66 gates needed for each component). The resulting flattened netlist contains 2356 (13702) gates in the case of the accurate 9-median (25-median).

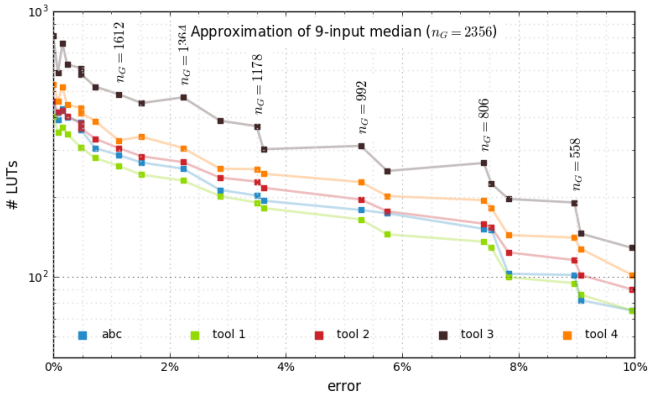

Fig. 6. Parameters of various approximations of the 9-input median circuit

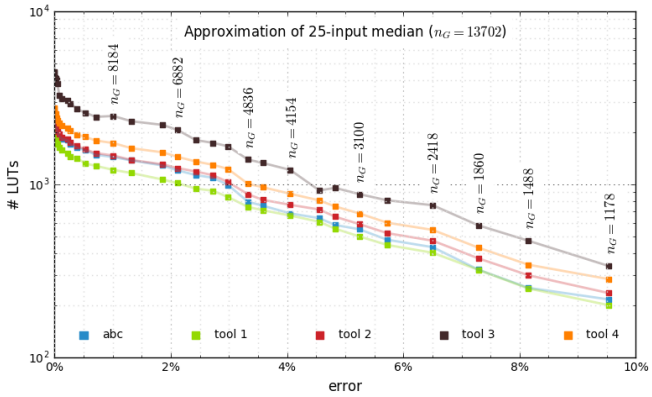

Fig. 8. Parameters of various approximations of the 25-input median circuit

Figure 6 and Figure 8 show the number of LUTs for a given mean absolute error. The tools compared in this study provide very different results (consider that the y-axis is logarithmic), where the area can differ by 100%. Hence in order to implement a median outputting circuit using constrained resources, it is recommended to use a better tool allowing a deep area optimization of the accurate solution rather than to introduce approximations using an average-performing synthesis tool.

Figure 7 shows relation between improvement in the number of LUTs and in the number of gates for approximate 25-input median. Despite the spread in the number of LUTs achieved across various tools, the improvement in the area obtained at the gate level is preserved at the LUT level almost perfectly.

C. Approximate Circuits in Real Applications

Not only the circuit parameters but also an impact on real applications needs to be quantified. In order to evaluate the effect of the proposed approximations, we approximated three basic image operators – Sobel operator, Gaussian filter and Median filter. The image operators were chosen intentionally because many problems from image processing domain exhibit a great degree of error resilience caused by limited human perception capabilities. Hence, it is possible to introduce an error to a corresponding image processing chain without a significant degradation in quality. This gives us a possibility to reduce the power consumption because the lower number the LUTS, the lower power consumption.

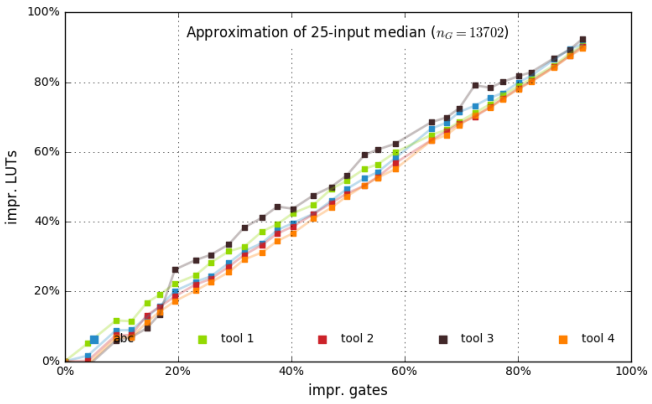

Fig. 7. The relation between improvement in the number of LUTs and in the number of gates for approximate 25-input median

The Sobel operator and Gaussian filter are typical examples of convolution filters, i.e. filters giving on their output a weighted sum of inputs pixels. While Gaussian filters need to be implemented using multipliers and adders, Sobel operator can be implemented solely from the adders because the convolution kernel contains only two coefficients. Both operations represent a basic building block that forms usually a part of more complex systems.

The Sobel operator is one of the most popular edge detectors that is defined on 3×3 pixel window. It is a discrete differentiation operator that computes an approximation of the gradient ($\mathbf{G}$) of the image intensity function. The gradient is determined using horizontal ($\mathbf{G}_x$) and vertial ($\mathbf{G}_y$) changes that are calculated by means of a convolution kernel that is used in direct and 90-degree rotated version. The gradient magnitude is computed using the square root function as $\mathbf{G} = \sqrt{\mathbf{G}_x{}^2 + \mathbf{G}_y{}^2}$ which is often replaced with the absolute value $\widetilde{\mathbf{G}} = |\mathbf{G}_x| + |\mathbf{G}_y|$ to reduce the computational requirements. The directional changes $\mathbf{G}_x$ and $\mathbf{G}_y$ as well as the final gradient $\mathbf{G}$ can be determined using adders as follows. The multiplication by two is implemented by arithmetic shifting. Subtraction is composed of adders and a set of inverters. The absolute value is obtained by an inversion controlled by the most significant bit representing a negative sign. In total 15 additions, 16 inverters and 15 XORs are required.

There exists several approaches to measure the quality of filtered images. The structural similarity index (SSIM) represents probably the most advanced approach which attempts to quantify the visibility of errors (differences) between a distorted image and a reference image [23].

SSIM calculated for various approximate Sobel operators evaluated on a set of 25 test images having 384x256 pixels each is given in Figure 9. Three architectures were considered: a) the accurate Sobel operator $\mathbf{G}$ whose output serves as a

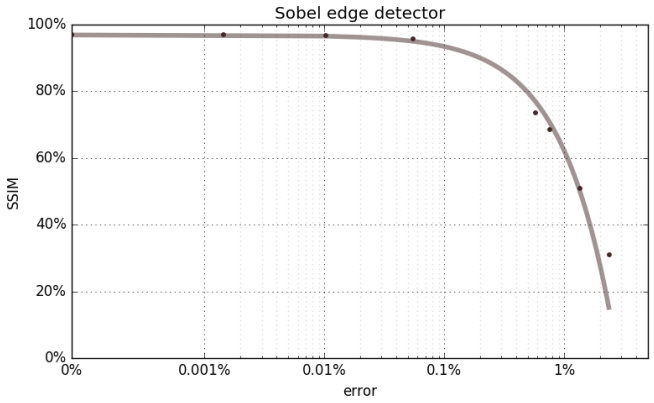

Fig. 9. Filtering quality of Sobel operator approximated using 8-bit precise adder (error=0) and various approximate adders whose average error is ranging from 0.0015% to 2.5%

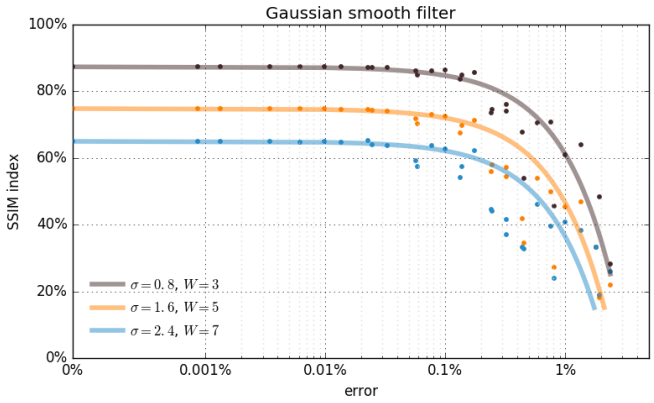

Fig. 10. Filtering quality of Gaussian filter approximated using 8-bit accurate (error=0) and approximate multipliers.

reference, b) the approximate Sobel operator $\widetilde{\mathbf{G}}$ implemented using accurate 8-bit adders, and c) the approximate Sobel operator implemented using approximate adders. Note that eight different approximate Sobel operators were created. For the simplicity, all additions were replaced with the same approximate adder exhibiting error e_i.

The similarity index changes only slightly when approximate adders exhibiting error lower than 0.1% are employed. As the error increases, however, the average similarity index decreases dramatically. According to the results, it is reasonable to employ adders exhibiting average arithmetic error not worse than 0.5%. Otherwise, the output quality is poor. We assume that the Sobel filter is very sensitive to the maximal absolute error and it would be probably necessary to include an additional constrain to the fitness function to improve the results.

SSIM for various 8-bit Gaussian filters evaluated on the same set of 25 test images are given in Figure 10. Three window sizes (3×3, 5×5, and 7×7) depending on the chosen standard deviation σ are considered. Two architectures were implemented: a) the Gaussian filter implemented using accurate 8-bit multipliers and a single accurate adder that sums up the products, and b) the approximate Gaussian filter implemented using approximate multipliers and an accurate adder. The test images were utilizes as reference for determining SSIM. Similarly to the Sobel operator, nearly no change in quality is observable when we employ an approximate multiplier with error not worse than 0.1%. In this case, the 0.1% error corresponds with a noticeable reduction of the number of LUTS (see Figure 4).

Finally, SSIM for median filters is summarized in Figure 11. The median filter is typically used in robust statistics to remove outliers. Its great resilience to the errors is kept even when it is approximated. For 9-input median filter, SSIM is larger than 99.9% for approximations exhibiting error $e_i < 0.7\%$. When $e_i = 10\%$, SSIM decreases to 92% which is a slight drop in performace compared to the results for Sobel operator and Gaussian filter. According to the results, it is possible to introduce a small error (1% in case of 25-median, 4% for 9-median) without any significant degradation in the output quality. This error is practically invisible in filtered images assuming that the approximate median is employed in image processing. As a consequence of that, 37% reduction in area is achieved for 25-input median, which is a significant result.

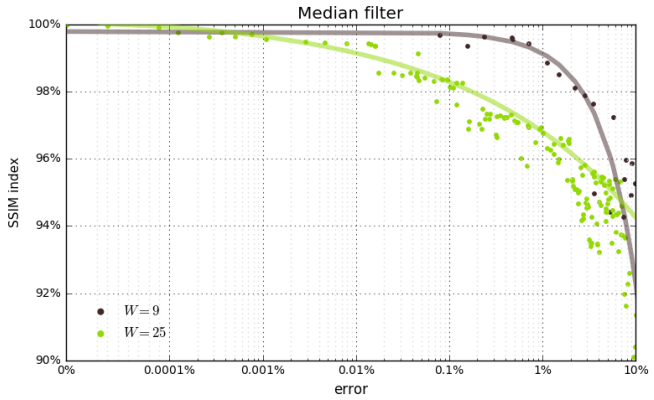

Fig. 11. Filtering quality of various 9-input and 25-input median approximations

V. Discussion

The complexity of the chosen benchmark circuits is roughly identical with circuits used for the evaluation of methods such as SASIMI [13], SALSA [14] and ABACUS [15]. Targeting these methods towards middle-size circuits is reasonable as approximations are typically introduced to carefully selected subcircuits which significantly contribute to power and area characteristics of the whole complex circuit. A typical case is the approximation of small multipliers employed in deep neural networks which enables to increase the number of neurons on a chip or reduce power consumption.

Although a direct comparison with other approximation methods is hard to perform (neither the implementations of the methods nor the benchmark circuits are available), we can provide only a rough comparison. For example, an 8-bit multiplier was approximated by SASIMI, which resulted in a 37% area reduction and the average error of 0.32% [13]. In

our case, the same error corresponds with reductions about 60% of LUTs. In the case of 9-median, Pareto fronts reported in this paper are at least of the same quality as reported in [24]. The CGP runtimes are typically in the order of tens of minutes for small circuits and up to three hours for medians (2.93 GHz CPU). As no execution times are usually reported in the literature dealing with circuit approximation, we can only give the execution times of SALSA [14] which, on a server with an AMD Opteron 6176 (2.29 GHz) processor, ranged from 4 minutes to 2.5 hours depending on the circuit complexity (2.29 GHz CPU).

An interesting result is that the gate-level optimization and approximation conducted by CGP is preserved by common FPGA synthesis tools, assuming that the original circuit is of a reasonable complexity and the allowed error is not very small. Interestingly, this result is valid even if the number of LUTs required to implement a given circuit is relatively low as it was demonstrated in the case of 8-bit adders consisting of less than 25 LUTs.

VI. Conclusions

We introduced a CGP-based methodology enabling to approximate various arithmetic circuits intended for FPGA implementations. The methodology was evaluated for two classes of circuits (arithmetic circuits, non-linear signal processing circuits) and using five FPGA synthesis tools. Due to the limited space we did not discuss the circuit delay; however, it has never been worsened by introducing the approximations in this study. By modifying only the fitness (error) function the method can easily be extended to approximate other types of combinational circuits such as common logic circuits. Here, the Hamming distance determined using BDDs may be employed.

We have shown that the results provided by commercial FPGA design tools vary significantly. This is noticeable especially in the case of median circuits consisting of a large number of gates. In this case, some tools provided very inefficient implementations independently whether it was approximate or accurate circuit. Hence, selection of the right synthesis tool seems to be a very important design step. In addition to that, we demonstrated that results provided by commercial FPGA design tools can significantly be improved by introducing an approximation conducted by CGP.

Our future work will be focused on circuit approximation at the LUT level and accelerating the design method.

Acknowledgments

This work was supported by Czech science foundation project GA16-17538S and The Ministry of Education, Youth and Sports of the Czech Republic from the National Programme of Sustainability (NPU II); project IT4Innovations excellence in science - LQ1602.

References

[1] A. Thompson, P. Layzell, and S. Zebulum, "Explorations in Design Space: Unconventional Electronics Design Through Artificial Evolution," *IEEE Trans. Evol. Comput.*, vol. 3, no. 3, pp. 167–196, 1999.

[2] J. Langeheine, "Intrinsic hardware evolution on the transistor level," Ph.D. dissertation, 2005.

[3] J. Walker, M. Trefzer, S. Bale, and A. Tyrrell, "PAnDA: A reconfigurable architecture that adapts to physical substrate variations," *IEEE Trans. Comput.*, vol. 62, no. 8, pp. 1584–1596, 8 2013.

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

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

[6] R. Dobai and L. Sekanina, "Low-level flexible architecture with hybrid reconfiguration for evolvable hardware," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 8, no. 3, pp. 20:1–20:24, May 2015.

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

[8] ——, "Search-based synthesis of approximate circuits implemented into fpgas," in *26th International Conference on Field Programmable Logic and Applications*. IEEE, 2016, pp. 1–4.

[9] A. Mishchenko. (2012) ABC: A system for sequential synthesis and verification, Berkley logic synthesis and verification group. [Online]. Available: http://www.eecs.berkeley.edu/˜ alanmi/abc/

[10] A. Mishchenko, S. Chatterjee, and R. Brayton, "Dag-aware AIG rewriting a fresh look at combinational logic synthesis," in *Proc. 43rd annu. Design Automation Conf.*, ser. DAC '06. ACM, 2006, pp. 532–535.

[11] P. Farm, E. Dubrova, and A. Kuehlmann, "Logic optimization using rule-based randomized search," in *Design Automation Conf., 2005. Proc. ASP-DAC 2005. Asia and South Pacific*, vol. 2, 2005, pp. 998–1001.

[12] Z. Vasicek and L. Sekanina, "A global postsynthesis optimization method for combinational circuits," in *Proc. Design, Automation and Test in Europe, DATE*. EDA Consortium, 2011, pp. 1525–1528.

[13] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-simplify: a unified design paradigm for approximate and quality configurable circuits," in *Design, Automation and Test in Europe, DATE'13*. EDA Consortium San Jose, CA, USA, 2013, pp. 1367–1372.

[14] S. Venkataramani, A. Sabne *et al.*, "SALSA: systematic logic synthesis of approximate circuits," in *49th Annu. Design Automation Conf. 2012, DAC '12*. ACM, 2012, pp. 796–801.

[15] K. Nepal, Y. Li *et al.*, "ABACUS: A technique for automated behavioral synthesis of approximate computing circuits," in *Proc. Conf. on Design, Automation and Test in Europe*, ser. DATE '14. EDA Consortium, 2014, pp. 1–6.

[16] A. Lotfi, A. Rahimi *et al.*, "Grater: An approximation workflow for exploiting data-level parallelism in FPGA acceleration," in *2016 Design, Automation Test in Europe Conf. Exhibition*, ser. DATE '16. EDA Consortium, March 2016, pp. 1279–1284.

[17] J. F. Miller, *Cartesian Genetic Programming*. Springer-Verlag, 2011.

[18] ——, "On the filtering properties of evolved gate arrays," in *1st NASA-DoD Workshop on Evolvable Hardware*, 1999, pp. 2–11.

[19] T. Knieper, P. Kaufmann *et al.*, "Coping with resource fluctuations: The run-time reconfigurable functional unit row classifier architecture," in *Proc. of the 9th Int. Conf. on Evolvable Systems: From Biology to Hardware*, ser. LNCS, vol. 6274. Springer, 2010, pp. 250–261.

[20] S. M. Cheang, K. H. Lee, and K. S. Leung, "Applying genetic parallel programming to synthesize combinational logic circuits," *IEEE Trans. Evol. Comput.*, vol. 11, no. 4, pp. 503–520, 2007.

[21] Z. Vasicek and K. Slany, "Efficient phenotype evaluation in cartesian genetic programming," in *Proc. 15th European Conf. on Genetic Programming*, ser. LNCS 7244. Springer Verlag, 2012, pp. 266–278.

[22] R. Hrbacek and L. Sekanina, "Towards highly optimized cartesian genetic programming: From sequential via simd and thread to massive parallel implementation," in *Proc. 2014 conf. Genetic and Evolutionary Computation*. ACM, 2014, pp. 1015–1022.

[23] Z. Wang, A. Bovik *et al.*, "Image quality assessment: from error visibility to structural similarity," *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–612, 2004.

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