2021 24th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS) | 978-1-6654-3595-6/20/$31.00 ©2021 IEEE | DOI: 10.1109/DDECS52668.2021.9417066

Synthesis of approximate circuits for LUT-based FPGAs

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

Abstract—**Approximate computing is an emerging paradigm that trades the accuracy of computation to achieve gain in terms of design area, critical path delay and/or power consumption. There is a rich body of literature showing that the approximate hardware components serving as basic building blocks for energy-efficient implementation of complex systems offer a remarkable gain in efficiency and/or performance in exchange for small losses in output quality. However, recent studies revealed that the approximate components optimized mainly for ASICs offer asymmetric gain when used in FPGAs. In this work, we present an iterative design method for automated synthesis of elementary approximate components natively optimized for usage in LUT-based FPGAs. The method takes into account the number of LUTs and LUT-level propagation delay instead of the number of gates and logic levels typically considered in other works. Using this method, we synthesized various approximate adders (up to 64-bit) and multipliers (8-bit and 16-bit). Compared to the current state-of-the-art, our designs achieve better trade-off when considered the worst case absolute error, number of LUTs and propagation delay. The discovered approximate adders and multipliers are available online in the form of Verilog netlists consisting of 4, 5 and 6-input LUTs.**

I. Introduction

There are many applications from various domains, including machine learning, computer vision, and signal processing, that exhibit a form of inherent resilience to small errors in the computation [1]. *Approximate computing* paradigm exploits the idea of accepting a certain level of inaccuracy in computations in order to reduce complexity and improve other parameters of digital systems. The typical goal is to improve design area, power consumption, performance or consumed energy. The current research covers the whole computing stack, integrating thus areas of microelectronics, circuits, components, architectures, networks, operating systems, compilers and applications. Approximations are conducted for embedded systems, ordinary computers, graphics processing units and even field-programmable gate arrays (FPGAs). For details, please refer to the recent surveys [2], [3], [4].

Functional approximation currently represents the preferred technique on how to introduce approximations to hardware components. The idea of functional approximation is to implement a slightly different function to the original one provided that the error is acceptable and other design parameters are reduced adequately. Transformation from the original (exact) circuit to its approximate counterpart can be performed in different ways and at different levels: netlist transformation, Boolean rewriting, and approximate high-level synthesis [5]. In netlist transformation, gate-level representation is typically used and modified to obtain an approximate circuit. Boolean rewriting manipulates with Truth table representation and uses common synthesis tools to obtain the implementation. High-level synthesis uses approximate building blocks to approximate a complex system. A detailed explanation of these principles and survey of the available approaches is provided, for example, in [5].

There is an extensive amount of research targeting the lowest level of abstraction, i.e. the approximation of basic building blocks represented mainly by the arithmetic circuits such as adders, multipliers or dividers. This area is particularly attractive since approximate circuits are employed as basic blocks for realizing application-specific accelerators that are highly relevant components of modern system-on-chips and hardware accelerators [6], [7]. The first works were the result of manual design, typically targeting a particular component and providing only one or few different circuit implementations optimized for a particular target technology. Hence, many interesting design points remained unexplored. Due to the limited scalability and the necessity of manual tuning, automated *approximate logic synthesis* (ALS) methods have been developed to provide approximate designs showing high-quality trade-offs between key design parameters for any circuit without a priori knowledge of its functionality [5]. Several approximate libraries consisting of approximate components have been created using various ALS approaches [4], [8], [9].

The approximate circuits included in the approximate libraries are typically designed and optimized to be used in ASICs. However, recent studies have demonstrated that the circuits offer asymmetric saving when used in LUT-based FPGAs [9], [10], [11]. A detailed analysis conducted on top of the state-of-the-art 8-bit multipliers revealed that the approximate circuits that might be optimal for ASICs are

This work was supported by Czech Science Foundation project 19-10137S.

978-1-6654-3595-6/21/$31.00 ©2021 IEEE

typically suboptimal when implemented in FPGAs [11]. In many cases, even negative gain was observed. To address this issue, some authors have started to optimize the approximate circuits manually to maximally exploit the structure of chosen target FPGAs [9], [10]. To avoid the manual LUT-level optimization, Prabakaran et al. applied machine learning approaches to identify the set of Pareto-optimal approximate circuits from already available approximate libraries [11]. The proposed method was used to construct several sets of Pareto-optimal arithmetic approximate circuits suitable for FPGAs that outperform current state-of-the-art implementations in many key parameters.

In this paper, we present an iterative design method for automated logic synthesis of basic approximate components that are inherently optimized for the usage in LUT-based FPGAs. The method takes into account the size of LUTs, the number of LUTs and propagation delay instead of the gate-level design parameters such as the number of gates or logic levels, typically considered in ALS approaches targeting ASICs.

II. Approximate logic synthesis for LUT-based FPGAs

A. Problem formulation

The design of approximate circuits represents a problem in which we know neither complete nor partial subset of input-output relations. To cope with this issue, the intended behaviour is typically defined relatively to the behaviour of a certain design point, typically an exact implementation of an approximate circuit. The design problem is then formulated as an optimization problem whose goal is to minimize the number of components and/or another criterion of the exact implementation provided that the distance (error) of the candidate solutions is not worse than a predefined error level.

Problem: *Given exact circuit C and a threshold ε, the goal is to find a circuit $\widetilde{C}$ with the minimal cost such that the error* $\mathrm{E}(C, \widetilde{C}) \leq \varepsilon$.

B. Overall principle

The ALS method used in this work is given in Algorithm 1. To simplify the problem, the design space exploration is based on a variant of hill climbing algorithm, an optimization technique which belongs to the family of local search strategies suitable for search in complex environments [12]. The algorithm starts with the netlist of an exact circuit (denoted as C) and attempts to find a better solution by making an incremental change to the netlist. If the change produces a better solution, another incremental change is made to the new solution, and so on until the terminating condition is satisfied (e.g. no further improvements can be found or the predefined amount of time is exhausted). Note that all accepted solutions have to satisfy condition on the target error level specified by the user. The best solution so far discovered is stored in p. Up to P_{size} variants of p are created in each iteration to foster the exploration performance. It means, a population of candidate solutions (denoted as P) consisting of the best design (p) and valid design alternatives (p') is generated.

Algorithm 1: The proposed LUT-aware ALS

```
Input: gate-level netlist of an exact circuit ($C$),
       target error level ($\varepsilon$), $P_{size} \geq 1$, $N_{ops} \geq 1$,
       $k \in \{4, 5, 6\}$
Output: approximate LUT-level netlist ($\widetilde{C}$) where
        $\mathrm{E}(C, \widetilde{C}) \leq \varepsilon$
1  $p \leftarrow C$;
2  while terminating condition not satisfied do
3      $P \leftarrow \{p\}$;
4      while $i < P_{size}$ do
           /* Apply $N_{ops}$ randomly chosen
              transform. operations to $p$ */
5          $p' \leftarrow \mathrm{Transform}(p)$;
           /* Perform k-input LUT mapping
              to determine the cost of $p'$ */
6          if $\mathrm{cost}(\mathrm{Map}(p')) \leq \mathrm{cost}(\mathrm{Map}(p))$ then
               /* Check if $p'$ satisfies the
                  target error level */
7              if $\mathrm{E}(C, p') \leq \varepsilon$ then
8                  $P \leftarrow P \cup \{p'\}$
9              end
10         end
11     end
12     $p \leftarrow \mathrm{SelectBest}(P)$;
13 end
   /* Perform k-input LUT mapping */
14 $\widetilde{C} \leftarrow \mathrm{Map}(p)$;
15 return $\widetilde{C}$;
```

C. Generating design alternatives

Circuits are represented as netlists composed of 2-input LUTs (nodes). The netlists correspond to directed acyclic graphs and are allowed to contain dangling nodes. The design alternatives are created using a set of randomly selected transformation operations. Up to N_{ops} operations are executed within a single transformation. Every operation can alter one of the following parts:

1) *node function*: e.g. AND node can be modified to inverter or vice versa, replaced by identity (wire), etc.; or
2) *node input connection*: a chosen input can be connected either to primary inputs, or to the output of any node provided that there will be no cycles in the resulting netlist; or
3) *primary output connection*: output can be connected to the output of any node, a primary input, or to logic constant '0'.

The proposed set of operations enables not only disconnect nodes but also introduce new ones (by modifying and re-activating dangling nodes). The nodes can become dangling by changing either primary output connection or node input connection or by changing node function resulting in decreasing of its arity.

D. Search space exploration

The search is guided by the objective function which determines how good a particular design (denoted as p') is. In accordance with literature and from practical reasons, the predefined error level ε is used as the constraint instead of being considered directly in the minimized objective function (see e.g. [13]). If a candidate circuit violates this constraint, it is simply discarded.

The goal of the search algorithm is to minimize the cost of a candidate solution which equals to the LUT count-level product (LLP):

$$cost(p') = \text{nodes}(\text{Map}(p')) \times \text{levels}(\text{Map}(p')), \qquad (1)$$

where $\text{Map}(p')$ refers to a netlist obtained from p' applying the k-input LUT mapping. Only better or equal candidate solutions are considered for the next iteration (included in P as shown in line 8). The cost of the candidate solutions is determined first (see line 6) because the error checking is usually more time consuming. Then, the error constraint is verified. This can be done either by a Monte-carlo simulation or by a SAT solver depending where there is need for error guarantee. The details can be found, for example, in [14]. In principle, the algorithm allows us to employ arbitrary error metric E in this step because one of the properties of the approximate circuits is that the error typically increases with increasing the number of removed nodes. If we consider this fact and construct the objective function to force the search towards more compact solutions (circuits having the minimal possible number of nodes for a given target error level), which is our case, the actual error of the candidate circuits will be implicitly forced to be as close as possible to the target ε.

Finally, the procedure SelectBest determines the best-scored individual from the population P at the end of each iteration. The selection prioritizes candidate solutions that differs from p (i.e. parental solution used to generate them). This strategy supports the diversity and prevents getting stuck in local optima.

III. Experimental Setup

The primary goal of our study is to evaluate the impact of the proposed approach on the quality of the obtained approximate circuits as well as on the time required to generate a particular result and compare the results with the recently released open-source library containing state-of-the-art approximate designs optimal for FPGAs [11].

A. Benchmarks

The proposed method is used to design approximate versions of various multipliers and adders whose error is no worse than a given threshold. In particular, 8-bit, 16-bit, 32-bit and 64-bit adders and 8-bit and 16-bit multipliers were used as benchmarks. Due to the limited space, only partial results are presented in this work.

TABLE I
Parameters of some benchmark circuits used in this work.

benchmark circuit	operand size	circuit architecture	gates count	gates levels	4-LUT [a] count	4-LUT [a] levels	6-LUT [b] count	6-LUT [b] levels
adder	16	RCA	77	31	31	15	24	8
		TA_BK	110	13	50	6	35	6
		TA_LF	113	11	52	5	35	5
		TA_SK	128	10	57	5	37	5
	64	RCA	317	127	127	63	96	32
		TA_BK	488	21	230	10	163	10
		TA_LF	557	15	266	7	170	7
		TA_SK	704	14	321	7	209	7
multiplier	8	CSAM_RCA	320	28	120	14	92	13
		CSAM_CSA	347	25	127	12	95	11
		WTM_CSA	396	26	172	11	123	9
		WTM_CLA	418	22	212	12	155	10

[a] The number of LUTs and LUT levels after mapping to 4-input LUTs.
[b] The number of LUTs and LUT levels after mapping to 6-input LUTs.

The approximation process starts with an accurate circuit (seed) which is then transformed to its approximate variant. We used several different architectures of the exact circuits. This helps us to evaluate the impact of the initial seed on the quality of the obtained approximate circuits which has not been investigated in the literature.

We chose four different ASIC architectures for each adder and multiplier. As depicted in Figure 1, the architectures form a Pareto frontier for the number of gates and the number of logic levels. The concrete values of the circuits referenced in this work are provided in Table I. Let us mention two prominent ones – the most known Ripple-Carry Adder (RCA) and Sklansky Tree Adder (TA_SK). The ripple carry adder (RCA) represents the most compact architecture exhibiting the largest propagation delay when implemented in ASICs as well as FPGAs. Sklansky Tree Adder (TA_SK) represent one of the tree-style parallel prefix adders (denoted as TA_XX) lying on the other side of the spectra. Compared to RCA, 64-bit TA_SK, for example, can operate at more than 9 times higher frequency but occupy 2.5× more resources. As shown in Table I, however, even those reference architectures offer asymmetric gain when implemented in LUT-based technologies. TA_SK does not offer any advantage compared to

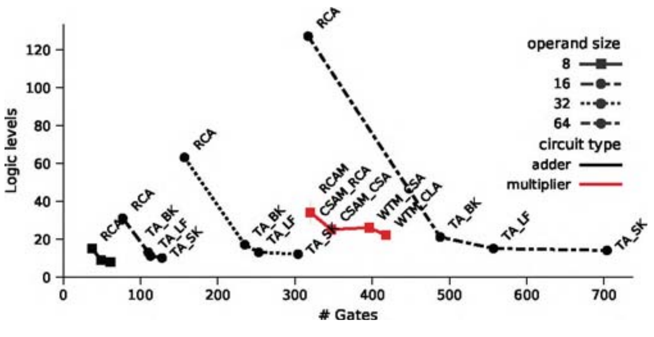

Fig. 1. Parameters of circuits used for benchmarking. Note that the 16-bit multipliers are omitted from the plot as they require more than 1400 gates (400 LUTs).

TA_LF. Both architectures have the same propagation delay independently of the number of LUT inputs. Carry save array multiplier with RCA in the final stage (CSAM_RCA) represents the most compact design both in ASICs as well as FPGAs. Wallace tree-based adders (denoted as WTM_xxx), on the other hand, have substantially better propagation delay for the price of higher occupied area.

B. *Experiments*

The proposed algorithm was implemented in C++. Several error metrics have been proposed in the literature to measure the distance between approximate and reference circuits. As a proof-of-concept we chose the worst case absolute error (WCE) defined, for example, in [14]. Compared to the other metrics, WCE can be determined formally without sacrificing error guarantees. In addition, it enables us to directly compare the results with [11]. Twenty WCE thresholds were considered for each benchmark circuit. The WCE violation was determined using a SAT solver and ABC as proposed in [15]. At the end of each experiment, exact WCE was calculated using Algorithm 5 proposed in [14]. For each threshold, five independent experiments were executed to obtain statistically sound results. The approximation process is terminated when there is no improvement for more than 10,000 iterations or a given runtime (3,600 seconds in our case) is exhausted.

LUT mapping is performed using Mockturtle library [16]. Every candidate netlist is represented using XAG network before LUT mapping. LUTs in the netlists are restricted to those hex values: 1, 6 (XOR), 7, 8 (AND), 9, A, E (OR). In addition to the LLP cost function described in Section II-D, we implemented and evaluated also the common cost function based on area-delay product (ADP). Three LUT sizes were considered: $k \in \{4, 5, 6\}$. The remaining parameters were chosen as follows: $P_{size} = 4$, $N_{ops} = 1$. In total, 14,400 independent experimental runs were executed on a server equipped with 2.4 GHz Intel Xeon CPU. At the end, we obtained many design alternatives described at the level of LUTs as well as gates.

IV. Results and Discussion

A. *Efficiency of the proposed cost function*

Figure 2 shows the overall results using boxplots calculated from all the experiments targeting the design of approximate adders. Each of four subplots shows the statistics related to a particular analysed parameter. We depicted the achieved reduction in the number of LUTs, gates, LUT levels, and logic levels depending on the allowed level of WCE and utilized cost function. In order to present a sound statistics, we plot the relative (i.e. normalized) values because of their independence of the used seed and operand size. WCE is normalized and expressed in terms of the percentage number of invalid bits (denoted as NWCE) because it enables us to compare parameters of circuits having different operand size. NWCE is calulated as $\mathrm{NWCE} = \log_2(\mathrm{WCE})/(w+1)$ for w-bit adders and $\mathrm{NWCE} = \log_2(\mathrm{WCE})/2w$ for w-bit multipliers, where WCE is the error of a particular approximate circuit (i.e. not

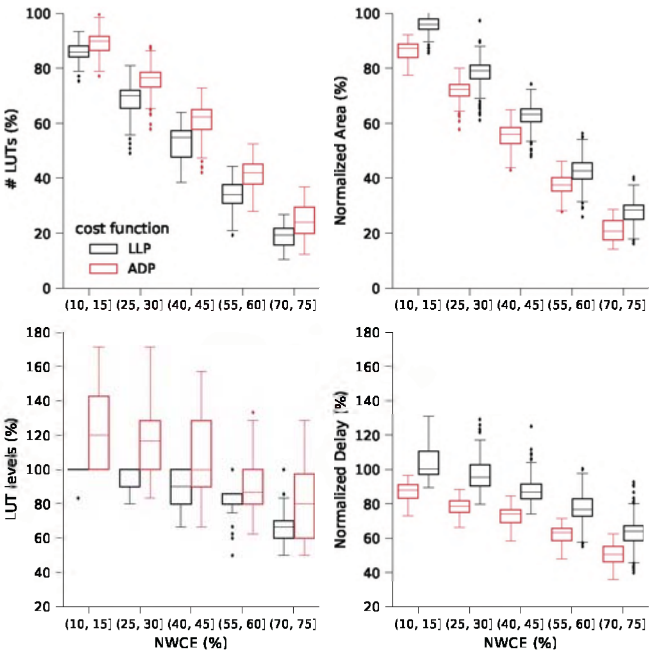

Fig. 2. Performance of the approximation depending on the chosen cost function. The lower value means the better result. Normalized values are shown on Y-axis; 100% refers to the parameters of the initial exact circuits as listed in Table I.

of the chosen target error level which may be higher in some cases). NWCE is categorized into four groups to illustrate its dependence of the achieved level of reduction.

The first subplot clearly demonstrates the advantage of the method utilizing LLP cost function. Considering the number of LUTs, the approximation based on the proposed cost function produces more compact circuits compared to the common cost function based on the number of gates (denoted as ADP). On the average, the number of reduced LUTs is by 5–7% higher in each category. But the main benefit of the LUT-aware cost function lies in the ability to control the propagation delay. ADP does not even guarantee that the number of LUT levels stays preserved. As shown in the left bottom plot of Figure 2, the number of levels can rise dramatically. For NWCE within $(10, 15]\%$, for example, the number of LUT levels increased by at least 20% for the majority of generated approximate circuits. There are even instances whose propagation delay rised by 70%. On contrary, LLP leads to approximate circuits exhibiting the same or better delay depending on the level of NWCE. The number of LUT levels is typically preserved for small NWCEs and decreases with the increasing NWCE.

The right column of Figure 2 shows that ADP leads to better results when we target the ASICs. The area on the chip expressed in terms of the number of 2-input gates is substantially better compared to the results obtained using LLP. Moreover, the propagation delay expressed in terms of the number of logic levels is also much better.

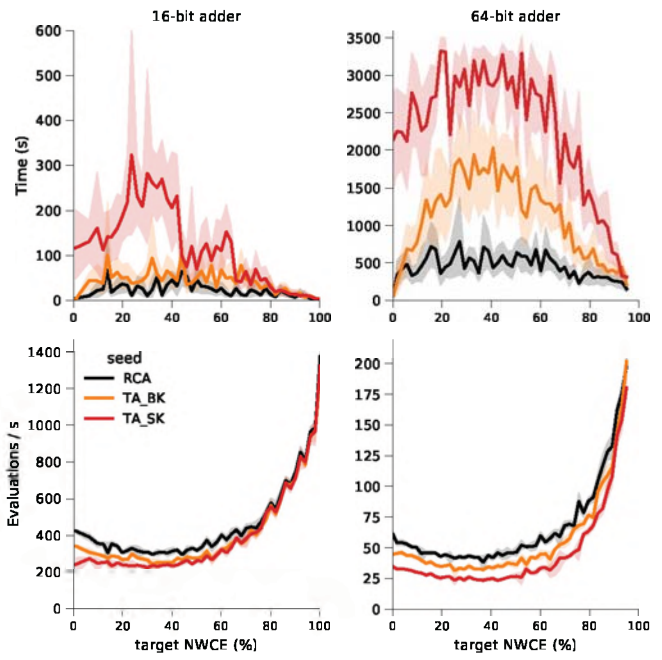

Fig. 3. Performance of the proposed method as a function of the used seed and required error level. The average time needed to design approximate adder with a given target error (top part) and the search performance expressed in terms of the number of evaluated candidate circuits per seconds (bottom part).

B. Computational complexity

The size of the approximate circuits is proportional to the level of WCE. The higher WCE, the higher reduction of LUT count (LUT levels) can be achieved. According to this dependence, one could expect the computational complexity grows with the increasing target WCE. Interestingly, an opposite behaviour is observable in reality. This fact is demonstrated in Figure 3.

The top row of Figure 3 shows the average time required to design an approximate circuit satisfying a given target WCE. It can be seen that the design time decreases with the increasing target error. On the average, more than 3,250 seconds are required to design a 64-bit approximate adder having NWCE around 20% from TA_SK adder. Less than 500 seconds are sufficient to design an approximate adder having the same bit-width but NWCE around 95%. This behaviour is caused mainly by the fact that the number of iterations evaluated within a certain period of time increases with increasing WCE (please see the bottom row of Figure 3 showing this dependence). There are two reasons for that. First, the size of the approximate circuit decreases in the course of the design process together with the increasing error that usually grows until it reaches the given target error level. Second, smaller circuits are easier to prove for WCE target level violation as they lead to more compact SAT instances.

We can observe that the search performance (i.e. the number of candidate circuits evaluated per second) increased dramatically. For 16-bit RCA adders, for example, around 350–410 candidate circuits can be evaluated per second when the target NWCE is less than 60%. The number of evaluations increases up to 1,400 per second (i.e. by 3–4 times) when the NWCE is around 95%. The similar situation is observable also for other widths as well as multipliers. It roughly holds for the adders that the number of evaluations per second decreases by the factor of two with doubling of the operand size. Around 800 evaluations per second can be achieved for 8-bit adders for NWCE less than 60%.

Apart from that, Figure 3 shows that the design time is dependent not only on target error level but also on the utilized seed which serves as a starting point for the search. Despite of the relatively stable search performance which changes only subtly with the seed (see the bottom part of Figure 3), the total design time may increase dramatically for some seeds and target errors (see the top part of Figure 3). While the dependence curve is nearly flat for the RCA architecture, it has a clear maximum in the middle of the NWCE range for more complex architectures (see e.g. the most complex TA_SK benchmark). This is caused by the increasing number of iterations needed to find an approximate circuit. In other words, it means that the fitness function changes regularly (at worse at every 10,000 iterations) and the search thus continues for longer period of time.

It can be concluded that the time of approximation is in reasonable bounds even for relatively complex arithmetic 64-bit instances. Five to ten minutes are required to approximate 16-bit adders and less than 60 minutes for 64-bit adders. More than 2,000 seconds are reported in [17] when approximating 64-bit RCA for the lowest considered WCE which was chosen to be 0.1% (i.e. NWCE equals to 86.7%). Less than 450 seconds are required in our method. In [15], the authors allocated two hours for the approximation of 64-bit adders.

C. Quality of the produced approximate circuits

Figure 4 reports the discovered Pareto optimal approximate adders and multipliers when considering the following three mutually conflicting parameters: the number of LUTs, the number of LUT levels and the level of WCE. Due to the limited space, we included the results only for 16-bit adders and 8-bit multipliers.

Looking at the top part of Figure 4 showing the best obtained 16-bit approximate adders, we can identify two imaginary chains of design points – the adders with low propagation delay but higher number of LUTs that are created from low-latency tree adders, and the adders obtained from RCA that occupy low area but are slow. Similar trend is observable also for multipliers. This phenomenon is valid independently on the chosen bit width. This result demonstrated that the proposed approximation method tends to preserve the main characteristics of the circuit used as the initial search point. This is especially important for lower error levels that are typically used in practice, because it gives us the ability to decide whether to prefer delay instead of area or vice versa. For higher error levels, it does not matter which architecture is used as the seed because the resulting approximate circuits have similar parameters. The distribution of the generated design points

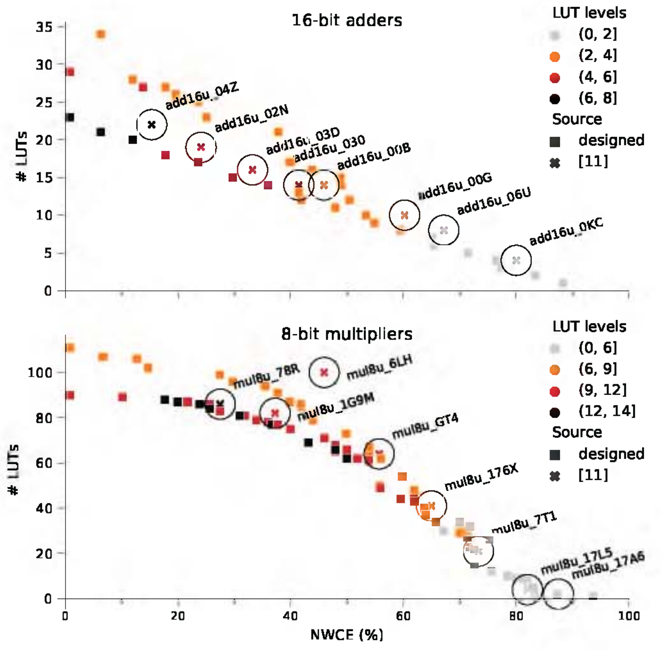

Fig. 4. Parameters of the state-of-the-art (see the highlighted points) and discovered Pareto-optimal (6-input LUTs count, LUT levels, WCE considered) approximate adders and multipliers.

also suggests that there must exist some approximate adders filling the design space between the imaginary chains.

Figure 4 also includes comparison with the state-of-the-art designs. For this purpose, we used the open-source Pareto-optimal arithmetic approximate circuits suitable for FPGAs introduced in [11]. To establish a fair evaluation, we chose the sub-set of circuits optimal with respect to the WCE and # LUTs. The circuits are synthesized using the same setup as used in this work.

The approximate circuits published in [11] were obtained by identifying implementations exhibiting the best LUT parameters from a rich set of approximate circuits obtained using an area-based cost function. The results shows that our method is able to achieve better trade-offs. This is consistent with the findings presented in Section IV-A demonstrating the superiority of the LUT-aware cost function.

V. Conclusions

We presented a novel iterative method for synthesis of approximate circuits optimized for LUT-based technologies. The method operates internally at the level of 2-input LUTs and produces LUT-level netlists that may consist of up to k-input LUTs, where the value of k can be chosen by the user depending on the intended target technology. Compared to the existing ALS methods, we used a cost-function which takes into account the effect of LUT mapping. This setup ensures that the search is forced towards more compact/faster design alternatives. One of the key properties of the proposed method is the ability to keep the propagation delay under control. As discussed, the number of LUT levels remains at worse preserved compared to the initial exact circuit. This allows to explore different portions of design space by changing the initial seed depending on the user requirements. For example, if there is a requirement to minimize the area by introducing some error, we can approximate slow but compact ripple-carry adder. On the other hand, if we need to improve the latency, we should use some tree adder as the initial point.

The experimental evaluation performed on common arithmetic benchmark circuits confirmed that the designs optimized for ASICs can be transferred to other technologies, but the achieved gain is highly suboptimal. If we need to achieve the best possible trade-off, we should always design the approximate circuits to match the target technology.

The set of the pareto-optimal arithmetic adders and multipliers designed within this work is available online at https://ehw.fit.vutbr.cz/pub/ddecs21 as open-source library which includes approximate circuits optimized for LUT-based architectures consisting of 4-, 5-, or 6-input LUTs.

References

[1] V. K. Chippa, S. T. Chakradhar *et al.*, “Analysis and characterization of inherent application resilience for approximate computing,” in *The 50th Annual Design Automation Conference 2013*. ACM, 2013, pp. 1–9.

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

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

[4] M. Shafique, R. Hafiz *et al.*, “Invited: Cross-layer approximate computing: From logic to architectures,” in *Proc. of DAC’16*, 2016, pp. 1–6.

[5] I. Scarabottolo, G. Ansaloni *et al.*, “Approximate logic synthesis: A survey,” *Proc. of the IEEE*, vol. 108, no. 12, pp. 2195–2213, 2020.

[6] S. Hashemi, H. Tann *et al.*, “Approximate computing for biometric security systems: A case study on iris scanning,” in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018, pp. 319–324.

[7] V. Mrazek, L. Sekanina, and Z. Vasicek, “Using libraries of approximate circuits in design of hardware accelerators of deep neural networks,” in *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2020, pp. 243–247.

[8] V. Mrazek, R. Hrbacek *et al.*, “Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods,” in *Proc. of DATE’17*, 2017, pp. 258–261.

[9] S. Ullah, S. S. Murthy, and A. Kumar, “SMApproxlib: Library of FPGA-based approximate multipliers,” in *Proceedings of the 55th Annual Design Automation Conference*. New York, NY, USA: ACM, 2018.

[10] B. S. Prabakaran, S. Rehman *et al.*, “DeMAS: An efficient design methodology for building approximate adders for FPGA-based systems,” in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2018, pp. 917–920.

[11] B. S. Prabakaran, V. Mrazek *et al.*, “ApproxFPGAs: Embracing ASIC-based approximate arithmetic components for FPGA-based systems,” in *2020 57th ACM/IEEE Design Automation Conference*, 2020, pp. 1–6.

[12] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall, 2010.

[13] S. Reda and M. Shafique, Eds., *Approximate Circuits*. Springer International Publishing, 2019.

[14] Z. Vasicek, “Formal methods for exact analysis of approximate circuits,” *IEEE Access*, vol. 7, no. 1, pp. 177 309–177 331, 2019.

[15] M. Ceska, J. Matyas *et al.*, “Approximating complex arithmetic circuits with formal error guarantees: 32-bit multipliers accomplished,” in *Proc. of 36th IEEE/ACM Int. Conf. On Computer Aided Design*. IEEE, 2017, pp. 416–423.

[16] M. Soeken, H. Riener *et al.*, “The EPFL logic synthesis libraries,” Nov. 2019, arXiv:1805.05121v2.

[17] V. G. U, V. V. S *et al.*, “LUT-based circuit approximation with targeted error guarantees,” in *2020 IEEE 29th Asian Test Symposium (ATS)*, 2020, pp. 1–6.