A Two-Level Approximate Logic Synthesis Combining Cube Insertion and Removal

Gabriel Ammes, Student Member, IEEE, Walter Lau Neto, Student Member, IEEE, Paulo Butzen, Senior Member, IEEE, Pierre-Emmanuel Gaillardon, Senior Member, IEEE, and Renato P. Ribas, Member, IEEE

Abstract—Approximate computing is an attractive paradigm for reducing the design complexity of error-resilient systems, therefore, improving performance and saving power consumption. In this work, we propose a new two-level approximate logic synthesis method based on cube insertion and removal procedures. The experimental results have shown significant literal count and runtime reduction compared to the state-of-the-art approach. The method scalability is illustrated for a high error threshold over large benchmark circuits. The obtained solutions have presented a literal number reduction up to 38%, 56%, and 93% with respect to an error rate of 1%, 3%, and 5%, respectively.

Index Terms—Approximate computing, approximate logic synthesis (ALS), digital design, sum of product (SOP), two-level (2L) circuit.

I. INTRODUCTION

In the last decades, the complexity of electronic systems has grown very fast, impacting the circuit power dissipation, performance, and area (PPA). Meanwhile, widely used applications, such as signal processing, machine learning, and data mining, exhibit error resilience properties. In this context, approximate computing has received special attention as a new design paradigm in recent years [1], [2]. Such a paradigm consists in modifying the functionality behavior of digital circuits to reduce PPA. When an approximate circuit is applied to an error-resilient application, the error introduced tends to be not so critical to the final operation, and improvements on PPA are expected. Particular effort has been made over adders and multipliers through handcrafted designing and systematic synthesis of such a regular arithmetic structure [3], [4].

This work exploits approximate logic synthesis (ALS), which automatically synthesizes approximate circuits for specified Boolean functions [5]. ALS approaches for two-level (2L) and multilevel combinational circuits as well as for sequential logic design have been presented in the literature [6]–[11]. In particular, 2L circuit synthesis consists in modifying a sum-of-products (SOP) expression of a given Boolean function aiming to minimize the literal count [6]–[9].

The main goal of 2L-ALS methods is identifying an approximate SOP expression with the fewest number of literals for a given original SOP and error threshold. This work applies the error rate (ER) metric as error constraint. The ER metric represents the probability that a given input vector leads to an erroneous output signal. Shin and Gupta [6] presented two techniques to approximate SOP: the insertion of cubes into the Boolean expression, by flipping the output from 0 to 1, and the removal of cubes from the expression, by switching the output from 1 to 0. They carried out experiments comparing both strategies and concluded that the cube insertion into the SOP leads to better results than the cube removal procedure. Hence, based on this assumption, related works have preferred the cube insertion to approximate SOP expressions.

In this work, we present a new 2L-ALS method that exploits cube insertion and removal by considering simultaneously both strategies without significant penalty in computation. First, the proposed approach applies a cube insertion procedure, similar to those presented in [6] and [9], to provide partial SOP solutions with fewer errors than the specified ER threshold. In the following, a cube removal procedure is applied over the obtained SOPs, taking into account the remaining error slack.

The major contribution of our approach is to exploit both cube insertion and removal procedures together in a unified 2L-ALS method. The experiments carried out over benchmark circuits, with a threshold of 16 errors, provided results with 8% fewer literals on average than the state-of-the-art method, without penalty in execution time [9]. Moreover, with a given ER percentage threshold, it reduced up to 38% and 96% in the literal count to an ER threshold of 1% and 5%, respectively. It is worth emphasizing that for the biggest benchmark circuit applied, the ER threshold of 5% comprised up to 6553 errors introduced. It illustrates the method scalability, knowing that, in [9], the error insertion is at most 16 for the same benchmark.

The remainder of this article is organized as follows. Section II presents some fundamentals, including the adopted terminology. Section III discusses Su’s work, taken here as the reference 2L-ALS method [9]. The proposed 2L-ALS method is described in Section IV, whereas experimental results are provided in Section V. Finally, Section VI concludes this article.

II. PRELIMINARIES

In this section, the fundamentals on ALS and error metrics are briefly reviewed. The adopted terminology is also presented for a better understanding of the proposed 2L-ALS method.

A variable corresponds to the symbol used to represent input and output signals. An occurrence of a variable in the Boolean expression is called a literal. It can represent an input or an output literal, being that the input literal can be direct or complemented. A product of literals where any variable appears at most once is a cube, and the sum of cubes results in a SOP. The particular case when a cube comprises
a single literal for each input variable and only one output literal is called a minterm. The size of a given cube is equal to the number of minterms it covers, whereas the expansion of a cube corresponds to the removal of one of the literals, turning it into a larger cube.

In the approximate circuit design, several metrics have been adopted to quantify and restrict the error introduced [14]. Our approach applies the ER metric for restraining the error occurrence. The ER metric corresponds to the ratio between input combinations that leads to output errors and the total input vectors allowed. It is also referred to as the probability of error occurring for a given input.

In terms of ALS, an input combination that results in one or more outputs with incorrect value is defined as an erroneous input combination (EIC). For instance, if two erroneous minterms present identical input literals but different output ones, only a single EIC is taken into account. Therefore, the number of errors (NoE) in a given SOP is equal to its number of EICs. When using NoE as the error threshold, it can be an arbitrary value or equal to $2^n \times er$, where $n$ is the number of inputs and $er$ corresponds to the ER threshold.

III. RELATED WORK

Su et al. [9] presented a heuristic search method to solve the 2L-ALS problem taking into account the ER constraint. This work can be considered as the state-of-the-art method in the subject, being presented in the following.

The main goal of Su’s approach is to identify the set of input combinations for 0-to-1 output complement (SICC) that maximize the literal count reduction on an approximate SOP. It is similar to selecting the set of EICs that results in the most compact SOP. They propose an SICC-cube tree (SCT) data structure, which groups a set of EICs to a set of cubes that depends on these EICs to be inserted into the SOP. It comprises a 2L tree where the root contains the EICs and the leaves represent the cubes to be added into the SOP. The number of EICs in the root is equal to the NoE inserted into the SOP.

Two conditions must be satisfied to ensure that SCT leaves lead to the optimization of the literal count. First, at least one cube must be removed from the SOP when a new cube is inserted. Second, the literal count in the removed cubes must be greater than the literals present in inserted cube.

Their initial task enumerates all possible multiple-output cubes of a function through the Hasse diagram structure. These cubes are used to build a set of SCTs. In the next task, the SCTs are combined because there are some with fewer errors than the maximum number allowed. After that, it is necessary to select the SCT that reduces the greatest number of literals.

A straightforward way to calculate the literal reduction in a given SCT is by using the Espresso tool [15], taking into account the EICs on the root as don’t cares to obtain an approximate SOP. The calculation of the literal reduction with Espresso presents a precise result, but the impact on the runtime is quite significant. Hence, a procedure that avoids the use of Espresso for estimating such a reduction is presented.

The procedure to predict the literal reduction on an SCT comprises mainly three steps. As the insertion of leaf cubes into the SOP does not guarantee a reduction in literal count, it first identifies the set of cubes that may be removed when the leaf cubes are inserted. Moreover, inserting all leaf cubes may increase the SOP literal count. Thus, it identifies the set of leaf cubes necessary to be inserted before removing the first set of cubes. Finally, it calculates the literal reduction between the sets of cubes removed and inserted.

Su et al. [9] presented four speed-up techniques to extend the application of their approach to large circuits.

1) As the basic algorithm time complexity grows exponentially with the NoE, the errors allowed for each execution are limited to two, therefore, generating partial approximate SOPs. All partial SOPs are approximated again until the accumulated NoE reaches the threshold allowed.

2) With the first speed-up technique, an exponential quantity of partial approximate SOPs is created, so impacting the final runtime. In order to reduce the number of partial SOPs, only the two expressions with the fewest number of literals are approximate again for a given partial NoE.

3) To reduce the number of combined SCTs, only a subset of all generated SCTs is taken into account. First, it estimates the literal count of all SCTs without combining them. In the next, for combining two SCTs, the first one must be within the 25% of the SCTs with the fewest number of literals whereas the second must be within the 80%.

4) The treatment of all cubes present on the Hasse diagram implies a high computational cost. In order to reduce such a cost, they only take the cubes on the diagram that are the parents of the cubes on the SOP, since it is improbable that any other than a parent of a SOP cube inserts less than two errors.

The values used in the first three speed-up techniques were obtained through an empirical analysis. Nevertheless, there is still an important runtime bottleneck for large NoE.

IV. PROPOSED 2L ALS METHOD

Existing 2L-ALS methods based on ER constraint have in common the adoption of cube insertion as the unique strategy adopted. On the other hand, the proposed approach exploits simultaneously both cube insertion and removal procedures, so aiming to reduce the literal count of a given optimized Boolean formulation $F$ and ER threshold $er$. First, a general description of the 2L-ALS method is presented, and the algorithms to approximate the SOP are described in the following.

The related works do not present good execution scalability. For instance, the approaches presented in [6] and in [9] limit the NoE to 8 and 16, respectively, due to the resulting runtime. The time complexity analysis of the proposed method is discussed at the end of this section.

A. General Description

First, the method applies a cube insertion procedure and generates multiple partial approximate SOPs with at most a given NoE constraint. For these partial SOPs, a cube removal procedure is then applied to approximate them assuming the remaining NoE. The cube insertion procedure is based on the approach presented in [9], and described in Section III.

The overall flow is illustrated in Algorithm 1. The SOP expression is stored using two maps. The first map groups each SOP cube with the minterms only covered by this cube. The second one groups each minterm covered by the SOP with the cubes that cover it. The original SOP $F$ is approximated taking into account an NoE threshold equal to 2. Two partial solutions comprising one and two errors are generated by applying the cube insertion algorithm. This procedure is based on the first speed-up technique presented in [9]. For each partial SOP solution, the cube removal procedure is applied considering the remaining slack NoE, which corresponds to the difference between the already inserted errors and the initial error constraint. Such a strategy helps in escaping from local minimum, thus leading to better solutions.

To prevent a critical increase in runtime and space complexity, only the initial SOP expressions are stored, and the partial solutions modify the original function $F$. Therefore, a partial solution comprises the set of cubes to be inserted and removed, along with the
Algorithm 1: Proposed 2L-ALS Method

Input: SOP expression $F$ and ER threshold $er$
Output: approximated SOP expression $F'$

1: $e \leftarrow er \ast 2^n$; Set $sols$ with $e+1$ sets of solutions;
2: $sols_0 \leftarrow \emptyset$ (empty solution);
3: for $i \leftarrow 0 \; \text{to} \; e$ do
4:  $topS \leftarrow$ the two best solutions in $sols_i$;
5:  for each solution $s$ in $topS$ do
6:      modifySOP($F$, $s$);
7:      $s1$, $s2$) $\leftarrow$ cubeInsertion($F$, $F'$, $s$);
8:      $sols_{i+1} \leftarrow sols_{i+1} \cup s1$;
9:      $sols_{i+2} \leftarrow sols_{i+2} \cup s2$;
10:     $s3 \leftarrow$ cubeRemoval($F$, $e-i$, $s$);
11:     $sMax \leftarrow \max(s1$, $s2$, $s3$);
12:     if $sMax > best$ then $best \leftarrow sMax$;
13:     restoreSOP($F$, $s$);
14:  end
15: end
16: return espresso(modifySOP($F$, best));

B. Cube Insertion Procedure

The cube insertion procedure is based on the heuristic search method presented in [9]. It uses the SCT as the primary data structure to perform the approximation. The main idea is to generate SCTs from cubes that do not exceed the threshold of EIC number, and then select the SCT with the most significant literal reduction. It is worth mentioning that since our NoE is equal to 2, the SCT root has two EICs at most.

Algorithm 2 presents the cube insertion flow. In line 1, it generates all SCTs. The expanded cubes from $F$ are considered as possible leaves to generate the SCTs. Using the expanded cubes simplifies the third speed-up technique, mentioned in Section III, as Su’s approach can insert any parent cube of a SOP cube. When an expanded cube is used as a possible SCT leaf, it guarantees the removal of the originating cube results in the literal count reduction. That way, it only has to verify the number of EICs needed to insert an expanded cube. The EICs of a cube comprise all minterms covered by it that are not covered by $F$, and its input combination that was not previously added as an EIC.

For SCTs $sct1$ and $sct2$ comprising one and two EICs in the root, if the $sct2$ root contains the $sct1$ root, the leaves of the $sct1$ are inserted into the $sct2$ leaves. This updated $sct2$ is called an augmented SCT, as seen in line 2.

Algorithm 2: cubeInsertion Procedure

Input: a simplified SOP expression $F$, an NoE threshold $e$, and the actual solution $s$
Output: two solutions with error 1 and 2

1: $trees \leftarrow$ generateSCT($F$, $e$, $s$.EIC);
2: augment($trees$);
3: $(s1, s2) \leftarrow$ combineAndEstimate($F$, $trees$);
4: $s1$ $\leftarrow$ updateSolution($s1$, $s$);
5: $s2$ $\leftarrow$ updateSolution($s2$, $s$);
6: return $(s1, s2)$;

Algorithm 3: cubeRemoval Procedure

Input: a simplified SOP expression $F$, an NoE threshold $e$, and the actual solution $s$
Output: a solution with at most $e$ errors

1: $error \leftarrow e$, newEIC $\leftarrow s$.EIC;
2: while $error > 0$ do
3:  for each Cube in $F$ do
4:      cubeEIC $\leftarrow$ getCubeEIC(cube, $F$, newEIC);
5:      gain $\leftarrow$ litCount(cube) / $\max(0.01, \#cubeEIC)$;
6:      if $gain > bestGain$ and $error \geq \#cubeEIC$ then
7:         bestGain $\leftarrow$ gain;
8:         bestCube $\leftarrow$ cube;
9:         bestEIC $\leftarrow$ cubeEIC;
10:    end
11: end
12: removeCubeFromSOP(bestCube, $F$);
13: removedCubes $\leftarrow$ removedCubes $\cup$ bestCube;
14: error $\leftarrow$ error - #bestEIC;
15: newEIC $\leftarrow$ updateEICs(newEIC, bestEIC);
16: end
17: insertCubes($F$, removedCubes);
18: $s3$ $\leftarrow$ updateSolution(removedCubes, newEic, $s$);
19: return $s3$;

In line 3, the combination of SCTs and the estimation of literal reduction are performed. First, the literal reduction of all generated SCTs is estimated. For all SCTs with one EIC in the root, their roots and leaves are combined two by two through the fourth speed-up technique presented in Section III. The two solutions that reduce more literals with NoE equal to 1 and 2 are returned and stored in $s1$ and $s2$.

The solutions $s1$ and $s2$ are updated in lines 4 and 5. This update comprises the following steps: adding the cubes inserted and removed within solution $s$ into solutions $s1$ and $s2$; estimating the new literal reduction; and updating the EICs. At the end, this procedure returns the solutions $s1$ and $s2$.

C. Cube Removal Procedure

Removing a cube implies that the cube literals are removed from the SOP. Therefore, the cube removal procedure is a greedy algorithm that selects the cube with the largest ratio between the numbers of literals and EICs. Algorithm 3 shows the flow of this procedure.

In the loop presented in line 3, the cube is chosen for removal. In this case, the EICs are obtained in line 4 and the gain in line 5. While removing a cube, its EICs are given by the minterms covered only by this cube in $F$ whose input combination was not previously added as an EIC. Those minterms are in the first map of the SOP data structure. In cases where the gain is greater than the actual bestGain,
that have their literals estimated is up to expanded cube generates one of them. Thus, the number of SCTs in worst case with one EIC is reached when each cube is executed.

D. Time Complexity Analysis

The cubeInsertion and cubeRemoval procedures present the most relevant impact on the time complexity of the proposed method. Each of them is executed $e$ times. For the sake of simplicity, we are omitting the Espresso complexity.

The combineAndEstimate task in the cubeInsertion procedure is the most time-consuming one. The most expensive phase of this procedure is to combine two by two the SCTs with one EIC on the root and estimate their literal count reduction. To generate the SCTs, the cubes on the SOP are expanded. As the expansion generates a new cube for every literal in a cube, the number of expanded cubes is equal to the number of literals in the SOP, represented by $L$. The worst case number of SCTs with one EIC is reached when each expanded cube generates one of them. Thus, the number of SCTs that have their literals estimated is up to $L^2$. The literal estimation depends on obtaining the covered minterms of each leaf cube. As the number of covered minterms by a cube is at most $m \times 2^n$, where $n$ and $m$ are the number of inputs and outputs of the function, respectively, the worst case time complexity of the cubeInsertion procedure is $O(L^2\times m + 2^n)$.

The cubeRemoval procedure, in turn, estimates the gain of removing each cube, represented by $C$, and removes the one with more gain until the limit error is reached. The gain depends on the number of EICs and cube literals. As obtaining the EICs relies on hash structures, its time complexity can be taken as constant. To obtain literal count, the cube is iterated $n + m$ times. Therefore, the worst case time complexity of the cubeRemoval procedure is $O(e + C \times (n + m))$.

The complete worst case time complexity of the proposed method is $O(e + L^2\times m + 2^n + e \times C \times (n + m))$.

V. EXPERIMENTAL RESULTS

The proposed algorithms have been implemented in the C++ programming language. Our experiments have been carried out over the IWLS’93 benchmark suite [16], in a computer with a quad-core i5-2400 CPU @ 3.10 GHz and 8 GB of RAM.

A. Comparison to Other Approaches

Su’s approach, presented in [9], is the state-of-the-art 2L-ALS published method, so it has been taken into account herein as our golden reference. The experiments consider the same circuits as in [9], with the NoE threshold that is equal to 16 in order to allow a fair comparison. Therefore, the designs have more than 6 and fewer than 20 inputs, and the sum of inputs and outputs has less than 34. As Su’s approach source code is not publicly available, we are comparing our results with the ones presented in [9], whose experiments were carried out with a quad-core i5-6500 CPU @ 3.20 GHz and 32 GB of RAM.

Table I shows the comparison results between Su’s method and our proposed approach. Column 1 presents the name of the circuits, as well as the number of inputs $(i)$ and the outputs $(o)$. Columns 2 shows the number of literals of the original SOPs, whereas columns 3 and 4 present the number of literals of the approximate circuits presented in [9] and obtained by our method, respectively. Columns 5 and 6 show the literal reduction rate between the literal count of the original SOP and our approximate solution, and between the literal count from the approximate SOP generated by our approach and the one presented in [9], respectively. Columns 7 and 8 present the runtime for both methods.

Our method has shown better results for all benchmark circuits treated, except the b12 one that both approaches were not able to optimize. The circuits con1, misex1, and b12 could not be approximated by Su’s method as it does not have SCT with size equal to one or two. On the other hand, con1 and misex1 have been approximated by our method due to the cube removal phase.

Moreover, the proposed method presented a better efficiency in general, with average runtime of around 4.69 s in comparison to 643 s presented in [9]. Such a difference is observed for circuits with more than ten inputs, where our method has a more scalable temporal behavior.

B. Results With ER

Fixing an NoE threshold may be a problem because the ER depends on the number of input combinations related to the target circuit. For instance, the 16 NoE applied before corresponds to an ER of 12.5% for a circuit with seven inputs but 0.01% for a circuit with 17 inputs. As our method presents a good runtime efficiency, it has been possible to apply a higher NoE and consequently allowed us to use a percentage ER. Table II shows the approximate SOP solutions considering an ER of 1%, 3%, and 5% over benchmark circuits with more than ten inputs. Column 1 gives the circuits and their input and output numbers. Columns 2 and 3 provide the ER in percentage and the corresponding NoE, respectively. Column 4 presents the literal count of the approximate SOPs obtained from our method. Column 5 shows the literal count reduction in percentage, whereas column 6 provides the runtime.
Our method reaches an average literal reduction of 38% with ER of 1%, 56% with ER of 3%, and 64% with ER of 5%. For sao2, table3, t481, and table5 circuits, we obtained a literal count reduction close to 90% with an ER of 5% and up to 93.9% with the same ER for the table5 circuit. Moreover, even though the b12 could not be approximated with a 16 NoE, we have shown that while considering an ER percentage as a constraint, it can be approximated with a literal count reduction up to 26.1%. Even with a higher NoE in circuits with many variables, the runtime remains under 5 min.

VI. CONCLUSION

This article presented the first 2L-ALS method that exploits both insertion and removal of cubes as strategies to approximate a given SOP expression taking into account the ER metric. The experimental results have shown that the proposed approach surpasses the state-of-the-art method in quality of results and scalability. The related source code, the applied benchmark circuits used to generate the experimental results, and their approximate descriptions are publicly available on GitHub.2

2https://github.com/GabrielAmmes/2LALS-IR

TABLE II
RESULT OF THE PROPOSED METHOD CONSIDERING ER THRESHOLD IN IWLS93 BENCHMARK SUIT

| Circuit | ER  | NoE | Literals | Original | Approximate | Time (s) |
|---------|-----|-----|----------|----------|-------------|----------|
| sao2    | 1%  | 10  | 496      | 274 (0.55)| 0.36        |          |
| i 10    | 3%  | 30  |          | 59 (0.15)| 2.16        |          |
| o 4     | 5%  | 51  |          | 37 (0.07)| 3.23        |          |
| ex1010  | 1%  | 10  | 2718     | 2659 (0.97)| 0.90        |          |
| i 10    | 3%  | 30  |          | 2588 (0.95)| 2.76        |          |
| o 10    | 5%  | 51  |          | 2511 (0.90)| 5.80        |          |
| alt4    | 1%  | 163 | 5087     | 3730 (0.73)| 96.07       |          |
| i 14    | 3%  | 491 |          | 2593 (0.52)| 188.18      |          |
| o 8     | 5%  | 819 |          | 2139 (0.42)| 279.35      |          |
| b12     | 1%  | 372 | 207      | 193 (0.93)| 1.87        |          |
| i 15    | 3%  | 983 |          | 170 (0.82)| 4.81        |          |
| o 9     | 5%  | 1638|          | 153 (0.73)| 10.98       |          |
| i881    | 1%  | 653 | 5233     | 542 (0.18)| 3.89        |          |
| i 16    | 3%  | 1966|          | 518 (0.11)| 5.92        |          |
| o 1     | 5%  | 3276|          | 578 (0.11)| 263.94      |          |
| table5  | 1%  | 1310| 2501     | 720 (0.28)| 100.98      |          |
| i 17    | 3%  | 3912|          | 280 (0.11)| 198.05      |          |
| o 15    | 5%  | 6593|          | 155 (0.06)| 245.94      |          |

REFERENCES

[1] Q. Xu, T. Mytkowicz, and N. Kim, “Approximate computing: A survey,” IEEE Des. Test., vol. 33, no. 1, pp. 8–22, Feb. 2016.
[2] S. Mittal, “A survey of techniques for approximate computing,” ACM Comput. Surveys, vol. 48, no. 4, pp. 1–33, 2016.
[3] V. Mrazek, R. Hrbacek, Z. Vasiccek, and L. Sekanina, “EvoApprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods,” in Proc. Design, Autom. Test Europe Conf. (DATE), 2017, pp. 258–261.
[4] M. E. Nojehdeh and M. Altun, “Systematic synthesis of approximate adders and multipliers with accurate error calculations,” Integration, vol. 70, pp. 99–107, Jan. 2020.
[5] I. Scarabottolo, G. Ansaloni, G. A. Constantinides, L. Pozzi, and S. Reda, “Approximate logic synthesis: A survey,” Proc. IEEE, vol. 108, no. 12, pp. 2195–2213, Dec. 2020.
[6] D. Shin and S. K. Gupta, “Approximate logic synthesis for error tolerant applications,” in Proc. Design Autom. Test Europe Conf. Exhibit. (DATE), 2010, pp. 957–960.
[7] J. Miao, A. Gerstlauer, and M. Orshansky, “Approximate logic synthesis under general error magnitude and frequency constraints,” in Proc. Int. Conf. Comput. Aided Design, 2013, pp. 779–786.
[8] C. Zou, W. Qian, and J. Han, “DPALS: A dynamic programming-based algorithm for two-level approximate logic synthesis,” in Proc. IEEE Int. Conf. ASIC, 2015, pp. 1–4.
[9] S. Su, C. Zou, W. Kong, J. Han, and W. Qian, “A novel heuristic search method for two-level approximate logic synthesis,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 39, no. 3, pp. 654–669, Mar. 2020.
[10] C. Meng, W. Qian, and A. Mishchenko, “ALSRC: Approximate logic synthesis by resubstitution with approximate care set,” in Proc. Design Autom. Conf. (DAC), 2020, pp. 1–6.
[11] A. Ranjan, A. Raha, S. Venkataramani, K. Roy, and A. Raghunathan, “ASLAN: Synthesis of approximate sequential circuits,” in Proc. Design Autom. Test Europe Conf. Exhibit. (DATE), 2014, pp. 1–6.
[12] M. Kubica and D. Kania, “Graph of outputs in the process of synthesis directed at CPLDs,” Mathematics, vol. 7, no. 12, p. 1171, 2019.
[13] C. Umans, T. Villa, and A. Sangiovanni-Vincentelli, “Complexity of two-level logic minimization,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 25, no. 7, pp. 1230–1246, Jul. 2006.
[14] S. Frohlich, D. Große, and R. Drechsler, “One method–All error-metrics: A three-stage approach for error-metric evaluation in approximate computing,” in Proc. Design Autom. Test Europe Conf. Exhibit., 2019, pp. 284–287.
[15] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, Logic Minimization Algorithms for VLSI Synthesis. Dordrecht, The Netherlands: Kluwer Acad. Publ., 1984.
[16] K. Mcelvain, “IWLS93 benchmark set 4.0,” in Proc. IWLS, 1993.