

QUADOL: A Quality-Driven Approximate Logic Synthesis Method Leveraging Dual-Output LUTs for Modern FPGAs

Jian Shi¹, Chang Meng², Xuan Wang³, and Weikang Qian¹

¹Global College, Shanghai Jiao Tong University, Shanghai, China; ³Synopsys, Inc., Shanghai, China

²Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, Netherlands

Emails: timeshi@sjtu.edu.cn, c.meng@tue.nl, wangxuan@synopsys.com, qianwk@sjtu.edu.cn

Abstract— Modern FPGAs support dual-output LUT to reduce the area of FPGA designs. Several existing works explored the use of dual-output LUTs in approximate computing. However, they are limited to small-scale arithmetic circuits. To address this issue, we propose QUADOL, a quality-driven approximate logic synthesis (ALS) method leveraging dual-output LUTs for modern FPGAs. It can approximately merge two single-output LUTs into a dual-output LUT. The selection of LUTs for approximate merging is formulated as a maximum matching problem to maximize area savings. To further enhance existing ALS methods, we also propose QUADOL+, a generic framework to integrate QUADOL into existing ALS methods. Experimental results showed that QUADOL+ achieves significant area reduction over prior works.

Index Terms—Approximate Computing, Approximate Logic Synthesis, Dual-output LUT, Approximate LUT Merging, FPGA

I. INTRODUCTION

Many modern applications, such as image processing and machine learning, are computationally intensive yet error-tolerant [1]. *Approximate computing* leverages this tolerance to trade accuracy for improved performance and energy efficiency. One important area of approximate computing is designing approximate circuits for *field-programmable gate array (FPGA)*, which is composed of reconfigurable *look-up tables (LUTs)*. Modern FPGAs support advanced dual-output LUTs, which enable dramatic reduction of the number of required LUTs [2]–[5]. A notable method was proposed to exactly merge two single-output LUTs into one dual-output LUT using local refinement strategies [6]. This process is called *LUT merging*, and the two merged LUTs are called a *LUT pair*.

Several works explored the use of dual-output LUTs in the design of approximate arithmetic units on FPGAs [7]–[14]. However, they can only be applied to specific types of circuits, such as multipliers [8]–[14] and adders [7]. In addition, they can only be applied to small-scale circuits. For example, the bit-widths of the multipliers in [8]–[13] do not exceed 8.

To design general approximate circuits, *approximate logic synthesis (ALS)* for FPGA was proposed [15]–[19]. It aims to automatically generate a high-quality circuit with a small error but a large area and delay reduction [20]. For example, Meng *et al.* proposed ALSRAC, which iteratively replaces signals in an *AND-inverter graph (AIG)* with new functions derived from existing signals, and maps the approximated AIG into LUTs [17]. Xiang *et al.* proposed to iteratively decompose a Boolean function approximately to minimize the number of

This work is supported by the National Natural Science Foundation of China under Grant 62574132 and the Natural Science Foundation of Shanghai under Grant 25ZR1401189. Corresponding author: Weikang Qian.

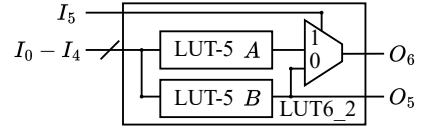


Fig. 1: Architecture of dual-output LUTs in the UltraScale+ series FPGA.

LUTs [18]. However, existing ALS methods on FPGA only consider single-output LUTs and fail to leverage dual-output LUTs available on modern FPGAs.

To solve the above issues, we propose QUADOL, a quality-driven ALS method leveraging dual-output LUTs for modern FPGAs. Our main contributions are as follows:

- We propose an *approximate LUT merging* technique, which approximately merges a LUT pair into a dual-output LUT to reduce the number of LUTs needed for a design, while minimizing the error introduced by approximation.
- To maximize area savings, we formulate a maximum matching problem to select a maximal number of LUT pairs for approximate merging.
- Based on the above techniques, we propose QUADOL, a new ALS method leveraging dual-output LUTs. We also propose to use binary search to accelerate QUADOL.
- To further enhance existing ALS methods, we also introduce QUADOL+, a generic framework for integrating QUADOL into existing ALS methods.
- We make the code for QUADOL and QUADOL+ open-sourced at <https://github.com/SJTU-ECTL/QUADOL>.

Experimental results showed that QUADOL+ reduces the LUT count by up to 17.9% compared to the state-of-the-art ALS method for FPGA. Moreover, the approximate multipliers optimized by QUADOL+ dominate most prior FPGA-based approximate multipliers in the area-error plane.

II. PRELIMINARIES

A. Dual-output LUT

A LUT with k inputs, denoted as *LUT- k* , can represent any Boolean function with k inputs. In several modern FPGA architectures, two small LUTs can be merged into a *dual-output LUT*, reducing the area of the design. In this work, we focus on the dual-output LUT architecture in the UltraScale+ series FPGA [2], which is shown in Fig. 1. The dual-output LUT, denoted as *LUT6_2*, is composed of two LUT-5s, *A* and *B*, sharing five same inputs I_0, \dots, I_4 . In Fig. 1, the output from LUT *B* provides the first output O_5 , while the second output

O_6 is determined by an additional input I_5 that selects between the outputs of the two LUT-5s.

B. Error Metrics

Error metrics are used to evaluate the accuracy of approximate circuits. In this work, two error metrics are used: *error rate* (ER) and *mean relative error distance* (MRED). ER is the probability that the circuit output is wrong. MRED measures the average relative error distance, calculated as $MRED = \sum_{i=1}^{2^I} \frac{|\hat{y}_i - y_i| \cdot p_i}{\max\{y_i, 1\}}$, where I is the number of primary inputs (PIs), y_i and \hat{y}_i are integers encoded by the exact and approximate outputs, respectively, for the i -th input combination, and p_i is the probability of the i -th input combination. The denominator is set as $\max\{y_i, 1\}$ to avoid division by zero.

In practice, for efficiency, ER and MRED are estimated via random logic simulation over M randomly sampled input combinations under a given distribution instead of using all possible input combinations [15], [17], [18].

III. METHODOLOGY

This section details the methodology of QUADOL and QUADOL+, both targeting combinational circuits. Section III-A introduces the basic idea of approximate LUT merging. To reduce the error introduced by approximate LUT merging, Section III-B presents a method to identify the optimal configuration for the dual-output LUT merged from a LUT pair. Section III-C formulates the LUT pair selection problem as a maximum matching problem to maximize merging opportunity. Based on the above techniques, Sections III-D and III-E finally present the flows of QUADOL and QUADOL+, respectively.

A. Approximate LUT Merging

Traditional LUT merging combines two single-output LUTs (*i.e.*, a LUT pair) into one LUT with dual outputs. The constraints for exactly merging two LUTs are strict. For instance, merging two distinct LUT-6s into a LUT6_2 shown in Fig. 1 is infeasible since the output O_5 can only express a function with up to five inputs. To further reduce the number of used LUTs, we propose *approximate LUT merging*. It allows approximately merging two single-output LUTs into a dual-output LUT, while only introducing a minimal error.

Suppose that the functions of the two single-output LUTs are F and G , respectively, and their input signals are \mathbb{I}_F and \mathbb{I}_G , respectively. After the approximate LUT merging, the dual-output LUT implements approximate versions of F and G , denoted as \tilde{F} and \tilde{G} , respectively, and the input signals of the dual-output LUT are also properly configured, denoted as $\mathbb{I}_{F,G}$. Fig. 2 shows an example of approximate LUT merging, where we assume that the single-output LUTs are LUT-3s and the truth table of a Boolean function is represented by a row vector. To implement F and G with a dual-output LUT, we introduce approximation by flipping some entries in their truth tables, *i.e.*, the red entries in Fig. 2(b).

In what follows, for simplicity, we use the term “merging functions F and G ” to mean merging their corresponding LUTs, and we denote the merged LUT pair F and G as (F, G) .

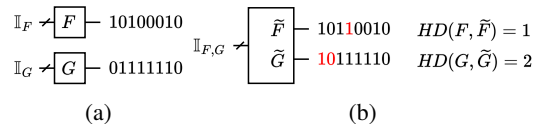


Fig. 2: Approximately merging two single-output LUTs into one dual-output LUT: (a) before merging; (b) after merging.

For a given LUT pair (F, G) , we aim to reduce the error introduced by the approximate functions \tilde{F} and \tilde{G} after approximate merging. To measure the error brought by different choices of \tilde{F} and \tilde{G} , we introduce the concept of *Hamming Distance* (HD). Specifically, the HD between an exact function Y and its approximate function \tilde{Y} is defined as the number of the input combinations where Y and \tilde{Y} have different output values. We denote it as $HD(Y, \tilde{Y})$. For the example in Fig. 2(b), $HD(F, \tilde{F}) = 1$ and $HD(G, \tilde{G}) = 2$. We define the error e brought by approximating F and G into \tilde{F} and \tilde{G} as:

$$e = HD(F, \tilde{F}) + HD(G, \tilde{G}). \quad (1)$$

The objective is to find functions \tilde{F} and \tilde{G} that minimize the error e under the constraint that \tilde{F} and \tilde{G} must be realizable by the dual-output LUT architecture.

B. Configuring Dual-output LUT for Approximate Merging

This section proposes a method to determine the optimal configuration for the dual-output LUT. Most commercial FPGAs use LUT-6s as their fundamental LUT units [2]–[5]. Once a circuit is mapped into such FPGAs, LUT-5s and LUT-6s occupy a large percentage of all mapped LUTs, and by approximately merging these LUTs into dual-output LUTs, we can substantially reduce the total LUT count. Therefore, we focus on approximately merging LUT-5s and LUT-6s in this work. Specifically, we target three types of LUT pairs for approximate merging: 1) two LUT-6s sharing six identical inputs; 2) two LUT-6s sharing five identical inputs; 3) one LUT-6 and one LUT-5 sharing five identical inputs. We call these three type of LUT pairs *mergeable LUT pairs*. Moreover, we focus on the dual-output LUT architecture shown in Fig. 1. However, it is worth noting that our method can be adapted to other architectures with simple modifications. Next, we discuss how we approximately merge the three types of LUT pairs.

1) *Merging two LUT-6s sharing six identical inputs*: Let F and G be the functions of two LUT-6s sharing six inputs x_0, x_1, \dots, x_5 . The configuration of LUT6_2 depends on the signals assigned to I_0, I_1, \dots, I_5 and the assignment of F and G to two outputs O_5 and O_6 of the LUT6_2. To minimize the error introduced by approximate LUT merging, we should keep as many original inputs as possible. Thus, five out of x_0, x_1, \dots, x_5 should be assigned to I_0, I_1, \dots, I_4 . The remaining input, denoted as x_i , is assigned to I_5 or not used. Therefore, I_5 is chosen from the set $\{1, x_i, \bar{x}_i\}$, where $i \in \{0, 1, \dots, 5\}$, and I_0, I_1, \dots, I_4 are assigned with the other inputs signals. Note that

- The choices of I_5 include \bar{x}_i . This is because due to the configurability of the truth table of a LUT, the signal \bar{x}_i can be generated by negating the output of the LUT of x_i . In this case, for each LUT originally using x_i as an input,

TABLE I: Configurations for LUT6_2 when merging two LUT-6s with six identical inputs.

I_5	F	G	LUT A	LUT B
x_i	O_6	O_5	F_{x_i}	$Maj(F_{\overline{x_i}}, G_{x_i}, G_{\overline{x_i}})$
x_i	O_5	O_6	G_{x_i}	$Maj(F_{x_i}, F_{\overline{x_i}}, G_{\overline{x_i}})$
$\overline{x_i}$	O_6	O_5	$F_{\overline{x_i}}$	$Maj(F_{x_i}, G_{x_i}, G_{\overline{x_i}})$
$\overline{x_i}$	O_5	O_6	$G_{\overline{x_i}}$	$Maj(F_{x_i}, F_{\overline{x_i}}, G_{x_i})$
x_i	O_6	O_6	F_{x_i} or G_{x_i}	$F_{\overline{x_i}}$ or $G_{\overline{x_i}}$
1	O_6	O_5	F_{x_i} or $F_{\overline{x_i}}$	G_{x_i} or $G_{\overline{x_i}}$

to keep its global function, we should change its local function by replacing input x_i with $\overline{x_i}$. The only exception is that when x_i is a PI or a primary output (PO), $\overline{x_i}$ is not considered, as we need additional LUTs to generate $\overline{x_i}$.

- The choices of I_5 do not include 0. This is because when the constant 0 is assigned to I_5 , LUT-5 A is not used (see Fig. 1), which has a larger error compared to the case where I_5 is assigned with constant 1.

For functions F and G , they are derived from the set $\{O_5, O_6\}$. Based on the above discussion, we list all configurations in Table I. Next, we will discuss how to derive the optimal functions for LUT-5s A and B for each configuration.

Row 1 of Table I corresponds to the case where x_i is assigned to I_5 , and F and G are derived from O_6 and O_5 , respectively. By Shannon decomposition, the approximate expressions for F and G can be written as $\tilde{F} = x_i\mathcal{A} + \overline{x_i}\mathcal{B}$ and $\tilde{G} = x_i\mathcal{B} + \overline{x_i}\mathcal{C}$, respectively, where \mathcal{A} and \mathcal{B} represent the functions for LUT-5s A and B in Fig. 1, respectively. Since $F = x_iF_{x_i} + \overline{x_i}F_{\overline{x_i}}$, where F_{x_i} and $F_{\overline{x_i}}$ are the cofactors of F when x_i is true and false, respectively, $HD(F, \tilde{F})$ equals $HD(\mathcal{A}, F_{x_i}) + HD(\mathcal{B}, F_{\overline{x_i}})$. Similarly, $HD(G, \tilde{G}) = HD(\mathcal{B}, G_{x_i}) + HD(\mathcal{C}, G_{\overline{x_i}})$. Therefore, $HD(F, \tilde{F}) + HD(G, \tilde{G})$ equals $HD(\mathcal{A}, F_{x_i}) + HD(\mathcal{B}, F_{\overline{x_i}}) + HD(\mathcal{B}, G_{x_i}) + HD(\mathcal{C}, G_{\overline{x_i}})$. To minimize the term $HD(\mathcal{A}, F_{x_i})$, \mathcal{A} should be set to F_{x_i} . To minimize the remaining parts, \mathcal{B} can be determined through the majority among $F_{\overline{x_i}}$, G_{x_i} and $G_{\overline{x_i}}$, i.e., $\mathcal{B} = Maj(F_{\overline{x_i}}, G_{x_i}, G_{\overline{x_i}})$ [21].

Row 2 of Table I corresponds to the case where x_i is assigned to I_5 , and G is mapped to O_6 and F to O_5 . It is equivalent to swapping F and G in row 1. Thus, \mathcal{A} and \mathcal{B} can be directly obtained by swapping F and G in the expressions for \mathcal{A} and \mathcal{B} in row 1. Rows 3 and 4 of Table I correspond to assigning $\overline{x_i}$ to I_5 , which is equivalent to swapping x_i and $\overline{x_i}$ in rows 1 and 2. Thus, \mathcal{A} and \mathcal{B} can be derived by swapping x_i and $\overline{x_i}$ in the expressions for \mathcal{A} and \mathcal{B} in rows 1 and 2.

Row 5 of Table I covers a special case where x_i is assigned to I_5 , and both F and G are derived from O_6 . In this case, $HD(F, \tilde{F}) + HD(G, \tilde{G})$ equals $HD(\mathcal{A}, F_{x_i}) + HD(\mathcal{B}, F_{\overline{x_i}}) + HD(\mathcal{A}, G_{x_i}) + HD(\mathcal{B}, G_{\overline{x_i}})$. Thus, \mathcal{A} is set to either F_{x_i} or G_{x_i} , and \mathcal{B} to either $F_{\overline{x_i}}$ or $G_{\overline{x_i}}$, as both options yield the same total HD. To determine the optimal choice, a simulation for each choice is performed, and the one causing the least error in the POs is chosen. Note that we ignore the case where $\overline{x_i}$ is assigned to I_5 and both F and G are derived from O_6 . Its optimal solution can be derived by swapping the expressions for \mathcal{A} and \mathcal{B} in row 5, and the optimal total HD is the same. Also, we ignore the case where both F and G are derived from O_5 , as it leaves LUT-5 A unused, causing a higher error than the case represented by row 5.

Row 6 of Table I presents another special case where the constant 1 is assigned to I_5 , making LUT6_2 operate as two separate LUT-5s, with one producing F and the other producing G . Suppose LUT A produces F and B produces G . As a result, F is derived from O_6 , and G from O_5 . Thus, \mathcal{A} should be either F_{x_i} or $F_{\overline{x_i}}$, and \mathcal{B} should be either G_{x_i} or $G_{\overline{x_i}}$. Note that there is no need to consider the case where LUT A produces G and B produces F , as it yields the same optimal total HD.

In the proposed approximate LUT merging method, the optimal total HD for each row in Table I is calculated. The configuration with the smallest total HD is selected as the optimal configuration for x_i . Since x_i is unknown, we perform the above procedure for each $x_i \in \{x_0, x_1, \dots, x_5\}$. Among these choices, we select the one with the smallest total HD as the optimal solution for merging F and G . To further improve accuracy, we actually consider two function pairs, (F, G) and $(\overline{F}, \overline{G})$, for approximate merging. The pair with the smallest HD is eventually selected. If $(\overline{F}, \overline{G})$ is selected, an input negation is performed on the LUTs with \overline{F} as an input to restore their functions. Note that the pairs $(\overline{F}, \overline{G})$ and (F, G) are not considered, as their configurations can be derived by negating \mathcal{A} and \mathcal{B} from those of (F, G) and $(\overline{F}, \overline{G})$, respectively, yielding the same HDs and errors in the POs.

2) *Merging two LUT-6s sharing five identical inputs:* Let the shared inputs of F and G be x_0, x_1, \dots, x_4 and the unique inputs of F and G be m and n , respectively. Thus, functions F and G are independent of n and m , respectively. In LUT6_2, both O_6 and O_5 depends on I_0, I_1, \dots, I_4 , while O_5 is independent of I_5 . Therefore, to minimize the approximation error, a good choice is to assign the shared inputs of F and G , x_0, x_1, \dots, x_4 , to I_0, I_1, \dots, I_4 . Similar to merging two LUT-6s with six identical inputs, the remaining input, I_5 , should be chosen from the set $\{1, m, \overline{m}, n, \overline{n}\}$. Table II lists all configurations for the LUT6_2.

As shown in Table I, when merging two LUT-6s with six identical inputs, after assigning x_i to I_5 , the assignment of F and G to two outputs O_6 and O_5 of the LUT6_2 should be further considered since F and G both depend on x_i . In contrast, merging two LUT-6s with five identical inputs yields more deterministic assignments. For example, when m is assigned to I_5 , G should be mapped to O_5 , as G is independent of m . Accordingly, F is assigned to O_6 . This principle also applies when I_5 is set to n , \overline{m} , or \overline{n} , where the function independent of the selected signal is always assigned to O_5 . Rows 1–4 in Table II correspond to these four cases. The optimal configurations for \mathcal{A} and \mathcal{B} for each case are derived based on cofactors and Shannon decomposition in a similar way as merging two LUT-6s sharing six identical inputs. Row 5 presents the case where constant 1 is assigned to I_5 . It is similar to the case corresponding to the last row of Table I and hence, its optimal configuration can be derived similarly.

3) *Merging one LUT-6 and one LUT-5 sharing five identical inputs:* Let F be the function for the LUT-6 and G be the function for the LUT-5. Denote their shared five inputs as x_0, x_1, \dots, x_4 and the unique input for F as m . Thus, G is independent of m . By a similar analysis used above, we can

TABLE II: Configurations for LUT6_2 when merging two LUT-6s with five identical inputs.

I_5	F	G	LUT A	LUT B
m	O_6	O_5	F_m	$Maj(F_{\bar{m}}, G_n, G_{\bar{n}})$
n	O_5	O_6	G_n	$Maj(F_m, F_{\bar{m}}, G_{\bar{n}})$
\bar{m}	O_6	O_5	$F_{\bar{m}}$	$Maj(F_m, G_n, G_{\bar{n}})$
\bar{n}	O_5	O_6	$G_{\bar{n}}$	$Maj(F_m, F_{\bar{m}}, G_n)$
1	O_6	O_5	F_m or $F_{\bar{m}}$	G_n or $G_{\bar{n}}$

TABLE III: Configurations for LUT6_2 when merging a LUT-6 and a LUT-5 with five identical inputs.

I_5	F	G	LUT A	LUT B
m	O_6	O_5	F_m	$F_{\bar{m}}$ or G
\bar{m}	O_6	O_5	$F_{\bar{m}}$	F_m or G
1	O_6	O_5	F_m or $F_{\bar{m}}$	G

obtain all configurations for LUT6_2 listed in Table III.

C. Selecting Multiple LUT Pairs

To reduce the number of LUTs used in the final design, we try to merge as many LUT pairs as possible. However, conflicts may arise when multiple candidate LUT pairs share a common LUT. For example, if LUT pairs (F, G) and (G, K) are individually mergeable, they cannot be merged simultaneously, as signal G cannot be driven by multiple sources. Therefore, when two LUT pairs share the same LUT member, they are *in conflict* and cannot be merged simultaneously. To maximize area reduction, we select the largest subset of *non-conflicting LUT pairs*. To achieve this, a graph is constructed where each node represents a LUT, and an edge between two nodes means that the pair of LUTs corresponding to the two nodes is mergeable, *i.e.*, belonging to the three types of LUT pairs described in Section III-B. Fig. 3(a) shows an example of such a graph with 6 LUTs. Our goal is to identify the maximum number of disjoint edges, which can be efficiently solved using a maximum matching algorithm [22]. All LUT pairs corresponding to the identified edges are then merged simultaneously to form an approximate circuit. For example, as shown in Fig. 3(a), the three LUT pairs denoted as the edges marked with red circles can be merged simultaneously.

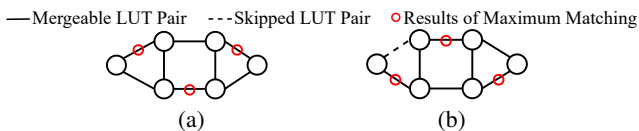


Fig. 3: Two different maximum matching solutions.

A given graph may have multiple maximum matching solutions. For example, Figs. 3(a) and (b) show two distinct maximum matchings for the same graph. As obtaining all such solutions is computationally expensive, we propose a randomized method to find k different maximum matching solutions for each graph, where k is a user-specified number. Specifically, once a new maximum matching solution is found, one of its LUT pairs is randomly chosen and excluded from the search for the next solution. For example, the dashline in Fig. 3(b) means that the LUT pair is randomly chosen and excluded after the solution in Fig. 3(a) is found. This ensures that the new solution in Fig. 3(b) is different from the initial solution in Fig. 3(a).

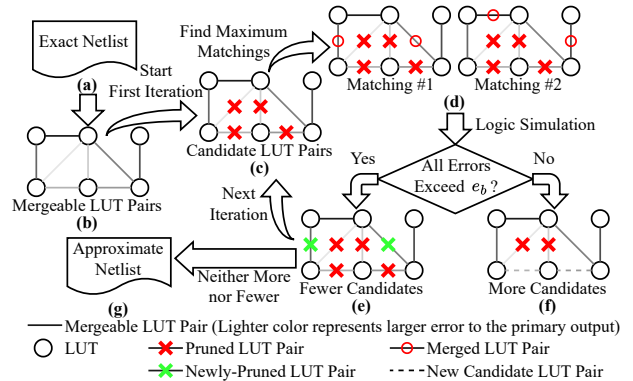


Fig. 4: The overall flow of QUADOL.

D. Flow of QUADOL

This section presents the entire flow of QUADOL, which is based on the approximate dual-output LUT configuration method in Section III-B and the LUT pair selection method in Section III-C. Fig. 4 illustrates the overall flow of QUADOL. It leverages the idea of binary search to speed up the process.

Initially, QUADOL identifies all mergeable LUT pairs in the given exact netlist (see Fig. 4(a)), which are represented by the edges in Fig. 4(b). However, some of them may induce large errors after approximate LUT merging and, hence, should be pruned before the search for optimal LUT pairs. Thus, for each LUT pair, QUADOL evaluates its error at the POs using random logic simulation (see Section II-B). In Fig. 4, lighter-colored edges represent LUT pairs with larger errors. Then, QUADOL starts an iterative process, illustrated in Figs. 4(c)–(f). In the first iteration, half of the LUT pairs with the largest errors are pruned, indicated by the red crosses in Fig. 4(c). The remaining LUT pairs are referred to as *candidate LUT pairs*. Then, as shown in Fig. 4(d), QUADOL searches for k maximum matching solutions, where $k = 2$ in this example. For each solution, all matched LUT pairs are merged, resulting in k approximate circuits. These circuits are evaluated via random simulation to get their errors. If their errors all exceed the given error bound e_b , half of the current candidate LUT pairs with larger errors are further pruned, indicated by the green crosses in Fig. 4(e). Otherwise, if any circuit satisfies the error bound, half of the previously pruned LUT pairs with smaller errors are reconsidered as the candidate LUT pairs in the next iteration, indicated by the dashed edges in Fig. 4(f). This binary search-inspired process continues until the candidate LUT pairs can neither be increased nor decreased. Then, k maximum matchings are randomly constructed over the current candidate LUT pairs, and the one with the lowest output error is selected as the final solution (see Fig. 4(g)).

E. Flow of QUADOL+

Since QUADOL explores a new dimension, *i.e.*, approximately merging LUT pairs into dual-output LUTs, which is never considered by the previous ALS methods, it can be integrated with any existing ALS methods to strengthen them. Thus, we also propose QUADOL+, a generic framework to integrate QUADOL into existing ALS methods. Given an exact netlist and several other ALS methods, QUADOL+ first applies

these ALS methods to the exact netlist to generate many intermediate approximate netlists. Then, it calls QUADOL to further optimize the intermediate netlists. Finally, it returns the best result optimized by QUADOL as the final result.

IV. EXPERIMENTAL RESULTS

This section presents the experimental results. We conduct all experiments on a 64-core AMD Threadripper PRO 5995WX CPU running at 2.7GHz with 512GB RAM and use 32 cores for our experiments. In our experiments, we set the inputs for each circuit to be uniformly distributed. Each random logic simulation used for measuring the error at the POs uses 131,072 (2^{17}) uniformly-distributed input patterns. The *area ratio*, defined as the number of LUTs in the approximate circuit over that of the exact one, and the runtime of various approaches are reported. The number of maximum matching solutions per iteration, k , is set to 16. All generated approximate circuits were successfully synthesized using AMD Xilinx Vivado 2022.2 [23].

A. Comparison with Prior ALS Methods

In this section, we compare QUADOL and QUADOL+ with two state-of-the-art ALS methods for FPGA, ALSRAC [17] and Xiang’s method [18]. Both ALSRAC and Xiang’s method are executed 10 times per error constraint. The best result among the 10 trials along with its corresponding runtime is reported for comparison. The intermediate AIGs from the best trial of ALSRAC are mapped to LUT-6 netlists, which are used as the intermediate netlists for QUADOL+. Both QUADOL and QUADOL+ are executed once.

Some large circuits from the EPFL [24] and IWLS [25] benchmark suites are used to evaluate the performance of QUADOL and QUADOL+. Table IV lists these circuits, along with their AIG node counts and the LUT numbers after mapping. EPFL circuits have already been mapped to LUT-6s. For IWLS circuits, which are not initially mapped to LUT-6s, we fully optimize them and map them to LUT-6s using ABC [26] with the script “*sweep;strash;resyn2 × 10; if -K 6;*”.

TABLE IV: Benchmarks used in our experiments.

IWLS Benchmark			EPFL Arithmetic		
Circuit	#AIG Nodes	#LUTs	Circuit	#AIG Nodes	#LUTs
apex1	1090	490	rd84	500	131
apex3	1143	527	rot	839	188
apex4	1423	799	seq	2952	883
cps	1329	434	table3	755	333
dalu	2626	228	table5	734	291
des	2509	1104	vda	198	237
ex5p	1471	526			

1) *Comparison under ER constraint:* We compare QUADOL+ with ALSRAC and Xiang’s method using IWLS circuits under ER constraint. Table V lists the area ratio and runtime for each circuit, averaged over two ER constraints, 1% and 2%. **Bold** entries highlight the cases where QUADOL+ outperforms both baselines. QUADOL+ consistently outperforms Xiang’s method in both circuit area and runtime. On average, QUADOL+ generates approximate circuits with an area ratio of **73.1%**, further improving Xiang’s method by **17.9%** and ALSRAC by **2.3%**. However,

QUADOL+ is $1.5\times$ slower than ALSRAC because it relies on the intermediate netlists generated by ALSRAC for further optimization. Hence, its runtime includes both the time for the best-performing ALSRAC trial and that of QUADOL to process the intermediate netlists.

Note that in this experiment, QUADOL is excluded as it outperforms none of the baselines under the ER constraint. This is because QUADOL only considers HD during approximate LUT merging, ignoring the input probabilities of merged LUT pairs, which are important under the ER constraint. In contrast, QUADOL+ can further enhance the performance of other ALS methods, leading to improved results.

TABLE V: Comparison of Xiang’s method [18], ALSRAC [17], and QUADOL+ under ER constraint.

Circuit	Average Area Ratio			Average Runtime (s)		
	[18]	[17]	QUADOL+	[18]	[17]	QUADOL+
apex1	95.7%	85.3%	84.0%	1946	973	1459
apex3	97.1%	90.4%	89.5%	2211	1106	1658
apex4	97.6%	93.2%	92.8%	5396	2698	4047
cps	55.5%	35.9%	34.9%	730	365	547
dalu	85.1%	73.9%	72.6%	305	153	229
des	96.0%	58.9%	57.5%	8124	4062	6093
ex5p	90.1%	75.6%	69.8%	2098	1049	1574
rd84	97.3%	70.2%	61.5%	115	58	86
rot	97.1%	89.1%	88.0%	152	76	114
seq	89.6%	79.2%	77.3%	5385	2693	4039
table3	93.2%	69.1%	67.9%	917	459	688
table5	92.3%	74.7%	72.7%	664	332	498
vda	96.0%	84.6%	82.3%	449	225	337
Arithmetic	91.0%	75.4%	73.1%	2192	1096	1644

2) *Comparison under MRED constraint:* We compare QUADOL and QUADOL+ with ALSRAC using EPFL circuits under MRED constraint. Xiang’s method is excluded as it does not support MRED. Table VI lists the area ratio and runtime, averaged across three MRED thresholds, 0.05%, 0.1%, and 0.5%. QUADOL often performs worse than ALSRAC, as ALSRAC works on AIG, which is more fine-grained than the LUT netlist that QUADOL works on, hence offering more optimization opportunity. In contrast, the combination of QUADOL and ALSRAC, *i.e.*, QUADOL+, improves ALSRAC’s area ratio by **6.3%** by utilizing dual-output LUTs, a feature not exploited by ALSRAC. Notably, for the *div* circuit, QUADOL+ reduces **22.0%** more area over ALSRAC. For the *multiplier* circuit, QUADOL+ shows less than 1% improvement, as ALSRAC has already reduced its area to 9.5% of the exact circuit. Nonetheless, QUADOL+ identifies and merges several LUT pairs within the error constraints, further decreasing the area. For the *sqrt* circuit, QUADOL+ shows no improvement over ALSRAC, for which ALSRAC already achieves a 96.8% area reduction. ALSRAC achieves such a large area reduction because it aggressively simplifies the logic cones of the least significant output bits, which dominate the area in *sqrt*.

In terms of delay, neither QUADOL nor QUADOL+ affects the delay of an FPGA design, as their key method, approximate LUT merging, only merges existing LUTs without affecting the length of each path in the design.

B. Comparison with Prior Approximate Multipliers for FPGA

In this section, we study the performance of QUADOL+ in synthesizing FPGA-based approximate multipliers. We use approximate multipliers from EvoApproxLib [27], a widely-used

TABLE VI: Comparison of ALSRAC [17], QUADOL, and QUADOL+ under MRED constraint.

Circuit	Average Area Ratio			Average Runtime (s)		
	[17]	QUADOL	QUADOL+	[17]	QUADOL	QUADOL+
div	96.0%	81.0%	74.0%	1711	7893	11244
log2	89.5%	91.7%	82.8%	912	24500	39155
multiplier	9.5%	85.9%	9.0%	18175	16638	24958
sin	96.0%	91.3%	88.9%	72	343	507
sqrt	3.2%	70.0%	3.2%	2725	2105	3067
square	14.8%	74.1%	13.6%	6284	9601	13168
Arithmetic	51.5%	82.3%	45.2%	4980	10180	15350

library of approximate adders and multipliers, as intermediate netlists for QUADOL+. We compare the multipliers generated by QUADOL+ with those from EvoApproxLib, LM [12], SMA [9], Ca [14], and Guo [13]. For each type of multiplier, the Pareto-optimal designs from QUADOL+ in terms of area and error are collected.

1) *MRED as the error metric*: We compare the Pareto-optimal approximate multipliers obtained by QUADOL+ with those from [9], [12]–[14] using MRED as the error metric. 8-bit and 16-bit unsigned multipliers are considered. The results are shown in Figs. 5(a) and (b), respectively. Each red point in the figures denotes a Pareto-optimal design from QUADOL+, while the other points correspond to prior designs. The ideal points are located in the bottom-left corners of the figures. We can see that QUADOL+’s results dominate most prior designs. It is worth noting that some prior designs use CARRY4 modules, which are carry-chain components in FPGA commonly used to reduce LUT usage [28]. In contrast, QUADOL+ uses only LUTs. This further highlights the superior quality of QUADOL+ to the existing designs. The better performance of QUADOL+ is partly because the 8-bit and 16-bit approximate multipliers in [12]–[14] are constructed from hand-crafted 4-bit unsigned approximate multipliers, which overlook the optimization opportunities across the 4-bit blocks. In contrast, QUADOL+ can automatically identify and merge LUT pairs in the whole circuit, leading to further area reduction.

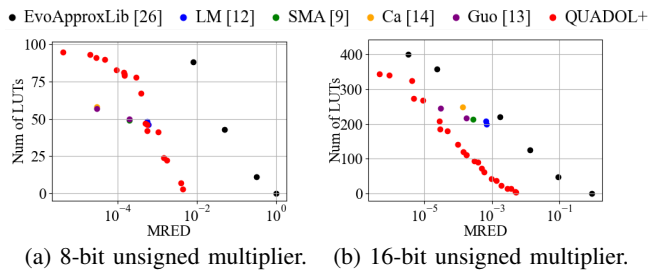


Fig. 5: Comparison among multipliers from prior works and QUADOL+ with MRED as the error metric.

As shown in Fig. 5, when the MRED ranges from 10^{-3} to 10^{-1} , QUADOL+ produces some designs with very few LUTs. This is because their least significant output bits are directly truncated to constants. For the most significant output bits, since one dual-output LUT can implement 2 Boolean functions, two dual-output LUTs can represent up to 4 such bits, which is enough for an MRED in the above range that is relatively large.

To further evaluate the real-world applicability of QUADOL+, we integrate one of its Pareto-optimal designs into a *Sobel edge detector*. The selected design is an 8-bit



(a) Original image. (b) Exact. (c) Approximate. Fig. 6: Comparison of results from Sobel edge detectors using exact and approximate multipliers, respectively.

unsigned approximate multiplier using 42 LUTs with an MRED of 0.057. Figs. 6(a)–(c) show the original image, the output of the exact Sobel edge detector, and the result using the approximate multiplier, respectively. The approximate image achieves a PSNR of 35.61 dB compared to the exact version, and the quality degradation is almost invisible.

2) *ER as the error metric*: We compare the Pareto-optimal approximate multipliers obtained by QUADOL+ with those from EvoApproxLib [27] using ER as the error metric. Designs from [9], [12]–[14] are excluded as they do not use ER as the error metric. Figs. 7(a) and (b) show the comparisons for four types of unsigned multiplier: 1) 7-bit, 2) 8-bit, 3) 8×2 , and 4) 8×3 . It can be seen that the approximate multipliers from EvoApproxLib are located at regions with large ERs, while QUADOL+’s results are at regions with small ERs. Although approximate computing tolerates some error, large errors are still unacceptable in practical applications. Hence, designs with small ERs are preferred, indicating that QUADOL+ produces more practically useful multipliers than EvoApproxLib. Approximate multipliers with large ERs are not produced by QUADOL+ because when ER is large, many POs of the intermediate netlists used within QUADOL+ are constants or just copies of the PIs, which results in less opportunity for QUADOL to perform approximate LUT merging.

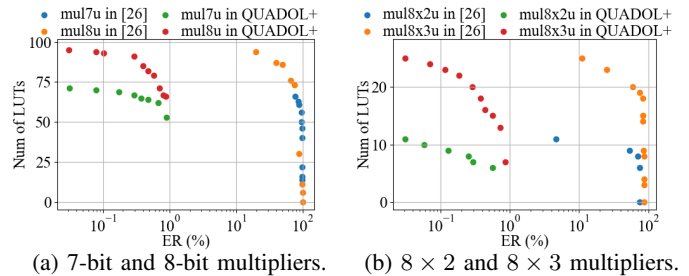


Fig. 7: Comparison among multipliers from EvoApproxLib [27] and QUADOL+ with ER as the error metric.

V. CONCLUSION

This paper proposes QUADOL, a quality-driven ALS method that uses approximate LUT merging to minimize the number of LUTs required in FPGA designs. To maximize the number of merged LUT pairs, the LUT pair selection problem is formulated as a maximum matching problem for solving, and a binary search-based algorithm is proposed to accelerate the search. Since QUADOL explores a new dimension, *i.e.*, approximately merging a LUT pair into a dual-output LUT, we also introduce QUADOL+, a generic framework to integrate QUADOL into existing ALS methods. Experimental results showed that QUADOL+ outperforms prior works in reducing LUT count. In the future, we plan to extend QUADOL and QUADOL+ to support other dual-output LUT architectures.

REFERENCES

- [1] M. Ahmadinejad *et al.*, “Energy- and quality-efficient approximate multipliers for neural network and image processing applications,” *TETC*, vol. 10, no. 2, pp. 1105–1116, 2022.
- [2] Xilinx, “LUT6_2,” in *UltraScale Architecture Libraries Guide*. AMD, Inc., 2024, ch. 5, pp. 437–441.
- [3] B. Gaide *et al.*, “Xilinx adaptive compute acceleration platform: Versal™ architecture,” in *FPGA*, 2019, pp. 84–93.
- [4] J. Chromczak *et al.*, “Architectural enhancements in Intel® Agilex™ FPGAs,” in *FPGA*, 2020, pp. 140–149.
- [5] Intel, “Adaptive Logic Module,” in *Intel® Arria® 10 Device Overview*. Intel, Inc., 2018, pp. 17–18.
- [6] F. Wang *et al.*, “Dual-output LUT merging during FPGA technology mapping,” in *ICCAD*, 2020, pp. 1–9.
- [7] B. S. Prabhakaran *et al.*, “DeMAS: An efficient design methodology for building approximate adders for FPGA-based systems,” in *DATE*, 2018, pp. 917–920.
- [8] S. Ullah *et al.*, “Area-optimized low-latency approximate multipliers for FPGA-based hardware accelerators,” in *DAC*, 2018, pp. 1–6.
- [9] —, “SMAApproxLib: Library of FPGA-based approximate multipliers,” in *DAC*, 2018, pp. 1–6.
- [10] Y. Guo *et al.*, “Small-area and low-power FPGA-based multipliers using approximate elementary modules,” in *ASP-DAC*, 2020, pp. 599–604.
- [11] S. Yao and L. Zhang, “FHAM: FPGA-based high-efficiency approximate multipliers via LUT encoding,” in *ICCD*, 2022, pp. 487–490.
- [12] —, “Hardware-efficient FPGA-based approximate multipliers for error-tolerant computing,” in *ICFPT*, 2022, pp. 1–8.
- [13] Y. Guo *et al.*, “Hardware-efficient multipliers with FPGA-based approximation for error-resilient applications,” *TCAS*, pp. 1–12, 2024.
- [14] S. Ullah *et al.*, “High-performance accurate and approximate multipliers for FPGA-based hardware accelerators,” *TCAD*, vol. 41, no. 2, pp. 211–224, 2022.
- [15] Y. Wu *et al.*, “Approximate logic synthesis for FPGA by wire removal and local function change,” in *ASP-DAC*, 2017, pp. 163–169.
- [16] B. S. Prabhakaran *et al.*, “ApproxFPGAs: Embracing ASIC-based approximate arithmetic components for FPGA-based systems,” in *DAC*, 2020, pp. 1–6.
- [17] C. Meng *et al.*, “ALSRAC: Approximate logic synthesis by resubstitution with approximate care set,” in *DAC*, 2020, pp. 1–6.
- [18] Z. Xiang *et al.*, “Approximate logic synthesis for FPGA by decomposition,” in *Approximate Computing*, W. Liu and F. Lombardi, Eds. Springer, 2022, ch. 7, pp. 149–174.
- [19] M. Barbareschi *et al.*, “FPGA approximate logic synthesis through catalog-based AIG-rewriting technique,” *JSA*, vol. 150, p. 103112, 2024.
- [20] I. Scarabottolo *et al.*, “Approximate logic synthesis: A survey,” *Proc. IEEE*, vol. 108, no. 12, pp. 2195–2213, 2020.
- [21] J. Chen *et al.*, “On computing centroids according to the p -norms of Hamming distance vectors,” in *ESA*, vol. 144, 2019, pp. 28:1–28:16.
- [22] Z. Galil, “Efficient algorithms for finding maximum matching in graphs,” *ACM Comput. Surv.*, vol. 18, no. 1, pp. 23–38, March 1986.
- [23] Xilinx, in *Vivado Design Suite User Guide*. AMD, Inc., 2024.
- [24] EPFL, “The EPFL combinational benchmark suite,” <https://www.epfl.ch/labs/lisi/page-102566-en-html/benchmarks/>, 2023.
- [25] IWLS, “IWLS 2005 benchmarks,” <https://iwls.org/iwls2005/benchmarks.html>, 2005.
- [26] A. Mishchenko *et al.*, “ABC: a system for sequential synthesis and verification, release 90703,” <http://people.eecs.berkeley.edu/~alanmi/abc/>, accessed July 3, 2019.
- [27] V. Mrazek *et al.*, “EvoApproxLib: Extended library of approximate arithmetic circuits,” in *WOSET*, 2019, p. 10.
- [28] Xilinx, “CARRY4,” in *Vivado Design Suite 7 Series FPGA and Zynq 7000 SoC Libraries Guide*. AMD, Inc., 2024, ch. 5, pp. 302–304.