electronics

MDPI

Article

An Efficient Multi-Output LUT Mapping Technique for Field-Programmable Gate Arrays

Sheng Lu [1], Liuting Shang [1], Qianhou Qu [1], Sungyong Jung [2], Qilian Liang [1] and Chenyun Pan [1,*]

[1] Department of Electrical Engineering, University of Texas at Arlington, Arlington, TX 76019, USA; sxl2408@mavs.uta.edu (S.L.); liuting.shang@mavs.uta.edu (L.S.); qxq9423@mavs.uta.edu (Q.Q.); liang@uta.edu (Q.L.)

[2] Department of Electrical Engineering and Computer Science, South Dakota State University, Brookings, SD 57007, USA; sungyong.jung@sdstate.edu

* Correspondence: chenyun.pan@uta.edu; Tel.: +1-817-272-5720

Abstract: The use of multi-output look-up tables (LUTs) is a widely adopted approach in contemporary commercial field-programmable gate arrays (FPGAs). Larger LUT configurations (e.g., six-input LUTs) can be partitioned into smaller LUTs (e.g., two five-input LUTs, maintaining a total input count of less than six). This capability of generating a second output from a larger LUT is not only crucial for reducing logic cell count and enhancing the utilization efficiency of logic resources—thus conserving area—but also plays a key role in optimizing system-level delays and energy consumption. In this paper, we propose an efficient multi-output LUT mapping technique, incorporating several highly efficient technology mapping algorithms, which focus on optimizing the mapping from an interconnection perspective as alternatives to directly merging smaller LUTs. These algorithms include a side-fanout insertion algorithm, and a runtime multi-output cut generation algorithm. The proposed methods improve mapping efficiency and enhance performance. The benchmarking results demonstrate that the dual-output mapping algorithms achieve LUT area reductions of up to 35% and 6%, compared to the state-of-the-art ABC six-input, single-output LUT mapping technique and previous work focusing on dual-output LUT mapping techniques that optimize cut generation parameters. Moreover, FPGA system-level simulations also show that area, delay, and energy can all be optimized based on this multi-output mapping technique.

Keywords: FPGA; technology mapping; dual-output LUT; CLB; VTR tool; ABC tool; Ultrascale+; Versal; benchmark

check for updates

Academic Editor: Alexander Barkalov

Received: 4 April 2025
Revised: 24 April 2025
Accepted: 25 April 2025
Published: 27 April 2025

Citation: Lu, S.; Shang, L.; Qu, Q.; Jung, S.; Liang, Q.; Pan, C. An Efficient Multi-Output LUT Mapping Technique for Field-Programmable Gate Arrays. *Electronics* **2025**, *14*, 1782. https://doi.org/10.3390/electronics14091782

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Background

A field-programmable gate array (FPGA) is a reconfigurable circuit system that can implement various circuit functions using an array of configurable logic blocks (CLBs). Compared to application-specific integrated circuits (ASICs), FPGAs reduce initial development costs, and when compared to central processing units (CPUs), they offer better performance, making them widely used in both industry and academia. Traditional FPGAs consist of basic computing units known as single-output look-up tables (LUTs), where the LUT, the output flip-flop (FF), and the multiplexer (MUX) form a basic logic element (BLE), and multiple BLEs constitute a CLB [1,2], as shown in Figure 1 [3]. However, in practice, when a circuit is mapped to a LUT by logic synthesis tools (e.g., ABC), some LUTs may have fewer inputs than expected. For instance, in a six-input LUT system, some mapped LUTs may occupy only three inputs, leading to inefficient resource utilization

of physical six-input LUTs. As illustrated by the simplified circuit directed acyclic graph (DAG) in Figure 2, Figure 2a presents the mapping result under a six-input, single-output scheme. Here, the network is an AIG (And-Inverter Graph), where each node in the graph represents an AND gate [4–6]. However, if the fanout number and LUT can be extended to a dual-output configuration, as shown in Figure 2b, the two single-output LUTs can be merged into a single dual-output LUT, reducing the area without degrading the delay.

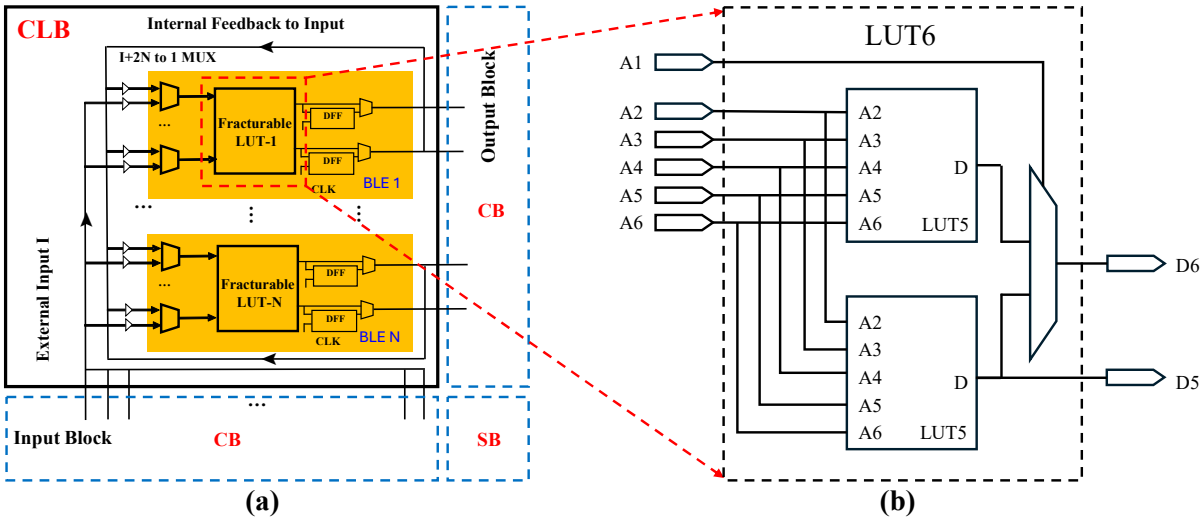


Figure 1. (**a**) Generic FPGA architecture with fracturable LUTs and (**b**) detailed structure of example LUT from Xilinx UltraScale+ series [3].

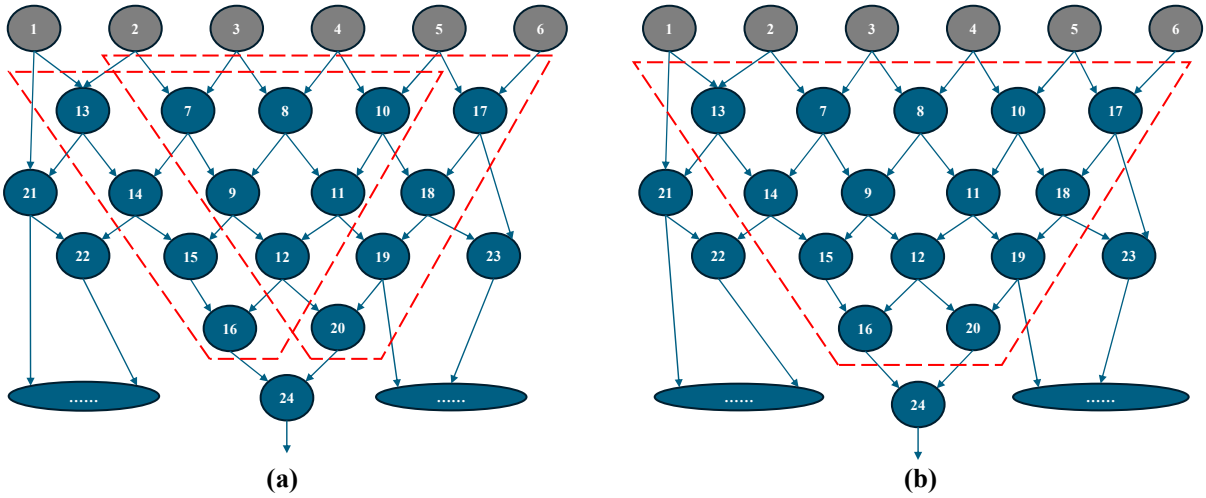


Figure 2. An example of mapping results under six-input (**a**) single- and (**b**) dual-output LUT schemes. Here, the network is an AIG, where each node in the graph represents an AND gate.

To address this issue, both industry and academia have proposed various methods for improvement. For example, Xilinx's Ultrascale+ series can fracture a six-input LUT into two five-input, single-output LUTs [3], while Intel's Stratix series can decompose an eight-input LUT into several smaller LUTs with fewer than seven inputs [7]. Similarly, the VTR FPGA simulation platform provides architecture files that allow large LUTs to be split into multiple smaller LUTs [8], which enhances logic resource utilization and reduces the circuit area. Additionally, several studies have pointed out that emerging multi-output reconfigurable logic units may provide potential performance improvements compared to traditional LUTs [9–12], leading to more application possibilities for multi-output mapping techniques.

In academia, the ABC tool is commonly used to map circuits to LUTs [13], providing an efficient and fast single-output LUT mapping based on graph theory. Based on this platform, some works have attempted to implement multi-output LUT generation at the technology mapping level, replacing the simple merging of smaller-input LUTs. Jang et al. proposed a method called Mapping with Structural Choices (MSCs), combined with WireMap, a heuristic metrics-based approach, to merge small-sized LUTs into a single dual-output LUT [14]. This technique achieves less than 10% LUT number reductions. To enhance LUT area savings, Wang et al. proposed a double-output cut enumeration process based on the ABC single-output cut enumeration process, optimizing parameter selection and achieving significant area savings [15]. However, this method highly relies on the parameters during the dual-output cut generation, and a larger priority cut number is required for certain benchmark circuits, which increases time complexity and computational cost. Similar methods have also proposed to enhance the mapping efficiency of multi-output LUTs [16–20].

In this paper, we propose an efficient multi-output LUT generation method based on the interconnection relationships between LUTs. Specifically, we perform side-fanout insertion on each LUT at each layer based on the ABC single-output mapping result to cover the leaves of the next layer with the minimum number of LUTs, while merging smaller input LUTs within the same layer. Then, we perform LUT expansion on the remaining single-output LUTs layer by layer, transforming them into multi-output LUTs to further reduce the number of LUTs. The proposed mapping technique is universally applicable and can handle larger-scale circuits with better LUT area saving.

The main contributions of this paper are as follows:

- We propose an efficient multi-output LUT generation method, which includes a side-fanout insertion algorithm and a runtime multi-output LUT expansion algorithm.
- We propose a bottom-up layer-based cut merging and reduction method, which can efficiently utilize the features of multi-output cuts without worsening the delay.
- We develop a comprehensive FPGA design framework with the proposed mapping algorithm to demonstrate the potential advantages in terms of area, delay, and energy at the system level.

The rest of the paper is organized as follows: In Section 2, we introduce the proposed multi-output mapping technique, which includes side-fanout insertion, runtime multi-output LUT expansion, and a LUT merging and covering algorithm to improve mapping efficiency. Section 3 describes the experimental setup, including details of the benchmark netlists, FPGA architecture specifications, and simulation environment. Section 4 presents a detailed analysis of the results, with comparisons between our proposed technique and state-of-the-art approaches. Sections 5 and 6 discuss a comparison of the proposed work with state-of-the-art methods and conclude with suggestions for future research.

2. Proposed Methods

In academic research, ABC is widely used for high-quality and efficient logic synthesis and FPGA LUT technology mapping [13]. In this section, building upon the ABC framework, we first introduce the fundamental concept of the cut-based LUT generation flow, followed by the proposed multi-output LUT mapping flow. This includes the side-fanout insertion and multi-output cut generation methods and algorithms, which emphasize interconnection-level high-quality multi-output cut generation. The overall flow of the proposed approach is depicted in Figure 3, with each step of the process discussed in detail in the following sections.

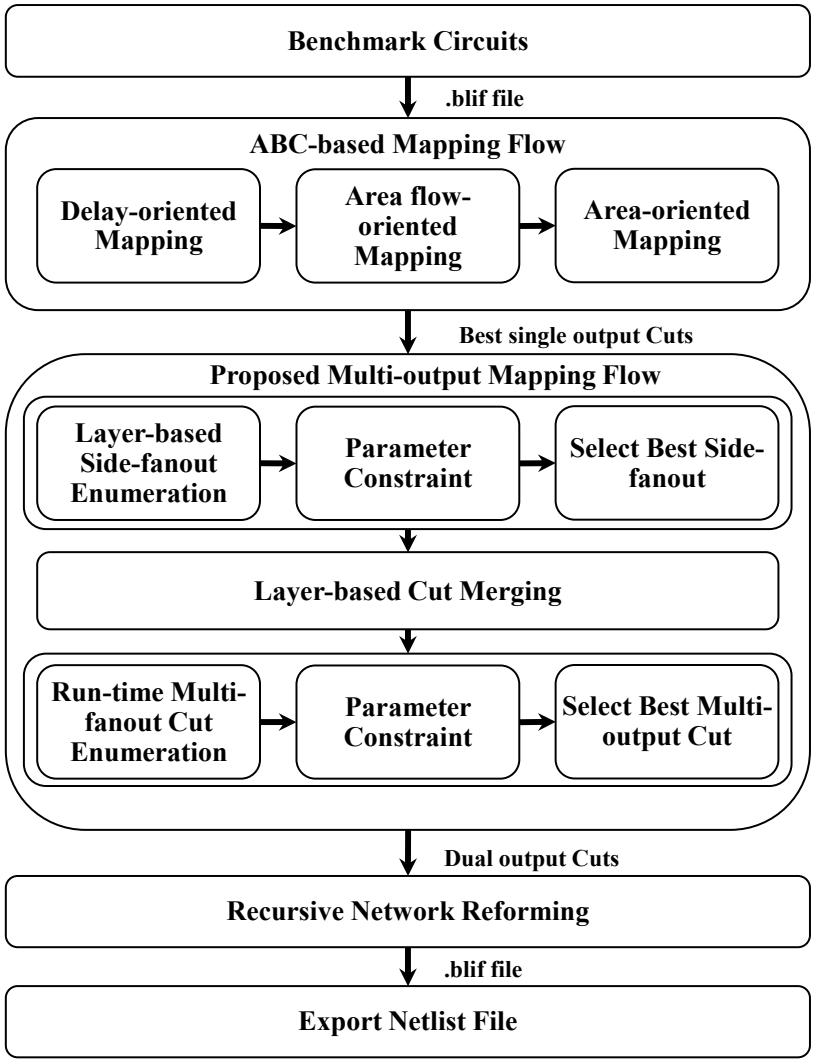


Figure 3. Overview of the proposed multi-output mapping technique.

2.1. Single-Output Cut Enumeration

The standard approach to computing K-feasible cuts in an AIG follows a systematic pass from Primary Inputs (PIs) to Primary Outputs (POs). The cut set for an AND node is derived by merging the cut sets of its children, incorporating the trivial cut. A cut is considered K-feasible if it contains no more than K leaves and is dominated if another cut at the same root subsumes its coverage [4–6]. In a K-bounded subject graph, K-feasible cuts for all nodes are enumerated in an interconnection order to ensure that fanin cuts are available when computing the root's cut set.

Defining $\Phi(\mathrm{n})$ as the K-feasible cut set of the node n, the operator $\oplus$ is defined as follows:

$$\Phi(\mathrm{n}_1) \oplus \Phi(\mathrm{n}_2) = \{c_1 \cup c_2 | c_1 \in \Phi(\mathrm{n}_1),\ c_2 \in \Phi(\mathrm{n}_2),\ |c_1 \cup c_2| \leq K\} \tag{1}$$

Then, the cut set of node n can be computed by combining the cut sets of its fanin nodes as follows:

$$\Phi(\mathrm{n}) = \left\{ \begin{array}{ll} \{\{n\}\} & : n \in PIs \\ \{\{n\}\} \cup [\Phi(\mathrm{n}_1) \oplus \Phi(\mathrm{n}_2)] & : otherwises \end{array} \right\} \tag{2}$$

Here, n_1 and n_2 are the fanin nodes of node n. Using this formula, we can generate the cut set for all nodes in the AIG network, progressing from PIs to POs. By selecting the optimal cut for each node based on factors, such as delay, area, and power, we can construct a mapped LUT network through a recursive process that reforms the structure from POs back to PIs, following the reverse direction of cut generation.

2.2. Cut Interconnection-Based Multi-Output LUT Generation

As introduced in the background section, multi-output LUTs can merge smaller single-output LUTs to enhance resource utilization efficiency. In this section, we present a methodology for generating multi-output LUTs based on the interconnection of single-output cuts. We propose two algorithms—the side-fanout insertion algorithm and the runtime multi-output cut generation algorithm—designed to effectively reduce the number of LUTs while maintaining delay performance.

2.2.1. Side-Fanout Insertion Algorithm

In the traditional single-output cut scheme, the cut structure resembles an inverted triangle, where all signals from the leaves propagate exclusively to the bottom output, which results in repeated nodes across different cuts. While this enumeration method is fast and efficient, it overlooks critical side-node interconnections that could be leveraged by multi-output cuts. As illustrated in Figure 4, side-fanouts of a single-output cut can serve as inputs for cuts in the next layer. By allowing a multi-output cut to designate these side-fanouts as additional outputs, the number of LUTs can be reduced during the recursive network reformation process from POs to PIs.

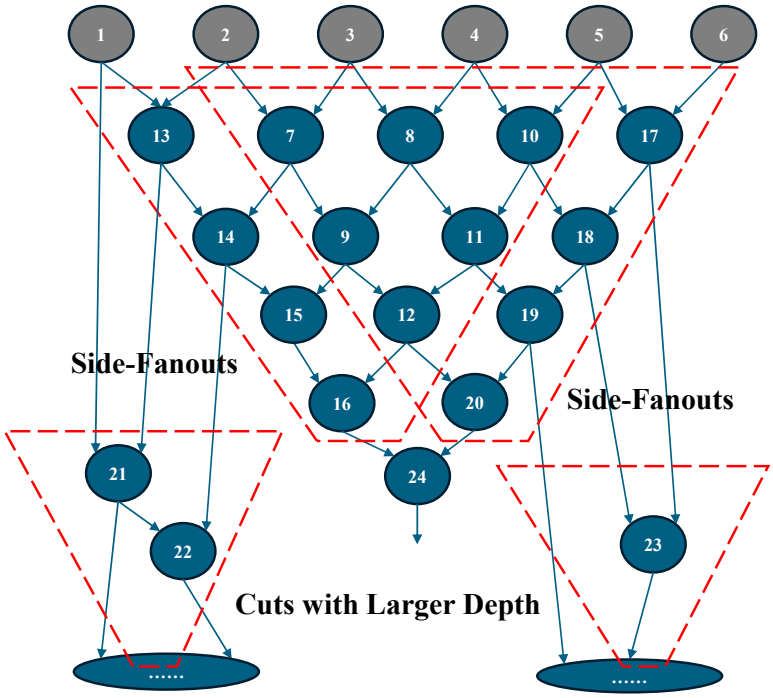


Figure 4. An example illustrating the potential application of side-fanouts and the generation of multi-output cuts.

In detail, we define the fanins of cuts at the same level i as FI_i, as shown in Figure 5b. For each single cut generated by ABC mapping, side-fanouts are considered for inclusion. Specifically, for a given cut that provides signals for FI_i, all possible side-fanouts are enumerated. Among these candidates, the side-fanout with the highest coverage of FI_i is selected as the second output of the cut. Once selected, FI_i is updated by removing the

covered fanins, and this process continues iteratively until all nodes in FI_i are covered. During the covering process, there is a high probability that the total number of cuts incorporating a second fanout is lower than that of the original single-output cuts, as indicated by the red-dashed block in Figure 5b. This reduction directly contributes to decreasing the overall number of LUTs. Afterward, the next-level fanins, FI_{i+1}, are generated based on the newly formed cut layer. We implement this process using a recursive loop, ensuring that the iteration continues until FI_i contains only PIs. The pseudocode for this process is presented in Algorithm 1.

Algorithm 1 Side-fanout Insertion Algorithm

Require: Layer leaves from POs to PIs
Ensure: Insert side-fanout and generation of the next layer's leaves

```
function SIDEFANOUTINSERTION(layer_leaves)
    Step 1: Enumerate possible side-fanouts
    for each node in leaves do
        cut_fanout_list = FindSideFanouts(current_node_cut)
    end for
    Step 2: Leaves coverage and cuts reduction
    while exist layer_leaves not visited do
        best_node: cut with most coverage with optimized side-fanout
        for each fanout_node in best_node cut do
            mark fanout_node as visited
        end for
    end while
    Step 3: Generate the next layer's leaves
    Initialize layer_leaves_next as empty
    for each selected node in leaves do
        Add current_cut leaves to layer_leaves_next
    end for
    Step 4: Recursive loop
    if layer_leaves_next is not empty then
        SideFanoutInsertion(layer_leaves_next)
    end if
end function
```

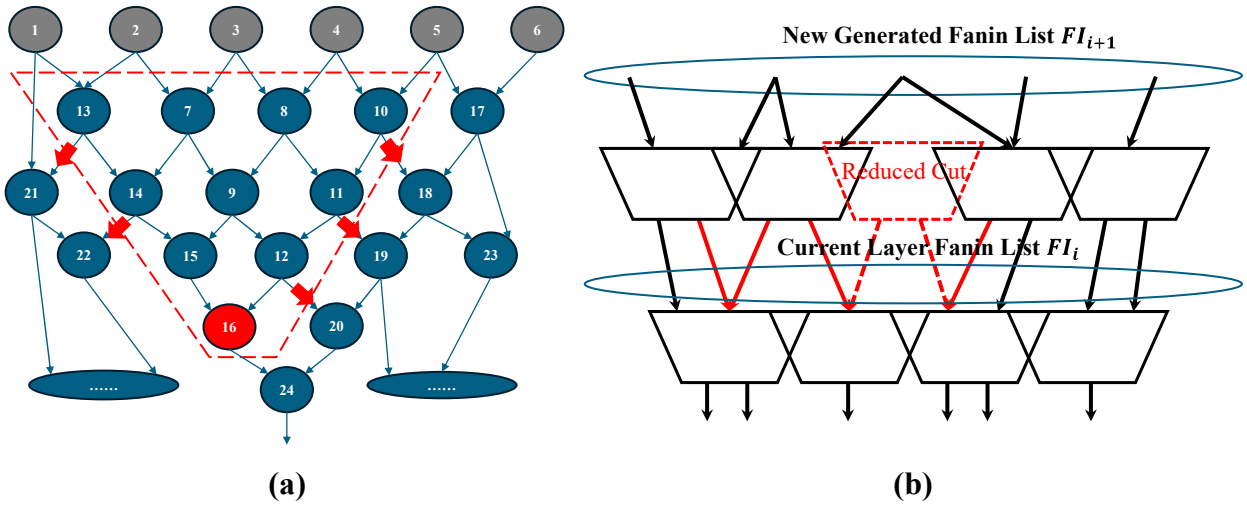


Figure 5. Layer-based side-fanout insertion and LUT reduction. (**a**) Enumeration of possible side-fanouts of a single-output cut; (**b**) merging and reduction of same layer cuts.

Meanwhile, within the same layer of cuts, multiple cuts exist with identical input net nodes. These cuts can be merged into a single dual-output LUT if the total number of inputs, I_{total}, does not exceed K. Thus, following the side-fanout insertion and coverage process, a layer-based cut merging procedure is executed to efficiently combine cuts with the same input pins. This approach not only reduces the number of LUTs but also improves system-level placement and routing considerations in FPGA implementation. Since cuts with identical inputs are merged, they tend to remain physically close, thereby enhancing spatial locality. In terms of C language implementation, this process is analogous to side-fanout insertion but is significantly simpler. A nested for loop can be employed to traverse the cuts within the same layer and perform the merging operation effectively.

2.2.2. Runtime Multi-Output LUT Expansion Algorithm

The side-fanout insertion and same-layer cut merging processes reduce the number of cuts based on the single-output cuts initially generated by ABC. In this section, we aim to expand the remaining unchanged single-output cuts after the previous processes. The key concept of this approach is to actively generate multi-output cuts, allowing multiple roots within the same cut. This process consists of two main components. The first component involves generating multi-output cuts using an optimized Breadth-First Search (BFS) recursive algorithm [21], which efficiently explores potential multi-output cuts for a given root. The second component is a layer-based coverage and merging procedure, similar to the side-fanout insertion algorithm.

As shown in Figure 6, the target root node is highlighted in red with an index of 16. The search algorithm first explores the fanout nodes of node 16, leading to the exploration of node 24. However, node 24 is not added to the cut because only nodes with a level that is not larger than the current root node's level are considered, ensuring compatibility with the layer-based analysis scheme.

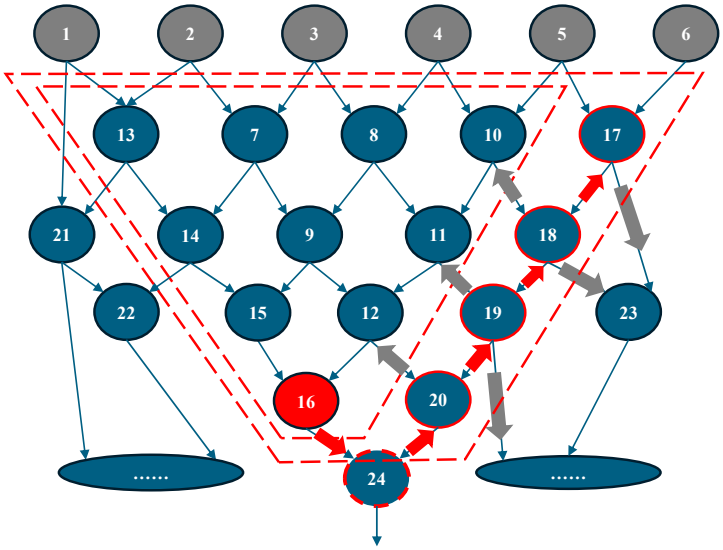


Figure 6. An example of the BFS-based multi-output cut generation algorithm. The red arrow represents successfully explored nodes, whereas the dark nodes indicate failed attempts—either because the node already exists in the current cut or because adding it would exceed the I/O number limit.

For the explored nodes with a level greater than the root node, only their fanin nodes are further searched. In this case, node 20 is added to the cut because it shares the same level as node 16, and the I/O count of the expanded cut does not exceed the predefined limit (6 inputs, 2 outputs). Next, nodes 12 and 19 are explored. Since node 20 has only one fanout, node 24, which has already been visited, only fanins of node 20 are further explored. For nodes with the same level as the root node, as well as those with a lower level, both their fanins and fanouts are explored. This process continues iteratively until there are no more candidate nodes to explore or the I/O size of the cut exceeds the predefined limit. The pseudocode for multi-output cut generation is presented in Algorithm 2.

Algorithm 2 Recursive Cut Expansion Algorithm

```
Require: Logic network represented as a set of nodes and cuts
Ensure: Recursive expansion of cuts satisfy fanin/fanout/level constraints
Declare Constraints: fanin/fanout/level/max enumeration number
function ExpandCut(current_node, level, current_cut)
    if current_node is not visited then
        Mark current_node as visited
        if satisfy Constraints then
            if NotRepeated(current_cut, cut_list) then
                Add current_cut to cut_list
            end if
            if curr_node.Level <= root_level then
                for each fanout node do
                    current_cut add fanout_node
                    ExpandCut(fanout_node, new_level, current_cut)
                end for
                for each fanin node do
                    current_cut add fanin_node
                    ExpandCut(fanin_node, new_level, current_cut)
                end for
            else
                for each fanin node do
                    current_cut add fanin_node
                    ExpandCut(fanin_node, new_level, current_cut)
                end for
            end if
        else
            Remove current_node from current_cut
        end if
        Add current_cut to cut_list
    end if
end function
```

The remaining part of the runtime multi-output LUT expansion algorithm is similar to the side-fanout insertion process, as both rely on a bottom-up recursive, layer-based cut merging and reduction strategy. A critical requirement for multi-output selection is that the chosen multi-output cut must minimize key parameters, such as area, delay, and edge count. Based on practical experiments, we adopt an area calculation method similar to ABC's exact area flow. This approach ensures that the area of each multi-output cut is accurately evaluated, guaranteeing that cut expansion does not degrade the area efficiency compared to the original single-output cuts. The pseudocode for this process is presented in Algorithm 3.

Algorithm 3 Runtime Multi-output Cut Expanding Algorithm

```
Require: Layer leaves (initial nodes for expansion)
Ensure: Recursive expansion of cuts and generation of the next layer's leaves
function RunTimeExpand(layer_leaves)
    Step 1: Recursive expansion based on ExpandCut
    for each node in leaves do
        Initialize current_cut = ABC_Best(current_node)
        cut_list = ExpandCut(current_node, level, current_cut)
        BestMultiCut = BestMultiCutSel(cut_list, leaves)
        Assign BestMultiCut to current_node
    end for
```

Algorithm 3 *Cont.*

```
    Step 2: Leaves coverage and cuts reduction
    while exist leaves not visited do
        best_node = CutWithMaxCoverage(leaves_unvisited)
        for each fanout_node in best_node cut do
            mark fanout_node as visited
        end for
    end while
    Step 3: Generate the next layer's leaves
    Initialize layer_leaves_next as empty
    for each selected node in leaves do
        Add current_cut leaves to layer_leaves_next
    end for
    Step 4: Recursive loop
    if layer_leaves_next is not empty then
        RunTimeExpand(layer_leaves_next)
    end if
end function

function BestMultiCutSel(cut_list, leaves)
    Goal: Ensure the selected multi-output cut has the smallest area
    for each cut in cut_list do
        cut_area = AreaCountRec(current_node)
    end for
    Cut_best = cut_with_minArea
end function
```

3. Experimental Configurations and Settings

This work is based on the well-known open-source synthesis tool, ABC, which can be found in [22]. ABC is developed in the C programming language and provides various functions for FPGA mapping. Our dual-fanout FPGA mapping method is implemented within this framework.

To evaluate our proposed mapping technique against existing works, we adopt the same FPGA architectures as referenced in [15], specifically Xilinx's Ultrascale+ and Versal series. The primary distinction between these two architectures lies in their LUT designs, which affect both the maximum number of unique inputs and the maximum number of shared inputs between fractured LUTs. The detailed specifications of these architectures are provided in Table 1, which is consistent with the reference work to ensure a fair comparison. While some parameters may differ slightly from the official Xilinx datasheets [3,23], this alignment maintains consistency in benchmarking and evaluation.

Table 1. Parameters of two typical FPGA series.

FPGA	UltraScale+	Versal
I_{single}	6	6
I_{dual}	5	6
$I_{sharing}$	5	6
K_{dual}	5	6

Additionally, we utilize the full-list EPFL benchmark suite [24], which includes large-scale circuits with up to 200,000 AIG nodes. To ensure a fair comparison with the reference work [15], we adopt two mapping strategies: (1) optimized benchmarks, where circuits are pre-processed using the ABC commands "resyn; resyn2" to refine the circuit structure, as done in the reference work, and (2) original benchmarks, where circuits are used in their unmodified form. These strategies allow for an equitable evaluation of our mapping technique against state-of-the-art methods while demonstrating its universal applicability to any combinational circuit. The detailed benchmark

netlist information is provided in Table 2. In the results section, we will present mapping results from three different mapping techniques: (1) the ABC-based single-output LUT as the baseline, (2) the dual-output mapping result from the reference work, and (3) our proposed multi-output mapping technique.

Table 2. Benchmark netlist information.

Netlist	Netlist Information				Original ABC	Resyn ABC	Original/Resyn Ratio
	Input	Output	AIG Node	Depth	LUT Area	LUT Area	
adder	256	129	1020	255	254	257	101.18%
arbiter	256	129	11,839	87	2722	2722	100.00%
bar	135	128	3336	12	512	512	100.00%
cavlc	8	256	304	3	116	118	101.72%
ctrl	7	26	174	10	29	29	100.00%
dec	8	256	304	3	287	287	100.00%
div	128	128	44,762	4470	22,113	9931	44.91%
hyp	256	128	214,335	24,801	44,508	44,338	99.62%
i2c	147	142	1342	20	353	316	89.52%
int2float	11	7	260	16	51	47	92.16%
log2	32	32	32,060	444	8092	8077	99.81%
max	512	130	2865	287	769	793	103.12%
mem_ctrl	1204	1231	46,836	114	12,084	11,747	97.21%
multiplier	128	128	27,062	274	5927	5922	99.92%
priority	128	8	978	250	210	178	84.76%
router	60	30	257	54	89	76	85.39%
sin	24	25	5416	225	1464	1477	100.89%
sqrt	128	64	24,618	5058	5711	4645	81.33%
square	64	128	18,484	250	3998	3949	98.77%
voter	1001	1	13,758	70	2695	1736	64.42%

Considering that the final performance of a circuit mapped to an FPGA is not solely determined by the LUT mapping phase, but is also significantly influenced by system-level placement and routing, it is essential to verify that the mapping results are compatible with the subsequent placement and routing stages. In other words, we need to ensure that the multi-output LUT maintains its advantages after placement and routing. To address this, we conduct system-level simulations using the open-source FPGA simulation tool VTR [1,2,8], which is widely used in academic research. For simplicity, we modeled the Xilinx FPGA architecture by fixing the number of BLEs per CLB. The detailed architecture-level parameters are provided in Table 3. Here, F_{cin} and F_{cout} are parameters that reflect the flexibility of the connection block, while F_s reflects the flexibility of the switch box. More details on these parameters can be found in [1]. For the switching energy calculation, the activity file is generated by ACE2.0 [25], which generates the effective activity factors for each connection in an FPGA system based on the circuit netlist generated by ABC. The interconnect capacitance and resistance per unit length of copper wires are calculated based on the following existing works [26,27].

Table 3. Parameter setting for VTR FPGA system-level simulation.

Parameter Type	Parameters	Values
FPGA Device/System-Level Variables	Vdd (V)	0.7
	# of LUT Inputs	$K = 6$
	# of BLEs per CLB	$N = 10$
FPGA System-Level Parameters	# of CLB Inputs	$I = (N+1) \times \frac{K}{2}$
	# of Channels (W)	$W = 300$
	F_{cin}	0.15 W
	F_{cout}	0.25 W
	Switch Box Style	Wilton Style
	F_s	3
	Wire Segments Length	4 CLB Width

The program runs on a desktop with an Intel i9-9900 CPU and 32 GB RAM. The program does not have specific requirements for the CPU, and it should be compatible with most modern processors. Based on our tests, the code requires no more than 16 GB of memory, even for the largest benchmark netlists. Therefore, it should be applicable to most mainstream personal computers.

4. Simulation Results

4.1. Comparison Between Different FPGA Architectures

As shown in Figure 7a, both the UltraScale+ and Versal dual-output architectures achieve significant area savings compared to the single-output architecture, with area reductions of up to 33.9% and 21.3%, respectively. Figure 7b illustrates that the Versal architecture exhibits a higher ratio of dual-output LUTs, with up to 59% of the mapped LUTs being dual-output, while this figure for the UltraScale+ architecture is 32%. The Versal architecture benefits more from dual-output LUTs than the UltraScale+ due to key differences in their flexibility. As shown in Table 1, the Versal architecture allows a LUT to be split into two sub-LUTs with up to six shared inputs, while the UltraScale+ architecture only supports a maximum of five shared inputs when a LUT is split. This additional flexibility in the Versal architecture enables more efficient utilization of dual-output LUTs, leading to improved mapping outcomes. The results demonstrate that architectures with a higher percentage of dual-output LUTs achieve greater area savings, confirming that dual-output LUTs effectively optimize the circuit area. The benchmark used for these results is the original EPFL benchmark list.

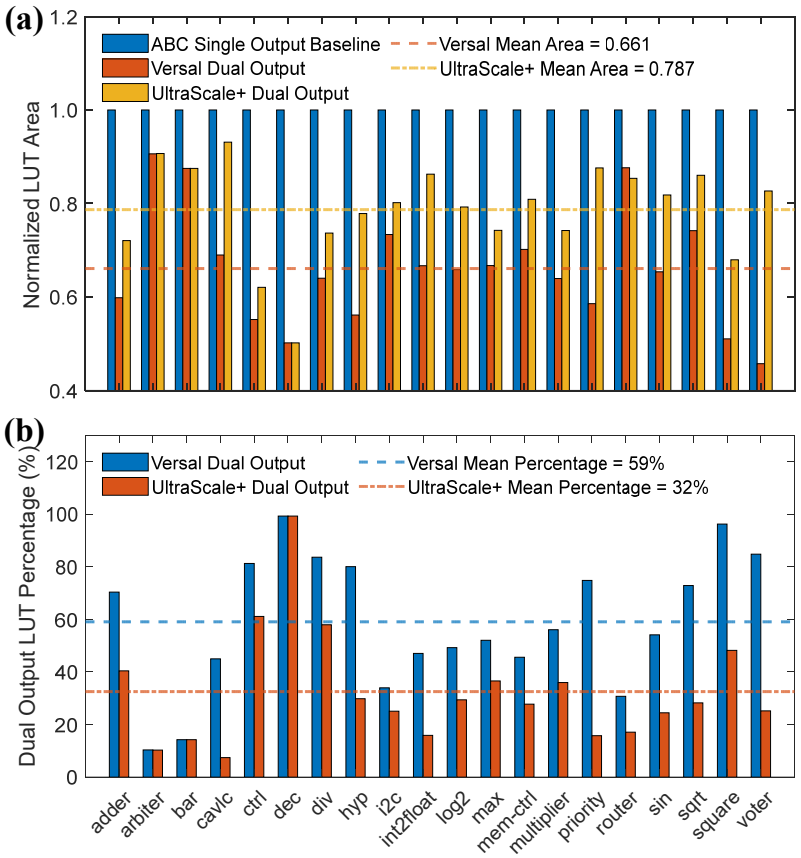


Figure 7. Comparison between different FPGA architectures. (**a**) Comparison of the number of LUTs among FPGAs using single-output LUTs, Versal architecture, and UltraScale+ architecture, where area is normalized by single-output LUT baseline. (**b**) Percentage of dual-output LUTs for different architectures.

4.2. Comparison with State-of-the-Art Dual-Output Mapping Technique

As shown in Table 4, we compare the mapping results with the work from [15]. All benchmarks are processed using the "resyn; resyn2" command as a preprocessing step to ensure a fair comparison. It should be noted that Table 4 only includes the 13 benchmarks used in the reference work, with

the full set of results available in Appendix A. The results demonstrate that the proposed mapping technique achieves up to 35% and 6% averaged area savings, compared to the state-of-the-art ABC six-input, single-output LUT mapping technique and previous work focusing on dual-output LUT mapping techniques that optimize cut generation parameters.

Table 4. Comparison with state-of-the-art dual-output mapping technique.

Architecture	Netlsit	Single Output LUT Area	Proposed Results with Resyn Benchmarks		Reference Results with Resyn Benchmarks	
			Dual Output LUT Area	Dual Output LUT Area Saving	Dual Output LUT Area	Dual Output LUT Area Saving
Versal	adder	257	151	41.25%	140	45.53%
	arbiter	2722	2467	9.37%	2457	9.74%
	bar	512	448	12.50%	448	12.50%
	cavlc	118	80	32.20%	80	32.20%
	ctrl	29	16	44.83%	15	48.28%
	dec	287	144	49.83%	137	52.26%
	i2c	316	221	30.06%	268	15.19%
	int2float	47	35	25.53%	35	25.53%
	max	793	531	33.04%	690	12.99%
	priority	178	107	39.89%	150	15.73%
	sin	1477	941	36.29%	1109	24.92%
	square	3949	1998	49.40%	2127	46.14%
	voter	1736	941	45.79%	1264	27.19%
	Average			34.61%		28.32%
Ultrascale+	adder	257	179	30.35%	189	26.46%
	arbiter	2722	2468	9.33%	2469	9.29%
	bar	512	448	12.50%	448	12.50%
	cavlc	118	105	11.02%	108	8.47%
	ctrl	29	18	37.93%	17	41.38%
	dec	287	144	49.83%	140	51.22%
	i2c	316	242	23.42%	300	5.06%
	int2float	47	41	12.77%	41	12.77%
	max	793	620	21.82%	822	−3.66%
	priority	178	159	10.67%	160	10.11%
	sin	1477	1203	18.55%	1426	3.45%
	square	3949	2692	31.83%	3241	17.93%
	voter	1736	1380	20.51%	1520	12.44%
	Average			22.35%		15.96%

In a detailed comparison between the reference work and our proposed method, three netlists—"adder", "arbiter" and "dec"—exhibit slightly greater area savings in the reference work when implemented on the Versal architecture. These few netlists feature relatively regular and well-structured hierarchical designs, making the parameter-based mapping technique is more efficient. However, for larger-scale netlists with more structurally random characteristics, the proposed method demonstrates a significant advantage. This further validates that the cut interconnection-based multi-output mapping technique is more universally applicable. In addition, for some small-sized netlists, such as "int2float", "cavlc", and "ctrl", the netlist sizes are relatively small, resulting in a similar LUT area utilization for both the reference work and the proposed method.

4.3. Runtime Analysis

As shown in Table 5, although the proposed mapping method demonstrates higher dual-output mapping efficiency, the trade-off is an increased execution time compared to the ABC single-output mapping. The time overhead is minimal when the input netlist size is small; however, as the netlist size increases, more significant time overhead is observed. As depicted in Figure 8, this is due to the linear time complexity of the proposed method at the code realization level. Future work, such as algorithm optimization, will be explored to address and reduce this time overhead.

Table 5. Program execution time for ABC and proposed mapping method with Ultrascale+.

Netlsit	Netlists Size	ABC Runtime (s)	Proposed Runtime (s)	Time Overhead (1×)
adder	1020	0.137	0.286	2.09
arbiter	11,839	0.516	2.982	5.78
bar	3336	0.139	0.394	2.83
cavlc	304	0.072	0.098	1.36
ctrl	174	0.060	0.059	0.98
dec	304	0.061	0.090	1.48
div	44,762	3.701	56.307	15.21
hyp	214,335	14.196	399.647	28.15
i2c	1342	0.089	0.214	2.40
int2float	260	0.055	0.065	1.18
log2	32,060	2.108	16.255	7.71
max	2865	0.174	0.535	3.07
mem_ctrl	46,836	2.436	59.881	24.58
multiplier	27,062	1.644	9.978	6.07
priority	978	0.086	0.138	1.60
router	257	0.059	0.074	1.25
sin	5416	0.362	1.330	3.67
sqrt	24,618	1.562	9.706	6.21
square	18,484	1.137	6.341	5.58
voter	13,758	0.718	3.893	5.42

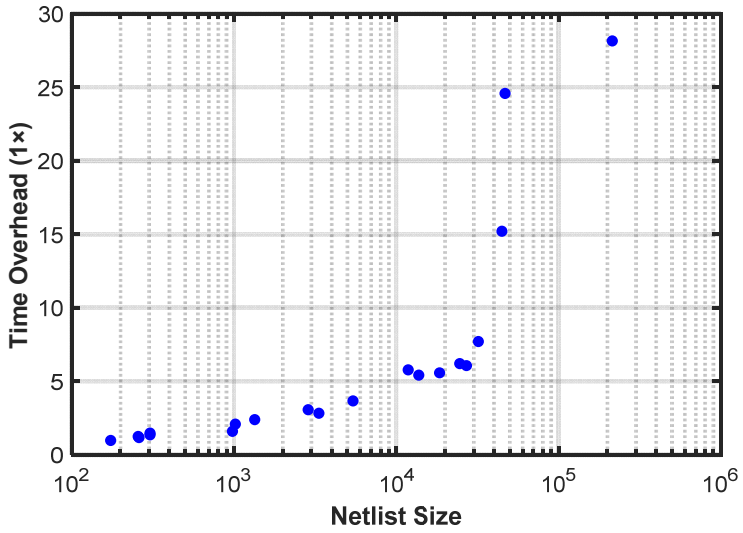


Figure 8. Relationship between program execution time overhead and netlist size (AIG node number).

4.4. FPGA System-Level Simulation Results

As shown in Table 6, both dual-output LUT architectures maintain an advantage in most parameters. Specifically, for the Versal series architecture, up to 13.92% area savings, 11.51% delay reductions, 7.31% wirelength savings, and 8.41% switching energy savings can be achieved compared to the single-output LUT baseline. However, for certain netlists, such as "priority", dual-output LUT architectures exhibit a higher delay, and for "arbiter", they show an increased wirelength. These "abnormal" results suggest that system-level factors related to FPGA placement and routing should be considered, even though dual-output LUT architectures already provide area advantages after technology mapping [28–30]. For example, although dual-output LUTs offer higher mapping efficiency, leading to a reduction in the LUT area, a larger number of input/output net nodes can increase placement and routing congestion, ultimately resulting in a longer wirelength. Therefore, future work will focus on optimizing the integration of LUT technology mapping with system-level placement and routing to mitigate these challenges.

Table 6. VTR-based FPGA system level simulation results.

ABC Single-Output LUT Baseline				
Netlist	**Area (μm^2)**	**Delay (ns)**	**Wire Length (μm)**	**Switch Energy (pJ)**
adder	1.03×10^6	6.98	6.85×10^4	1.21×10^{-2}
arbiter	1.25×10^7	3.96	8.27×10^5	1.77×10^{-2}
bar	1.96×10^6	1.55	1.25×10^5	1.05×10^{-2}
cavlc	4.82×10^5	1.06	8.83×10^3	1.63×10^{-3}
ctrl	1.29×10^5	0.52	1.40×10^3	7.79×10^{-4}
dec	1.02×10^6	1.13	3.71×10^4	5.43×10^{-3}
i2c	1.17×10^6	1.40	7.62×10^4	9.57×10^{-3}
int2float	1.68×10^5	0.88	2.83×10^3	1.30×10^{-3}
max	3.18×10^6	6.13	2.46×10^5	2.72×10^{-2}
priority	7.71×10^5	2.02	3.12×10^4	4.33×10^{-3}
sin	6.47×10^6	8.52	4.14×10^5	1.87×10^{-2}
square	1.56×10^7	7.55	7.72×10^5	5.78×10^{-2}
voter	7.17×10^6	4.08	5.16×10^5	5.21×10^{-2}
Versal Series Architecture				
Netlist	**Area (μm^2)**	**Delay (ns)**	**Wire Length (μm)**	**Switch Energy (pJ)**
adder	8.63×10^5	1.58	6.11×10^4	1.13×10^{-2}
arbiter	1.57×10^7	4.19	9.06×10^5	2.21×10^{-2}
bar	1.99×10^6	1.61	1.29×10^5	9.52×10^{-3}
cavlc	3.11×10^5	0.97	4.79×10^3	1.34×10^{-3}
ctrl	1.11×10^5	0.52	1.07×10^3	8.13×10^{-4}
dec	6.36×10^5	1.25	5.15×10^4	2.43×10^{-3}
i2c	1.53×10^6	1.36	7.95×10^4	9.94×10^{-3}
int2float	1.49×10^5	0.75	2.01×10^3	1.21×10^{-3}
max	2.72×10^6	7.50	2.81×10^5	3.00×10^{-2}
priority	5.70×10^5	3.04	2.74×10^4	4.63×10^{-3}
sin	5.14×10^6	2.65	3.20×10^5	1.42×10^{-2}
square	1.12×10^7	5.42	6.18×10^5	4.02×10^{-2}
voter	4.65×10^6	2.34	5.13×10^5	4.73×10^{-2}
Average Savings	13.92%	11.51%	7.31%	8.41%
UltraScale+ Series Architecture				
Netlist	**Area (μm^2)**	**Delay (ns)**	**Wire Length (μm)**	**Switch Energy (pJ)**
adder	9.92×10^5	1.75	6.95×10^4	1.23×10^{-2}
arbiter	1.41×10^7	3.79	8.53×10^5	1.74×10^{-2}
bar	1.99×10^6	1.54	1.30×10^5	1.07×10^{-2}
cavlc	4.39×10^5	1.02	8.43×10^3	1.57×10^{-3}
ctrl	1.11×10^5	0.52	1.07×10^3	7.51×10^{-4}
dec	6.36×10^5	1.25	5.15×10^4	2.43×10^{-3}
i2c	1.62×10^6	1.25	7.53×10^4	9.96×10^{-3}
int2float	1.94×10^5	0.84	3.20×10^3	1.10×10^{-3}
max	2.94×10^6	5.59	2.82×10^5	2.93×10^{-2}
priority	7.33×10^5	3.20	2.83×10^4	4.61×10^{-3}
sin	5.89×10^6	5.41	3.58×10^5	1.68×10^{-2}
square	1.40×10^7	4.54	6.57×10^5	4.82×10^{-2}
voter	6.69×10^6	4.95	5.53×10^5	5.45×10^{-2}
Average Savings	2.66%	7.28%	−1.17%	6.13%

5. Discussion

The simulation results demonstrate that the proposed dual-output mapping technique achieves significant improvements over existing approaches. Specifically, compared to state-of-the-art ABC

single-output mapping, our method reduces the LUT area by up to 35%. When compared to existing dual-output mapping techniques, our approach still delivers notable area savings of up to 6%.

Both our proposed work and the referenced work are based on the ABC synthesis platform and aim to optimize the mapping flow for dual-output LUTs. The key difference lies in the method of dual-output cut generation. The referenced work employs a k-input, l-output cut-based approach, which directly enumerates potential dual-output cuts during the enumeration phase. While effective in certain scenarios, this method requires exploring many possible cuts, making it memory-intensive and time-consuming, and thus less suitable for large or complex benchmark netlists.

In contrast, our proposed method takes an interconnection-oriented perspective. Instead of directly enumerating dual-output cuts, we focus on merging and combining highly related single-output cuts that are generated by the standard ABC mapping flow. This reformulates the dual-cut generation problem as a mathematical covering and merging problem, which significantly simplifies the process. As a result, our method is more scalable and broadly applicable across a wide range of benchmark circuits, providing a practical and efficient alternative to exhaustive enumeration.

However, in terms of runtime, there is a noticeable time overhead compared to the ABC single-output mapping flow. This is primarily due to the inclusion of repeated executions, such as checking the flow of the layer-based cut merging, which involves identifying common nodes between two cuts. These steps can be further optimized at the code realization level. When compared to the state-of-the-art dual-output method in the referenced work, which incurs a 3.5× time overhead, our proposed method achieves a smaller time overhead of 2.8× with the same benchmark netlists. Reducing runtime during the mapping process is a critical aspect, and optimization efforts in this area will be addressed in future work.

6. Conclusions

In this work, we propose a cut interconnection-based multi-output LUT mapping technique that effectively utilizes the interconnection information of single-output cuts to enable efficient dual-output mapping. The approach integrates several key algorithmic components, including side-fanout insertion, runtime multi-output LUT expansion, and a LUT merging and covering algorithm, all designed to enhance the overall mapping quality and reduce area usage.

We evaluate the proposed method using the complete set of EPFL benchmark circuits. The experimental results show that our technique achieves significant improvements over existing approaches. Specifically, compared to the state-of-the-art ABC single-output mapping, our method reduces the LUT area by up to 35%. When compared to existing dual-output mapping techniques, our approach still delivers notable area savings of up to 6%. However, in terms of code execution time, the trade-off involves an increased execution duration compared to the ABC single-output mapping. While the time overhead remains minimal for smaller input netlists, it becomes more pronounced as the netlist size grows. This is attributable to the linear time complexity of the proposed method at the code realization level. Future work will focus on reducing the time complexity at the code level and extending the approach to support other emerging multi-output logic units.

Author Contributions: Algorithm development, simulation, and manuscript preparation were conducted by S.L., L.S. and Q.Q. Paper writing guidance was provided by Q.L. and S.J. Project administration and supervision were carried out by C.P. All authors have read and agreed to the published version of the manuscript.

Funding: This work is funded by the Advanced Scientific Computing Research (ASCR) program of the Department of Energy (DoE) through the award DE-SC0022881, imec, and National Science Foundation (NSF) under grant CCF-2219753.

Data Availability Statement: All the data and code can be downloaded from GitHub with the following link: https://github.com/Sheng1994/abcpercy_merge.git (accessed on 26 April 2025). Please email the corresponding author for download permission.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

FPGA	Field-Programmable Gate Array
CPU	Central Processing Unit
ASIC	Application-Specific Integrated Circuit
LUT	Look-Up Table
BLE	Basic Logic Element
CLB	Configurable Logic Blocks
FF	Flip-Flop
DAG	Directed Acyclic Graph
AIG	And Inverter Graph
PIs	Primary Inputs
POs	Primary Outputs

Appendix A

FPGA Type	Netlsit	Proposed Simulation Results with Original Benchmarks				Proposed Simulation Results with Resyn Benchmarks				Reference Simulation Results with Resyn Benchmarks		
		Dual Output LUT Area	Depth	Dual-output Percentage	Dual Output LUT Area Saving	Dual Output LUT Area	Depth	Dual-output Percentage	Dual Output LUT Area Saving	Dual Output LUT Area	Depth	Dual Output LUT Area Saving
Versal	adder	152	51	70.39%	40.16%	151	51	71.52%	41.25%	140	51	45.53%
	arbiter	2467	18	10.34%	9.37%	2467	18	10.34%	9.37%	2457	18	9.74%
	bar	448	4	14.29%	12.50%	448	4	14.29%	12.50%	448	4	12.50%
	cavlc	80	4	45.00%	31.03%	80	4	50.00%	32.20%	80	4	32.20%
	ctrl	16	2	81.25%	44.83%	16	2	81.25%	44.83%	15	2	48.28%
	dec	144	2	99.31%	49.83%	144	2	99.31%	49.83%	137	2	52.26%
	div	14155	936	83.64%	35.99%	5524	861	87.09%	44.38%	Not Provided	Not Provided	Not Provided
	hyp	24983	4214	80.07%	43.87%	24609	4198	81.64%	44.50%	Not Provided	Not Provided	Not Provided
	i2c	259	4	33.98%	26.63%	221	4	42.53%	30.06%	268	4	15.19%
	int2float	34	3	47.06%	33.33%	35	3	28.57%	25.53%	35	3	25.53%
	log2	5330	74	49.25%	34.13%	5405	76	48.46%	33.08%	Not Provided	Not Provided	Not Provided
	max	513	45	52.05%	33.29%	531	39	49.72%	33.04%	690	39	12.99%
	mem_ctrl	8482	27	45.59%	29.81%	8295	29	44.76%	29.39%	Not Provided	Not Provided	Not Provided
	multiplier	3790	58	56.07%	36.06%	3801	53	55.70%	35.82%	Not Provided	Not Provided	Not Provided
	priority	123	31	74.80%	41.43%	107	26	68.22%	39.89%	150	26	15.73%
	router	78	7	30.77%	12.36%	66	6	16.67%	13.16%	Not Provided	Not Provided	Not Provided
	sin	957	40	54.13%	34.63%	941	42	55.58%	36.29%	1109	36	24.92%
	sqrt	4236	1025	72.85%	25.83%	2859	1066	80.69%	38.45%	Not Provided	Not Provided	Not Provided
	square	2041	51	96.28%	48.95%	1998	50	96.75%	49.40%	2127	50	46.14%
	voter	1232	20	84.82%	54.29%	941	15	79.28%	45.79%	1264	13	27.19%
	Average			**59.10%**	**33.91%**			**58.12%**	**34.44%**			**28.32%**
UltraScale+	adder	183	51	40.44%	27.95%	179	51	44.69%	30.35%	189	26	26.46%
	arbiter	2468	18	10.29%	9.33%	2468	18	10.29%	9.33%	2469	18	9.29%
	bar	448	4	14.29%	12.50%	448	4	14.29%	12.50%	448	4	12.50%
	cavlc	108	4	7.41%	6.90%	105	4	12.38%	11.02%	108	4	8.47%
	ctrl	18	2	61.11%	37.93%	18	2	61.11%	37.93%	17	2	41.38%
	dec	144	2	99.31%	49.83%	144	2	99.31%	49.83%	140	2	51.22%
	div	16289	943	57.97%	26.34%	7279	866	38.27%	26.70%	Not Provided	Not Provided	Not Provided
	hyp	25007	4214	79.90%	43.81%	24617	4198	81.58%	44.48%	Not Provided	Not Provided	Not Provided
	i2c	283	5	25.09%	19.83%	242	4	30.58%	23.42%	300	4	5.06%
	int2float	44	3	15.91%	13.73%	41	3	14.63%	12.77%	41	3	12.77%
	log2	6412	77	29.43%	20.76%	6476	76	27.53%	19.82%	Not Provided	Not Provided	Not Provided
	max	571	45	36.60%	25.75%	620	39	28.55%	21.82%	822	39	-3.66%
	mem_ctrl	9773	30	27.79%	19.12%	9475	30	27.91%	19.34%	Not Provided	Not Provided	Not Provided
	multiplier	4398	56	36.02%	25.80%	4324	54	37.44%	26.98%	Not Provided	Not Provided	Not Provided
	priority	184	31	15.76%	12.38%	159	26	12.58%	10.67%	160	26	10.11%
	router	76	7	17.11%	14.61%	72	6	5.56%	5.26%	Not Provided	Not Provided	Not Provided
	sin	1198	41	24.46%	18.17%	1203	40	25.35%	18.55%	1426	36	3.45%
	sqrt	4913	1051	28.25%	13.97%	3908	1118	42.14%	15.87%	Not Provided	Not Provided	Not Provided
	square	2717	51	48.25%	32.04%	2692	50	46.84%	31.83%	3241	50	17.93%
	voter	2228	20	25.22%	17.33%	1380	16	28.48%	20.51%	1520	13	12.44%
	Average			**35.03%**	**22.40%**			**34.48%**	**22.45%**			**15.96%**

Figure A1. Full simulation results.

References

1. Betz, V.; Rose, J.; Marquardt, A. *Architecture and CAD for Deep-Submicron FPGAs*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2012; Volume 497.
2. Ahmed, E.; Rose, J. The effect of LUT and cluster size on deep-submicron FPGA performance and density. In Proceedings of the 2000 ACM/SIGDA Eighth International Symposium on Field Programmable Gate Arrays, Monterey, CA, USA, 10–11 February 2000; pp. 3–12.
3. AMD. *Vivado UltraScale Libraries User Guide (UG974)—LUT6*; AMD: Santa Clara, CA, USA, 2024.
4. Cong, J.; Liu, B.; Neuendorffer, S.; Noguera, J.; Vissers, K.; Zhang, Z. High-level synthesis for FPGAs: From prototyping to deployment. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2011**, *30*, 473–491. [CrossRef]
5. Cong, J.; Ding, Y. FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **1994**, *13*, 1–12. [CrossRef]
6. Mishchenko, A.; Chatterjee, S.; Brayton, R. Improvements to technology mapping for LUT-based FPGAs. In Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2006; pp. 41–49.
7. Intel Stratix FPGA. Available online: https://www.intel.com/content/www/us/en/products/details/fpga/stratix.html (accessed on 21 March 2025).
8. Murray, K.E.; Petelin, O.; Zhong, S.; Wang, J.M.; Eldafrawy, M.; Legault, J.-P.; Sha, E.; Graham, A.G.; Wu, J.; Walker, M.J. VTR 8: High-performance CAD and customizable FPGA architecture modelling. *ACM Trans. Reconfigurable Technol. Syst.* **2020**, *13*, 9. [CrossRef]
9. Shang, L.; Lu, S.; Zhang, Y.; Jung, S.; Pan, C. Directed Acyclic Graph-Based Datapath Synthesis Using Graph Isomorphism and Gate Reconfiguration. *Chips* **2024**, *3*, 182–195. [CrossRef]
10. Shang, L.; Lu, S.; Jung, S.; Pan, C. Novel Fence Generation Methods for Accelerating Reconfigurable Exact Synthesis. In Proceedings of the 2023 IEEE 66th International Midwest Symposium on Circuits and Systems (MWSCAS), Tempe, AZ, USA, 6–9 August 2023; pp. 506–510.
11. Lu, S.; Shang, L.; Jung, S.; Zhang, Y.; Pan, C. A Novel Delay-Aware Packing Algorithm for FPGA Architecture Using RFET. In Proceedings of the 2024 IEEE 67th International Midwest Symposium on Circuits and Systems (MWSCAS), Springfield, MA, USA, 11–14 August 2024; pp. 362–366.
12. Gaillardon, P.-E.; Tang, X.; Kim, G.; De Micheli, G. A novel FPGA architecture based on ultrafine grain reconfigurable logic cells. *IEEE Trans. Very Large Scale Integr. Syst.* **2014**, *23*, 2187–2197. [CrossRef]
13. Brayton, R.; Mishchenko, A. ABC: An academic industrial-strength verification tool. In Proceedings of the Computer Aided Verification: 22nd International Conference, CAV 2010, Edinburgh, UK, 15–19 July 2010; pp. 24–40.
14. Jang, S.; Chan, B.; Chung, K.; Mishchenko, A. Wiremap: FPGA technology mapping for improved routability and enhanced LUT merging. *ACM Trans. Reconfigurable Technol. Syst.* **2009**, *2*, 14. [CrossRef]
15. Wang, F.; Zhu, L.; Zhang, J.; Li, L.; Zhang, Y.; Luo, G. Dual-output LUT merging during FPGA technology mapping. In Proceedings of the 39th International Conference on Computer-Aided Design, San Diego, CA, USA, 2–5 November 2020; pp. 1–9.
16. Calvino, A.T.; De Micheli, G. Technology mapping using multi-output library cells. In Proceedings of the 2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD), San Francisco, CA, USA, 28 October–2 November 2023; pp. 1–9.
17. Shi, J.; Wang, X.; Meng, C.; Qian, W. QUADOL: A Quality-Driven Approximate Logic Synthesis Method Exploiting Dual-Output LUTs for Modern FPGAs. *arXiv* **2024**, arXiv:2411.18330.
18. Jiang, Z.; Lin, C.Y.; Yang, L.; Wang, F.; Yang, H. Exploring architecture parameters for dual-output LUT based FPGAs. In Proceedings of the 2014 24th International Conference on Field Programmable Logic and Applications (FPL), Munich, Germany, 2–4 September 2014; pp. 1–6.
19. Lee, J.-Y.; Hu, Y.; Majumdar, R.; He, L.; Li, M. Fault-tolerant resynthesis with dual-output LUTs. In Proceedings of the 2010 15th Asia and South Pacific Design Automation Conference (ASP-DAC), Taipei, Taiwan, 18–21 January 2010; pp. 325–330.
20. Hu, Y.; Shih, V.; Majumdar, R.; He, L. FPGA area reduction by multi-output function based sequential resynthesis. In Proceedings of the 45th Annual Design Automation Conference, Anaheim, CA, USA, 8–13 June 2008; pp. 24–29.
21. Beamer, S.; Asanović, K.; Patterson, D. Direction-optimizing breadth-first search. *Sci. Program.* **2013**, *21*, 137–148. [CrossRef]
22. ABC: A System for Sequential Logic Synthesis and Verification. Available online: https://github.com/berkeley-abc/abc (accessed on 23 April 2025).
23. AMD. *Versal Adaptive SoC Configurable Logic Block Architecture Manual (AM005)*; AMD: Santa Clara, CA, USA, 2024.
24. Amarú, L.; Gaillardon, P.-E.; De Micheli, G. The EPFL combinational benchmark suite. In Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS), Mountain View, CA, USA, 12–13 June 2015.
25. Lamoureux, J.; Wilton, S.J. Activity estimation for field-programmable gate arrays. In Proceedings of the 2006 International Conference on Field Programmable Logic and Applications, Madrid, Spain, 28–30 August 2006; pp. 1–8.

26. Pan, C.; Naeemi, A. A proposal for a novel hybrid interconnect technology for the end of roadmap. *IEEE Electron Device Lett.* **2013**, *35*, 250–252. [CrossRef]
27. Pan, C.; Naeemi, A. A paradigm shift in local interconnect technology design in the era of nanoscale multigate and gate-all-around devices. *IEEE Electron Device Lett.* **2015**, *36*, 274–276. [CrossRef]
28. Elgammal, M.A.; Murray, K.E.; Betz, V. RLPlace: Using reinforcement learning and smart perturbations to optimize FPGA placement. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2021**, *41*, 2532–2545. [CrossRef]
29. Ababei, C.; Feng, Y.; Goplen, B.; Mogal, H.; Zhang, T.; Bazargan, K.; Sapatnekar, S. Placement and routing in 3D integrated circuits. *IEEE Des. Test Comput.* **2005**, *22*, 520–531. [CrossRef]
30. Li, W.; Dhar, S.; Pan, D.Z. UTPlaceF: A routability-driven FPGA placer with physical and congestion aware packing. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2017**, *37*, 869–882. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.