ELMap: Area-Driven LUT Mapping with k-LUT Network Exact Synthesis

Hongyang Pan[1], Keren Zhu[1], Fan Yang[1], Zhufei Chu[2*], and Xuan Zeng[1*]
[1]State Key Lab of Integrated Circuits and Systems, School of Microelectronics, Fudan University, Shanghai, China
[2]Faculty of Electrical Engineering and Computer Science, Ningbo University, Ningbo, China.
Emails: chuzhufei@nbu.edu.cn, zengxuan@fudan.edu.cn

***Abstract*—Mapping to k-input lookup tables (k-LUTs) is a critical process in field-programmable gate array (FPGA) synthesis. However, the structure of the subject graph can introduce structural bias, which refers to the dependency of mapping results on the inherent graph structure, often leading to suboptimal results. To address this, we present `ELMap`, an area-driven LUT mapping framework. It incorporates structural choice during the collapsing phase. This enables dynamic decomposition, maximizing local-to-global optimization transfer. To ensure seamless integration between the optimization and mapping processes, `ELMap` leverages exact k-LUT synthesis to generate area-optimal sub-LUT networks. Experiments on the EPFL benchmark suite demonstrate that `ELMap` significantly outperforms state-of-the-art methods. Specifically, in 6-LUT mapping, `ELMap` reduces the average LUT area by 8.5% and improves the area-depth-product (ADP) by 5.8%. In 4-LUT remapping, it reduces the average LUT area by 17.6% and improves the ADP by 2.4%.**

***Index Terms*—Logic synthesis, Technology mapping, LUT mapping, Exact synthesis of k-LUT networks**

I. Introduction

LUT mapping is the process of converting circuits composed of simple gates into lookup tables (LUTs) for implementation on field-programmable gate arrays (FPGAs). This step is critical in FPGA synthesis, aims to minimize the area or depth of the LUT network.

LUT mapping is typically formulated as a covering problem, where local transformations are effectively applied to the subject graph [1]. Depth-oriented mapping uses duplication to reduce depth [2], while area-oriented mapping limits duplication to minimize the total area [3]. The structure of the subject graph influences the quality of mapping, known as structural bias. Recent efforts to mitigate structural bias can be categorized into two approaches: 1) **Structural choices**: This approach improves quality of results (QoR) by considering multiple structural variations. However, it introduces many choice nodes, which increase complexity and reduce mapping efficiency [4, 5]. 2) **Collapsing and decomposition**: This method allows for rapid local optimizations but struggles to ensure optimal subcircuits and translate local improvements into global benefits [6–8]. Moreover, current logic synthesis techniques exhibit a gap between technology-independent optimization and LUT mapping. The enhancements of technology-independent optimizations do not always translate into better results in the final LUT networks during the mapping phase [9]. The limitations of both approaches, combined with this gap, present significant challenges in ensuring that early-stage optimizations lead to improvements in the post-mapping netlist and enhancement of mapping algorithms.

This paper presents a novel LUT mapping framework (`ELMap`) that incorporates the concept of structural choice into the collapsing phase. By generating optimal representations for multiple decomposed functions, we maximize the transfer of local optimizations to global optimizations. We also introduce a semi-tensor product (STP)-based exact synthesis method for k-LUT networks, which generates optimal sub-LUT networks for direct mapping of the replaced subcircuits. This approach ensures that the local optimizations align seamlessly with the LUT mapping process. The main contributions of our work are summarized as follows:

- **Area-driven LUT mapping framework**: Integrates choice into the collapsing phase to maximize local optimizations. By generating area-optimal LUT sub-networks, the framework enables direct mapping during local replacement.
- **Choice collapsing**: Efficiently generates multiple optimal sub-LUT networks by collapsing larger LUTs, with the best sub-LUT selected during the cover phase.
- **Exact synthesis of k-LUT networks**: Introduces a new area-optimal exact synthesis of k-LUT networks, integrating topological constraints to improve runtime.
- The proposed framework is implemented in the opensource logic synthesis tool phyLS [1]. Experiments demonstrate that `ELMap` achieves an 8.5% LUT area reduction and 5.8% area-depth-product (ADP) improvement on 6-LUT mapping, and a 17.6% LUT area reduction and 2.4% ADP improvement on 4-LUT remapping.

II. Preliminaries

This section reviews the background of Boolean networks, FPGA mapping, recent advancements in exact synthesis, and the use of STP in logic networks. Key terms and their descriptions are summarized in TABLE I.

A. Boolean Networks

A Boolean network can be represented as a directed acyclic graph (DAG) $G = (V, E)$ with a set of nodes V and edges E, where V correspond to logic gates (Boolean functions) and E represent the wires connecting these gates. The input (output) of each node is termed its *fanin* (*fanout*). Node

[1] https://github.com/panhomyoung/phyLS

978-3-9826741-0-0/DATE25/© 2025 EDAA

TABLE I: Terminology and Notation in this paper

Notation	Description
$G = (V, E)$	Graph G with vertices V and edges E
$\mathbf{c}_v$	Cut of node v
k	Input size of LUT
f	Function of exact synthesis
r, $\mathcal{C}_r$	Number of gate in exact synthesis, and encoding CNF formula for exact synthesis
$\mathbf{M}_{x\times y}$	Boolean matrix $\mathbf{M}$ with dimension $x \times y$
$\mathbf{I}_x$	Identity matrix $\mathbf{I}$ with dimension $x \times x$
$\mathbb{B}$	The set of binary Boolean variables
δ_j^i, $\Delta_j[i, i, ...]$	The i-th column of I_j, and the set of δ_j^i
pDAG	Possible DAGs constraint with r gates
$\mathcal{F}_r$, $\mathcal{P}_r$	Fence, and partial-DAG constraint with r gates
pLUT	Optimal sub-LUT networks
$\mathbf{M}_\Phi$, $\mathbf{M}_f$	STP form of *pDAG*, and structural matrix of f

without *fanins* or *fanouts*, serve as the *primary input* (PI) or *primary output* (PO) of the graph. The level of a node is defined by counting the nodes along the longest structural path from any PI to the node, inclusive. A cut of node v ($\mathbf{c}_v$) includes a set of nodes within the network. The nodes included in the $\mathbf{c}_v$ of node are called *leaves*, such that each path from a PI to node passes through at least one leaf. The node v is called the *root* of the $\mathbf{c}_v$, and the cut size is the number of its leaves. A cut is k-feasible if its size does not exceed k.

B. FPGA Technology Mapping

In LUT mapping, the objective is to convert a Boolean network into a representation using k-LUTs. The initial network, known as the subject graph, is a k-bounded Boolean network, where each node has a fanin of at most k. The LUT mapping process begins with cut enumeration [10], where the subject graph is partitioned into multiple subcircuits. Local replacements are then applied to these subcircuits, ultimately mapping each one to a k-LUT. An LUT mapper computes a mapping solution, referred to as a cover, by selecting a subset of cuts that fully cover the subject graph. Conventional LUT mappers use heuristic techniques based on delay, area, and edge count to compute the cuts and iteratively refine the cover through multiple mapping passes [2].

C. Exact Synthesis

Exact synthesis refers to produce optimal circuits by finding the best possible implementation of a given Boolean function under specific constraints. It is widely used in logic optimization [11], NPN matching [12], and standard cell library construction [13]. When optimizing for minimal circuit size, a SAT solver is utilized to determine whether a logic network with r gates can implement a given function f. The algorithm starts with $r = 0$ and incrementally increases r until the SAT solver finds a satisfiable solution. Each formula $\mathcal{C}_r$ corresponds to the conjunctive normal form (CNF) encoding for a network with r steps. This method proves that no logic network with r or fewer steps can realize f.

D. Semi-Tensor Product of Matrices

The STP introduces a novel method for logic representation by encoding logical reasoning using Boolean matrices. For more detailed information, we refer readers to [14].

Definition 1. *Let $\boldsymbol{X} \in \boldsymbol{M}_{a\times b}$ and $\boldsymbol{Y} \in \boldsymbol{M}_{c\times d}$, the STP of $\boldsymbol{X}$ and $\boldsymbol{Y}$, denoted by $\boldsymbol{X} \ltimes \boldsymbol{Y}$, is defined as:*

$$\boldsymbol{X} \ltimes \boldsymbol{Y} = (\boldsymbol{X} \otimes \boldsymbol{I}_{t/b}) \cdot (\boldsymbol{Y} \otimes \boldsymbol{I}_{t/c}),$$

where $\cdot$ represents the common matrix product, t is the least common multiple of b and c, and $\otimes$ is kronecker product. We refer to the matrix product as the STP in this paper and omit the symbol "$\ltimes$" hereinafter.

Definition 2. *STP denotes binary Boolean variables as $\mathbb{B}$:* $\left\{\text{True} = \begin{bmatrix}1\\0\end{bmatrix} = \delta_2^1, \text{False} = \begin{bmatrix}0\\1\end{bmatrix} = \delta_2^2\right\}$.

Definition 3. *The matrix $\boldsymbol{M}_\sigma$ is named the of logic operator σ, if each column of $\boldsymbol{M}_\sigma$ is consistent with the truth table of logic operator σ. Assume $a, b \in \mathbb{B}$, the logic representation can be converted as $a\sigma b = \boldsymbol{M}_\sigma ab$.*

Property 1. *1. To swap structural matrix $\boldsymbol{M}_\sigma$ and Boolean variable a, STP has $a\boldsymbol{M}_\sigma = (\boldsymbol{I}_2 \otimes \boldsymbol{M}_\sigma)a$.*

2. For multiple Boolean variables multiply $a \cdot a$, the STP has power-reducing matrix ($\boldsymbol{M}_r = \Delta_4[1, 4]$) for $a^2 = \boldsymbol{M}_r a$.

3. The variable swapping matrix ($\boldsymbol{M}_w = \Delta_4[1, 3, 2, 4]$) is used to sort the Boolean variables in order as $\boldsymbol{M}_w ba = ab$.

III. Observation and Motivation

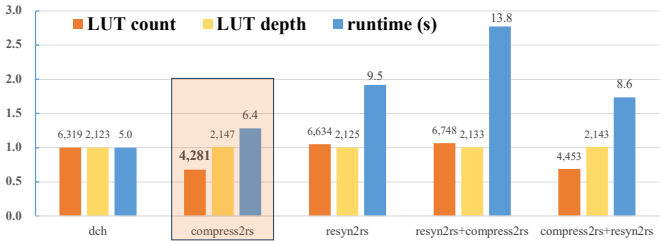
We conduct a case study on existing mapping methods using the *div* circuit from the EPFL benchmark suite [15] to evaluate their impact on LUT mapping performance. Structural choices introduce complexity into the exploration process by increasing the number of equivalent substructures through extensive optimizations. The efficiency of collapsing and decomposition-based methods depends on the collapsing size and decomposition quality. The case study uses optimization strategies provided by the logic synthesis tool ABC [16], with all tests mapped using the area-driven command `if -K 4 -a`. Two main aspects are investigated:

- We generated choice nodes using the `compress` and `compress2` strategies via the `dch` command. These results were then compared with alternative strategies (`resyn2rs`, `compress2rs`), as illustrated in Fig. 1(a).
- We applied LUT collapsing with varying k-values and tested decomposition techniques [17], as shown in Fig. 1(b).

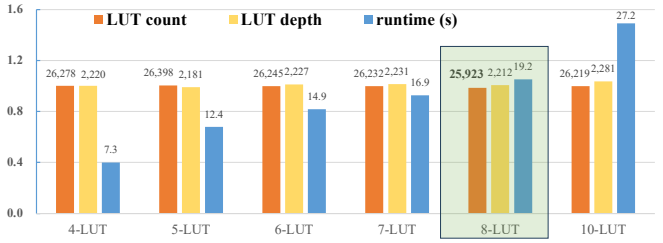
This case study highlights optimization opportunities by analyzing the limitations of both approaches. Based on the experimental results in Fig. 1, we identify two key observations:

Observation 1: More choice nodes do not necessarily lead to better mapping results. The best QoR for the LUT netlist is achieved using choice nodes generated by a single `compress2rs` optimization (highlighted in red). However, too few choice nodes result in suboptimal mapping, highlighting the importance of incorporating an adequate number of choice nodes.

Observation 2: Larger k-values during the collapsing phase do not always improve QoR. Beyond $k > 8$, increasing the k-value does not necessarily yield better decomposition results (highlighted in green), though this varies by design.



(a) structural choice



(b) collapsing and decomposition

Fig. 1: In benchmark *div*, Y-axis shows the ratio of the result, while the X-axis represents the various methods compared.

Larger k-values can lead to redundant node coverage, where duplication negatively impacts QoR.

These observations emphasize the importance of determining the appropriate number of choice nodes and employing a more dynamic collapsing algorithm. Therefore, this study inspires us to develop a new framework that integrates choice directly into the collapsing phase, generating multiple functions for decomposition, with the best sub-LUT selected during the cover phase. This framework addresses challenges related to subcircuit optimization and the synchronization between optimization and mapping processes.

IV. LUT Mapping with Exact Synthesis

In this section, we introduce `ELMap`, which consists of two main components (area-driven LUT mapping (ALM) and exact synthesis of k-LUT network (ELN)). In the mapping stage, ALM generates multiple decomposed subcircuits and optimizes the area of the LUT netlist by dynamically selecting the optimal sub-LUT. ELN uses exact k-LUT synthesis to store multiple optimal sub-LUT netlists for each node.

- ALM begins by traversing the subject graph in topological order. For each node, ALM collapses multiple larger k-LUTs, generating multiple functions to be synthesized. It then uses ELN to generate and store the optimal LUT network for each function. Finally, in reverse topological order, ALM covers the subject graph by visiting the leaves and combining the "best sub-LUT" saved in each visited node, thereby generating the final netlist. [Section IV-A]
- ELN is based on matrix factorization using the STP. ELN starts by generating possible DAG structures that satisfy the topological constraints and obtaining their corresponding STP forms. It then attempts STP-based matrix factorization on these structures. If the factorization is successful, an area-optimal sub-LUT network is obtained. If not, new topological constraints are generated, and the process repeats until a successful factorization is achieved. [Section IV-B]

A. Area-driven LUT mapping

In this subsection, we introduce the area-driven LUT mapping process employed by `ELMap`. In LUT mapping, structural bias is often addressed by either constructing structural choices (choice nodes) or by collaborative mapping through collapsing and decomposition. However, the former approach is limited by excessive runtime, while the latter fails to propagate local optimizations to the global effectively. `ELMap` overcomes these limitations by directly constructing choice nodes during the collapsing phase and utilizing exact synthesis for local replacements to obtain optimal local expressions. Informally, `ELMap` can be viewed as a scheme that integrates choice directly within the mapping algorithm itself.

Algorithm 1: Area-driven LUT mapping algorithm.

Input: Input network $\mathbf{A}$, LUT input size k, number of cut choices n
Output: LUT network $\mathbf{L}$

1 **foreach** $v \in \mathbf{A}$ *in topological order* **do**
2 $\quad f_k, \cdots, f_{k+n} \leftarrow$ `cut_enu`$(k, \cdots, k+n)$;
3 $\quad$ *best_subLUT*$_v \leftarrow$ `exact`$(f_k, \cdots, f_{k+n})$; `// ALgorithm 2`
4 **end**
5 **foreach** $v \in \mathbf{A}$ *in reverse topological order* **do**
6 $\quad$ *best_LUT* $\leftarrow$ `compare_area`(*best_subLUT*$_v$);
7 $\quad \mathbf{L}_v \leftarrow$ `LUT_map`(*best_LUT*, $\mathbf{A}_v$);
8 $\quad$ `visited`(*best_LUT*);
9 **end**

The detailed framework of `ELMap` is presented in Algorithm 1. The process begins by traversing each node in topological order and performing cut enumeration to generate cuts of varying input sizes for each node (Algorithm 1 [Lines 1-2]). Increasing the number of cut choices, n, allows `ELMap` to generate more sub-cuts during the collapsing phase, offering more options during the covering phase. For each cut, the corresponding function is used as input for STP-based exact synthesis, which produces the optimal sub-LUT. These optimal sub-LUTs are stored as the "best subLUT" for each node (Algorithm 1 [Line 3]). Subsequently, we traverse the subject graph in reverse topological order to cover the graph by selecting the optimal LUT area for each node. For each node, we compare all the stored 'best subLUT" and select the optimal one to replace the current node, directly performing a one-to-one LUT mapping on the replaced subgraph (Algorithm 1 [Lines 5-7]). Nodes within each replaced sub-LUT are marked as visited to prevent them from being revisited during subsequent traversals (Algorithm 1 [Line 8]).

B. Exact synthesis of k-LUT network

In exact synthesis, the main task is to find an optimal Boolean chain by determining a DAG structure and assigning Boolean operators to its vertices. When performing SAT-based exact synthesis for k-LUT networks, the large number of inputs for each operator leads to excessive enumerations. This causes the encoded CNF to increase exponentially, making the runtime unpredictable and potentially very slow. To address this challenge, we present an exact synthesis method for k-LUT networks based on the STP, which utilizes matrix factorization to generate optimal Boolean chains. An overview of the algorithm is presented in Algorithm 2. It takes a Boolean

function as input and generates a set of optimal Boolean chains as output. The detailed steps are outlined as follows: (i) **Initialization**: Set the number of logic gates $r = 1$. Generate all possible DAGs with r nodes that satisfy the topological constraints, denoted as *pDAG*s (Algorithm 2 [Lines 1, 3]). (ii) **Matrix Factorization**: For each *pDAG*, attempt STP-based matrix factorization to obtain candidate optimal LUT networks *pLUT*. If factorization fails, terminate the exact synthesis process (Algorithm 2 [Lines 4-11]). (iii) **Verification**: Simulate each *pLUT* using a circuit-based AllSAT solver [18]. If the simulation results match the target function f, the optimal LUT network is found and stored in L_r. If not, increment $r = r + 1$ and return to step (ii) (Algorithm 2 [Lines 12-16]).

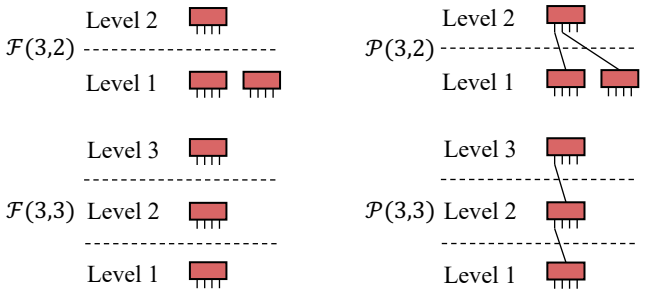
Algorithm 2: k-LUT exact synthesis algorithm.

```
   Input: Boolean function f, the input size of LUT k
   Output: The set of optimal LUT network L
1  Initialize r ← 1;
2  while True do
3      pDAGs ← topo_const(r, k);
4      foreach pDAG ∈ pDAGs do
5          pLUT ← matrix_factor(pDAG);
6          if simulate(pLUT) == f then
7              L_r += pLUT;
8          else
9              continue;
10         end
11     end
12     if L_r ≠ 0 then
13         return L_r;
14     else
15         r ← r + 1;
16     end
17 end
```

As r increases, the search space for enumerating *pDAG* expands, leading to a space complexity of $O(r^2)$. However, since the STP-based method represents logic networks as matrices, it maintains consistent complexity for the exact synthesis of arbitrary networks, making it particularly well-suited for k-LUT networks. Consequently, the exact synthesis of k-LUT networks remains efficient even when the input function has $k+4$ inputs. For instance, synthesizing an optimal 2-LUT for a 6-input function has the same complexity as synthesizing an optimal 6-LUT for a 10-input function using the STP-based method. Additionally, since a failed factorization in step (ii) leads to the immediate termination of that synthesis path, we avoid the lengthy CNF-solving process of SAT-based methods, thereby accelerating the overall synthesis process.

1) LUT-based topological constraints: Topology-based exact synthesis can reduce the enumeration search space by providing additional constraints, thereby decreasing the number of potential graph structures. To apply topology constraints in LUT-based networks, ELN have extended existing *partial DAG* based methods [19] from 2-input gates to k-input gates.

Definition 4. *The fence of k-LUT network is a partition of r nodes, where each level contains at least one node, denoted by $\mathcal{F}_r$. The partial DAG ($\mathcal{P}_r$) is a further specification of fence, where the number of connections for each gate is specified, but the connections of the PIs remain unspecified.*



(a) *fence* $\mathcal{F}_3$ (b) *partial DAG* $\mathcal{P}_3$

Fig. 2: *fence* $\mathcal{F}_3$ and *partial DAG* $\mathcal{P}_3$ of 4-LUT network

To further reduce the number of topological representations, ELN applies additional constraints. First, since we are synthesizing single-output functions, we ensure that each fence has only one node at its highest logic level. Second, ELN limits the difference in the number of nodes between any higher logic level and its immediate lower logic level to not exceed k. This ensures that each operator has at most k-inputs. As illustrated in Fig. 2 with $\mathcal{F}_3$ and $\mathcal{P}_3$, each node's child nodes can be either PIs or nodes from lower logic levels. Furthermore, only the PIs are allowed to be reused throughout the network. This restriction helps avoid unnecessary duplication of logic, thereby reducing overall complexity.

2) LUT-based matrix factorization: By applying LUT-based topological constraints, ELN transforms the *pDAG*s into STP form. It then utilizes STP-based matrix factorization to attempt decomposing the function f into multiple k-input Boolean operators, assigning these operators to the vertices of the *pDAG*s. Any *pDAG* corresponding to f that cannot be decomposed is pruned from the synthesis process. To the best of our knowledge, this is the first work to achieve exact synthesis of k-LUT network. Unlike traditional SAT-based exact synthesis methods, which often suffer from exponential growth in CNF size and unpredictable runtimes, the STP approach represents logic networks as matrices. This representation allows STP to be compatible with arbitrary networks, with the only difference being the dimensions of the structural matrices. By extending previous STP-based matrix factorization methods from 2-input to k-input Boolean operators, ELN realizes exact synthesis of k-LUT network.

For any *pDAG*, we can convert it into STP form. First, suppose the function to be synthesized, f, has n inputs, where the structural matrix $\mathbf{M}_f \in \mathbf{M}_{2\times 2^n}$, defined as:

$$\mathbf{M}_f = \begin{bmatrix} M_1 & M_2 & \cdots & M_{2^n} \end{bmatrix}, \tag{1}$$

where $M_i \in \mathbb{B}$. If the function f already has a *pDAG*, the STP form of the *pDAG* ($\mathbf{M}_\Phi$) can be represented as:

$$\mathbf{M}_\Phi = \mathbf{M}_f x_1 ... x_n = \mathbf{M}_{\Phi_1} \cdots \mathbf{M}_{\Phi_i} \mathbf{M}_r \cdots \mathbf{M}_w x_1 \cdots x_n, \tag{2}$$

where $\mathbf{M}_{\Phi_i}$ is the structural matrix representing Boolean operators for each node in the *pDAG*, $\mathbf{M}_r$ is the power-reducing matrix, and $\mathbf{M}_w$ is the variable swapping matrix.

In equation (2), to ensure that all Boolean variables are sorted in order, we use power-reducing and variable-swapping

matrices, and $\mathbf{M}_{\Phi_i}$ is the matrix to be decomposed. Therefore, after calculation, the matrix equation to be decomposed is:

$$\mathbf{M}_{\Phi_1} \cdots \mathbf{M}_{\Phi_i} \mathbf{M}_r \cdots \mathbf{M}_w = \begin{bmatrix} M_1 & M_2 & \cdots & M_{2^n} \end{bmatrix}. \quad (3)$$

The matrix can be decomposed into several logic matrices by STP-based matrix factorization, that realize the exact synthesis.

Property 2. *Consider the case where $\boldsymbol{M}_{\Phi_1} \cdots \boldsymbol{M}_{\Phi_j} = \begin{bmatrix} M_1 & M_2 & \cdots & M_{2^n} \end{bmatrix}$, and structural matrix of $\boldsymbol{M}_{\Phi_j} \in \boldsymbol{M}_{2\times 2^k}$, indicating that $\boldsymbol{M}_{\Phi_j}$ corresponds to a k-input Boolean operator. $\boldsymbol{M}_{\Phi_j}$ can be decomposed if and only if, after dividing $\boldsymbol{M}_f$ equally into 2^k parts, there exist only two distinct types of sub-matrices.*

Example 1. *1. Assume that the $\boldsymbol{M}_{\Phi_j}$ is a 3-input Boolean operator, and the $\boldsymbol{M}_f = \begin{bmatrix} M_1 & M_2 & M_2 & M_2 & M_2 & M_1 & M_1 & M_2 \end{bmatrix}$, which is divided into $2^3 = 8$ equal parts. We have $\boldsymbol{M}_{\Phi_i}\boldsymbol{M}_{\Phi_j} = \boldsymbol{M}_f$. Then, $\boldsymbol{M}_f$ can be factored as:*

$$\boldsymbol{M}_{\Phi_i} = \begin{bmatrix} M_1 & M_2 \end{bmatrix}, \boldsymbol{M}_{\Phi_j} = \Delta_2[1,2,2,2,2,1,1,2]; \text{ or}$$

$$\boldsymbol{M}_{\Phi_i} = \begin{bmatrix} M_2 & M_1 \end{bmatrix}, \boldsymbol{M}_{\Phi_j} = \Delta_2[2,1,1,1,1,2,2,1];$$

2. If $\boldsymbol{M}_f = \begin{bmatrix} M_1 & M_2 & M_3 & M_2 & M_2 & M_1 & M_1 & M_2 \end{bmatrix}$, and we have $\boldsymbol{M}_{\Phi_i}\boldsymbol{M}_{\Phi_j} = \boldsymbol{M}_f$. The $\boldsymbol{M}_f$ can not be factored.

Property 3. *1. Assume that we have $\boldsymbol{M}_{\Phi_i}\boldsymbol{M}_r = \boldsymbol{M}_f$, and the $\boldsymbol{M}_f = \begin{bmatrix} M_1 & M_2 & M_3 & M_4 \end{bmatrix}$, which is divided into four equal parts. The matrix $\boldsymbol{M}_f$ can be factored as:*

$$\boldsymbol{M}_{\Phi_i} = \begin{bmatrix} M_1 & M_2 & M_x & M_x & M_x & M_x & M_3 & M_4 \end{bmatrix},$$

where the 'M_x' is any Boolean matrix. As a result, the dimensions of $\boldsymbol{M}_f$ go from $(2 \times n)$ to $(2 \times 2n)$.

2. Assume that we have $\boldsymbol{M}_{\Phi_i}\boldsymbol{M}_w = \boldsymbol{M}_f$, and the $\boldsymbol{M}_f = \begin{bmatrix} M_5 & M_6 & M_7 & M_8 \end{bmatrix}$. The matrix $\boldsymbol{M}_f$ can be factored as:

$$\boldsymbol{M}_{\Phi_i} = \begin{bmatrix} M_5 & M_7 & M_6 & M_8 \end{bmatrix},$$

where the second and third parts of $\boldsymbol{M}_f$ are swapped.

V. Experiments and evaluations

We implement our `ELMap` framework in the C++ programming language, with the logic synthesis library mockturtle [21] for cut enumeration. For area-driven k-LUT mapping, we utilized the '`if -K k -a -C 12`' command from ABC. We run all experiments on a Linux server with a 3.20 GHz Apple M1 CPU with 8GB of main memory. We conduct experiments on the EPFL combinational benchmarks suite and verify all results with the '`cec`' command of the ABC to ensure the correctness. It is essential to clarify that our objective is not necessarily to achieve comprehensive improvements in LUT are and depth, but rather to demonstrate the effectiveness of `ELMap`'s choice collapsing algorithm and ELN in the context of area-driven mapping.

A. Area-driven LUT mapping

In this subsection, we compare `ELMap` with state-of-the-art (SOTA) area-oriented LUT mapping methods. To demonstrate `ELMap`'s efficiency and scalability, we set the LUT input size to $k = 6$ and evaluate it across two different scenarios:

- *Choice* [5]: a new method to construct structural choices in a subject graph (`lch;if -K 6 -a -C 12`).
- *Window* [20]: an area-oriented reconvergence-driven window optimization method (`wr;if -K 6 -a -C 12`).

Since the primary goal is to optimize LUT area, we focus on three key metrics: LUT count (LUTs), logic levels of LUT (depth), and runtime efficiency (T(s)) measured in seconds. Exact synthesis can be time-consuming, so to improve efficiency, each optimal LUT sub-network is stored in a hash table with the function as the key. When the same function or its NPN transformation [22] needs to be synthesized again, we simply retrieve the optimal LUT sub-network from the hash table, reducing synthesis time. Additionally, since we perform exact synthesis for multiple functions per node, this process is parallelized, with the number of threads determined by the number of functions being synthesized. In this experiment, we set the number of cut choices $n = 2$. As a result, `ELMap`'s runtime consists of two main components: T_N(s) (the time for NPN transformation) and T_P(s) (the time for parallel synthesis). Improvements are evaluated by comparing the geometric mean improvement ratio (**Impro.**) from optimization to initialization (**Geo.**). For a fair comparison, we also evaluate the average improvement in the ADP (**Average**).

The experimental results for these methods are summarized in TABLE II. The *Choice* and *Window* methods achieve average LUT area improvements of 11.0% and 13.7%, respectively, while *ELMap* deliver a higher improvement of 19.5%. In terms of ADP, *ELMap* outperforms both methods, achieving a 5.8% improvement over *Choice* and 3.6% over *Window*. *ELMap* also shows LUT area improvements for 14 out of 20 benchmarks. When it comes to runtime, *Choice*, which is based on structural choices, is the fastest, producing results in just 0.49 seconds on average. In comparison, *Window* requires 2.2× longer to achieve similar improvements. *ELMap* maintains competitive runtime performance. Specifically, T_N(s) takes 22.29× longer, while T_P(s) requires only 7.49× more runtime than *Choice*. Considering the QoR, the runtime for achieving these substantial improvements is reasonable. While exact synthesis can be computationally intensive, *ELMap* is able to return results quickly for certain functions. For instance, in the *Bar* and *Max* circuits, *ELMap* achieves the fastest synthesis times, demonstrating both its scalability and its ability to match SOTA methods in terms of performance and efficiency.

B. Area-driven remapping

Next, we evaluate the performance of `ELMap` in LUT remapping. Since the complexity of STP-based exact synthesis remains consistent for arbitrary networks, `ELMap` can efficiently perform synthesis even on post-mapping netlists. Here, we evaluate the performance with a LUT input size of $k = 4$. To demonstrate its effectiveness and efficiency, we assess and compare `ELMap` in the following two scenarios:

- *ABC-opt*: a comprehensive area-driven optimization command designed specifically to be LUT mapping-friendly.
- *mfs2*: an area-oriented resynthesis engine for network

TABLE II: Comparison of different 6-LUT mapping methods.

Benchmarks				*Choice* [5]			*Window* [20]			*ELMap*			
name	LUTs	depth	T(s)	LUTs	depth	T(s)	LUTs	depth	T(s)	LUTs	depth	T_N(s)	T_P(s)
Adder	249	121	0.01	252	124	0.06	244	116	0.13	**192**	64	2.66	0.89
Bar	512	4	0.02	512	4	0.16	512	4	0.62	**512**	4	0.00	0.00
Div	23,852	2,113	0.91	6,055	2,059	5.42	8,339	2,070	156.74	**5,762**	2,157	305.10	102.54
Hyp	47,558	8,400	3.97	50,660	8,457	325.74	47,545	8,399	245.42	**45,862**	8,520	1,877.07	630.84
Log2	7,558	159	0.76	7,498	153	5.53	7,571	158	19.90	**7,360**	169	321.46	108.04
Max	724	114	0.05	**701**	103	0.25	719	118	0.49	704	126	0.09	0.03
Multiplier	5,680	126	0.53	5,723	126	2.36	5,708	126	13.38	**5,305**	127	188.11	63.22
Sin	1,449	75	0.13	1,437	75	0.89	1,433	73	1.56	**1,334**	74	300.65	101.04
Sqrt	8,034	3,930	0.48	7,035	3,411	2.85	**4,253**	2,067	33.22	5,004	2,997	128.31	43.12
Square	3,994	122	0.35	4,373	123	1.72	3,884	122	7.25	**3,862**	122	53.31	17.92
Arbiter	2,602	19	0.18	2,602	20	2.72	2,602	19	1.77	**2,595**	19	22.14	7.44
Cavlc	115	7	0.01	114	7	0.05	115	7	0.06	**109**	6	0.22	0.07
Ctrl	29	2	0.00	29	2	0.01	28	2	0.03	**28**	2	0.08	0.03
Dec	287	2	0.00	287	2	0.02	273	2	0.02	**273**	2	27.68	9.30
I2c	354	7	0.01	**308**	7	0.10	339	7	0.09	311	7	15.53	5.22
Int2float	47	5	0.00	**43**	5	0.02	47	5	0.02	47	5	0.76	0.25
Mem_ctrl	11,613	53	0.60	11,095	54	7.67	11,275	53	7.82	**10,754**	53	326.29	109.66
Priority	266	105	0.01	**181**	40	0.10	217	59	0.12	189	85	9.38	3.15
Router	71	15	0.00	74	14	0.03	47	18	0.05	**32**	17	6.35	2.13
Voter	2,541	36	0.25	1,839	20	2.10	**1,643**	19	7.73	1,679	26	11.39	3.83
Geo.	1,082	54	0.07	963	49	0.49	933	49	1.10	871	50	10.92	3.67
Impro.	1.0	1.0		0.890	0.913	1.0	0.863	0.917	2.2	**0.805**	0.938	**22.29**	**7.49**
Average		**1.0**			**0.813**			**0.791**			**0.755**		

Geo.: The geometric mean of the above dataset, **Ratio.**: Average geometric mean improvement (new/old)
Average: Average ADP improvement (new/old).

mapped into k-LUTs [23] (command `mfs2`).

ABC-opt is used as the baseline to generate an optimized 4-LUT netlist, while *mfs2* and *ELMap* are evaluated as resynthesis engines for remapping. The experiments are reported in TABLE III. *mfs2* achieves only a 0.6% improvement in LUT area, and it shows improvement in just 11 out of 20 benchmarks, with minimal gains in each case. In contrast, for 4-LUT remapping, exact synthesis yields more substantial results. Consequently, *ELMap* improves LUT area by 17.6% and achieves a 2.4% improvement in the ADP. In terms of runtime, *mfs2* is extremely fast, producing results in just 0.04 seconds, but this is largely due to its inability to find significant optimization opportunities. In comparison, *ELMap* demonstrates strong runtime performance for 4-LUT remapping. Specifically, T_N(s) requires 1.23 seconds, and T_P(s) takes only 0.45 seconds, which is just 0.5× of the runtime of *ABC-opt*, while delivering more substantial improvements. These findings highlight `ELMap`'s ability to effectively enhance LUT area, establishing it as a robust and efficient optimization approach for FPGA LUT mapping.

TABLE III: Comparison of different remapping method.

Benchmark	*ABC-opt*			*mfs2* [23]			*ELMap*			
	LUTs	Depth	T(s)	LUTs	Depth	T(s)	LUTs	Depth	T_N(s)	T_P(s)
Adder	255	127	0.14	**255**	127	0.01	256	127	0.04	0.01
Bar	1,280	10	0.45	1,280	10	0.15	**896**	7	0.63	0.23
Div	7,656	2,150	9.80	7,656	2,150	0.02	**7,190**	1,629	16.44	5.96
Hyp	60,544	8,593	82.87	60,526	8,593	1.54	**59,502**	8,770	112.73	40.88
Log2	9,982	160	8.10	9,981	160	0.30	**8,890**	181	13.63	4.94
Max	909	149	0.56	909	149	0.01	**893**	242	0.94	0.34
Multiplier	7,282	130	6.07	7,274	130	0.15	**6,161**	187	3.91	1.42
Sin	1,922	86	1.44	1,922	86	0.06	**1,757**	98	0.76	0.28
Sqrt	4,283	2,110	4.72	4,281	2,109	0.02	**3,638**	2,671	3.05	1.11
Square	5,252	124	4.12	5,250	124	0.32	**5,134**	167	1.26	0.46
Arbiter	2,118	15	0.83	2,113	15	0.18	**1,422**	27	0.34	0.12
Cavlc	297	10	0.21	294	10	0.03	**214**	13	1.10	0.40
Ctrl	47	5	0.04	**47**	5	0.00	48	5	0.32	0.12
Dec	288	2	0.08	288	2	0.01	**286**	3	0.09	0.03
I2c	429	8	0.22	416	8	0.02	**284**	11	2.65	0.96
Int2float	82	8	0.06	82	8	0.01	**74**	9	1.36	0.49
Mem_ctrl	14,771	52	8.35	14,411	52	0.79	**11,153**	80	3.55	1.29
Priority	192	29	0.12	191	29	0.01	**171**	36	0.39	0.14
Router	90	11	0.05	86	11	0.00	**37**	9	0.17	0.06
Voter	2,518	24	2.19	2,518	24	0.10	**1,770**	30	1.02	0.37
Geo.	1,247	55	0.84	1,240	55	0.04	1,028	65	1.23	0.45
Impro.	1.0	1.0	1.0	0.994	1.000	0.1	**0.824**	1.184	**1.5**	**0.5**
Average		1.0			0.994			**0.976**		

ABC-opt: `share;compress2rs;dc2;&get -n;&st -s; if -K 4 -a -C 12;`

VI. Conclusion

The paper introduces `ELMap`, an area-driven LUT mapping framework designed to address structural bias and improve the efficiency of FPGA technology mapping. `ELMap` integrates the concept of choice into the collapsing phase, maximizing the transfer of local optimizations to global optimization. Additionally, it incorporates a novel STP-based exact synthesis for k-LUT networks, enabling efficient generation of optimal sub-LUT networks. Experiments on the EPFL benchmark suite indicate that `ELMap` significantly outperforms SOTA methods, achieving an average LUT area improvement by 8.5% and ADP improvement by 5.8% in 6-LUT mapping, and an average LUT area improvement of 17.6% and ADP improvement by 2.4% in 4-LUT remapping. `ELMap` proves to be a scalable and effective solution for both area-driven mapping and remapping scenarios in FPGA synthesis.

VII. Acknowledgements

This work was supported partly by National Natural Science Foundation of China (NSFC) Research Projects under Grant 92373207, partly by the State Key Laboratory of Integrated Chips and Systems at Fudan University under Grant SKLICS-Z202404, partly by NSFC under Grant 62274100, partly by Key Research and Development Program of Zhejiang Province under Grant 2024C01111, and partly by Ningbo City under Grant 2023Z071.

REFERENCES

[1] M. Kubica, A. Opara, and D. Kania, *Technology Mapping for LUT-based FPGA*. Springer, 2021, vol. 1.

[2] S. Ray, A. Mishchenko, N. Een, R. Brayton, S. Jang, and C. Chen, "Mapping into lut structures," in *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2012, pp. 1579–1584.

[3] V. Manohararajah, S. D. Brown, and Z. G. Vranesic, "Heuristics for area minimization in lut-based fpga technology mapping," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 11, pp. 2331–2340, 2006.

[4] A. Mishchenko, R. Brayton, and S. Jang, "Global delay optimization using structural choices," in *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*, 2010, pp. 181–184.

[5] A. Grosnit, M. Zimmer, R. Tutunov, X. Li, L. Chen, F. Yang, M. Yuan, and H. Bou-Ammar, "Lightweight structural choices operator for technology mapping," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[6] A. Mishchenko, S. Chatterjee, and R. Brayton, "Improvements to technology mapping for lut-based fpgas," in *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*, 2006, pp. 41–49.

[7] W. J. Haaswijk, M. Soeken, L. Amaru, P.-E. Gaillardon, and G. De Micheli, "Lut mapping and optimization for majority-inverter graphs," in *Proceedings of the 25th International Workshop on Logic & Synthesis (IWLS)*, 2016.

[8] L. Machado and J. Cortadella, "Support-reducing decomposition for fpga mapping," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 1, pp. 213–224, 2018.

[9] A. T. Calvino, G. De Micheli, A. Mishchenko, and R. Brayton, "Enhancing delay-driven lut mapping with boolean decomposition," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[10] J. Cong, C. Wu, and Y. Ding, "Cut ranking and pruning: Enabling a general and efficient fpga mapping solution," in *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, 1999, pp. 29–35.

[11] H. Riener, A. Mishchenko, and M. Soeken, "Exact dag-aware rewriting," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 732–737.

[12] W. Haaswijk, E. Testa, M. Soeken, and G. De Micheli, "Classifying functions with exact synthesis," in *2017 IEEE 47th International Symposium on Multiple-Valued Logic (ISMVL)*. IEEE, 2017, pp. 272–277.

[13] W. Xiao, S. Han, Y. Yang, S. Yang, C. Zheng, J. Chen, T. Liang, L. Li, and W. Qian, "Minitntk: An exact synthesis-based method for minimizing transistor network," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 01–09.

[14] H. Pan, Y. Xia, L. Wang, and Z. Chu, "Semi-tensor product based exact synthesis for logic rewriting," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[15] EPFL, "EPFL combinational benchmark suite," *https://github.com/lsils/benchmarks*, 2023.

[16] A. Mishchenko, "ABC: System for sequential logic synthesis and formal verification," *https://github.com/berkeley-abc/abc*, 2023.

[17] Z. Chu, M. Soeken, Y. Xia, L. Wang, and G. De Micheli, "Advanced functional decomposition using majority and its applications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 8, pp. 1621–1634, 2019.

[18] H. Pan and Z. Chu, "Exact synthesis based on semi-tensor product circuit solver," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2023, pp. 1–6.

[19] W. Haaswijk, M. Soeken, A. Mishchenko, and G. De Micheli, "Sat-based exact synthesis: Encodings, topology families, and parallelism," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 4, pp. 871–884, 2019.

[20] H. Riener, S.-Y. Lee, A. Mishchenko, and G. De Micheli, "Boolean rewriting strikes back: Reconvergence-driven windowing meets resynthesis," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2022, pp. 395–402.

[21] EPFL, "Mockturtle," *https://github.com/lsils/mockturtle*, 2024.

[22] A. Petkovska, M. Soeken, G. De Micheli, P. Ienne, and A. Mishchenko, "Fast hierarchical NPN classification," in *2016 International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2016.

[23] A. Mishchenko, R. Brayton, J.-H. R. Jiang, and S. Jang, "Scalable don't-care-based logic optimization and resynthesis," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 4, no. 4, pp. 1–23, 2011.